Experiences with Software Quality Metrics in the EMI middleware

Experiences with Software Quality Metrics in the EMI middleware

M Alandes[1] E M Kenny[2], D Meneses[1] and G Pucciani[1]

[1] European Organization for Nuclear Research, CERN CH-1211, Genève 23, Switzerland

[2] Trinity College Dublin, College Green, Dublin 2, Ireland

E-mail: maria.alandes.pradillo@cern.ch

Abstract. The EMI Quality Model has been created to define, and later review, the EMI (European Middleware Initiative) software product and process quality. A quality model is based on a set of software quality metrics and helps to set clear and measurable quality goals for software products and processes. The EMI Quality Model follows the ISO/IEC 9126 Software Engineering – Product Quality to identify a set of characteristics that need to be present in the EMI software. For each software characteristic, such as portability, maintainability, compliance, etc, a set of associated metrics and KPIs (Key Performance Indicators) are identified. This article presents how the EMI Quality Model and the EMI Metrics have been defined in the context of the software quality assurance activities carried out in EMI. It also describes the measurement plan and presents some of the metrics reports that have been produced for the EMI releases and updates. It also covers which tools and techniques can be used by any software project to extract "code metrics" on the status of the software products and "process metrics" related to the quality of the development and support process such as reaction time to critical bugs, requirements tracking and delays in product releases.

1. Introduction

According to the standard ISO 9001, the quality of something can be determined by comparing a set of inherent characteristics with a set of requirements. If those inherent characteristics meet all requirements, high or excellent quality is achieved. If those characteristics do not meet all requirements, a low or poor level of quality is achieved.

Quality is, therefore, a question of degree. As a result, the central quality question is: How well does this set of inherent characteristics comply with this set of requirements? In short, the quality of something depends on a set of inherent characteristics and a set of requirements and how well the former complies with the latter.

Software Quality Engineering (SQE) is the process that evaluates, assesses, and improves the quality of software. Software quality is often defined as the degree to which software meets requirements for reliability, maintainability, transportability, etc, as contrasted with functional, performance, and interface requirements that are satisfied as a result of software engineering.

A quality model helps evaluating the software product and process quality. It helps to set quality goals for software products and processes.

The European Middleware Initiative project (EMI) [1] is comprised of 28 software development teams, called product teams (PTs), who develop the 56 EMI software products. EMI PTs are coming from major middleware providers like ARC, dCache, gLite and UNICORE who have been developing software in the grid domain for the past several years. The EMI Quality Model [3] helps to evaluate the quality of the EMI software products taking into account the existing working methods and tools of the PTs.

2. EMI Quality Model

The EMI Quality Model uses the *ISO/IEC 9126 Software Engineering – Product Quality* standard to identify a set of characteristics that need to be present in EMI software products and processes to be able to meet EMI quality requirements. EMI quality requirements are based on Distributed Computer Infrastructure's (DCI) quality requirements, like the UMD (Unified Middleware Distribution) Quality Criteria [8] from the EGI (European Grid Infrastructure) project [2], and internal project objectives that influence qualitative aspects of the EMI software, as specified in the EMI Description of Work [9].

2.1. Quality Requirements

EMI quality requirements are defined by taking into account internal EMI quality criteria and quality criteria coming from EMI users, like EGI as defined in the UMD Quality Criteria.
UMD Quality Criteria is summarized below:

1. Functional Description: all products must provide a document with a brief functional description of the product.
2. Release Notes: all products must provide a document with the release notes.
3. User Documentation: all products must provide a document describing how to use it.
4. Online help (man pages): all products with end user command line tools must include man pages or online help.
5. API Documentation: public API of products must be documented.
6. Administrator Documentation: products must provide an administrator guide describing installation, configuration and operation of the system.
7. Service Reference Card: for each of the services that a product runs, document its characteristics with a reference card.
8. Software License: products must have a compatible license for using them in the EGI infrastructure.
9. Release changes testing: changes in a release of a product must be tested.
10. Source Code Availability: products should provide their source code.
11. Source Distribution: technology providers should provide buildable source distributions of products.
12. Binary Distribution: products must be available in the native packaging format of the supported platform.
13. Backwards compatibility: minor/revision releases of a product must be backwards compatible.
14. Service control and status: services run by the product must provide a mechanism for starting, stopping and querying the status of services.
15. Log files: all services should create log files where the service administrator can trace most relevant actions taken.
16. Service Reliability: services must maintain a good performance and reliability over long periods of time with normal operations.

17. Service Robustness: services should not produce unexpected results or become uncontrollable when taxed beyond normal capacity.
18. Automatic configuration: products that provide tools for configuration that cover typical deployments must assure tools work as documented.
19. World writeable files: products must not create world-writeable files or directories.
20. Directory traversal attacks testing: products should assure that directory traversal exploits are not possible using their interfaces.
21. Incident Tracking: EMI must enroll as 3rd level support in the EGI Helpdesk.

EMI internal quality criteria are defined as complementary criteria to that of the EMI users and it is based from the EMI project objectives as described in the EMI Description of Work.

- EMI Objective 1: Simplify and organize the different middleware services implementations by delivering a streamlined, coherent, tested and standard compliant distribution able to meet and exceed the requirements of EGI, PRACE and other distributed computing infrastructures and their user communities.
- EMI Objective 2: Increase the interoperability, manageability, usability and efficiency of the services by developing or integrating new functionality as needed following existing and new requirement of EGI, PRACE and other infrastructures and their user communities.
- EMI Objective 3: Support efficient, reliable operations of EGI, PRACE and other DCIs by reactively and proactively supporting and maintaining the middleware distribution and providing users with increasingly user-friendly, maintainable, reliable, stable, and scalable software.
- EMI Objective 4: Strengthen the participation and support for user communities in the definition and evolution of middleware services by promoting the EMI achievements, objectives and plans, and move the EMI middleware towards a more sustainable model by expanding the collaboration with national and international research agencies, scientific research programs and with industrial providers.

