C++ Software Quality in the ATLAS experiment: Tools and Experience

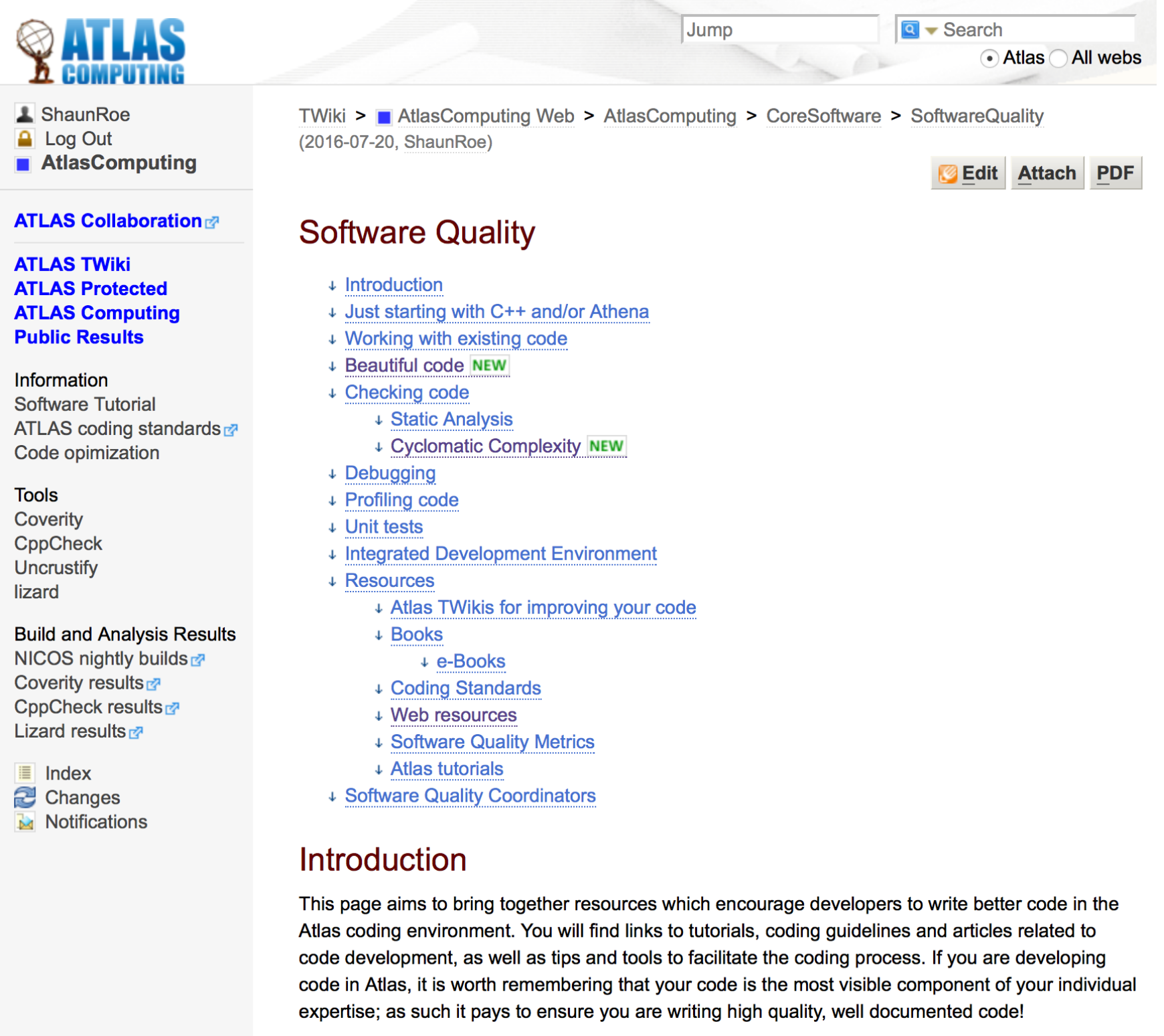
Stewart Martin-Haugh (STFC - Rutherford Appleton Lab. (GB)) Stefan Kluth (Max-Planck-Institut für Physik (Werner-Heisenberg-Institut) (D)) Rolf Seuster (University of Victoria (CA)) Scott Snyder (Brookhaven National Laboratory (US)) Emil Obreshkov (University of Texas at Arlington (US)) Shaun Roe (CERN) Peter Sherwood (University College London (UK)) Graeme Stewart (University of Glasgow (GB))

ATLAS Software:

~6 000 000 lines of Code; **140** Teams; **420** Developers; **2** Software Quality Coordinators

Information: TWikis provide education, coding tips, access to tools.
Tools: Continuous build, Static Analysis (Coverity®,cppcheck), Sanitizers, gcc plugins, Unit tests, "lizard".

