The gLite
Software Development Process

Alberto Di Meglio
CERN

Lightweight Middleware for Grid Computing

Information Society and Media

Overview

- Software configuration management and tools
- Release process
- QA Metrics and Process Auditing

Software Process

- **JRA1 Software Process is based on an iterative method loosely based on RUP and some XP practices**
- **It comprises two main 12-month development cycles divided in shorter *development-integration-test-release* cycles lasting from 2 to 6 weeks**
- **The two main cycles starts with full Architecture and Design phases, but the architecture and design are periodically reviewed and verified.**
- **The process is fully documented in a number of standard document:**
  - Software Configuration Management Plan (SCM)
  - Test Plan
  - Quality Assurance Plan
  - Developer's Guide

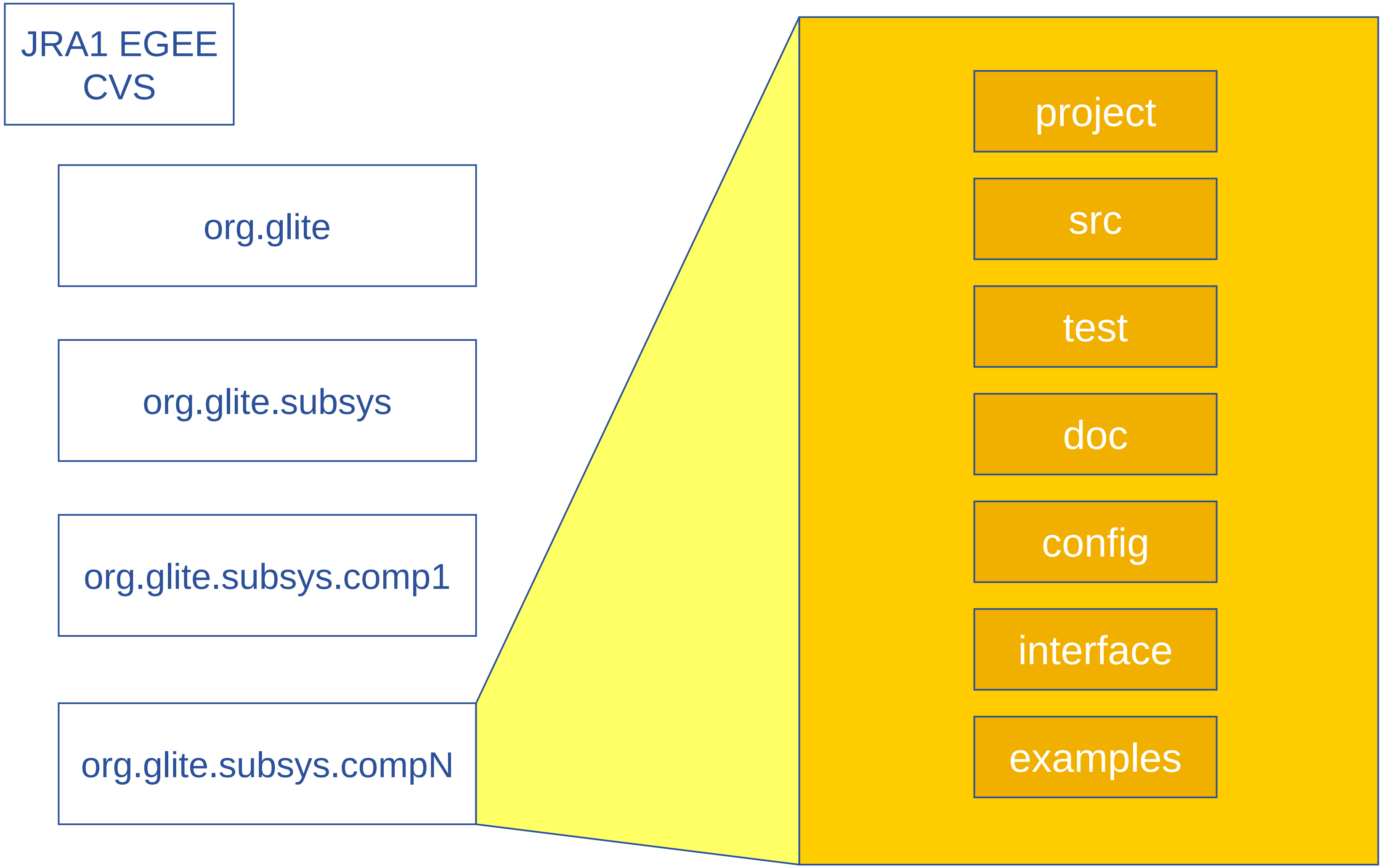
- **The SCM Plan is the core document of the Software Process**
- **It contains a description of the processes and the procedures to be applied to the six SCM activity areas:**
  - Software configuration and versioning, tagging and branching conventions
  - Build Tools Systems
  - Bug Tracking
  - Change Control and the Change Control Board (CCB)
  - Release Process
  - Process Auditing and QA Metrics
- **It is based on a number of standard methods and frameworks including:**
  - ISO 10007:2003 - Quality management systems -- Guidelines for configuration management, ISO, 2003
  - IEEE Software Engineering Guidelines (http://standards.ieee.org/reading/ieee/std/se)
  - The Rational Unified Process (http://www-306.ibm.com/software/awdtools/rup/)
- **In addition it adopts best-practice solutions[1] to guarantee the highest possible quality in a very distributed and heterogeneous collaboration**

[1]S.P. Berczuk, Software Configuration Management Patterns, Software Patterns Series, Addison-Wesley, 2002
A. Di Meglio et al., A Pattern-based Continuous Integration Framework for Distributed EGEE Grid Middleware Development, Proc. CHEP 2004

Version and Configuration Control

- Based on CVS using CERN Central IT CVS service
- Fixed directory structure for each module
- Rules for tagging and branching (e.g. bug fix branches)
- Common naming conventions for baseline and release tags
- Configuration files to automate creation of workspaces
  - Used to enforce build reproducibility
  - Used to create private workspaces for developers
- Rules apply also to external dependencies, all third-party packages and versions are controlled by the build systems, not the developers

CVS Structure



Build Tools

- Ant: used for the general build management and all Java modules.
- Make + GNU Autotools: for C/C++, Perl and other languages as necessary. Some effort has been done to port the Makefiles to Windows or use Cygwin, but with very limited success
- CruiseControl: used to automate the nightly and integration builds on the central servers
- An abstraction layer has been created on top of these tools to provide a common interface to all build tools independently of the language, platform and tool used

