Continuous Software Quality analysis for the ATLAS experiment at CERN

Andrew Washbrook *on behalf of the ATLAS collaboration*
University of Edinburgh
WSSSPE5.1 Workshop, Manchester
6th September 2017

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 **software 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

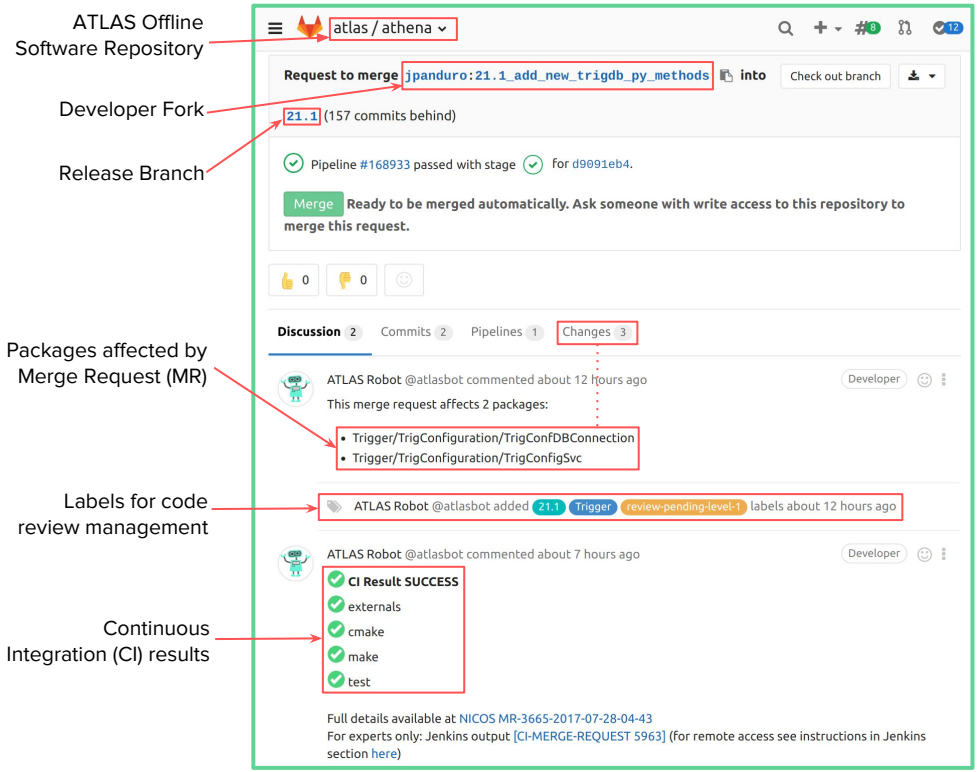
	HIGH	MEDIUM	LOW	TOTAL
< 3 MONTHS	9	76	10	**95**
3-6 MONTHS	13	85	7	**105**
6-12 MONTHS	54	531	50	**635**
>12 MONTHS	83	1205	460	**1748**

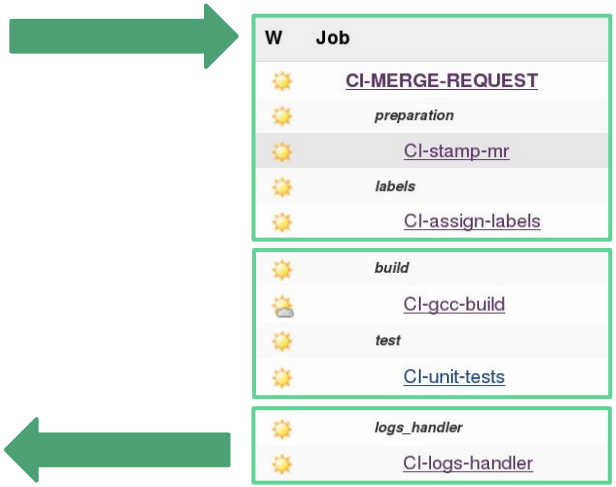
Defects by age of first detection

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

ATLAS Code Review Process





- Code reviews performed by a **dedicated rota of shifters** to validate any changes
- Lightweight testing and build correctness checking for each proposed code change