TWikis



Jump
Search
○ Atlas ○ All webs

ShaunRoe
Log Out
AtlasComputing

ATLAS Collaboration

ATLAS TWiki
ATLAS Protected
ATLAS Computing
Public Results

Information
Software Tutorial
ATLAS coding standards
Code opimization

Tools
Coverity
CppCheck
Uncrustify
lizard

Build and Analysis Results
NICOS nightly builds
Coverity results
CppCheck results
Lizard results

Index
Changes
Notifications

TWiki > AtlasComputing Web > AtlasComputing > CoreSoftware > SoftwareQuality
(2016-07-20, ShaunRoe)

Edit Attach PDF

Software Quality

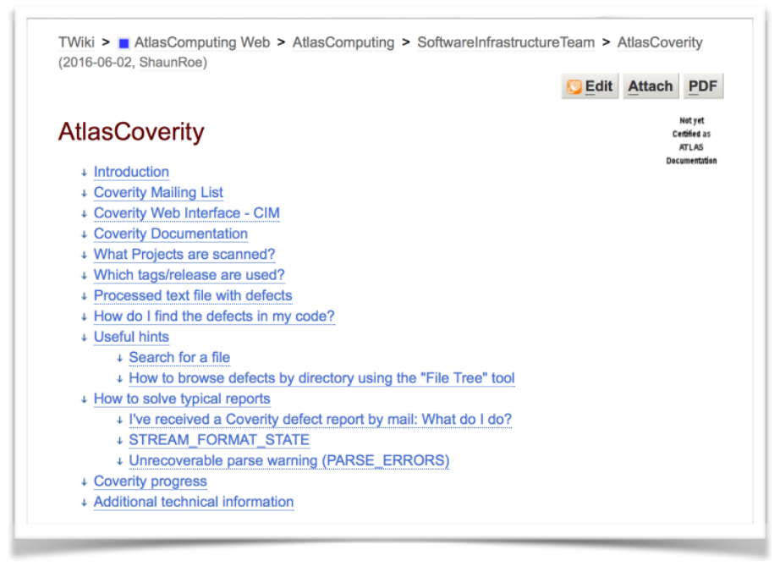
- Introduction
- Just starting with C++ and/or Athena
- Working with existing code
- Beautiful code NEW
- Checking code
  - Static Analysis
  - Cyclomatic Complexity NEW
- Debugging
- Profiling code
- Unit tests
- Integrated Development Environment
- Resources
  - Atlas TWikis for improving your code
  - Books
    - e-Books
  - Coding Standards
  - Web resources
  - Software Quality Metrics
  - Atlas tutorials
- Software Quality Coordinators

Introduction

This page aims to bring together resources which encourage developers to write better code in the Atlas coding environment. You will find links to tutorials, coding guidelines and articles related to code development, as well as tips and tools to facilitate the coding process. If you are developing code in Atlas, it is worth remembering that your code is the most visible component of your individual expertise; as such it pays to ensure you are writing high quality, well documented code!

https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/SoftwareQuality

Top level access



TWiki > AtlasComputing Web > AtlasComputing > SoftwareInfrastructureTeam > AtlasCoverity
(2016-06-02, ShaunRoe)

