

# *Introduction to Bug Tracking*

Gérald Monard

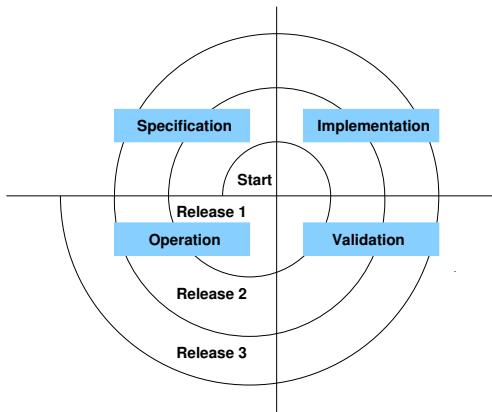
Ecole GDR CORREL - April 16, 2013

[www.monard.info](http://www.monard.info)

# INTRODUCTION

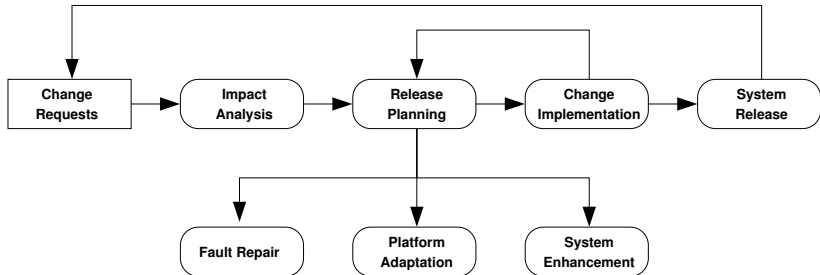
## *Software evolution*

- Software engineering can be seen as a spiral process with requirements, design, implementation, and testing going on throughout the lifetime of the system
- Development is performed within a team
- Developers can be spread out all over the world



# INTRODUCTION

## *Overview of the evolution process*



- Decisions should be made to define what should be included or not in the software
- Different roles: Users, Developers, but also Reviewers, Integrators, Managers, Administrators, etc.

# BUG TRACKING

## *Outline*

1. The Aegis example
2. Bugzilla
3. Trac
4. CruiseControl

# THE AEGIS EXAMPLE

## *Aegis: A Project Change Supervisor*

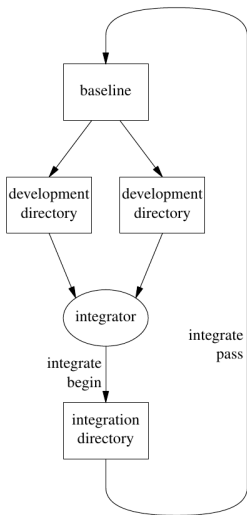
- Software developed by Peter Miller (GNU License, available in SourceForge)
- Aegis is a **Software Configuration Management**
- Aegis provides a framework within which a team of developers may work on many changes to a program independently, and coordinates integrating these changes back into the master source of the program.

# THE AEGIS EXAMPLE

## *The Aegis model*

- The repository is called a **baseline**
- The baseline has one particular attribute: **it always works**
- The baseline contains not only the source of a project, but also the tests for a project.
- Tests are treated just like any other source file. The baseline is defined to “work” if and only if it passes all of its own tests.
- The Aegis program has mandatory testing, to ensure that all changes to the baseline are accompanied by tests, and that those tests have been run and are known to pass.

# THE AEGIS EXAMPLE



- The model may be summarized briefly: it consists of a *baseline* (master source), updated through the agency of an *integrator*, who is in turn fed *changes* by a team of *developers*.

# THE AEGIS EXAMPLE

## *The Change Mechanism*

- Any changes to the baseline are made by atomic increments, known as *changes*.
- A change is a collection of files to be added to, modified in, or deleted from, the baseline.
- Changes must be accompanied by tests.
- Tests will either establish that a bug has been fixed (bug fix), or will establish that new functionality works (enhancement)

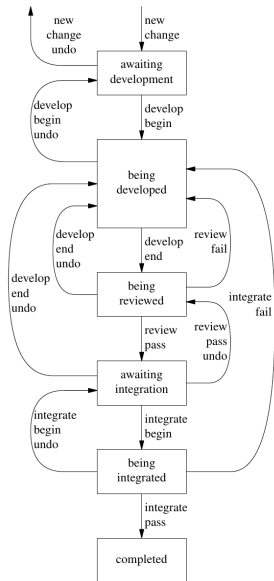


# THE AEGIS EXAMPLE

## *The Change States*

As a change is developed using Aegis, it passes through seven states.