Build Infrastructure

- **Two nightly build servers on RH Linux 3.0 (ia32)**
  - Clean builds out of HEAD and v. 1.x every night of all components
  - Results are published to the gLite web site
  - Tagged every night and totally reproducible
- **One continuous build server on RH Linux 3.0 (ia32)**
  - Incremental builds out of v. 1.x every 60 minutes
  - Results published to CruiseControl web site
  - Automated build error notifications to developers and Integration Team
- **One nightly build server on RH Linux 3.0 (ia64)**
  - Clean builds every night of all components
- **One nightly build server on Windows XP**
  - Clean builds every night of all components currently ported to Windows
- **Build system supported platforms:**
  - Red Hat Linux 3.0 and binary compatible platforms (SLC 3, CentOS, etc), 32 and 64-bit (gcc)
  - Windows XP/2003

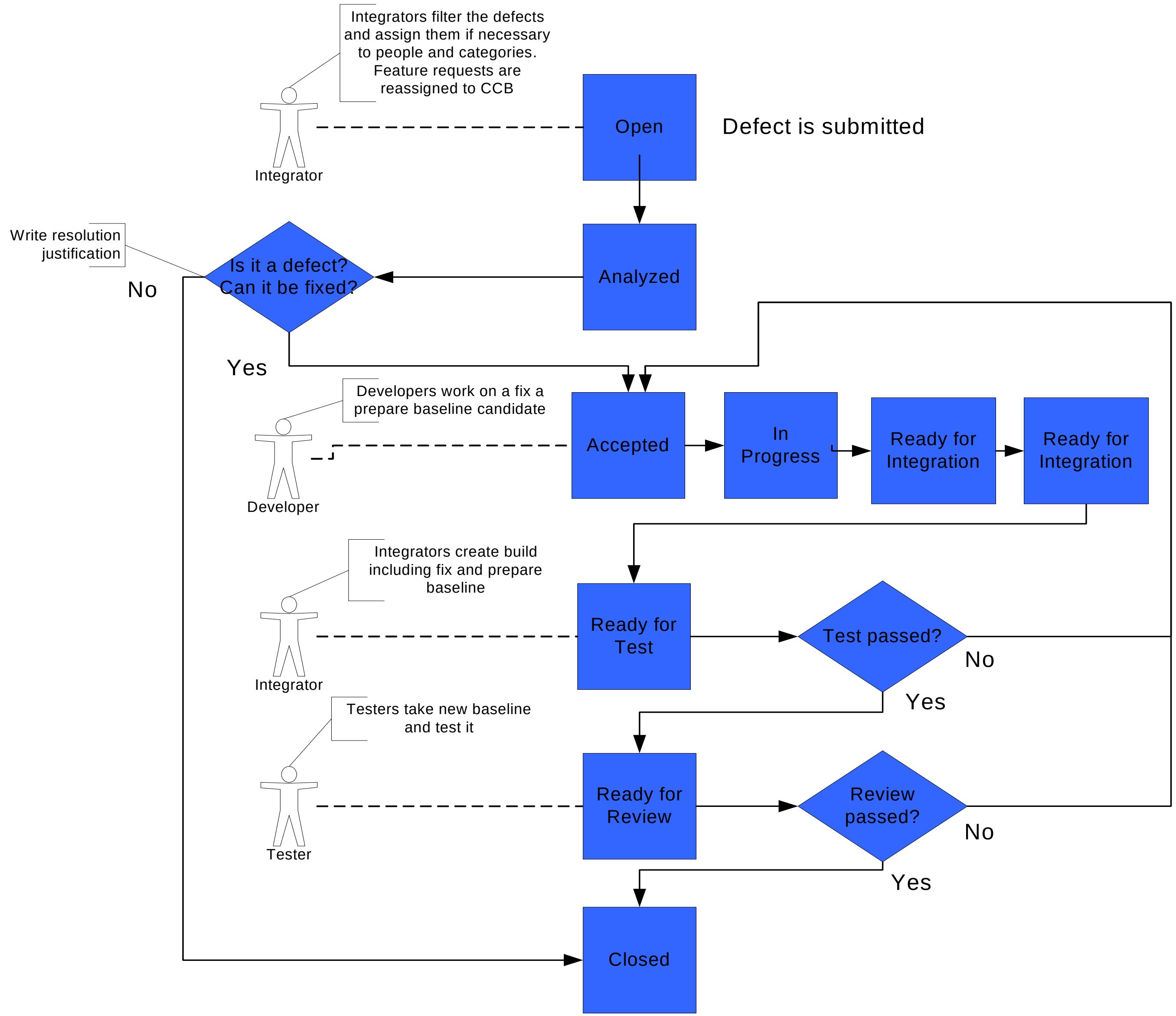
Defect Tracking System

- Based on the Savannah project portal at CERN
- Used also for change requests (for example API changes, external libraries version changes, etc). In this case, request are assigned to the Change Control Board for further evaluation (explained later)
- Heavily customized to provide several additional bug status on top of the default Open and Closed (!)
- Each gLite subsystem is tracked as a separate category and related bugs are assigned to the responsible clusters

Defect Tracking Configuration

- Two conditions: Open, Closed
- Ten main states: None, Accepted, In Progress, Integration Candidate, Ready for Integration, Ready for Test, Ready for Review, Fixed, Duplicate, Invalid
- Transitions between two states is subject to specific checks (Is it a defect? Can it be fixed? Test passed, Review passed?)
- Not all transitions are allowed and additional automated changes can be triggered by changing the bug status (condition, assignment, etc)