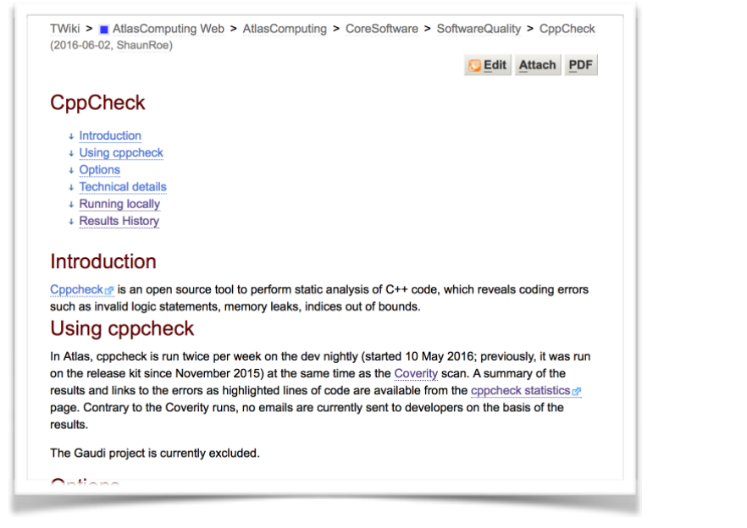
Edit Attach PDF

AtlasCoverity

Not yet Certified as ATLAS Documentation

- Introduction
- Coverity Mailing List
- Coverity Web Interface - CIM
- Coverity Documentation
- What Projects are scanned?
- Which tags/release are used?
- Processed text file with defects
- How do I find the defects in my code?
- Useful hints
  - Search for a file
  - How to browse defects by directory using the "File Tree" tool
- How to solve typical reports
  - I've received a Coverity defect report by mail: What do I do?
  - STREAM_FORMAT_STATE
  - Unrecoverable parse warning (PARSE_ERRORS)
- Coverity progress
- Additional technical information

https://twiki.cern.ch/twiki/bin/view/AtlasComputing/AtlasCoverity



TWiki > AtlasComputing Web > AtlasComputing > CoreSoftware > SoftwareQuality > CppCheck
(2016-06-02, ShaunRoe)

Edit Attach PDF

CppCheck

- Introduction
- Using cppcheck
- Options
- Technical details
- Running locally
- Results History

Introduction

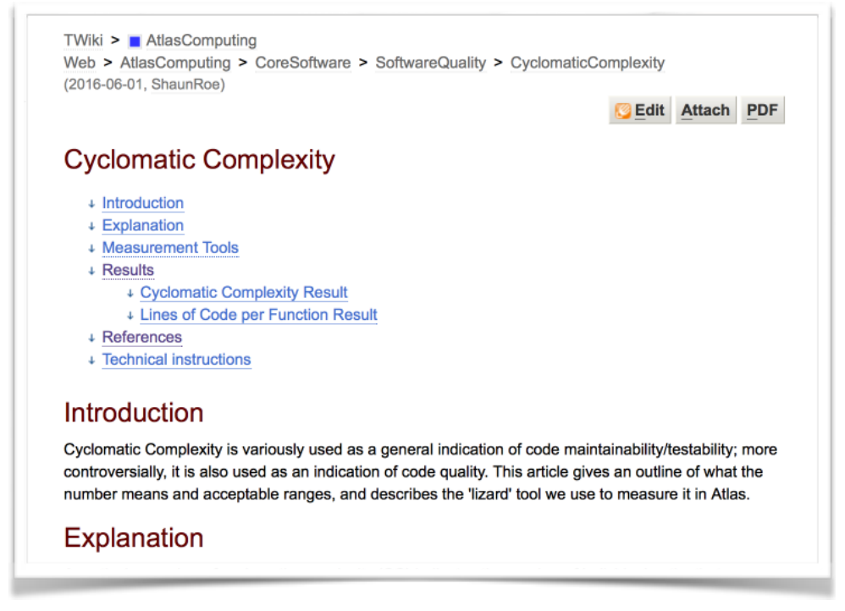
Cppcheck is an open source tool to perform static analysis of C++ code, which reveals coding errors such as invalid logic statements, memory leaks, indices out of bounds.

Using cppcheck

In Atlas, cppcheck is run twice per week on the dev nightly (started 10 May 2016; previously, it was run on the release kit since November 2015) at the same time as the Coverity scan. A summary of the results and links to the errors as highlighted lines of code are available from the cppcheck statistics page. Contrary to the Coverity runs, no emails are currently sent to developers on the basis of the results.