2.2. Quality Characteristics

Once EMI software product quality requirements are defined, the software product quality characteristics which define the quality requirements can be determined. In order to do this, quality characteristics from ISO/IEC 9126 are analysed within the context and objectives of the EMI project. Two more characteristics have been taken into account as well: EPEL and Debian repositories compliance. One of the main goals of the EMI project is to provide a sustainable model at the end of the project. Being able to deliver middleware packages into EPEL and Debian repositories is fundamental to move towards an open source like model where middleware developers can distribute their packages through EPEL and Debian to their user community.

For each of the defined characteristics, the following areas have been analysed:

- **Importance for EMI:** the aim is to determine how much attention should be paid to the characteristic to meet EMI quality requirements. Possible values are High, Medium and Low.
- **Risks**: the aim is to determine the possible effects of the problems caused when the characteristic is not present in the EMI software.
- **Indicators**: the aim is to determine which indicators can be used to make the presence of the characteristic visible.
- **Measures**: the aim is to determine which measures are necessary to control the characteristic.

Table 1 shows a summary of the thorough analysis of each characteristic:

Characteristic	Subcharacteristic	*Quality Requirement*	Importance
Functionality	Suitability	*EMI Objective 2* *UMD 14,15,18,19,20,21*	High
	Accuracy	*EMI Objective 3*	Low
	Interoperability	*EMI Objective 2*	High
	Security	*EMI Objective 1* *UMD 19,20*	High
	Functionality compliance	*EMI Objective 1*	High
Reliability	Maturity	*EMI Objective 3* *UMD 16,17*	High
	Recoverability	*EMI Objective 3* *UMD 16,17*	Medium
Usability	Understandability	*EMI Objective 1* *UMD 1,2,3,4,5,6,7,10*	High
	Operability	*EMI Objective 3* *UMD 14,18*	High
Efficiency	Resource utilisation	*EMI Objective 3* *UMD 17*	Low
Maintainability	Changeability	*EMI Objective 2*	High
	Stability	*EMI Objective 3* *UMD 13,16,17*	High
	Testability	*EMI Objective 1* *UMD 9*	High
	Maintainability Compliance	*EMI Objective 3*	High
Portability	Adaptability	*EMI Objective 1 and 4*	High
	Installability	*EMI Objective 1* *UMD 8,11,12*	High
	Replaceability	*Objective 2*	Medium
	Co-existence	*Objective 1*	High
EPEL and Debian Compliance		*EMI sustainability plan objective.*	High

Table 1 - Quality Characteristics vs. Quality Requirements

Identifying the characteristics that need to be present in the software to meet the existing quality requirements, and understanding what we needs to done to measure whether they are present or not, is the basis of the quality model. In the next section, we define which metrics are needed to be able to measure the presence of the software characteristics.

3. Metrics and KPIs

EMI metrics are calculated to measure the presence of those quality characteristics evaluated as highly important for the EMI middleware. KPIs are also calculated. They are defined in the EMI Description of Work and they are normally calculated every quarter. KPIs also relate to the analysed software characteristics.

Table 2 presents a summary of the metrics that are needed to evaluate each quality characteristic.

Metrics	**Quality characteristic**
Number of technical objectives	Suitability
Number of user requirements	
Number of development tasks.	
Number of GGUS tickets related to lack of accuracy.	Accuracy
KPI KJRA1.2 *Number of Interoperable Interface Usage*.	Interoperability
Number of EMI security assessments.	Security
Number of fixed security vulnerabilities.	
Number of EGI SVG tickets still opened after the defined deadline.	
KPI KJRA1.1 *Number of Adopted Open Standard Interfaces*	Functionality compliance
KPI KSA1.4 *Number of urgent changes*.	Maturity
Number of services providing high-availability setups.	Recoverability
Number of missing mandatory documents.	Understandability
Number of EMT tasks tracking documentation issues.	
Number of services providing service control and status mechanisms.	Operability
Number of services providing configuration tools.	
Number of GGUS tickets related to resource utilisation issues.	Resource utilisation
KPI KSA1.2 *Incident Resolution Time*.	Changeability
KPI KSA1.5 *Change Application Time*	
KPI KSA1.1 *Number of incidents*	Stability
KPI KSA1.3 *Number of problems*.	
Number of Test Plans.	Testability
Number of Test Reports per released EMI software product.	
Number of mandatory tests per EMI software product.	
Number of RfCs tracking a defect with an associated regression test.	
Number of RfCs tracking a new feature with an associated functionality test.	
Number of development tasks tracking a new feature with an associated functionality test.	
Number of passed certification checks.	
KPI KJRA1.3 *Number of Reduced lines of code.*	Maintainability compliance
KPI KJRA1.4 *Number of reduced released products*.	
Number of supported platforms.	Adaptability
Number of standard installation tools per supported platform.	Installability
Number of standard package formats per supported platforms per released product.	
KPI KNA2.4 *Number of EMI products included in standard repositories, Linux distributions, etc*	Co-existence
RPMlint and Lintian	EPEL and Debian compliance

Table 2 - EMI Metrics

Metrics in the EMI project are described in detail in the EMI Metrics Specification and are divided into:

- Process related metrics: they are related to software changes and user support. They use information stored in the tracking tools and user support tools, as explained in the upcoming sections.
- Product related metrics: they are related to the software itself, like RPMlint or SLOC.

3. Tools

To cope with the task of producing reports in a regular fashion including all the many metrics that need to be calculated, a high level of automation is needed. Moreover, the sources of information within the EMI project are very heterogeneous: different tracking tools and programming languages are used by the product teams. A common layer on top of the existing tools is also necessary to be able to automate the calculation of metrics in an easy way.

The following subsections describe the tools and dashboards that have been developed by the QA team in order to automate the generation of metrics reports.

3.1. ETICS plugins

ETICS [10] is the tool that provides a build and packaging infrastructure for the EMI project. The ETICS plugin framework provides the ability of collecting metrics during build and test execution. RPMlint (RPM common problems) and SLOC (number of lines of Code) are some examples of the used ETICS plugins. Figure 1 and figure 2 show a graphical representation of RPMlint and SLOC measurements taken for EMI software products.