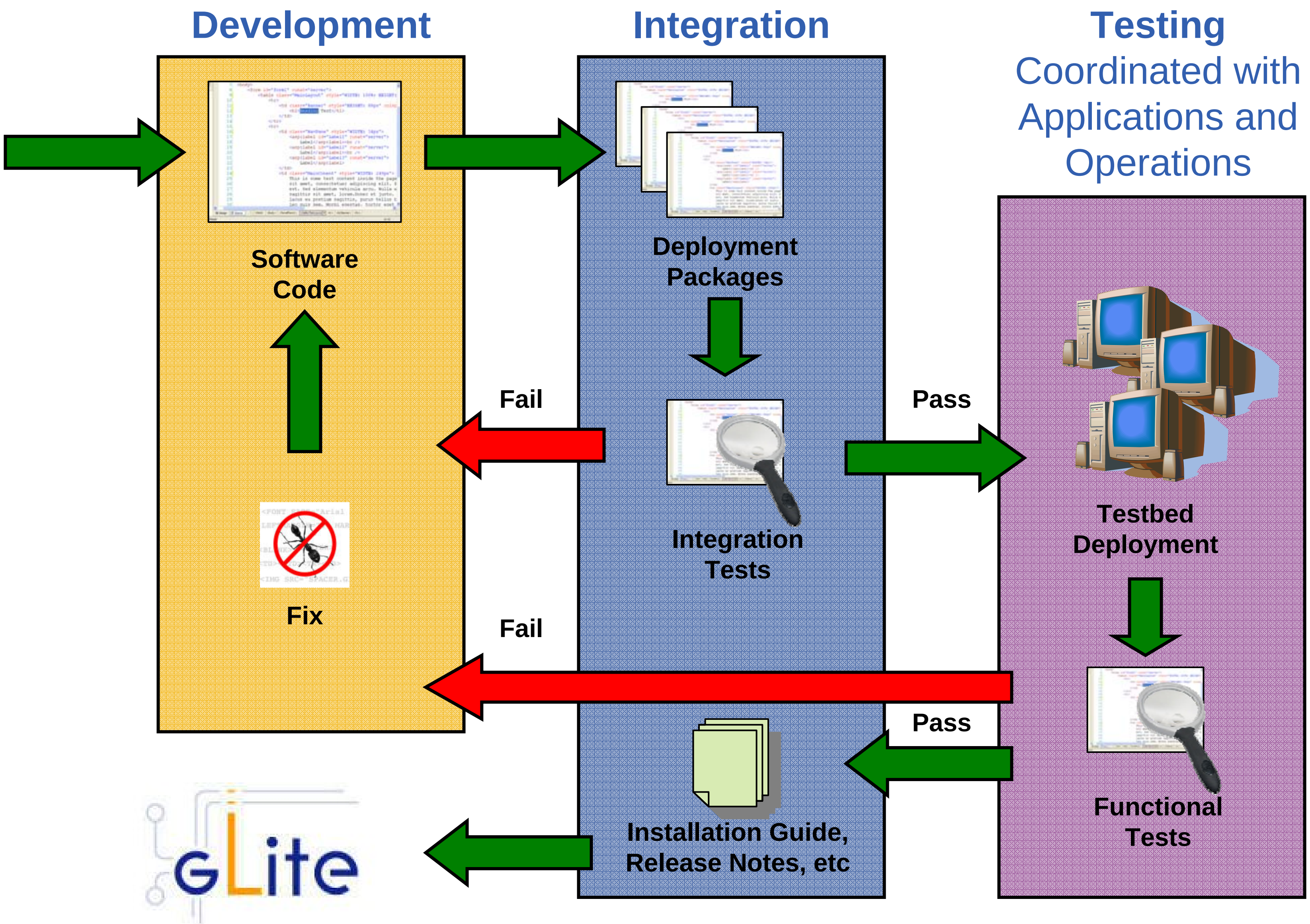
Defect Tracking Cycle



Change Control Board

- All public changes must go through a formal approval process
- The CCB is tasked to collect and examine the change requests
- Changes are tracked and handled as quickly as possible
- The CCB is not a physical team, but a role that is assumed by more than one team or group depending on the type of change (interface changes, bug fixes, software configuration changes, etc)

Release Process



Software Quality Assurance

- Software Metrics are collected as part of the build process
- Failure to pass a quality check fails the build
- Additional checks are implemented in the version control system (coding style, documentation tags)
- Software Defect and QA Metrics are collected from the defect tracking system
- Reports and graphs are published on the project web site

Code Style and Conventions

gLite coding style report

system

org.glite

Summary			
Subsystems	**SLOC**	**Errors**	**Errors / line**
13	1091608	52792	0.0484

Subsystems			
Name	**Errors**	**Lines**	**Errors / line**
org.glite.jdl	2572	3847	0.6686
org.glite.ce	9575	26410	0.3626
org.glite.wms-ui	18785	93834	0.2002
org.glite.rgma	7677	110008	0.0698
org.glite.gpbox	6645	108168	0.0614
org.glite.service-discovery	336	7508	0.0448
org.glite.amga	1448	41802	0.0346
org.glite.security	3225	108056	0.0298
org.glite.wms	1888	297219	0.0064
org.glite.data	641	196888	0.0033
org.glite.dgas	0	31226	0.0000
org.glite.testsuites	0	63582	0.0000
org.glite.wms-utils	0	3060	0.0000

Coding conventions checked by CHECKSTYLE and CODEWIZARD
using the gLite coding conventions.
Line count by SLOCCOunt.

Internet

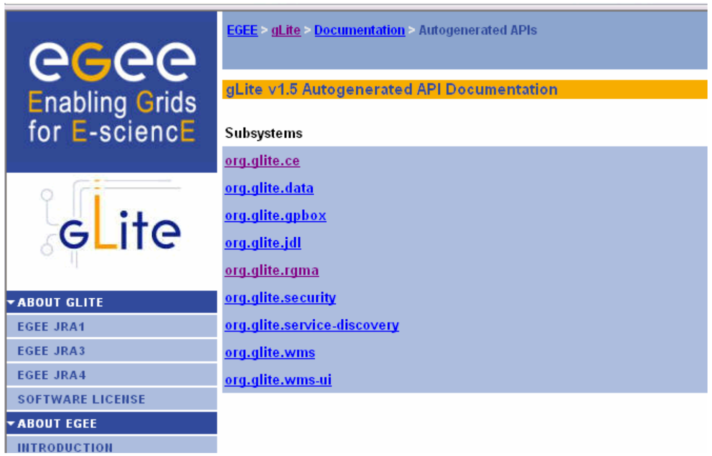
Unit Tests Reports

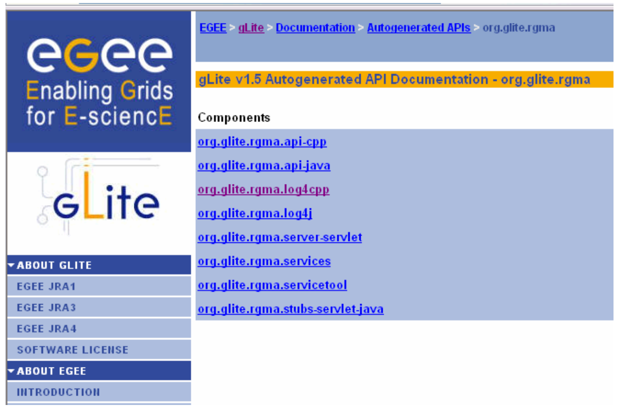
Build Results | Test Results | XML Log File | Control Panel