The Gaudi project is currently excluded.

https://twiki.cern.ch/twiki/bin/view/AtlasComputing/CppCheck



TWiki > AtlasComputing Web > AtlasComputing > CoreSoftware > SoftwareQuality > CyclomaticComplexity
(2016-06-01, ShaunRoe)

Edit Attach PDF

Cyclomatic Complexity

- Introduction
- Explanation
- Measurement Tools
- Results
  - Cyclomatic Complexity Result
  - Lines of Code per Function Result
- References
- Technical instructions

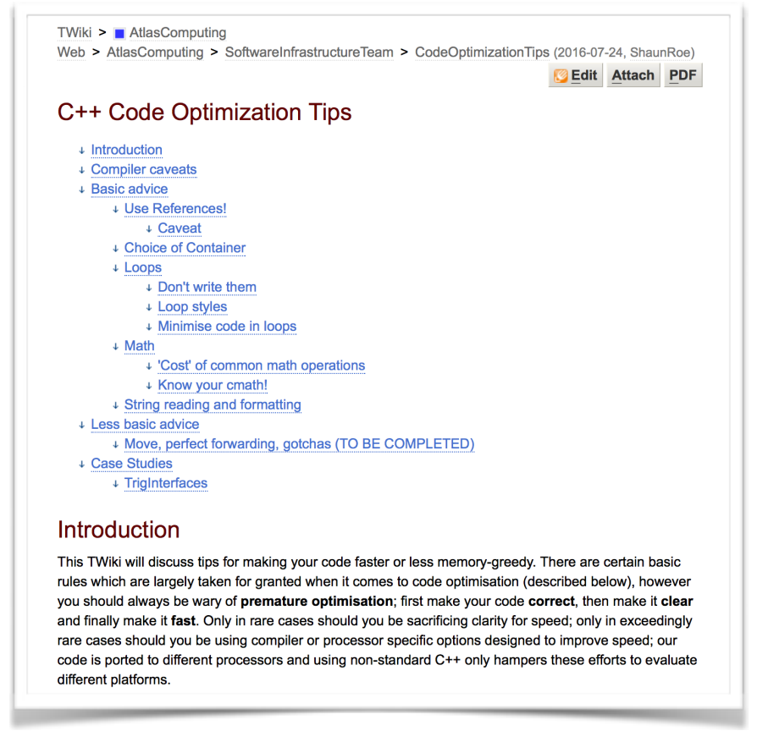
Introduction

Cyclomatic Complexity is variously used as a general indication of code maintainability/testability; more controversially, it is also used as an indication of code quality. This article gives an outline of what the number means and acceptable ranges, and describes the 'lizard' tool we use to measure it in Atlas.

Explanation

https://twiki.cern.ch/twiki/bin/view/AtlasComputing/CyclomaticComplexity

Tools



TWiki > AtlasComputing Web > AtlasComputing > SoftwareInfrastructureTeam > CodeOptimizationTips (2016-07-24, ShaunRoe)