The metrics plugins are executed during some of the build steps in ETICS. The data generated by the plugins is stored in the ETICS repository. Once the data is stored, it can be queried at any time. In order to generate charts or statistics, the ETICS repository is queried using a web service and converting the data into a specific XML format. A chart generation framework is used later on in the process. The chart generation framework processes the XML data in several ways as defined by its extensions, producing the datasets for the different charts. Its extensibility makes it easy to produce any new charts from the data collected during the builds.

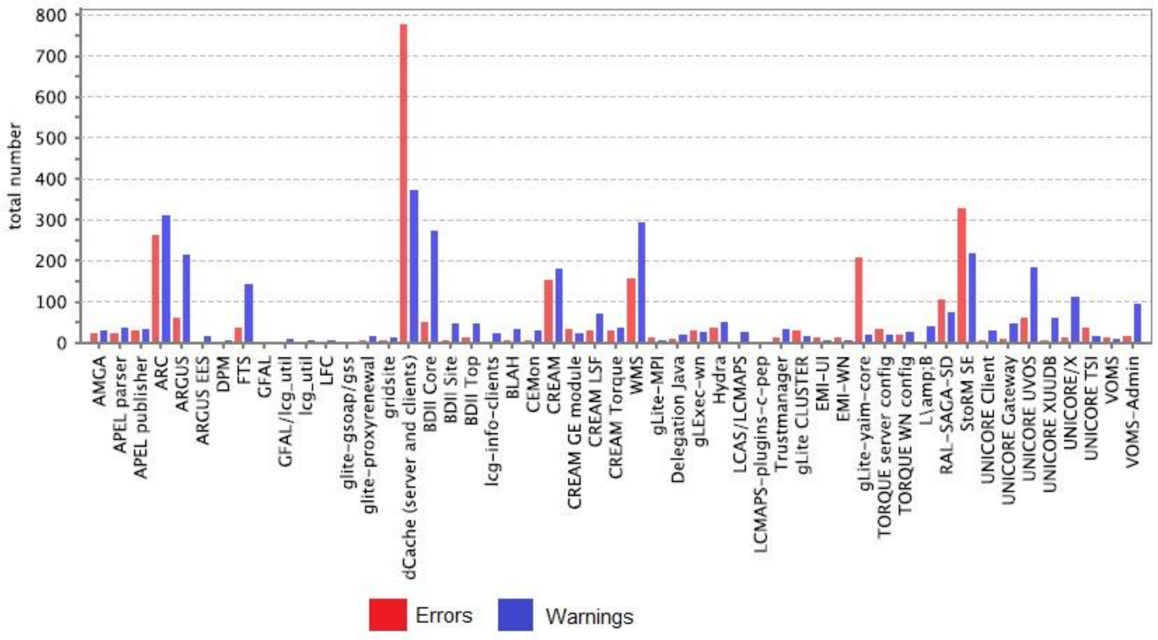


Figure 1 – RPMlint errors and warnings per EMI software product

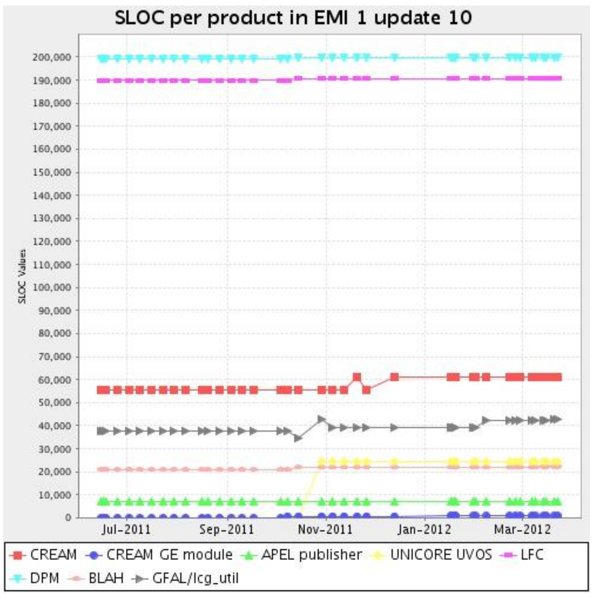
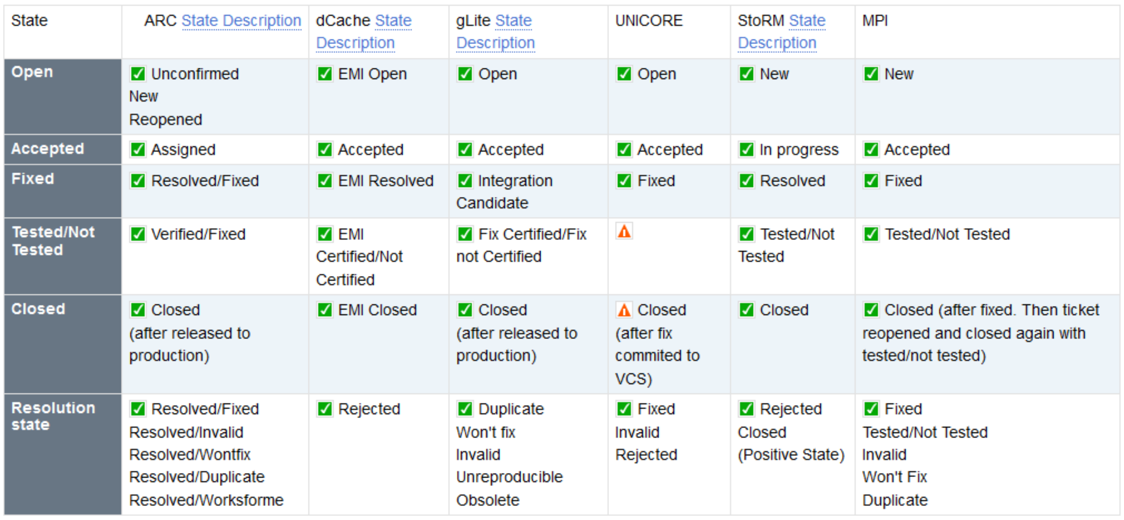


