Continuous Software Quality analysis for the ATLAS experiment

Andrew Washbrook on behalf of the ATLAS collaboration
University of Edinburgh
ACAT 2017, Seattle
22nd August 2017
Software Defects

Examples
- Redundant code paths
- Errors of omission
- Inefficient use of allocated memory
- Software defects may not be flagged by compilers

Why is resolving software defects important?
- If left unchecked the accumulation of defects can result in:
  - Performance degradation at scale
  - Problems with the long-term sustainability of the software

Examples
```
int *particleID = new int;
*particleID = newValue;
...```
Simple example of a C++ software defect (memory leak)
Software Quality Evaluation on ATLAS

The regular application of software quality tools in large collaborative projects is required to reduce software defects to an acceptable level

- Software quality tools are used by the ATLAS developer community to identify, track and resolve any defects in close to 6 million lines of code
- cppcheck and the Synopsys Static Analysis Tool (Coverity) regularly scan the entirety of the main software release
  - Results are available in custom portals accessible for all developers
  - Scheduled notifications of any urgent defects to code maintainers

- More general code quality indicators, coverage testing tools and code formatting checkers are also used as part of the development and build process
Limitations and new approaches

- **Uninitialised variables** and **sources of memory leaks** are usually dealt with promptly.
- Other defects in non-critical sections of code often remain unresolved.
- This leads to a backlog of legacy defects where:
  - Responsibility and provenance of the code is unrecorded.
  - Developer effort is re-organised or not retained.

### How can this be addressed?

- Defects periodically re-evaluated and disregarded if their impact is marginal.
- Identify and address defects **before** they are introduced into a software release.

### Defects by age of first detection

<table>
<thead>
<tr>
<th></th>
<th>HIGH</th>
<th>MEDIUM</th>
<th>LOW</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 3 MONTHS</td>
<td>9</td>
<td>76</td>
<td>10</td>
<td>95</td>
</tr>
<tr>
<td>3-6 MONTHS</td>
<td>13</td>
<td>85</td>
<td>7</td>
<td>105</td>
</tr>
<tr>
<td>6-12 MONTHS</td>
<td>54</td>
<td>531</td>
<td>50</td>
<td>635</td>
</tr>
<tr>
<td>&gt;12 MONTHS</td>
<td>83</td>
<td>1205</td>
<td>460</td>
<td>1748</td>
</tr>
</tbody>
</table>
ATLAS Software Build Infrastructure

**Source Code Management**
- *git* version control
- *Gitlab* social coding platform

**Merge Request Process**
- Code reviews performed by a dedicated rota of shifters to validate any changes

**Continuous Integration (CI)**
- Lightweight testing and build correctness checking for each proposed code modification
ATLAS Code Review Process

ATLAS Offline Software Repository

Developer Fork

Release Branch

Packages affected by Merge Request (MR)

Labels for code review management

Continuous Integration (CI) results

GitLab

Jenkins

W Job

CI-MERGE-REQUEST

preparation

CI-stamp-mr

labels

CI-assign-labels

build

CI-gcc-build

test

CI-unit-tests

logs_handler

CI-logs-handler

Request to merge panduro:21.1_add_new_trigdb.py_methods into 21.1 (157 commits behind)

Pipeline #169933 passed with stage for 0001404.

Merge Ready to be merged automatically. Ask someone with write access to this repository to merge this request.

Discussion | Commits | Pipelines | Changes

ATLAS Robot @atlasbot commented about 12 hours ago

This merge request affects 2 packages:

• TrigEvent/TrigConfiguration/TrigConfDBConnection
• TrigEvent/TrigConfiguration/TrigConfSvc

ATLAS Robot @atlasbot added 11 labels about 12 hours ago

ATLAS Robot @atlasbot commented about 7 hours ago

CI Result SUCCESS

externals

make

make

test

Full details available at NICOS MR-3665-2017-08-04-43

For experts only: Jenkins output [CI-MERGE-REQUEST 5963] (for remote access see instructions in Jenkins section here)
Continuous Software Quality Evaluation

Ideal opportunity to apply software quality checks as part of the new code review process

- Code review shifters can catch defects as they are introduced
- Defects are audited at source for free as part of the merge request discussion

Some practicalities

- Software Quality CI tests should be quick (less than 5 minutes)
  - Avoid additional load on CI servers
  - Reasonable response time expected by shifters to progress review
- Ideally perform checks only on the code directly affected by any changes in a given merge request
- Test results should be only used as advisory information in the review discussion
Testing Infrastructure

- **Distributed** testbed provides a development sandbox without interruption to the production ATLAS CI System.
- **Container images** of key services easily instantiated across multiple sites.
- Instance configuration snapshots stored in a common Gitlab container registry.
- Test harness emulates representative merge request patterns.
- Software quality CI tests deployed to production once fully validated.
Continuous Integration `cppcheck` Test

- Feasibility testing using a lightweight static code analysis application (`cppcheck`).
- Feedback in code review indicates a state change based on the modified code.