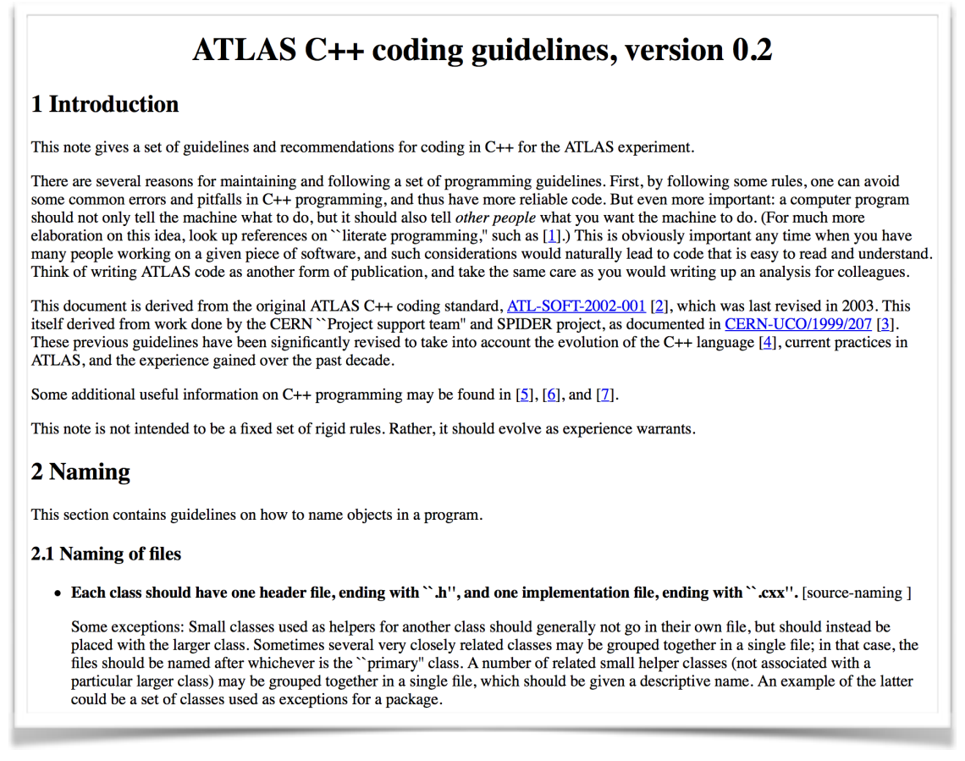
Edit Attach PDF

C++ Code Optimization Tips

- Introduction
- Compiler caveats
- Basic advice
  - Use References!
    - Caveat
  - Choice of Container
  - Loops
    - Don't write them
    - Loop styles
    - Minimise code in loops
  - Math
    - 'Cost' of common math operations
    - Know your cmath!
  - String reading and formatting
- Less basic advice
  - Move, perfect forwarding, gotchas (TO BE COMPLETED)
- Case Studies
  - TrigInterfaces

Introduction

This TWiki will discuss tips for making your code faster or less memory-greedy. There are certain basic rules which are largely taken for granted when it comes to code optimisation (described below), however you should always be wary of **premature optimisation**; first make your code **correct**, then make it **clear** and finally make it **fast**. Only in rare cases should you be sacrificing clarity for speed; only in exceedingly rare cases should you be using compiler or processor specific options designed to improve speed; our code is ported to different processors and using non-standard C++ only hampers these efforts to evaluate different platforms.



ATLAS C++ coding guidelines, version 0.2

1 Introduction

This note gives a set of guidelines and recommendations for coding in C++ for the ATLAS experiment.

There are several reasons for maintaining and following a set of programming guidelines. First, by following some rules, one can avoid some common errors and pitfalls in C++ programming, and thus have more reliable code. But even more important: a computer program should not only tell the machine what to do, but it should also tell *other people* what you want the machine to do. (For much more elaboration on this idea, look up references on ``literate programming,'' such as [1].) This is obviously important any time when you have many people working on a given piece of software, and such considerations would naturally lead to code that is easy to read and understand. Think of writing ATLAS code as another form of publication, and take the same care as you would writing up an analysis for colleagues.

This document is derived from the original ATLAS C++ coding standard, ATL-SOFT-2002-001 [2], which was last revised in 2003. This itself derived from work done by the CERN ``Project support team'' and SPIDER project, as documented in CERN-UCO/1999/207 [3]. These previous guidelines have been significantly revised to take into account the evolution of the C++ language [4], current practices in ATLAS, and the experience gained over the past decade.

Some additional useful information on C++ programming may be found in [5], [6], and [7].

This note is not intended to be a fixed set of rigid rules. Rather, it should evolve as experience warrants.

2 Naming

This section contains guidelines on how to name objects in a program.