Continuous Software Quality Evaluation

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

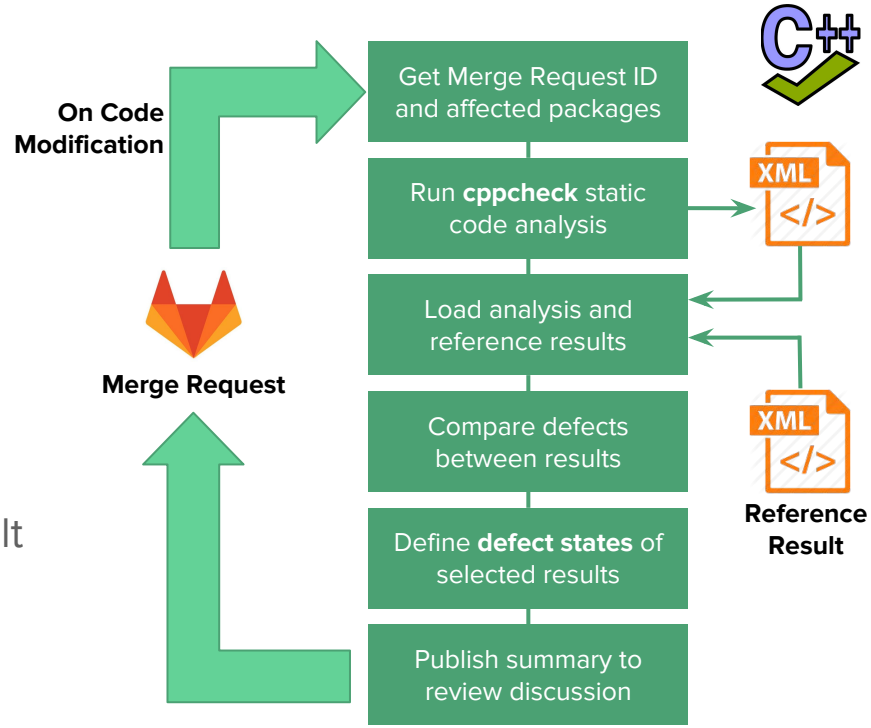
- Code review shifters can catch defects **as they are introduced**
- Defects are audited at source **for free** as part of the merge request discussion

