

## Common Framework for Extracting Information and Metrics from Multiple Change Trackers

---

**Eamonn Kenny\*, Duarte Meneses†**

*Trinity College Dublin*

*E-mail: ekenny@scss.tcd.ie*

An important aspect of EMI is the delivery of ‘quality software’. For this reason the quality assurance (QA) group was introduced. There are a key number of beneficiaries of this work in several work packages. These EMI development and support activities are required to produce key performance indicators (KPIs) and metrics for milestone, quarterly and yearly deliverables based on the information provided by the QA group. However, EMI has a large number of varying sized products, different and existing middlewares and various bug/feature request for change (RfC) trackers used by each product team. The only way to reliably produce KPIs and metrics related to change management in such a varied project is to introduce simplifying, common environments that are readily accessible by all the different customers of the project. For this reason an extensible XML-based framework was defined for storing, plotting, querying and tabulating change tracker information for all its customers.

*EGI Community Forum 2012 / EMI Second Technical Conference,  
26-30 March, 2012  
Munich, Germany*

---

\*Speaker.

†Thanks to Duarte Meneses for his work on providing much of the implementation.

## 1. Introduction

Quality Assurance is a very important aspect of a large projects lifespan, especially when developing and troubleshooting software products originating from many institutions, each producing many components using various bug/feature trackers. Acquiring reliable and consistent data sets across multiple institutions provides the only mechanism to produce meaningful metrics and key perform indicators (KPIs) (see Linda Westfall[1]).

There currently exists many issue trackers such as Bugzilla[2] and Trac[3] containing report generation tools to correlate or query data, producing specified trends. There also exists many static analyser suites such as the Sonar Project[4] allowing the user to produce code based metrics from their source code using Eclipse and Hudson. However, it is difficult to find a tool that produces multiple issue tracker reports and static analysis metrics for a wide variety of software languages and bug-trackers. The aim of the common metric framework is to make this possible.

The analysis leading to the choice of metrics/KPIs is not within the scope of this paper. Instead, the mechanisms for obtaining the inputs, framework for storing the metrics and outputs will be discussed, to keep the concepts as generalisable as possible.

There is little work in this area, however Rex Black[5] does present the work of Entomology by Matt Barringer describing a tracker client that supports Mantis, Bugzilla and Trac. However, in EMI there are far more than three bug-trackers to be integrated. For instance, there are 6 trackers alone from UNICORE that must be integrated to separately manage bugs, features and testing.

The key to producing a uniform framework for dealing with metrics and KPIs is the production of a set of policies that every party adheres to. Firstly, there was a clear definition of the common parameters that was then successfully exported by each middleware. These parameters form the inputs for any metric engine. It was imperative that metrics receive exactly the same form of data from multiple institutions and produce outputs in a form that is relevant to each individual customer.

The metric outputs in the EMI project are used by the release management team to produce key performance indicators (KPIs), by the quality control group to monitor the objectives of the project and internally by the quality assurance group to assess the correction of the values/outputs. The metrics and KPIs should assist in governing and streamlining the product development and release process.

## 2. Policy and Architecture

Figure 1 shows a diagrammatic overview of the different aspects of the common framework for producing metrics. The product development teams are responsible for producing an export/dump of their request for change (RfC) tracker in a form that is specified by the metrics group complying to policies set down by the project. Each of the product teams is responsible for producing any and all of the information necessary to product the output format which is then used by the metrics group.

One of the main advantages of such an export is that it creates an impartial view of the data that is not easy to manipulate at a future date. It is human nature to try to produce the best metrics one can by doctoring data so that a better view of the data is obtained. The current scheme avoids this possibility.