2.1 Naming of files

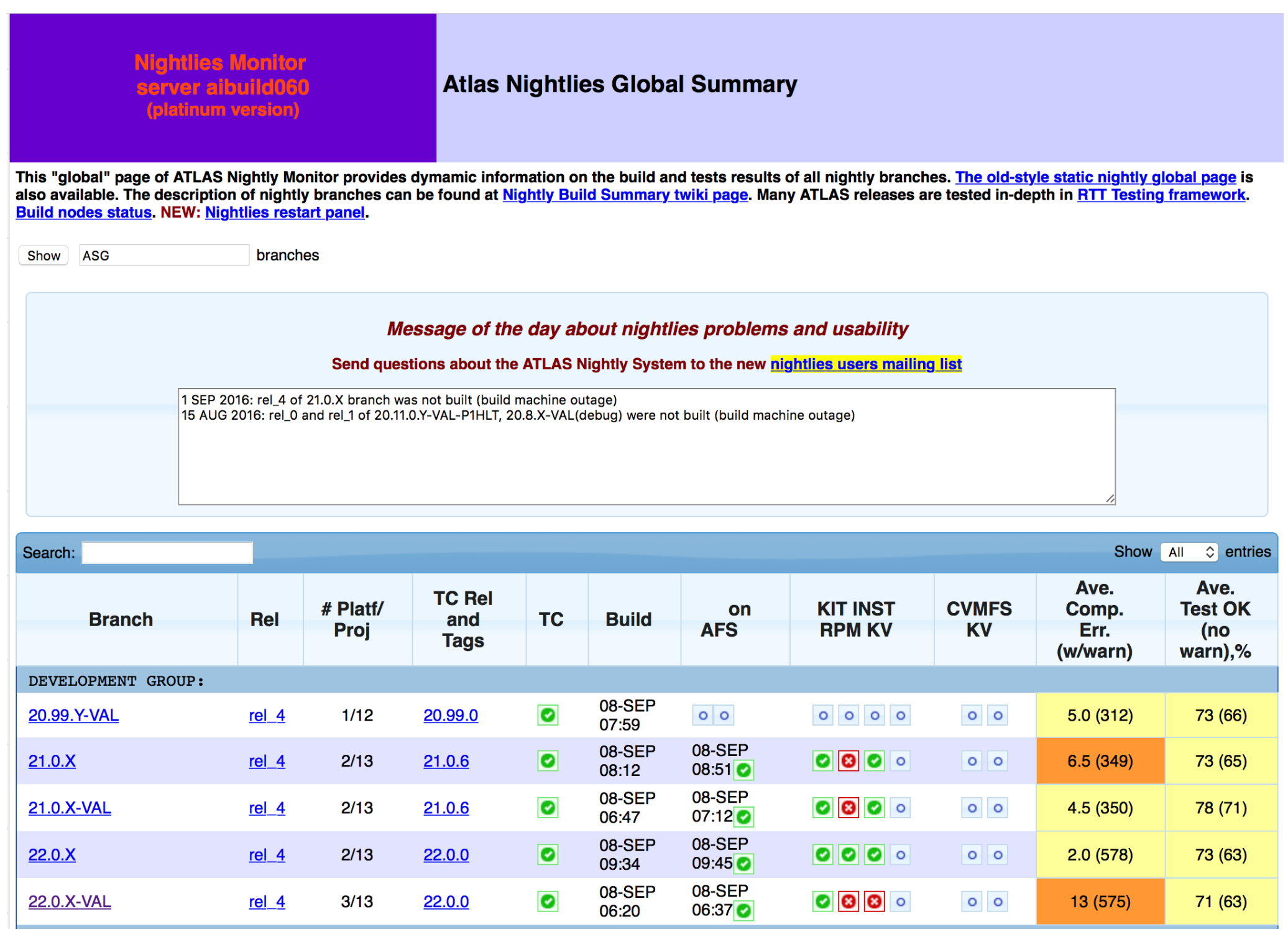
- **Each class should have one header file, ending with ``.h'', and one implementation file, ending with ``.cxx''.** [source-naming]

  Some exceptions: Small classes used as helpers for another class should generally not go in their own file, but should instead be placed with the larger class. Sometimes several very closely related classes may be grouped together in a single file; in that case, the files should be named after whichever is the ``primary'' class. A number of related small helper classes (not associated with a particular larger class) may be grouped together in a single file, which should be given a descriptive name. An example of the latter could be a set of classes used as exceptions for a package.

Information

The 'front line' in maintaining software quality is education and awareness. TWikis provide a structured navigable access to information and tools for the developers, but it is challenging to maintain awareness of the importance of software quality. We notice renewed interest in quality issues with the introduction of new tools or presentation of the issues at collaboration weeks. Thus novelty and continuous discussion are important to maintain motivation.

Nightly Build



Nightlies Monitor server aibuild060 (platinum version)

Atlas Nightlies Global Summary

This "global" page of ATLAS Nightly Monitor provides dymamic information on the build and tests results of all nightly branches. The old-style static nightly global page is also available. The description of nightly branches can be found at Nightly Build Summary twiki page. Many ATLAS releases are tested in-depth in RTT Testing framework. Build nodes status. NEW: Nightlies restart panel.

Show ASG branches

Message of the day about nightlies problems and usability

Send questions about the ATLAS Nightly System to the new nightlies users mailing list

1 SEP 2016: rel_4 of 21.0.X branch was not built (build machine outage)
15 AUG 2016: rel_0 and rel_1 of 20.11.0.Y-VAL-P1HLT, 20.8.X-VAL(debug) were not built (build machine outage)