1. Awaiting development
2. Being developed
3. Awaiting review
4. Being reviewed
5. Awaiting integration
6. Being integrated
7. Completed



# THE AEGIS EXAMPLE

## *The Software Engineers*

- The model of software development used by Aegis has four different roles for software engineers to fill.
  - These four roles may be overlapping sets of people, or be distinct, as appropriate for the project.
1. **Developer** This is the only role allowed to edit a source file of the project.

# THE AEGIS EXAMPLE

## *The Software Engineers*

- The model of software development used by Aegis has four different roles for software engineers to fill.
- These four roles may be overlapping sets of people, or be distinct, as appropriate for the project.

### 1. Developer

2. **Reviewer** Its role is to check a developer's work. This review consists of peer examining the code.

The reviewer must pass or fail each change that he/she reviews.

Before the change comes up for review: it builds, it has tests, they have run successfully.

# THE AEGIS EXAMPLE

## *The Software Engineers*

- The model of software development used by Aegis has four different roles for software engineers to fill.
- These four roles may be overlapping sets of people, or be distinct, as appropriate for the project.

1. **Developer**

2. **Reviewer**

3. **Integrator** Its role is to take a change which has already been reviewed and integrate it with the baseline, to form a new baseline. The baseline is readable by all developers, but not writable. All updates of the baseline to reflect changes produced by developers are performed through the agency of the integrator.

# THE AEGIS EXAMPLE

## *The Software Engineers*

- The model of software development used by Aegis has four different roles for software engineers to fill.
- These four roles may be overlapping sets of people, or be distinct, as appropriate for the project.

1. Developer

2. Reviewer

3. Integrator

4. Administrator He/she has the following duties:

- ✓ Create new changes
- ✓ Grant testing exemptions
- ✓ Add or remove staff
- ✓ Edit project attributes
- ✓ Edit change attributes

# THE AEGIS EXAMPLE

## *The development phase*

- From the list of changes “awaiting development”, the developer assigns that change to herself. The change is then “being developed”.
- To leave the “being developed” state to enter the “being reviewed” state:
  1. the change must have source files
  2. it must have tests
  3. it must have built successfully
  4. it must have passed all its own tests
  5. it must have been differenced

# THE AEGIS EXAMPLE

## *The tests*

Tests are (almost) mandatory for any change. There are 3 kinds of tests

1. If a change contains a new test or a test which is being modified, this test must pass against the code compiled and linked in the change.
2. If a change contains a new test and the change is a bug fix, this test must *fail* against the old code in the baseline. This is to confirm that the bug has been fixed. This is referred to as a “baseline test”.
3. Tests which already exist in the baseline may be run against the code compiled and linked in the change. These tests must pass. This is to confirm that the project has not regressed. These tests are referred to as “regression tests”.

# THE AEGIS EXAMPLE

## *Further reading*

- *Aegis - A Project Change Supervisor's User Guide*  
(Peter Miller, <http://aegis.sourceforge.net>)
- <http://aegis.sourceforge.net/propaganda/index.html>



# BUGZILLA

## *What is Bugzilla*

- Bugzilla is a Web-based general-purpose **bugtracker** and **testing tool** originally developed and used by the Mozilla project
- A **bug tracking** system or defect tracking system is a software application that is designed to help keep track of reported software bugs in software development efforts. It may be regarded as a type of **issue tracking** system.

# BUGZILLA

## *Issue tracking systems*

- An issue tracking system is a computer software package that manages and maintains lists of **issues**, as needed by an organization.
- Issue tracking systems are commonly used in an organization's customer support call center to create, update, and resolve reported customer issues
- An issue tracking system often also contains a knowledge base containing information on each customer, resolutions to common problems, etc.
- A **ticket** is an element which contains information about support interventions made by technical support staff
- Tickets are commonly created in a help desk or call center environment.
- Typically the ticket will have a unique reference number which is used to allow the user or support staff to quickly locate, add to or communicate the status of the user's issue or request.