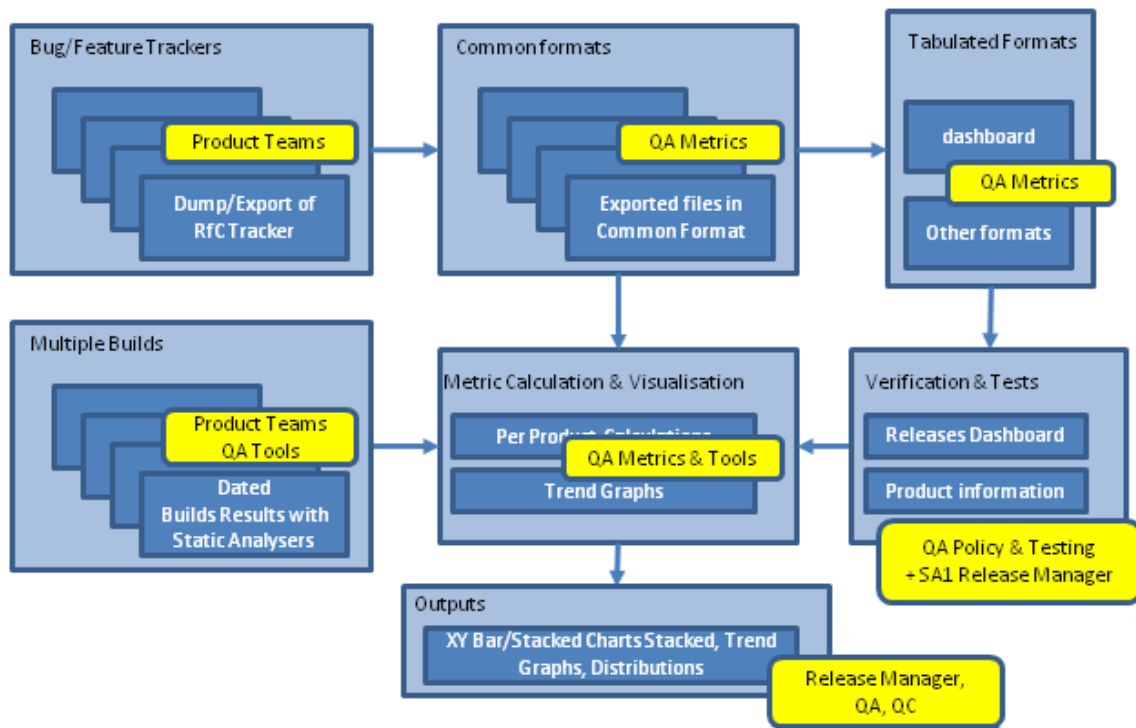


Figure 1: Architecture of the Framework

Next the exported data is passed to the metrics group where it can be validated for correctness. At this point some clarifications are needed to ensure that policies are adhered to, and that any missing data is added. The data is now in a raw form, where it can be combined easily into one larger view. This allows it to be passed to the metrics generation engine or to web based dashboards, where it can be used to product tabular output or charts/plots.

The metrics generation at this point only includes the bug-tracker related information, so there is also the requirement to include software build and test related metrics. These take the form of static analysers and testing verification dashboard inputs respectively.

Once all information is obtained the metrics generation engine can calculate metrics based on data from requests for change (RfC), static analysers performed at build time and the quality controls testing/verification dashboard.

### 3. The XML based Implementation

It became apparent early on in the EMI project that a uniformly extensible framework for dealing with inputs from each middleware product teams change tracker (i.e bug/feature tracker) could easily be achieved using an XML based format starting with a lightly constrained XML schema and progressing to a very tightly controlled schema.

The defining of a common schema means that some trackers must combine their change management states into one common state  $S$  whilst other trackers are required to introduce new states.