Figure 2 – SLOC trend for the EMI Software products released in EMI 1 Update 10

3.2. RfC Dashboard

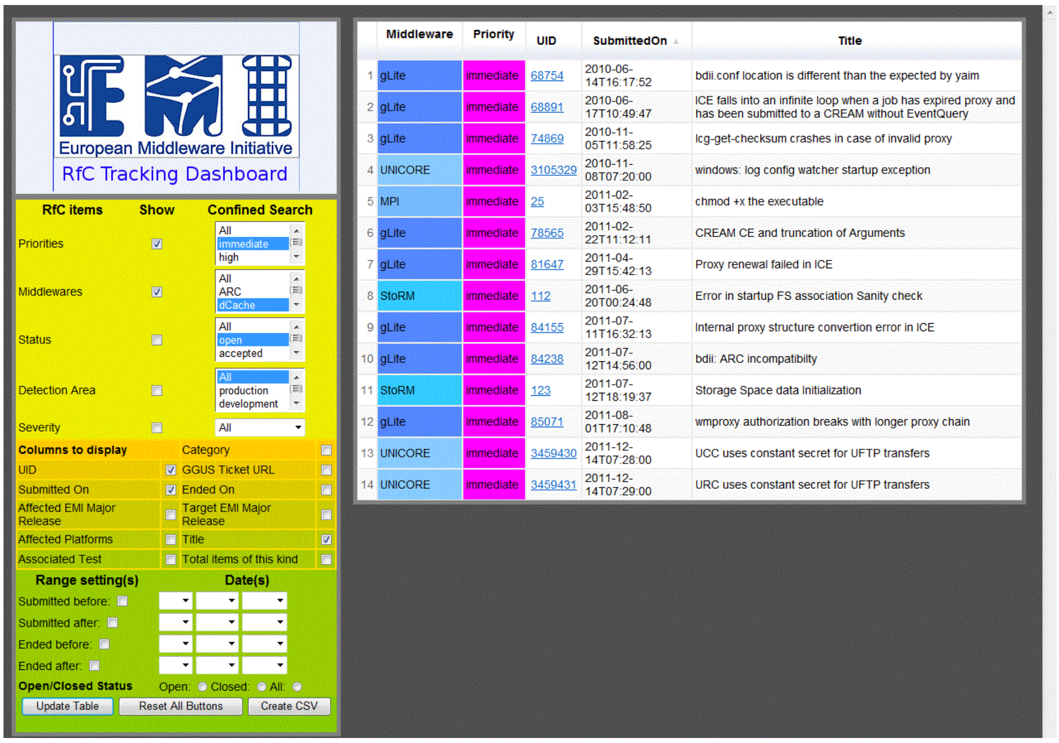
The RfC (Request for Change) Dashboard [5] is a tool that offers a unique entry point to track software changes, like defects and new features, for all EMI products. EMI product teams use different tracking tools of their choice. The EMI QA policies [11] define a common release process that is followed by all product teams, including which is the minimum set of states that need to be present in the tracking tools. Figure 3 shows how the different tracking tools of the middleware providers involved in EMI map to the states defined by the policies.



State	ARC State Description	dCache State Description	gLite State Description	UNICORE	StoRM State Description	MPI
Open	✓ Unconfirmed New Reopened	✓ EMI Open	✓ Open	✓ Open	✓ New	✓ New
Accepted	✓ Assigned	✓ Accepted	✓ Accepted	✓ Accepted	✓ In progress	✓ Accepted
Fixed	✓ Resolved/Fixed	✓ EMI Resolved	✓ Integration Candidate	✓ Fixed	✓ Resolved	✓ Fixed
Tested/Not Tested	✓ Verified/Fixed	✓ EMI Certified/Not Certified	✓ Fix Certified/Fix not Certified	⚠	✓ Tested/Not Tested	✓ Tested/Not Tested
Closed	✓ Closed (after released to production)	✓ EMI Closed	✓ Closed (after released to production)	⚠ Closed (after fix commited to VCS)	✓ Closed	✓ Closed (after fixed. Then ticket reopened and closed again with tested/not tested)
Resolution state	✓ Resolved/Fixed Resolved/Invalid Resolved/Wontfix Resolved/Duplicate Resolved/Worksforme	✓ Rejected	✓ Duplicate Won't fix Invalid Unreproducible Obsolete	✓ Fixed Invalid Rejected	✓ Rejected Closed (Positive State)	✓ Fixed Tested/Not Tested Invalid Won't Fix Duplicate

Figure 3 - Mapping of product team tracker states

The different tracking tools export their data in an XML file that is used by the RfC Dashboard. In this way the software changes of all products can be tracked in a single place and metrics can be calculated for all of them. The RfC Dashboard uses PHP and HTML forms to provide input to a python based query engine. The query engine produces tabulated results based on the XML files. Figure 4 is a snapshot of the RfC Dashboard that shows a query retrieving results from different product teams.



	Middleware	Priority	UID	SubmittedOn	Title
1	gLite	immediate	68754	2010-06-14T16:17:52	bdii.conf location is different than the expected by yaim
2	gLite	immediate	68891	2010-06-17T10:49:47	ICE falls into an infinite loop when a job has expired proxy and has been submitted to a CREAM without EventQuery
3	gLite	immediate	74869	2010-11-05T11:58:25	lcg-get-checksum crashes in case of invalid proxy
4	UNICORE	immediate	3105329	2010-11-08T07:20:00	windows: log config watcher startup exception
5	MPI	immediate	25	2011-02-03T15:48:50	chmod +x the executable
6	gLite	immediate	78565	2011-02-22T11:12:11	CREAM CE and truncation of Arguments
7	gLite	immediate	81647	2011-04-29T15:42:13	Proxy renewal failed in ICE
8	StoRM	immediate	112	2011-06-20T00:24:48	Error in startup FS association Sanity check
9	gLite	immediate	84155	2011-07-11T16:32:13	Internal proxy structure convertion error in ICE
10	gLite	immediate	84238	2011-07-12T14:56:00	bdii: ARC incompatibilty
11	StoRM	immediate	123	2011-07-12T18:19:37	Storage Space data Initialization
12	gLite	immediate	85071	2011-08-01T17:10:48	wmproxy authorization breaks with longer proxy chain
13	UNICORE	immediate	3459430	2011-12-14T07:28:00	UCC uses constant secret for UFTP transfers
14	UNICORE	immediate	3459431	2011-12-14T07:29:00	URC uses constant secret for UFTP transfers

Figure 4 - EMI RfC Dashboard

Figure 5 is a graphic showing the number of reported problems per EMI product classified per priority. Metrics like this one can be easily generated after the data collected in the RfC Dashboard.