Search: Show All entries

Branch	Rel	# Platf/ Proj	TC Rel and Tags	TC	Build	on AFS	KIT INST RPM KV	CVMFS KV	Ave. Comp. Err. (w/warn)	Ave. Test OK (no warn),%
DEVELOPMENT GROUP:										
20.99.Y-VAL	rel_4	1/12	20.99.0	✓	08-SEP 07:59	○ ○	○ ○ ○ ○	○ ○	5.0 (312)	73 (66)
21.0.X	rel_4	2/13	21.0.6	✓	08-SEP 08:12	08-SEP 08:51 ✓	✓ ✗ ✓ ○	○ ○	6.5 (349)	73 (65)
21.0.X-VAL	rel_4	2/13	21.0.6	✓	08-SEP 06:47	08-SEP 07:12 ✓	✓ ✗ ✓ ○	○ ○	4.5 (350)	78 (71)
22.0.X	rel_4	2/13	22.0.0	✓	08-SEP 09:34	08-SEP 09:45 ✓	✓ ✓ ✓ ○	○ ○	2.0 (578)	73 (63)
22.0.X-VAL	rel_4	3/13	22.0.0	✓	08-SEP 06:20	08-SEP 06:37 ✓	✓ ✗ ✗ ○	○ ○	13 (575)	71 (63)

Nightly Builds are performed across different compilers (e.g. Clang, gcc49), and developers are informed of errors by email.

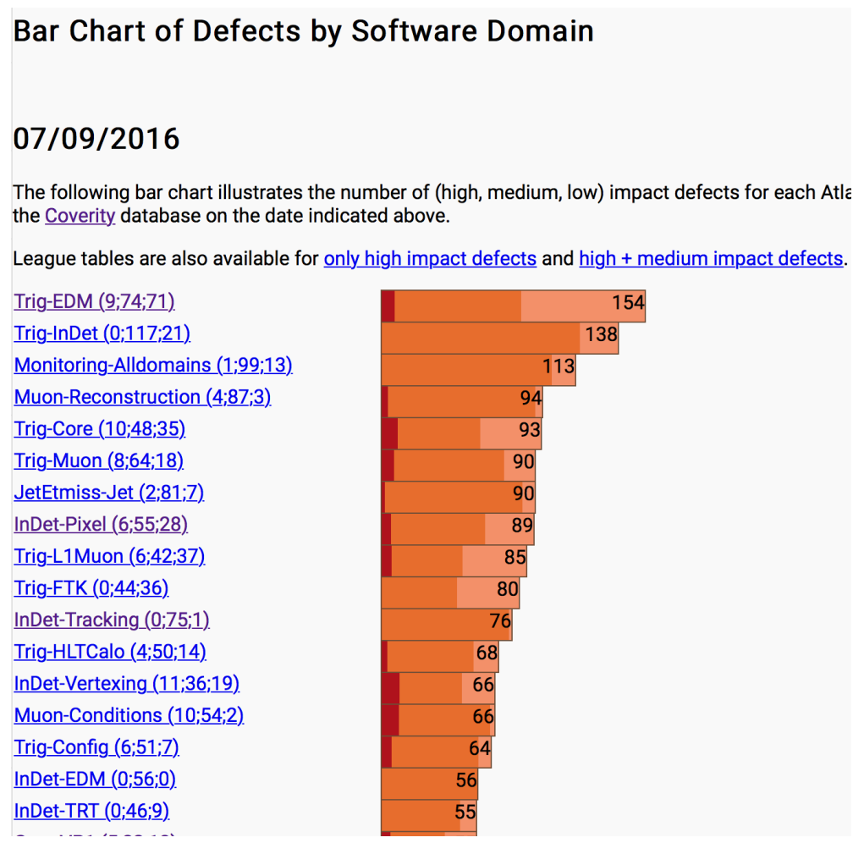
In addition, custom gcc plugins have been written to check for violation of internal standards (e.g. naming conventions, inheritance conventions)

The Undefined Behaviour Sanitizer (UBSan) is active in debug builds (one example: left shift of negative number).

Unit tests are performed as part of the build, including (more recently) tests in the GoogleMock framework which allow testing of complex objects.