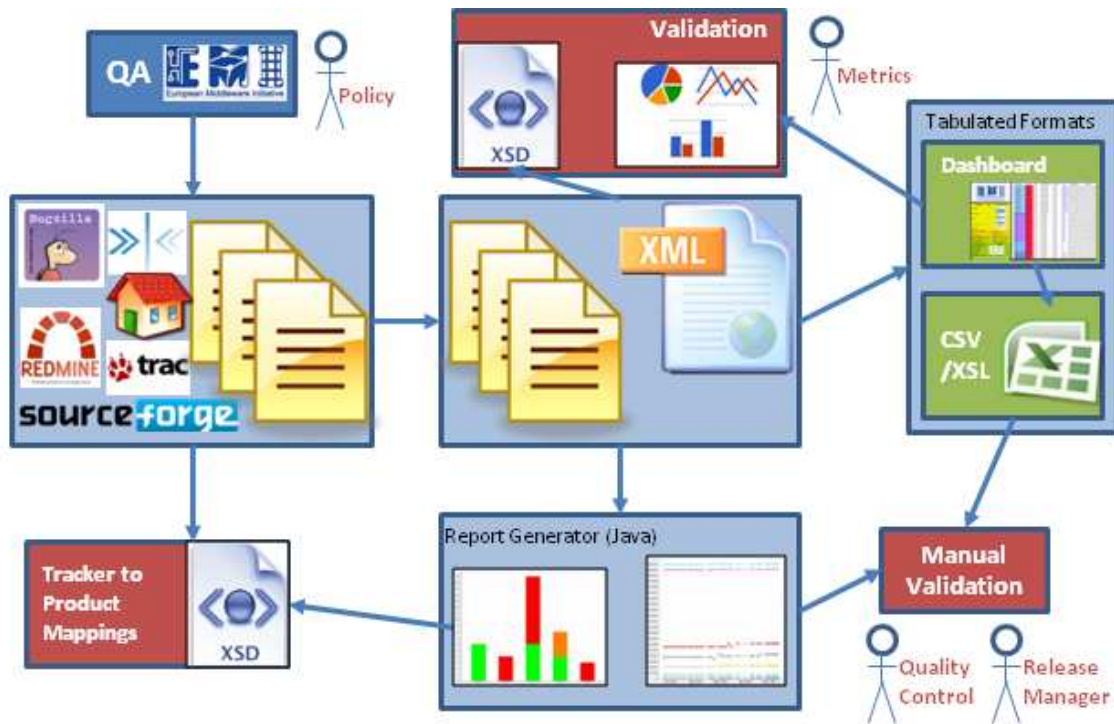


Figure 2: Framework Implementation (Year 1)

An example of this is expressed by the following:

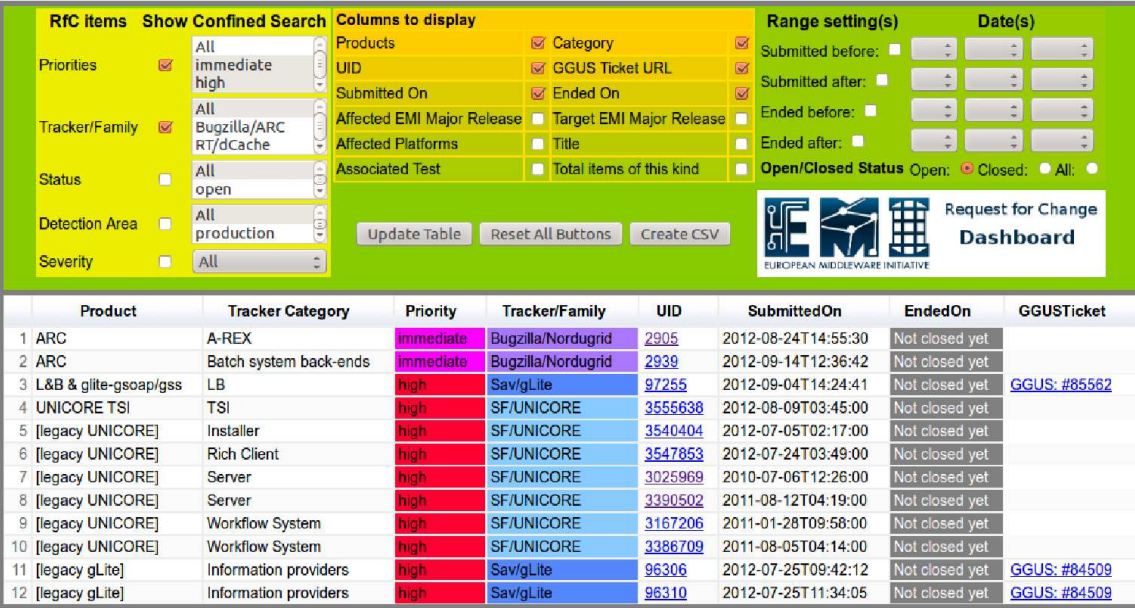
$$S_{reject} = \{S_{duplicate} \cup S_{won't\ fix} \cup S_{unreproducible} \cup S_{obsolete} \cup S_{invalid}\} \quad (3.1)$$