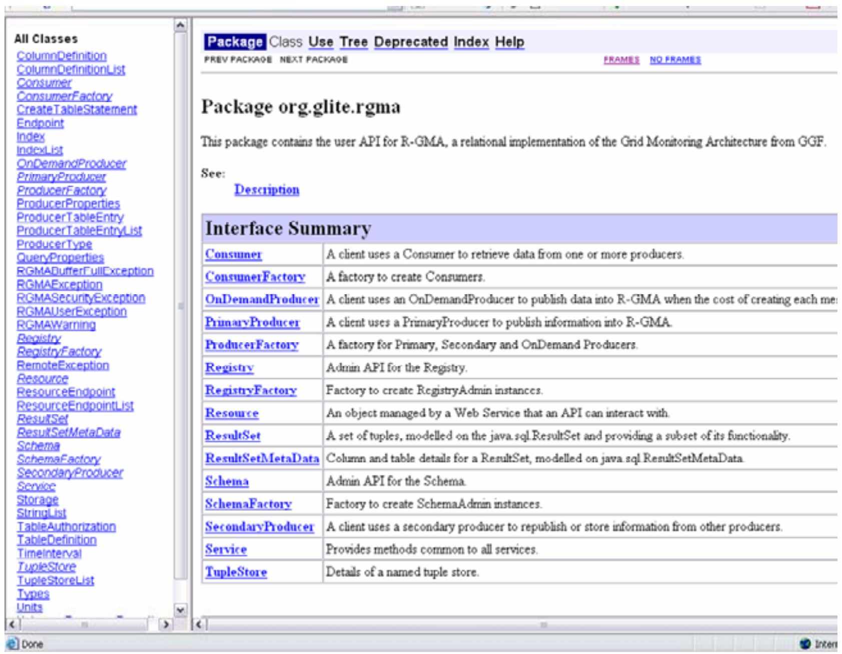
Name		Status	Time(s)
.org.glite.rgma.ProducerPropertiesTest			
	testIsHistory	Success	0.008
	testIsLatest	Success	0.000
Properties »			
.org.glite.rgma.QueryPropertiesTest			
	testIsHistory	Success	0.008
	testIsLatest	Success	0.000
	testIsContinuous	Success	0.000
	testEquals	Success	0.000
Properties »			
.org.glite.rgma.StorageTest			
	testEquals	Success	0.006
	testGetPassword	Success	0.000
	testGetLocation	Success	0.000
	testGetUserName	Success	0.000
	testIsDatabase	Success	0.000
	testIsMemory	Success	0.001
	testHasDetails	Success	0.000
Properties »			
.org.glite.rgma.TimeIntervalTest			
	testValueAsMillis	Success	0.006
	testValueAsSeconds	Success	0.000
	testValueAsMinutes	Success	0.000
	testValueAsHours	Success	0.000
	testValueAsDays	Success	0.000
Properties »			

Autogenerated Documentation

Enabling Grids for E-sciencE







SLOC

Total Physical Source Lines of Code (SLOC)

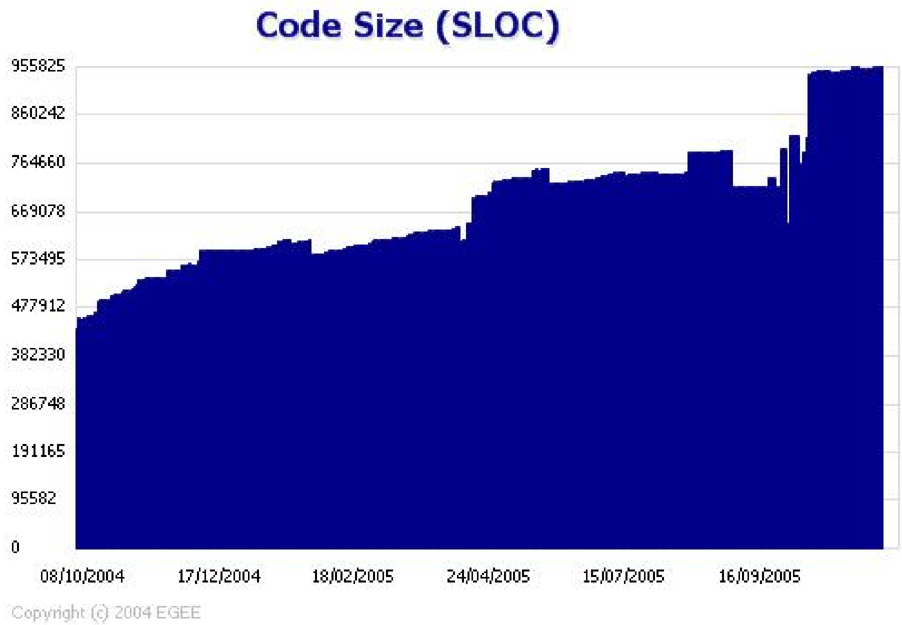
- SLOC = 955,825 (as of 21 November 2005)

Total SLOC by language (dominant language first)

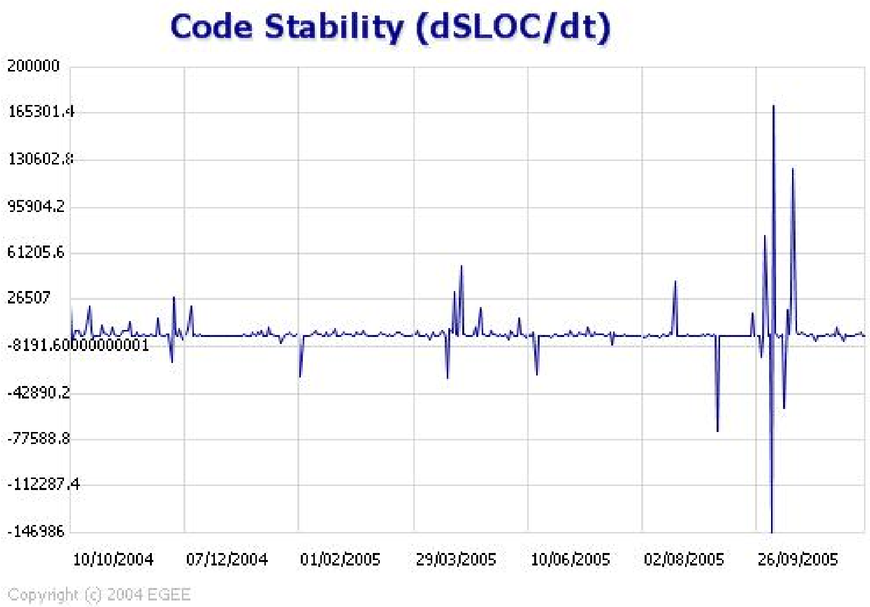
Java	285271 (29.85%)
C++	266828 (27.92%)
Ansi C	209326 (21.90%)
Perl	75386 (7.89%)
sh	70904 (7.42%)
Python	43459 (4.55%)