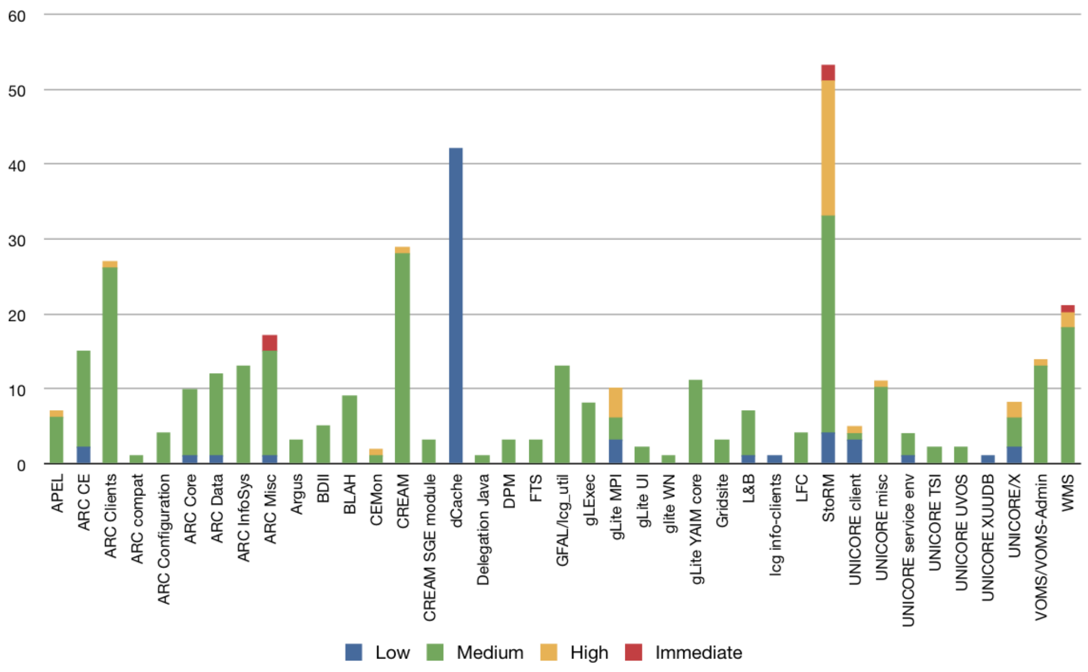


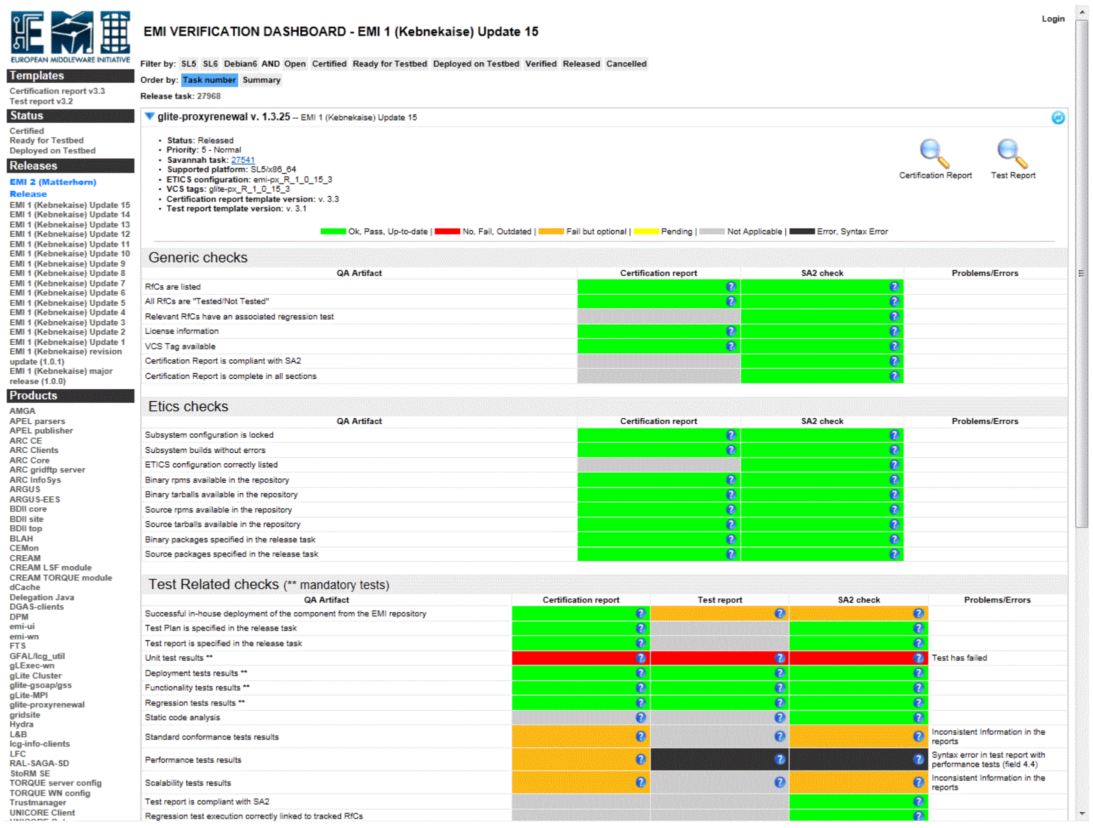
Figure 5 - Number of Problems per EMI product classified per priority

3.3. Verification Dashboard

The EMI Verification Dashboard [6] automates the verification of EMI releases in terms of quality. EMI releases are verified against the Production Release Criteria [7], which is the set of mandatory criteria defined in the EMI QA policies [11]. Most of the checks are done automatically, but some others, like the documentation review, are done manually. The EMI verification dashboard displays all this information for each EMI release assisting the quality control team to carry out this task. Figure 6 presents a snapshot of the dashboard.