The union in eqn 3.1 was applied to a number of, but not all of the trackers, in line with the management policies of the project. The mapping was defined within an XML description and adhered to the management policy by conforming to an XML schema.

Once all input data was defined for the QA group, it was highlighted that there was a fundamental issue of mapping multiple middleware change tracker category names to an individual product. This required a specific XML mapping description to allow the QA group to ascertain how each tracker maps its changes to each individual product. This mapping is very tightly controlled by an XML schema (XSD). For example, suppose a core component of software has bug tracking category  $C_{core}$ . It is conceivable that fixes in software described in the category  $C_{core}$  will eventually appear in multiple products each consisting of many packages linked with various bug/feature category.

With the tracker to product mapping XML and change tracker XML in place this paved the way for defining a Java based framework for producing automated daily or periodic charts, producing metrics and KPIs for each of the customers within EMI.

One very useful requirement for customers was a dashboard (Figure 3) showing tabulated data obtained from the change tracker XML. This was built with a back-end query engine giving various views of the data to many customers. It also produces data sets that the user can easily plot. The simplest format of choice for such output was comma separated versioning (CSV) to be used in Excel.



The dashboard includes the following sections:

- RfC Items:** Filtered by 'All immediate high'.
- Show Confined Search:** Filtered by 'All Bugzilla/ARC RT/dCache'.
- Status:** Filtered by 'All open'.
- Detection Area:** Filtered by 'All production'.
- Severity:** Filtered by 'All'.
- Columns to display:** Products, UID, Submitted On, Affected EMI Major Release, Affected Platforms, Associated Test, Category, GGUS Ticket URL, Ended On, Target EMI Major Release, Title, Total items of this kind.
- Range setting(s):** Submitted before, Submitted after, Ended before, Ended after.
- Date(s):** Open, Closed, All.
- Buttons:** Update Table, Reset All Buttons, Create CSV.
- Request for Change Dashboard:** EUROPEAN MIDDLEWARE INITIATIVE logo.

	Product	Tracker Category	Priority	Tracker/Family	UID	SubmittedOn	EndedOn	GGUSTicket
1	ARC	A-REX	immediate	Bugzilla/Nordugrid	2905	2012-08-24T14:55:30	Not closed yet	
2	ARC	Batch system back-ends	immediate	Bugzilla/Nordugrid	2939	2012-09-14T12:36:42	Not closed yet	
3	L&B & glite-gsoap/gss	LB	high	Sav/gLite	97255	2012-09-04T14:24:41	Not closed yet	GGUS: #85562
4	UNICORE TSI	TSI	high	SF/UNICORE	3556638	2012-08-09T03:45:00	Not closed yet	
5	[legacy UNICORE]	Installer	high	SF/UNICORE	3540404	2012-07-05T02:17:00	Not closed yet	
6	[legacy UNICORE]	Rich Client	high	SF/UNICORE	3547853	2012-07-24T03:49:00	Not closed yet	
7	[legacy UNICORE]	Server	high	SF/UNICORE	3025969	2010-07-06T12:26:00	Not closed yet	
8	[legacy UNICORE]	Server	high	SF/UNICORE	3390502	2011-08-12T04:19:00	Not closed yet	
9	[legacy UNICORE]	Workflow System	high	SF/UNICORE	3167206	2011-01-28T09:58:00	Not closed yet	
10	[legacy UNICORE]	Workflow System	high	SF/UNICORE	3386709	2011-08-05T04:14:00	Not closed yet	
11	[legacy gLite]	Information providers	high	Sav/gLite	96306	2012-07-25T09:42:12	Not closed yet	GGUS: #84509
12	[legacy gLite]	Information providers	high	Sav/gLite	96310	2012-07-25T11:34:05	Not closed yet	GGUS: #84509