- Total complete builds: 641 (all 1.x branches), 236 (HEAD)
- Number of subsystems: 21
- Number of CVS modules: 454
- Pre-Release Defects/KSLOC = 2.75
- Post-Release Defects/KSLOC = 1.10

Code Size and Stability

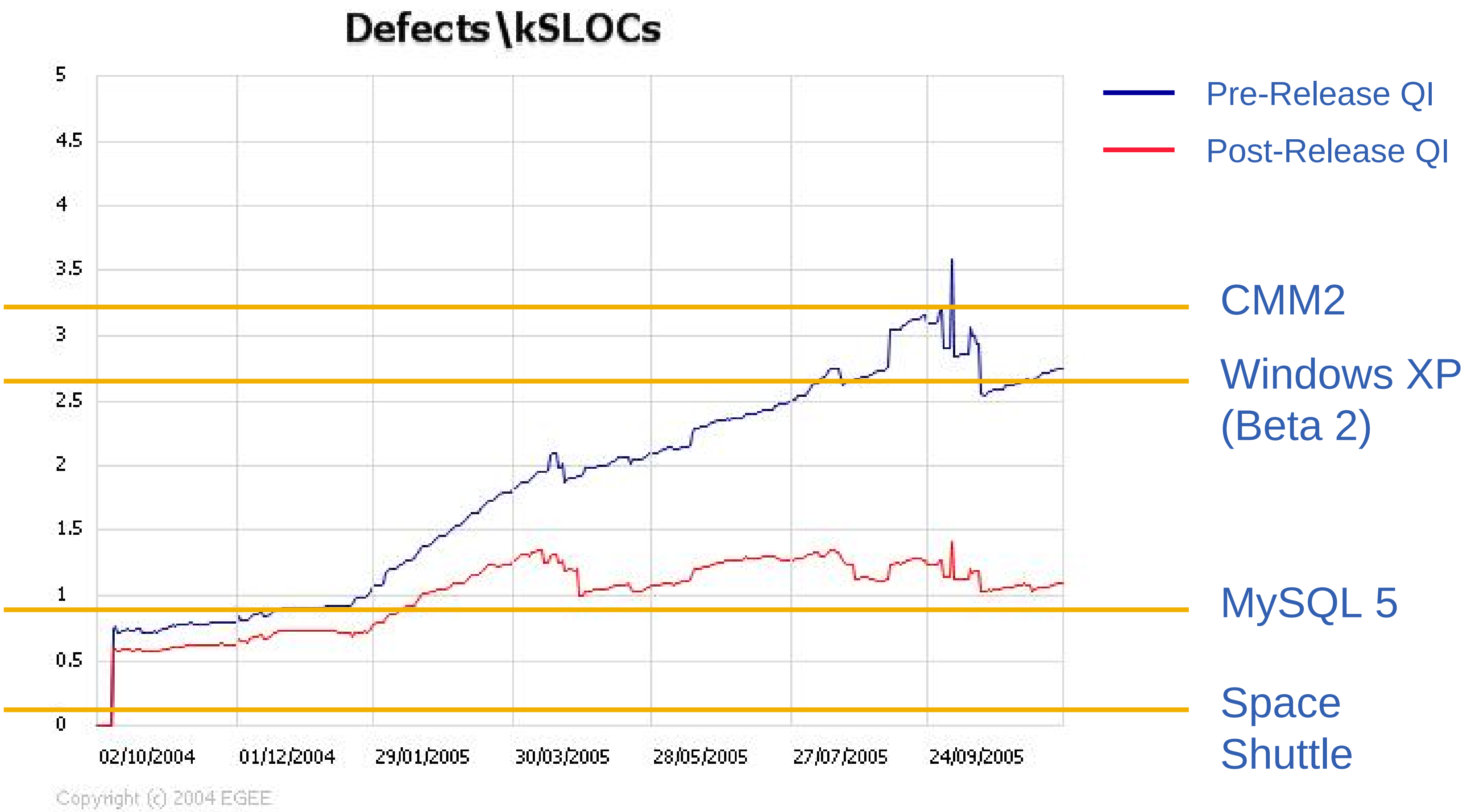


The Code Size chart shows the changes in total number of SLOCs during the life of the project