The EMI verification dashboard retrieves information from the Savannah tool (where EMI releases are tracked) and presents different views to users and other applications, like the RfC Dashboard. It is written in Python, using the web framework Django to present the information through a web interface using standard HTML+ CSS. As a storage backend, it uses the MySQL database, but due to Django's abstraction other databases could be used. For the information retrieval and parsing from Savannah, the BeautifulSoup library is used, solving the problems of inconsistencies in the source HTML code.

The EMI Verification Dashboard is not only a very useful tool to automate quality control checks but also a way to easily calculate process and product metrics in various aspects of the software. Figure 7 show graphics on metrics calculated thanks to the data stored in the Verification Dashboard. Thanks to the Dashboard it is possible to calculate statistics and trend diagrams on testing, packaging, documentation and certification.



EMI VERIFICATION DASHBOARD - EMI 1 (Kebnekaise) Update 15

Templates
Certification report v3.3
Test report v3.2

Status
Certified
Ready for Testbed
Deployed on Testbed

Releases
EMI 2 (Matterhorn) Release
EMI 1 (Kebnekaise) Update 15
EMI 1 (Kebnekaise) Update 14
EMI 1 (Kebnekaise) Update 13
EMI 1 (Kebnekaise) Update 12
EMI 1 (Kebnekaise) Update 11
EMI 1 (Kebnekaise) Update 10
EMI 1 (Kebnekaise) Update 9
EMI 1 (Kebnekaise) Update 8
EMI 1 (Kebnekaise) Update 7
EMI 1 (Kebnekaise) Update 6
EMI 1 (Kebnekaise) Update 5
EMI 1 (Kebnekaise) Update 4
EMI 1 (Kebnekaise) Update 3
EMI 1 (Kebnekaise) Update 2
EMI 1 (Kebnekaise) Update 1
EMI 1 (Kebnekaise) revision update (1.0.1)
EMI 1 (Kebnekaise) major release (1.0.0)

Products
AMGA
APEL parsers
APEL publisher
ARC CE
ARC Clients
ARC Core
ARC gridftp server
ARC InfoSys
ARGUS
ARGUS-EES
BDII core
BDII site
BDII top
BLAH
CEMon
CREAM
CREAM LSF module
CREAM TORQUE module
dCache
Delegation Java
DGAS-clients
DPM
emi-ui
emi-wn
FTS
GFAL/lcg_util
gLExec-wn
gLite Cluster
glite-gsoap/gss
gLite-MPI
glite-proxyrenewal
gridsite
Hydra
L&B
lcg-info-clients
LFC
RAL-SAGA-SD
StoRM SE
TORQUE server config
TORQUE WN config
Trustmanager
UNICORE Client

Filter by: SL5 SL6 Debian6 AND Open Certified Ready for Testbed Deployed on Testbed Verified Released Cancelled
Order by: Task number Summary
Release task: 27968

glite-proxyrenewal v. 1.3.25 -- EMI 1 (Kebnekaise) Update 15

- **Status:** Released
- **Priority:** 5 - Normal
- **Savannah task:** 27541
- **Supported platform:** SL5/x86_64
- **ETICS configuration:** emi-px_R_1_0_15_3
- **VCS tags:** glite-px_R_1_0_15_3
- **Certification report template version:** v. 3.3
- **Test report template version:** v. 3.1

🟩 Ok, Pass, Up-to-date | 🟥 No, Fail, Outdated | 🟧 Fail but optional | 🟨 Pending | ▨ Not Applicable | ⬛ Error, Syntax Error

Generic checks

QA Artifact	Certification report	SA2 check	Problems/Errors
RfCs are listed	🟩	🟩	
All RfCs are "Tested/Not Tested"	🟩	🟩	
Relevant RfCs have an associated regression test	▨	🟩	
License information	🟩	🟩	
VCS Tag available	🟩	🟩	
Certification Report is compliant with SA2	▨	🟩	
Certification Report is complete in all sections	▨	🟩	

Etics checks

QA Artifact	Certification report	SA2 check	Problems/Errors
Subsystem configuration is locked	🟩	🟩	
Subsystem builds without errors	🟩	🟩	
ETICS configuration correctly listed	▨	🟩	
Binary rpms available in the repository	🟩	🟩	
Binary tarballs available in the repository	🟩	🟩	
Source rpms available in the repository	🟩	🟩	
Source tarballs available in the repository	🟩	🟩	
Binary packages specified in the release task	🟩	🟩	
Source packages specified in the release task	🟩	🟩	

Test Related checks (** mandatory tests)

QA Artifact	Certification report	Test report	SA2 check	Problems/Errors
Successful in-house deployment of the component from the EMI repository	🟩	🟧	🟧	
Test Plan is specified in the release task	🟩	▨	🟩	
Test report is specified in the release task	🟩	▨	🟩	
Unit test results **	🟥	🟥	🟥	Test has failed
Deployment tests results **	🟩	🟩	🟩	
Functionality tests results **	🟩	🟩	🟩	
Regression tests results **	🟩	🟩	🟩	
Static code analysis	▨	▨	🟩	
Standard conformance tests results	🟧	▨	🟧	Inconsistent Information in the reports
Performance tests results	🟧	⬛	⬛	Syntax error in test report with performance tests (field 4.4)
Scalability tests results	🟧	▨	🟧	Inconsistent Information in the reports
Test report is compliant with SA2	▨	▨	🟩	
Regression test execution correctly linked to tracked RfCs	▨	▨	🟩	

Figure 6 - EMI Verification Dashboard

4. Measurement plan

EMI major releases have five major phases from the quality measurement perspective:

- EMI major release planning: it's when the different work area plans containing the technical objectives to be achieved are written and user requirements are gathered.
- EMI major release software coding and testing: it is the preparation of the release. The result is a set of packages per software product with the corresponding test and certification reports.
- EMI major release availability: it is the moment when software documentation and software repositories are ready and all the new required functionality is available for the users.
- EMI major release maintenance: Once the release has made available, software changes to fix defects or introduce new features are released as long as the EMI major release is supported.
- EMI major release user support: Once the release has made available, user support is provided as long as the EMI major release is supported.