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

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

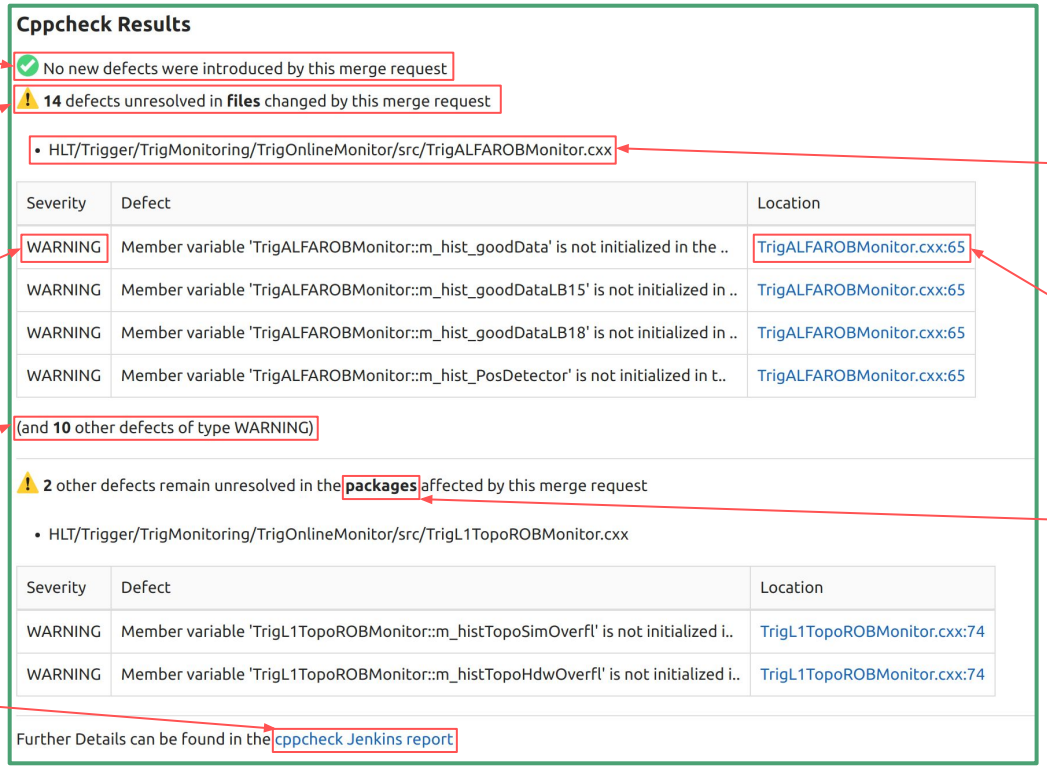
First check for introduced defects

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



Cppcheck Results

No new defects were introduced by this merge request

14 defects unresolved in **files** changed by this merge request

- HLT/Trigger/TrigMonitoring/TrigOnlineMonitor/src/TrigALFAROBMonitor.cxx

Severity	Defect	Location
WARNING	Member variable 'TrigALFAROBMonitor::m_hist_goodData' is not initialized in the ..	TrigALFAROBMonitor.cxx:65
WARNING	Member variable 'TrigALFAROBMonitor::m_hist_goodDataLB15' is not initialized in ..	TrigALFAROBMonitor.cxx:65
WARNING	Member variable 'TrigALFAROBMonitor::m_hist_goodDataLB18' is not initialized in ..	TrigALFAROBMonitor.cxx:65
WARNING	Member variable 'TrigALFAROBMonitor::m_hist_PosDetector' is not initialized in t..	TrigALFAROBMonitor.cxx:65

(and **10** other defects of type WARNING)

2 other defects remain unresolved in the **packages** affected by this merge request

- HLT/Trigger/TrigMonitoring/TrigOnlineMonitor/src/TrigL1TopoROBMonitor.cxx

Severity	Defect	Location
WARNING	Member variable 'TrigL1TopoROBMonitor::m_histTopoSimOverfl' is not initialized i..	TrigL1TopoROBMonitor.cxx:74
WARNING	Member variable 'TrigL1TopoROBMonitor::m_histTopoHdwOverfl' is not initialized i..	TrigL1TopoROBMonitor.cxx:74

Further Details can be found in the cppcheck Jenkins report

Sample summary report in Merge Request discussion

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

Software Quality Trend Analysis

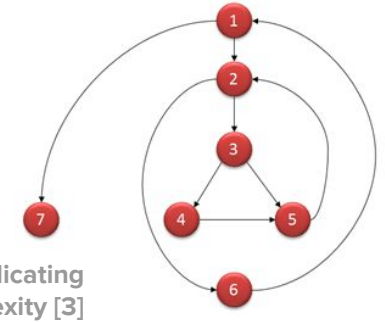
- Also possible to apply *holistic* measurements of code quality to the review process

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



Control flow diagram indicating cyclomatic complexity [3]

[1] https://github.com/terryyin/lizard
[2] https://www.imagix.com/user_guide/software-metrics.html
[3] https://www.guru99.com/cyclomatic-complexity.html

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

Additional Material

Software Defects

Examples

- Redundant code paths
- Errors of omission
- Inefficient use of allocated memory

```
int *particleID = new int;
*particleID = newValue;
...
```

Simple example of a C++ software defect (memory leak)

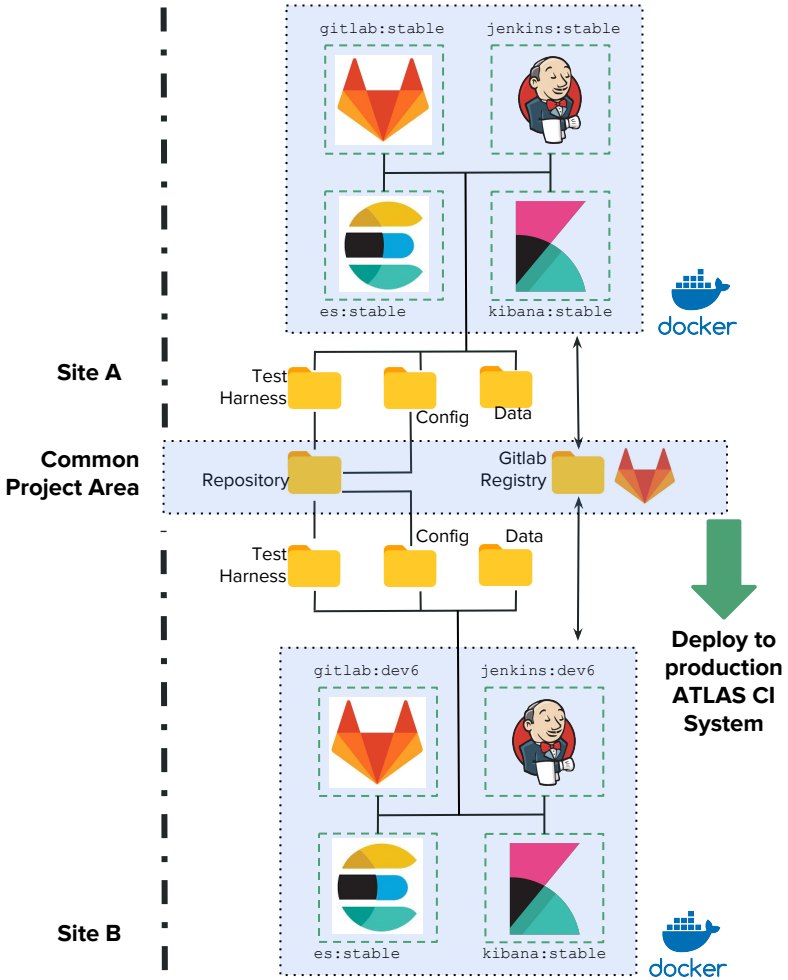
- **Software defects may not be flagged by compilers**