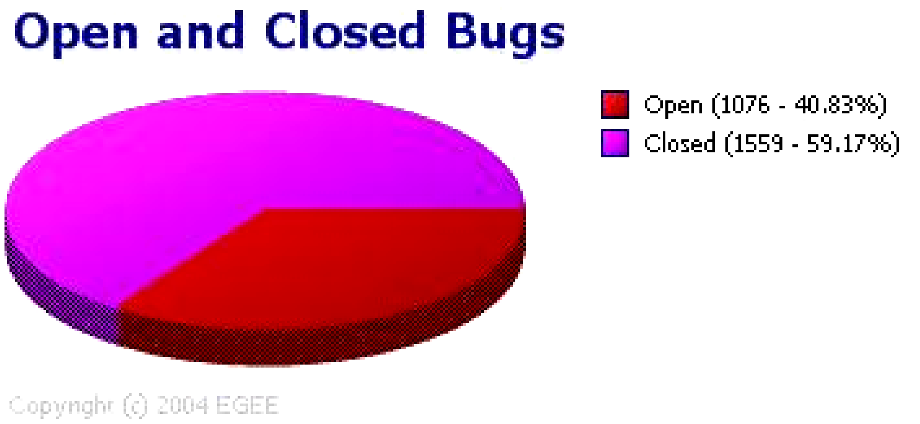


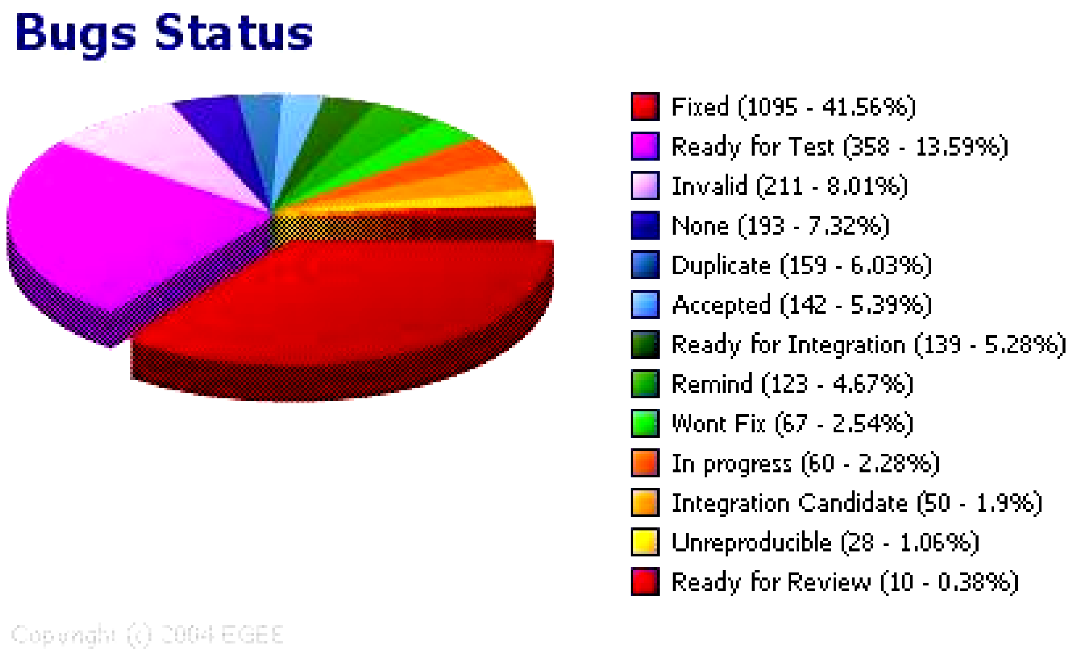
The Code Stability chart shows the change rate of code size during the life of the project.
As the project nears completion the rate should approach 0