- Defects are either introduced, removed or remain unresolved against a reference result generated from the main development branch.
Continuous Integration **cppcheck** Report

**Sample summary report in Merge Request discussion**

**First check for introduced defects**

Then flag any defects contained in files modified by developer

Order by Severity

Truncate list if large number of defects found

Link to full test results on Jenkins server

**Cpcheck Results**

- No new defects were introduced by this merge request.
- 14 defects unresolved in files changed by this merge request.

<table>
<thead>
<tr>
<th>Severity</th>
<th>Defect</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigALFAROBMonitor::m_hist_goodData' is not initialized in the ..</td>
<td>TrigALFAROBMonitor.cxx:65</td>
</tr>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigALFAROBMonitor::m_hist_goodDataLB15' is not initialized in ..</td>
<td>TrigALFAROBMonitor.cxx:65</td>
</tr>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigALFAROBMonitor::m_hist_goodDataLB18' is not initialized in ..</td>
<td>TrigALFAROBMonitor.cxx:65</td>
</tr>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigALFAROBMonitor::m_hist_PosDetector' is not initialized in t..</td>
<td>TrigALFAROBMonitor.cxx:65</td>
</tr>
</tbody>
</table>

(and 10 other defects of type WARNING)

2 other defects remain unresolved in these packages affected by this merge request:

<table>
<thead>
<tr>
<th>Severity</th>
<th>Defect</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigL1TopoROBMonitor::m_histTopoSimOverfl' is not initialized i..</td>
<td>TrigL1TopoROBMonitor.cxx:74</td>
</tr>
<tr>
<td>WARNING</td>
<td>Member variable 'TrigL1TopoROBMonitor::m_histTopoHdwOverfl' is not initialized i..</td>
<td>TrigL1TopoROBMonitor.cxx:74</td>
</tr>
</tbody>
</table>

Further Details can be found in the **cppcheck Jenkins report**

Drill down into defects by file ordered by severity and amount

Link to code browsing location in Gitlab

Finally show remaining defects from affected packages
Coverity Quality Analysis

- More comprehensive defect coverage provided by **Coverity** static analysis
- However full build, analysis and commit steps are not suitable for CI running
  - Over 30 hours to run across entire release
- Investigating use of **incremental builds** or **desktop analysis** methods as part of a CI test job
- Apply similar comparison and reporting mechanism

**Can this defect comparison workflow be extended to other code quality tools?**

- Only limited to the format of the results output so methods could be abstracted to parse results from elsewhere
Software Quality Trend Analysis

- Also possible to apply *holistic* measurements of code quality to the review process

**How can these indicators be best interpreted?**

- Single value quality metrics are not instructive
- Instead capture *trend information* through the evolution of the code to put any reported value into context
- Define acceptable thresholds before developer action should be taken

**Example Code Quality Indicators** [1,2]

- Lines of code with comments
- Cyclomatic Complexity
- Halstead Program Difficulty
- Class Coupling
- Function Decision Depth

[1] https://github.com/terryyin/lizard
Trend Analysis Example

- Use **Lizard** as an example code quality indicator tool
- Captured code quality data for **15** snapshots of full release
- Each release has over **51,000** files and **219,000** functions

Injection of highly-branched code section to test cyclomatic complexity monitoring

![Diagram of Trend Analysis Example](image)
Defect Triage Methods

- Promote defect resolution and assign responsibility through **reviewer-led triage**
- Unimportant or incorrectly identified defects need to be flagged to aid future identification

**Possible Methods**

- Check and maintain defect suppression lists
- Make Coverity-based triage data accessible to Gitlab and issue tracking (JIRA)
- Use **Gitlab webhooks** to monitor triage trigger actions in the merge request discussion
Other Considerations

Alternative Solutions

- [Sonarqube](https://www.sonarqube.org/) provides an alternative option for continuous software quality evaluation
- Current emphasis is on integrating services into our chosen workflow rather than inclusion of new tools

Attribution

- Responsive contact person for every line of code is not trivial in a large collaboration
- Trace roles and responsibility through [git](https://git-scm.com/) commit history and project mapping

Developer-led defect analysis

- Defects can be caught by the developer **before they commit their code**
  - [cppcheck](https://cppcheck.net/) git pre-commit hooks
  - Coverity desktop analysis tools and IDE plugins
Outlook

- **Continuous software quality evaluation** for ATLAS can be achieved by including lightweight defect testing into the code review process.

- Accumulation of experience from review shifters and developers will help with optimising defect tests and results presentation.

- More extensive code quality reporting mechanisms are being evaluated.

- Chosen solutions aim to be project agnostic
  - Greatly helped by recent migration from bespoke and legacy tools.
  - Similar approaches could be applied elsewhere.