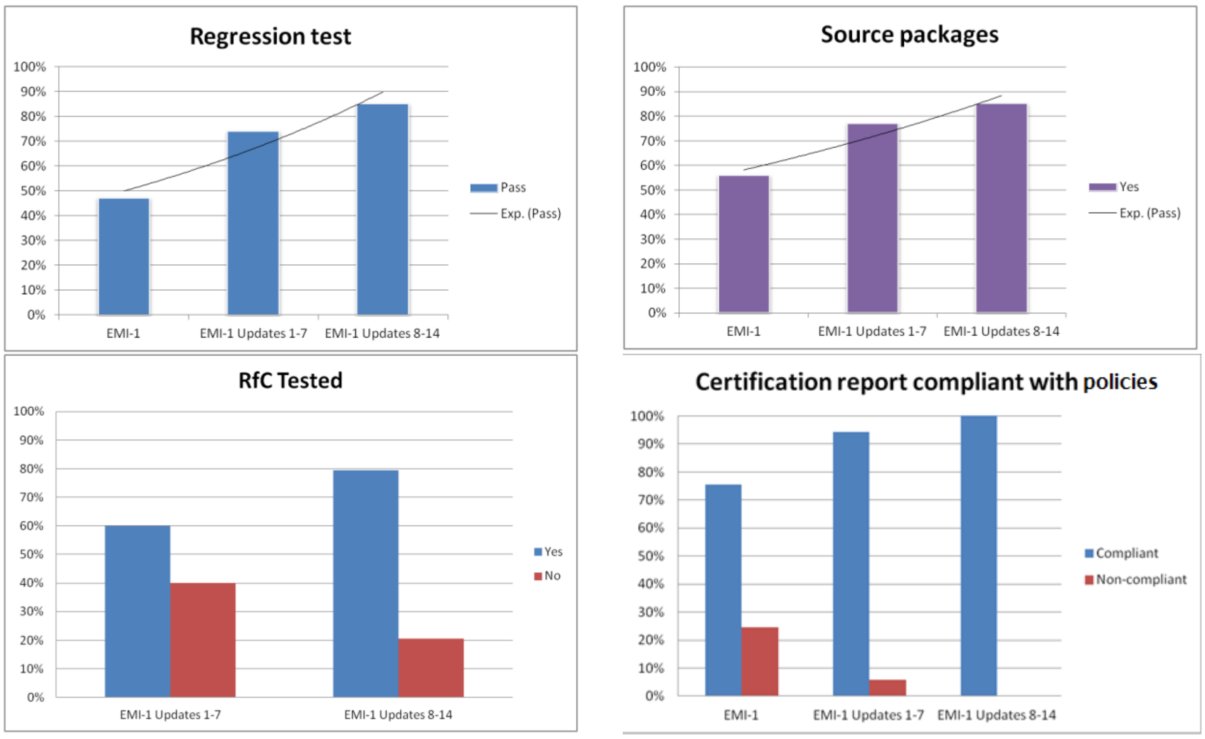


Figure 7 – Metrics calculated with the EMI Verification Dashboard information

Metrics are associated to the different phases as presented in table 3. Metrics should be calculated periodically as presented in the *Frequency* column.

Phase	Deliverable	Metrics	Frequency	Metrics Report Name
EMI major release planning	Work Area Plans, user requirements and development tasks.	• Number of technical objectives • Number of user requirements • Number of total development tasks.	Every major release	EMI_X_planning_MetricsReport, where X is the EMI major release 1, 2 or 3.

EMI major release software coding and testing	Software packages, Certification reports, Test reports.	• KJRA1.3 • Number of Test Plans per software product. • Number of Test Reports per released product. • Number of mandatory tests per released product. • Number of RfCs with regression test. • Number of RfCs/Development tasks with functionality tests. • Number of passed certification checks.	Every major release and update.	EMI_X_[Update_Y]_CodeTest_MetricsReport, where X is the EMI major release 1, 2 or 3, and Y is the number of update.
EMI major release availability	Software Documentation, Software Repositories.	• KJRA1.2 • KJRA1.1 • KJRA1.4 • KNA2.4 • Number of implemented development tasks. • Number of EMI Security Assessments. • Number of fixed EMI security vulnerabilities. • Number of fixed EGI SVG tickets. • Number of services providing high-availability setups. • Number of missing mandatory documents. • Number of services providing service control and status mechanisms. • Number of services providing configuration tools. • Number of Test Plans • Number of supported platforms. • Number of standard installation tools per supported platform.	Every major release.	EMI_X_general_MetricsReport, where X is the EMI major release 1, 2 or 3.

		• Number of standard package formats per supported platforms per released product. • Number of non compatible software licenses. • Number of closed development tasks.		
EMI major release maintenance	RfCs	• KSA1.3 • KSA1.4 • KSA1.5 • RPMlint • Lintian • Number of fixed EMI security vulnerabilities. • Number of fixed EGI SVG. • Number of EMT Tasks tracking Documentation issues	Every week in the EMT meetings.	EMT_MetricsReport
EMI major release user support	GGUS tickets	• KSA1.1 • KSA1.2 • Number of GGUS tickets related to lack of accuracy and resource utilisation issues.	Every week in the EMT meetings.	EMT_MetricsReport

Table 3 – EMI Metrics reports

Metrics reports basically contain the following information:

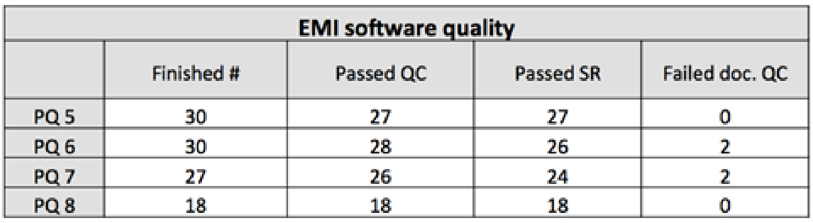
- A table summarising the assessment result per metric for each characteristic that is relevant in the metrics report.
- Plots showing the results of the assessment and historic data of the metrics.
- A list of corrective actions when the assessed value of the metric is under the required level.