The Defects/kSLOC Chart

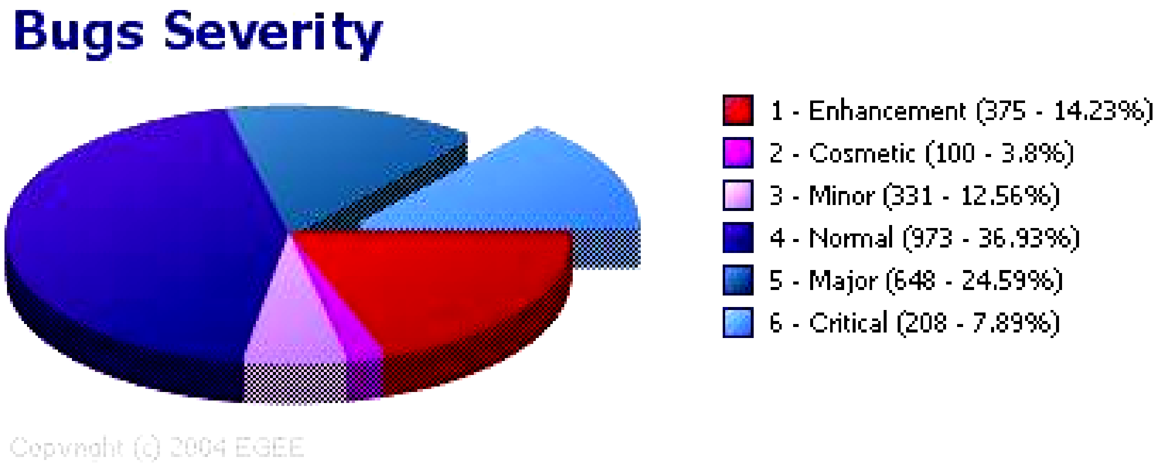


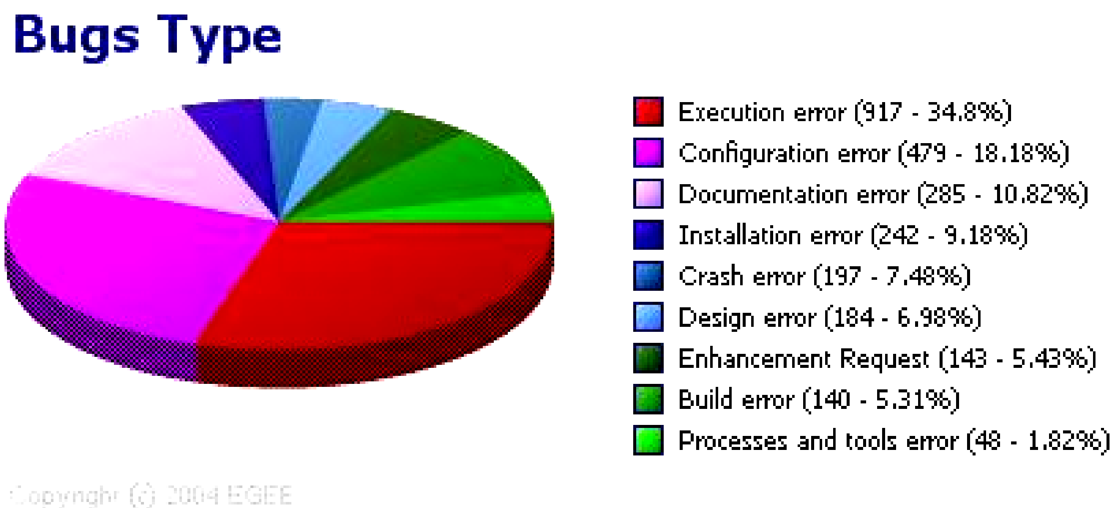
Software Defects Statistics





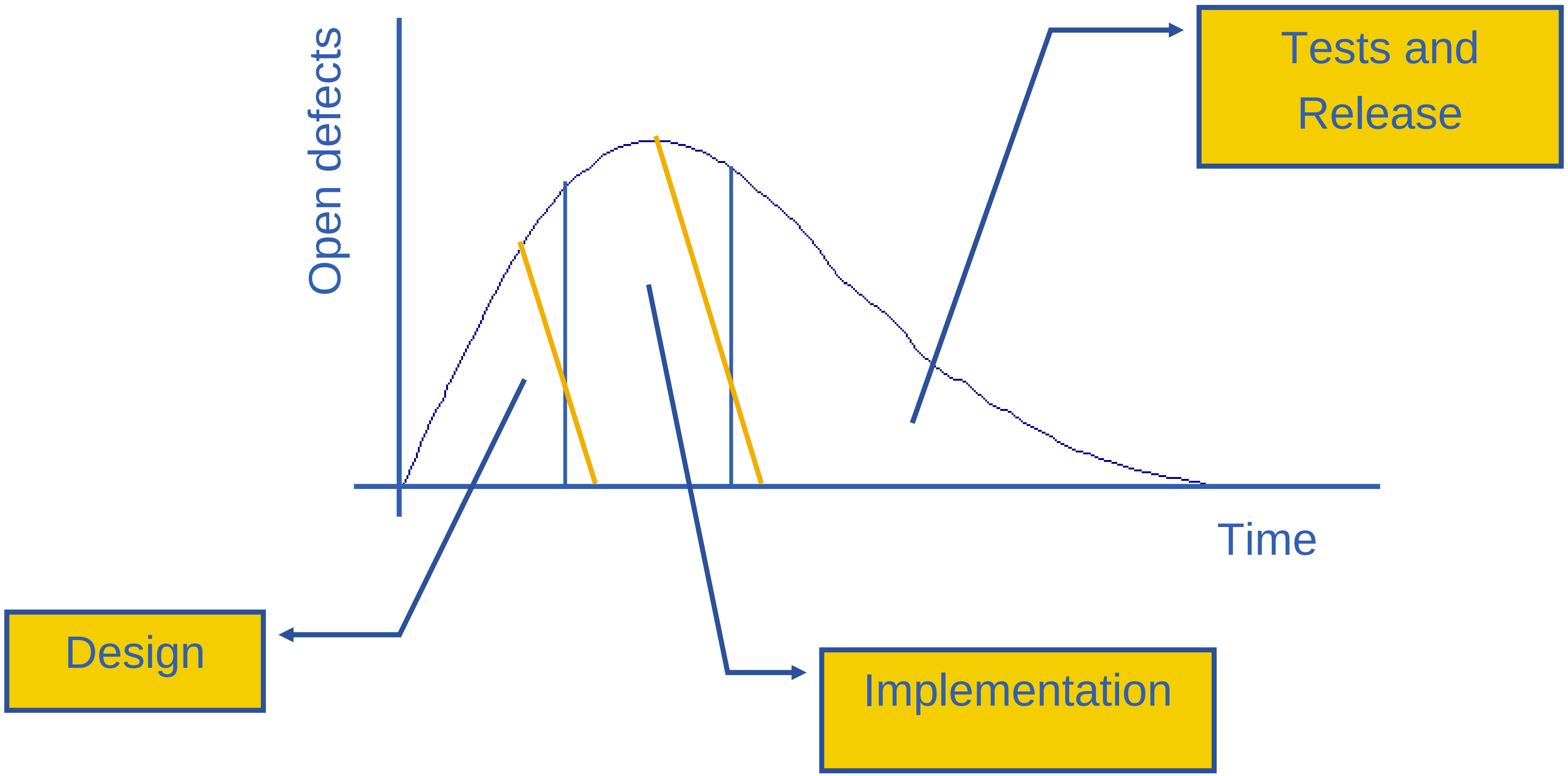
Software Defects Statistics



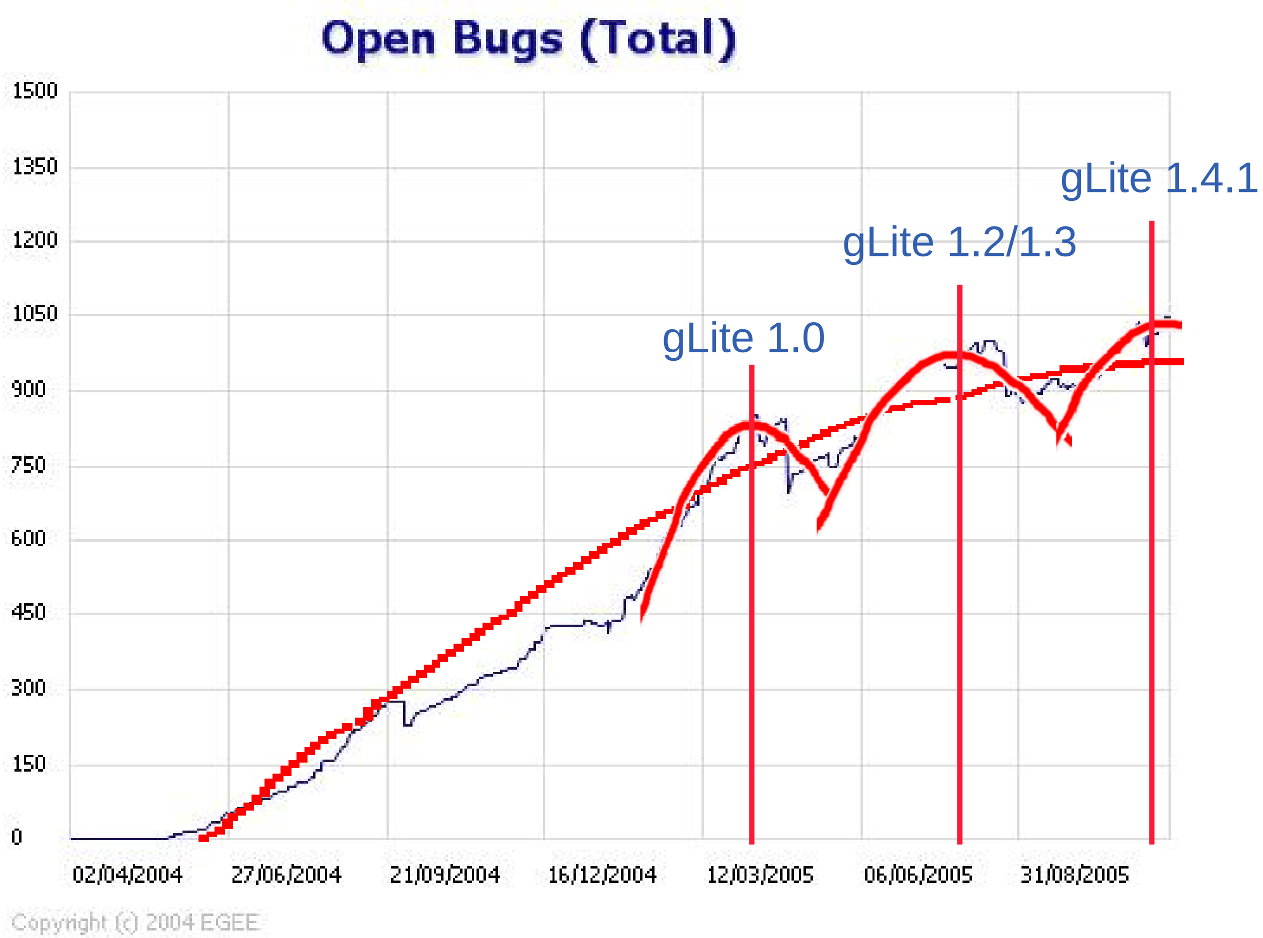


Defects Trends (I)

The Rayleigh Defect Prediction Model