Figure 3: RfC Dashboard using Google Apps Engine (Charts)

### 3.1 Revised Implementation

In year one of the EMI project, a complete Report Generator framework was produced for producing PDF and Word Documents as well as the much needed charts/plots. However, in year two of the project, it was decided to strip the Java plotting framework back to a minimal set of operations (see Figure 4), so that specially tailored charts could be generated for any customer that requests them. This has proved invaluable from the point of view of being able to corroborate the KPIs of customers and to give quick turnaround times on producing new statistics.

In Figure 4 there are four main points to consider when producing new metrics or revising the schema's that govern the XML mappings:

1. When a schema is modified such as the bug-tracking/RfC schema, bug-mapping schema, verification dashboard schema or tracker-to-product mapping schema, there is an easy mechanism to expose the elements of the XML to the Java implementation. This is achieved using JAXB stubs which automatically generates the methods to manipulate XML elements. For instance, the text value of newly introduced XML elements such as *AssociatedTest* can be accessed using functions *getAssociateTest()* and *setAssociatedTest()* within the Chart Generator Framework.
2. Producing new metrics even if they are not that similar to the existing metrics usually just requires cloning of existing metrics and changing their functionality.
3. Producing new plotting tools uses an abstract class object meaning that you can clone an existing component and produce a new type of plotting mechanism much more quickly.
4. In the case of item 2 above, there is always the requirement to define a configuration file to expose the new metric(s). There are a number of predefined variables such as the main title,