# BUGZILLA

## *Issue tracking system: architecture*

The most common issue tracking system's design is relatively simple.

- A database is the main storage repository for all data.
- The architecture is a typical Model-View-Controller architecture
- The end-user can create entirely new issues, read existing issues, add details to existing issues, or resolve an issue.
- When a user of the system makes a change, the issue tracking system will record the action and who made it, so as to maintain a history of the actions taken.
- Each user of the system may have issues assigned to them.
- For security, an issue tracking system will authenticate its users before allowing access to the systems.

# BUGZILLA

## *Issue tracking system: Issues*

- Each issue in the system may have an urgency value assigned to it, based on the overall importance of that issue.
- Critical issues are the most severe and should be resolved in the most expedient way possible, taking precedence over all other issues.
- Low or zero urgency issues are minor, and should be resolved as time permits.
- Other details of issues include
  - ✓ the customer experiencing the issue (whether external or internal),
  - ✓ date of submission,
  - ✓ detailed descriptions of the problem being experienced,
  - ✓ attempted solutions or work-arounds
  - ✓ etc.

# BUGZILLA

## *What Does Bugzilla Do?*

- Track bugs and code changes
- Communicate with teammates
- Submit and review patches
- Manage quality assurance (QA)

# BUGZILLA

## *Features (from Bugzilla website)*

### For Users

- Advanced Search Capabilities
- Email Notifications Controlled By User Preferences
- Bug Lists in Multiple Formats (Atom, iCal, etc.)
- Scheduled Reports (Daily, Weekly, Hourly, etc.) by Email
- Reports and Charts
- Automatic Duplicate Bug Detection
- File/Modify Bugs By Email
- Time Tracking
- Request System
- Patch Viewer
- "Watch" Other Users
- Move Bugs Between Installs

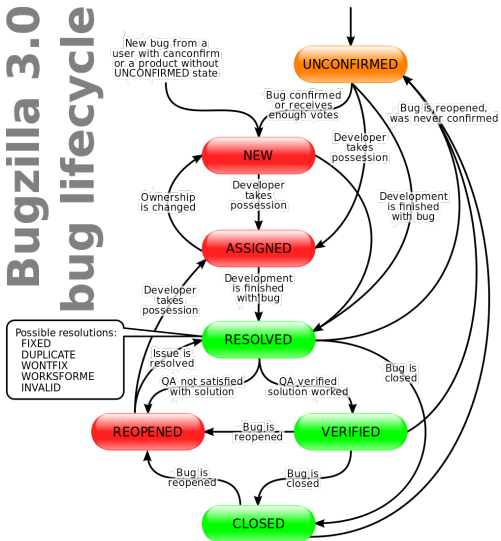
### For Administrators

- Excellent Security
- Extension Mechanism for Highly Customizable Installations
- Custom Fields
- Custom Workflow
- Full Unicode Support
- Localization
- Webservices (XML-RPC) Interface
- Control Bug Visibility/Editing with Groups
- Impersonate Users
- Multiple Authentication Methods
- Support for Multiple Database Engines
- Sanity Check

# BUGZILLA

## Lifecycle

### Bugzilla 3.0 bug lifecycle



# TRAC

## *What is Trac?*

- Trac is an open source web-based project management and bug tracking system.
- Trac is written in the Python programming language.
- Trac allows hyperlinking information between a bug database, revision control and wiki content.
- It also serves as a web interface to the following revision control systems: Subversion, Git, Mercurial, Bazaar, Perforce and Darcs.



# TRAC

## *Other features*

- Project management (Roadmap, Milestones, etc.)
- Ticket system (bug tracking, tasks, etc.)
- Fine-grained permissions
- Timeline of all recent activity
- Wiki (syntax similar to MoinMoin)
- Customized reporting
- version control system web interface
- RSS feeds
- Multiple project support
- Environment extensibility (via Python plugins)
- iCalendar export
- Multiple repository Support per environment
- Interface localizations

# CRUISECONTROL

## *What is CruiseControl*

- CruiseControl is a Java-based framework for a continuous build process.
- It allows one to perform a continuous integration of any software development process.
- CruiseControl is free, open-source software, distributed under a BSD-style license.