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

Testing Infrastructure

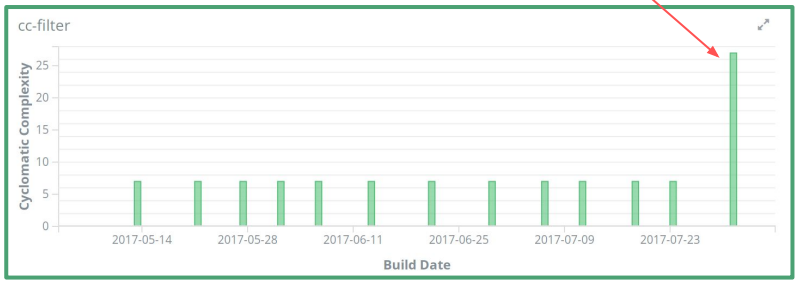
- **Distributed** testbed provides a development sandbox without interruption to the production ATLAS CI System
- **Container images** of key services easily instantiated across multiple sites
- Instance configuration snapshots stored in a common Gitlab container registry
- Test harness emulates representative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

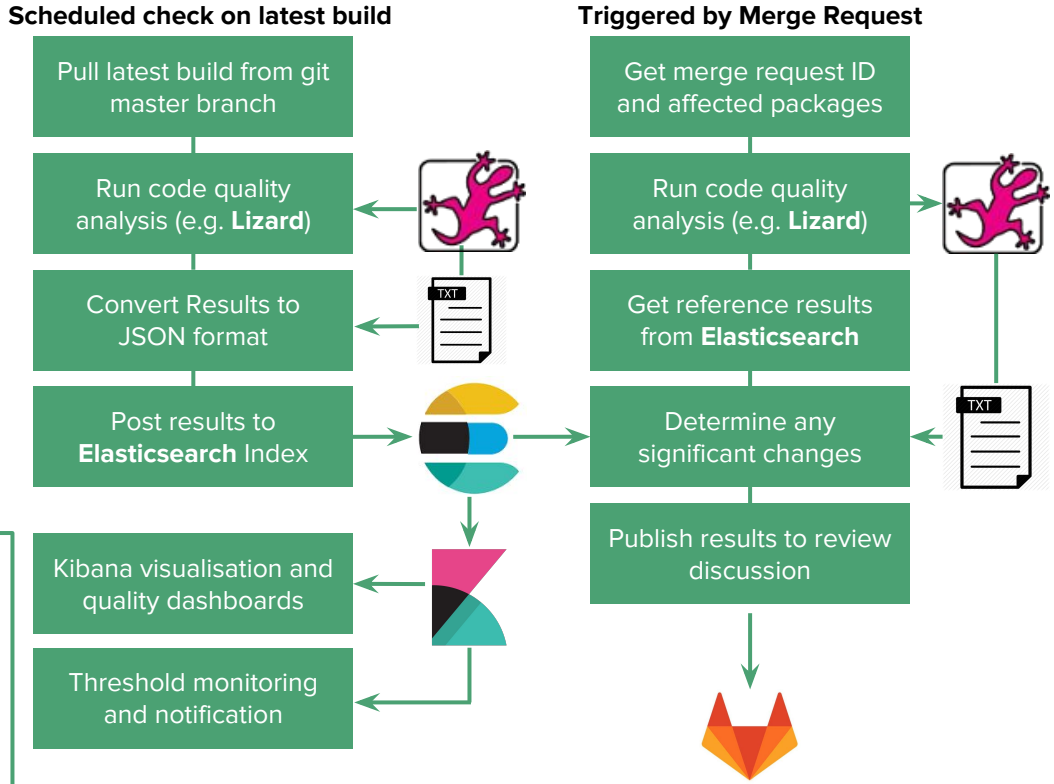


Trend Analysis Example

- 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





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