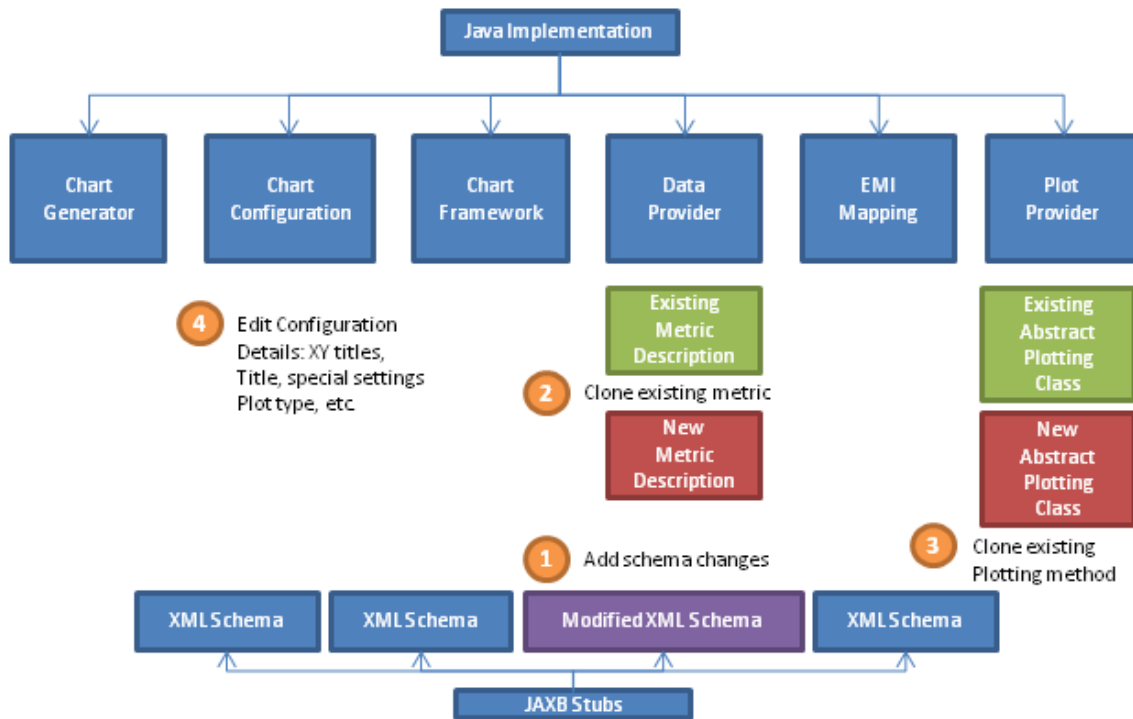


Figure 4: Framework Implementation (Year 2)

x-axis title, y-axis title and metric name. However, there is also the possibility to provide user defined internal variables to be used by specific metrics making the configurations extensible.

4. Current Status

In all, so far, six different bug-trackers (Savannah, SourceForge, RT, Bugzilla, Trac and Redmine) have been integrated into the common XML based format. This does not mean that there are only six trackers sets of data. For instance, there are six UNICORE feature and defect related tracker streams integrated into the common XML format. It is possible that the JIRA tracker will be integrated in the future.

Currently the RfC dashboard (Figure 3) is used to verify the correctness of the RfC report produced by the tools group for the executive management team (EMT) weekly meeting. The release management group use it to export data which is then used as input data to produce their quarterly reports. The metrics group use it internally to verify whether each element of an RfC has valid values. The Quality Control team use it to corroborate which products generate valid tests at the time of each release update and major release.

In year one of EMI, the development of the Java based framework made it possible to automate the production of a weekly PDF report of all immediate and high priority tickets in each of their transition states, for given time periods, for varying defect/feature categories and ascertain whether they arose from production, development or testing.

In year two of EMI, it became increasingly important for the Quality Control (QC) team to see which requests for change (RfCs) due for release included regression tests in the case of defects

and functional tests in the case of feature requests. This feature was slowly integrated into the change management XML format using a lightly constrained schema, without any disruption to existing customers. Once fully integrated, the XML relating to testing had to conform to a much more heavily constrained schema.

Visually speaking (see Figure 3), the RfC dashboard produces columns of data such as priorities, severities and detection areas for each RfC, details about each individual ticket and links back to the originating GGUS ticket if it exists. The output of the dashboard is a table of subsets, presented as Google Apps based tables or data sets that can be imported by customers into Excel so that they can produce their own plots.

## 5. Conclusions

The production of a common XML format for change management avoided the painful migration of each bug/feature tracker to a new system where each middleware had little or no expertise.

All middlewares were able to produce and export the common XML format. Since it was simple in structure, it was relatively straightforward to wrap with a query engine to produce a dynamic dashboard.

The Chart Generator Framework uses the common change tracker XML and mapping XML to produce many different KPIs in the context of a product. Without these mappings this would be next to impossible, requiring manual queries on each bug-tracker, correlation of data, and manual plotting of KPIs for every milestone or deliverable.

Without a general framework in place, consulting the change trackers for more than 50 Products would have resulted in time-consuming and error-prone activities across approximately 30 Product Teams while the proposed approach results in a much more agile and efficient control of the information flow.

## 6. Acknowledgements

Special thanks due to Gabriele Pierantoni and Alberto Aimar for reviewing this paper. This work was performed as part of the EMI project and partially funded by the European Commission under Grant Agreement INFSO-RI-261611.

## References

- [1] L. Westfall, *12 Steps to Useful Software Metrics*, Proceedings of the Seventeenth Annual Pacific Northwest Software Quality Conference, Volume: 57 Suppl 1, Issue: May 2006.
- [2] Bugzilla, <http://bugzilla.org>.
- [3] Trac, <http://trac.edgewall.org>.
- [4] Sonar, <http://sonarsource.org>.
- [5] Rex Black, *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*, John Wiley & Sons, Chapter: 4, August 2009.