# CRUISECONTROL

*CruiseControl is composed of 3 main modules*

- The **build loop** is designed to run as a daemon process, which periodically checks the revision control system for changes to the codebase, builds if necessary, and publishes a notice regarding the status of the software build
- the **jsp reporting** application allows the users to browse the results of the builds and access the artifacts
- the **dashboard** provides a visual representation of all project build statuses.

# CRUISECONTROL

## *What is the Build Loop?*

The Build Loop is designed to run as a daemon process which will periodically

- check your source control tool for changes to your codebase,
- build if necessary,
- send out a notification regarding the status of the build.

## *How Does It Work?*

CruiseControl defines a build cycle:

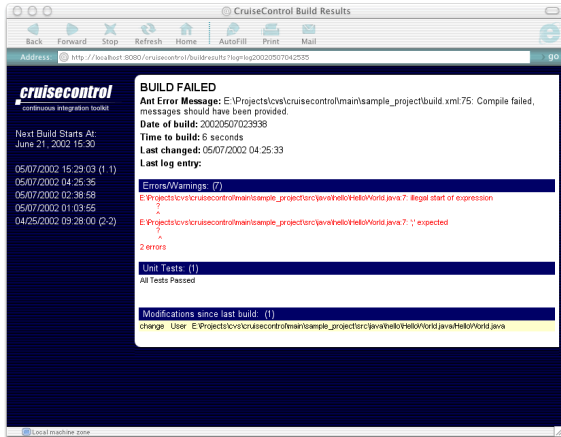
- determine if a build is necessary,
- build,
- create a log file,
- send notifications.

The daemon process will wake up at a user defined time interval and attempt a build cycle.

# CRUISECONTROL

## *What is the Build Results JSP?*

The Build Results JSP is designed to present the results of the cruisecontrol build loop.



The screenshot shows a web browser window titled "CruiseControl Build Results". The address bar shows the URL: `http://localhost:8080/cruisecontrol/buildresults?log=log20020507042535`. The browser's navigation bar includes buttons for Back, Forward, Stop, Refresh, Home, AutoFill, Print, and Mail. The main content area has a dark blue background with white text. On the left, the "cruisecontrol" logo is displayed with the tagline "continuous integration toolkit". Below the logo, it indicates the next build start time: "Next Build Starts At: June 21, 2002 15:30". A list of recent build times is shown: "05/07/2002 15:29:03 (1, 1)", "05/07/2002 04:25:36", "05/07/2002 02:38:58", "05/07/2002 01:03:55", and "04/25/2002 09:28:00 (2-2)". The main content area displays the following information:

- BUILD FAILED**
- Ant Error Message:** E:\Projects\cvsc\cruisecontrol\main\sample\_project\build.xml:75: Compile failed, messages should have been provided.
- Date of build:** 20020507023938
- Time to build:** 6 seconds
- Last changed:** 05/07/2002 04:25:33
- Last log entry:**

Below this information, there are three expandable sections:

- Errors/Warnings: (7)**  
E:\Projects\cvsc\cruisecontrol\main\sample\_project\src\java\hello\HelloWorld.java:7: Illegal start of expression  
?  
^  
E:\Projects\cvsc\cruisecontrol\main\sample\_project\src\java\hello\HelloWorld.java:7: ';' expected  
?  
^  
2 errors
- Unit Tests: (1)**  
All Tests Passed
- Modifications since last build: (1)**  
change User E:\Projects\cvsc\cruisecontrol\main\sample\_project\src\java\hello\HelloWorld.java\HelloWorld.java

The browser's status bar at the bottom indicates "Local machine zone".

# CRUISECONTROL

## What is the CruiseControl Dashboard?

- The CruiseControl Dashboard is a tool to help visualize the project statuses.
- Previous project builds result are color-coded
- The build results are overlaid with icons showing the current status of the project (for example paused, queued or building).

AMBER Automated Build System Dashboard



Dashboard Builds

### NOTE:

This information is provided for the 50 most recent builds, with most recent being to the right.  
The number of tests that fail is shown above each data point  
CUDA tests are currently not running correctly due to hardware issues.  
**All builds experienced errors due to hardware issues.**