5. Cost and impact of quality

The EMI project has a dedicated activity for Quality Assurance. This activity is responsible for defining and establishing a common software quality assurance process and metrics for all software engineering activities. It is also responsible for consistently pass the customer acceptance criteria and continually improve the software quality and the process itself by monitoring the metrics value trends and reviewing quality control activities. This activity has an effort of 324 person-months out of the total EMI project effort which is 2436 person-months. The EMI project has a total duration of three years.

The impact of the quality assurance activities can be seen in the software provided to the EGI project. In Figure 8 it can be seen the evolution of the EMI software quality per project quarters (PQ) as evaluated by the EGI project. The first column gives information about the number of released products, the second column gives information about the number of products that have passed the EGI quality control checks for the UMD Quality Criteria [8]. The third column indicates the number of products that successfully passed the Staged Rollout phase. The EGI Staged Rollout is a procedure by which certified updates of the middleware are first released to and tested by Early Adopter sites before being made available to all sites through the production repositories. Finally, the last column summarises how many products have been rejected as they don't meet the UMD Quality Criteria.

Figure 8 summarises the impact of the quality assurance activity demonstrating that EMI releases have improved over time meeting the required quality criteria and successfully passing the Staged Rollout phase.



EMI software quality				
	Finished #	Passed QC	Passed SR	Failed doc. QC
PQ 5	30	27	27	0
PQ 6	30	28	26	2
PQ 7	27	26	24	2
PQ 8	18	18	18	0

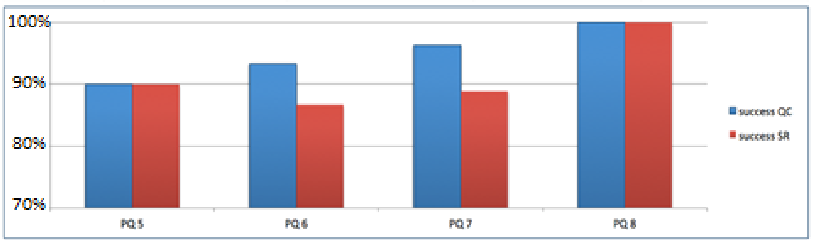


Figure 8 - EMI software quality evolution

6. Lessons learned

6.1. Useful even if it arrives a bit late…

The definition of a quality model helps to set up quality goals for the software and metrics to measure whether those goals have been achieved or not in the developed software. It is a very good starting point to organise the work of the quality control activity. The definition of a measurement plan with metrics report templates and clear dates on when the reports have to be generated, also helps to organise a working plan on how the quality control activities are going to be carried out and what the outcome is going to be. However, defining a quality model takes time and it is not always easy to do since the beginning of the project, when software processes may not be clearly defined yet. Even if the quality model arrives in a later stage, it helps to consolidate the quality control activity.

In EMI, the quality model was defined after the first year of the project when EMI QA policies [11] were stable and it was clear how EMI releases were going to be managed. Quality Control activities took place during the first year as well, but tools like the RfC Dashboard or the Verification Dashboard were not ready until the end of the first year. Experience was gathered during the first year and this helped to create the quality model, to develop tools and to tune existing metrics, fitting better the needs of the project.

6.2. A model that improves

The quality model is something alive that evolves throughout the lifetime of the project. In the case of EMI, the metrics specification has changed many times. In the first year of the EMI project, focus was put to calculate static code analysis metrics like *PyUnit* or *FindBugs*. In the second year of the project, these metrics were no longer calculated and effort was put in *RPMlint*. Thanks to the implementation of tools like the RfC Dashboard and the Verification Dashboard, the calculation of certain metrics was made possible and they could be automatically calculated with little effort. All these examples show that the quality model evolves with time and that this helps to make it really useful for the project.

6.3. Metrics on demand

The quality model basically identifies metrics that measure the presence of certain (high level) characteristics in the software. It does not take into account requirements from project members who need metrics to better do their work. Metrics should also be defined and calculated in these cases. For instance, in EMI metrics are calculated every week for the release manager who is interested in successful builds and in tracking software changes. Metrics are also calculated every quarter for technical managers who want to see how well product teams have performed in terms of user support tickets and software changes implementation. It is difficult that the quality model covers all possible metrics needed in the project. This is why it is important to have a good communication with project members who can benefit from the existing tools and knowledge to calculate metrics.

7. Acknowledgements

This work was partially funded by the EMI project under European Commission Grant Agreement INFSO-RI-261611.

8. References

[1] European Middleware Initiative project (EMI) *http://www.eu-emi.eu/*
[2] European Grid Initiative project (EGI) *http://www.egi.eu/*
[3] EMI Quality Model *https://twiki.cern.ch/twiki/bin/view/EMI/QualityModel*
[4] EMI Software Quality Assurance Plan *https://twiki.cern.ch/twiki/bin/view/EMI/SQAP*
[5] EMI RfC Tracker Dashboard *http://emi-rfc.cern.ch/*
[6] EMI Verification Dashboard *http://emi-dashboard.cern.ch/*
[7] EMI Production Release Criteria *https://twiki.cern.ch/twiki/bin/view/EMI/ProductionReleaseCriteria*
[8] Unified Middleware Distribution (UMD) Quality Criteria *https://documents.egi.eu/public/RetrieveFile?docid=364&version=5&filename=EGI-GENERIC-QC-V2.pdf*
[9] EMI Description of Work *https://twiki.cern.ch/twiki/pub/EMI/EmiDocuments/EMI-Part_B_20100624-PUBLIC.pdf*
[10] eInfrastructure for Testing, Integration and Configuration of Software (ETICS) *http://etics.web.cern.ch/etics*
[11] EMI Quality Assurance Policies *https://twiki.cern.ch/twiki/bin/view/EMI/SA2*