As a final step, Run Time Tests provide physics parameters (e.g. pT or angular distributions) as a holistic test of the software for comparison against known references

Static Analysis: Coverity®, cppcheck, lizard

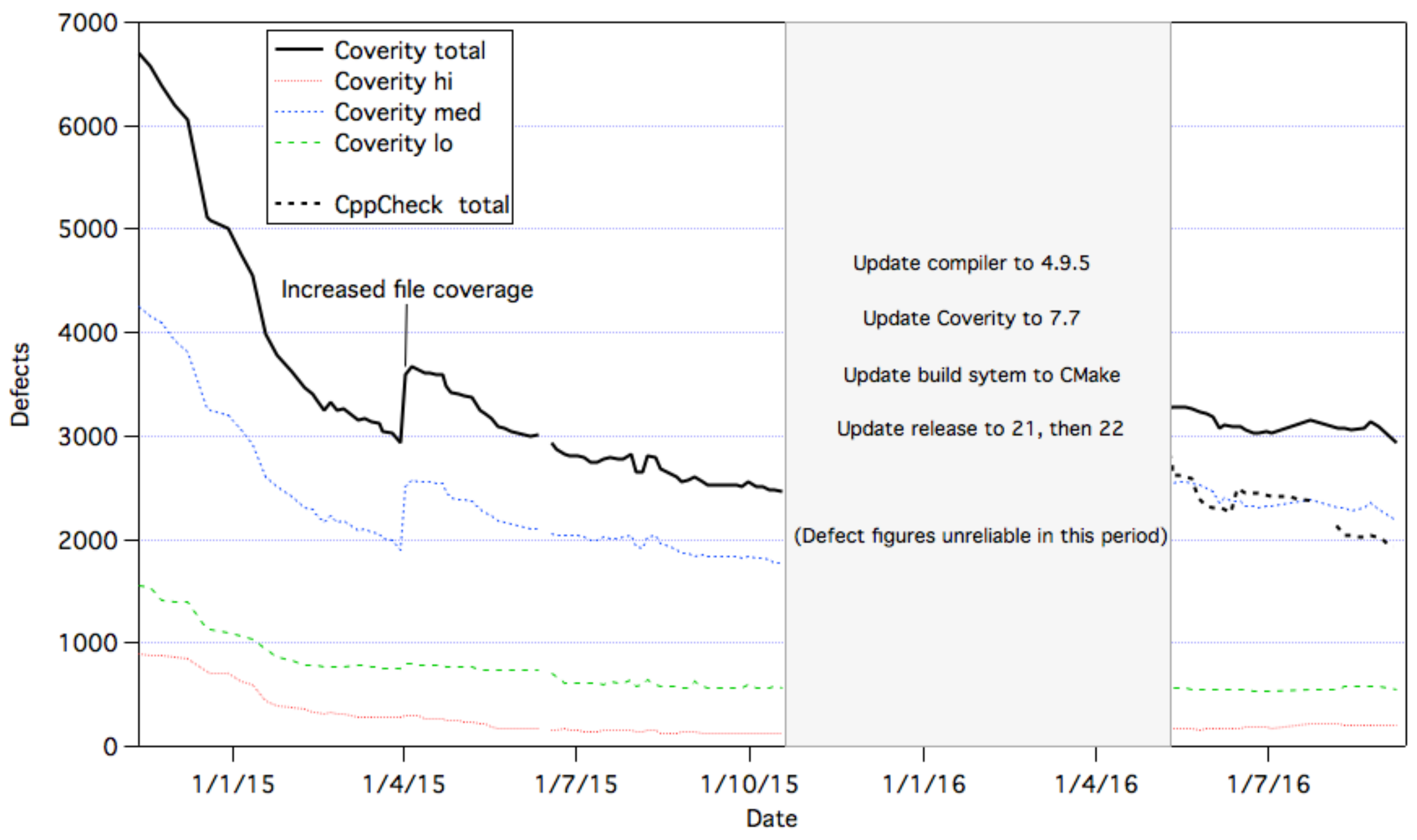


Bar Chart of Defects by Software Domain

07/09/2016

The following bar chart illustrates the number of (high, medium, low) impact defects for each Atla the Coverity database on the date indicated above.

League tables are also available for only high impact defects and high + medium impact defects.



Coverity and cppcheck are run twice-weekly. Coverity reports are sent once per week to developers. 'Lizard' provides cyclomatic-complexity and line count metrics which can be useful to spot rotten code. These tools all have results presented as a league table of defects per software team, intended to motivate the teams. Coverity is the most comprehensive, and has resulted in many defects being found; the progress over time is shown left. Current defect densities are 0.2 - 0.8 defects/1000-lines-of-code

The Future

ATLAS is migrating from SVN to Git, which allows a formal code review before acceptance into the repository. The tools shown above will likely form one input to the review process, but finally code quality depends on the developers motivation and the reviewers conscientiousness.