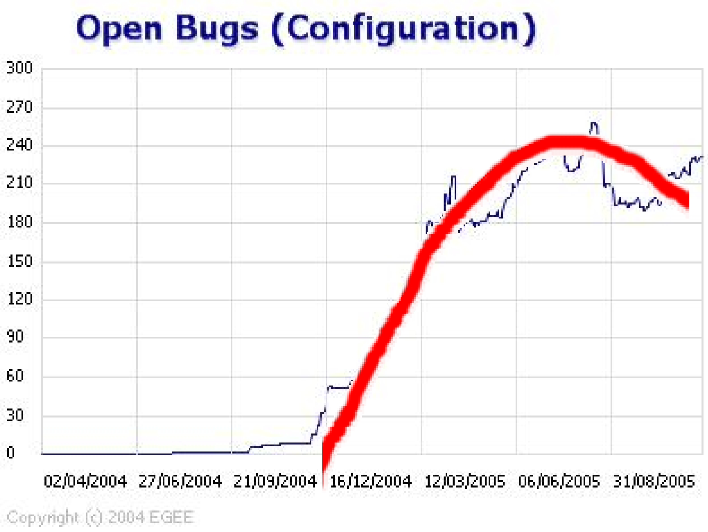


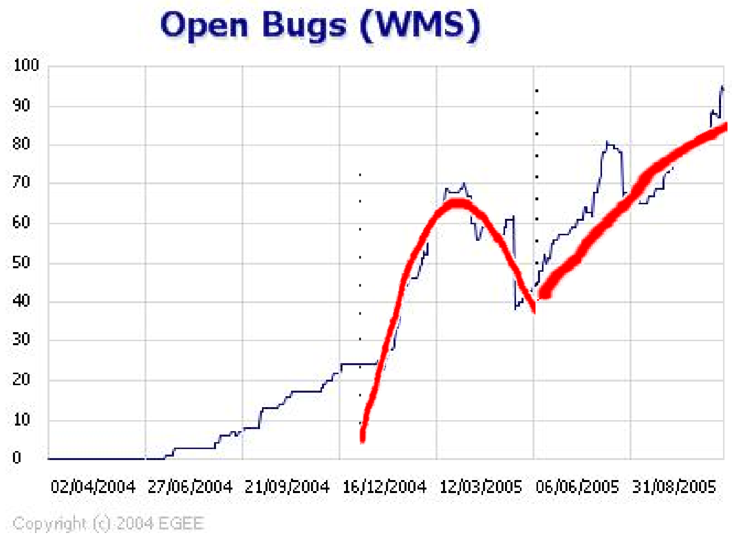
Defects Trends (II)

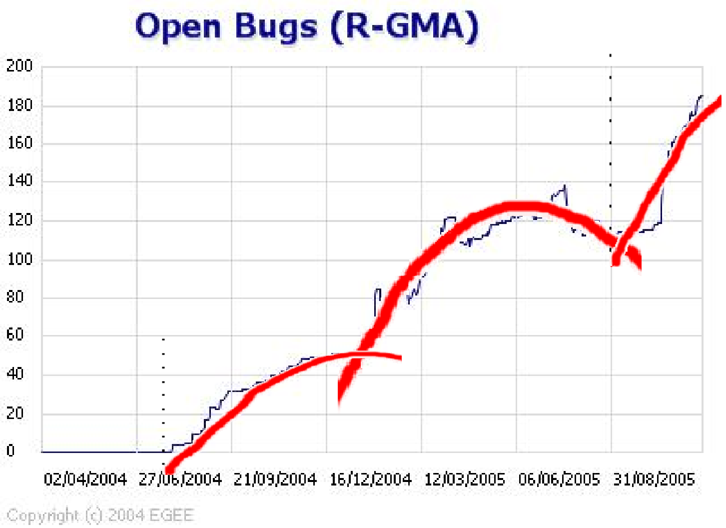


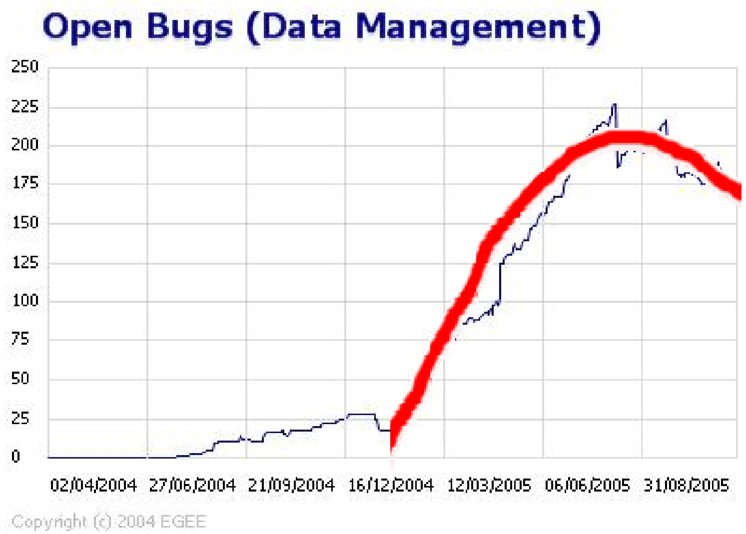
The Rayleigh Defect Prediction Model applied to gLite

Defects Trends (III)









More information

http://www.glite.org

http://cern.ch/egee-jra1