Experience with C++ Code Quality in ATLAS

Stewart Martin-Haugh, Emil Obreshkov, Shaun Roe, Rolf Seuster, Scott Snyder, Graeme Stewart

Tools in use
Experience
Progress, Pleasures and Pitfalls,

TOOLS

Compilation * Static Analysis * run-time behaviour

Atlas code

2

6

0

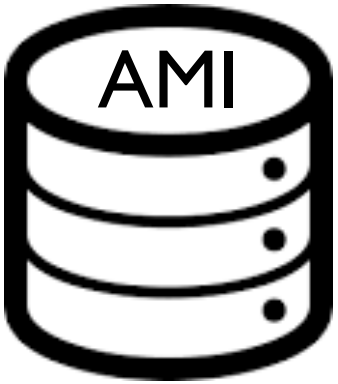
loc

(millions)

10

~420

developers



~140 'domains'

e.g. Tracking-Fitting

(gcc 4.9)

Compiler checks

Code should always, where possible, compile without warnings.

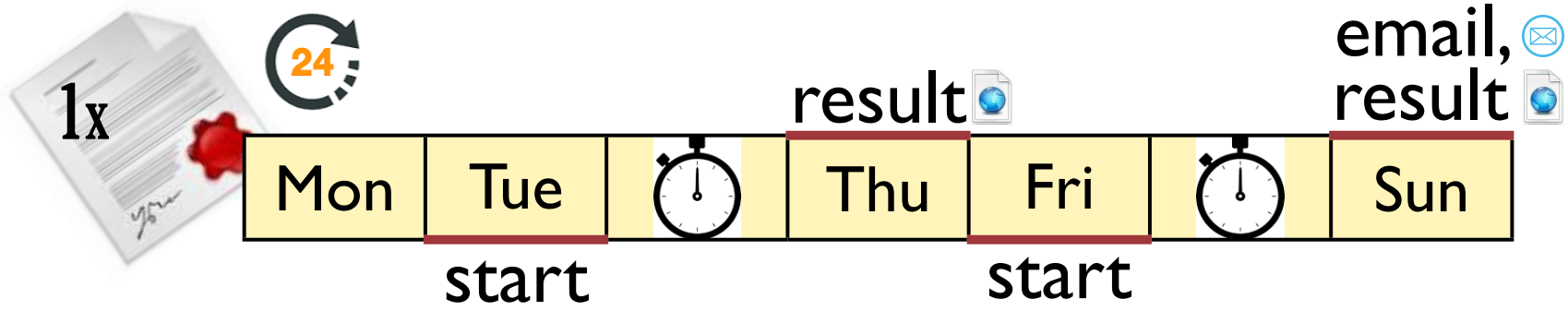
It is beneficial to expose the code to different compilers; e.g. **Clang builds** give additional info:

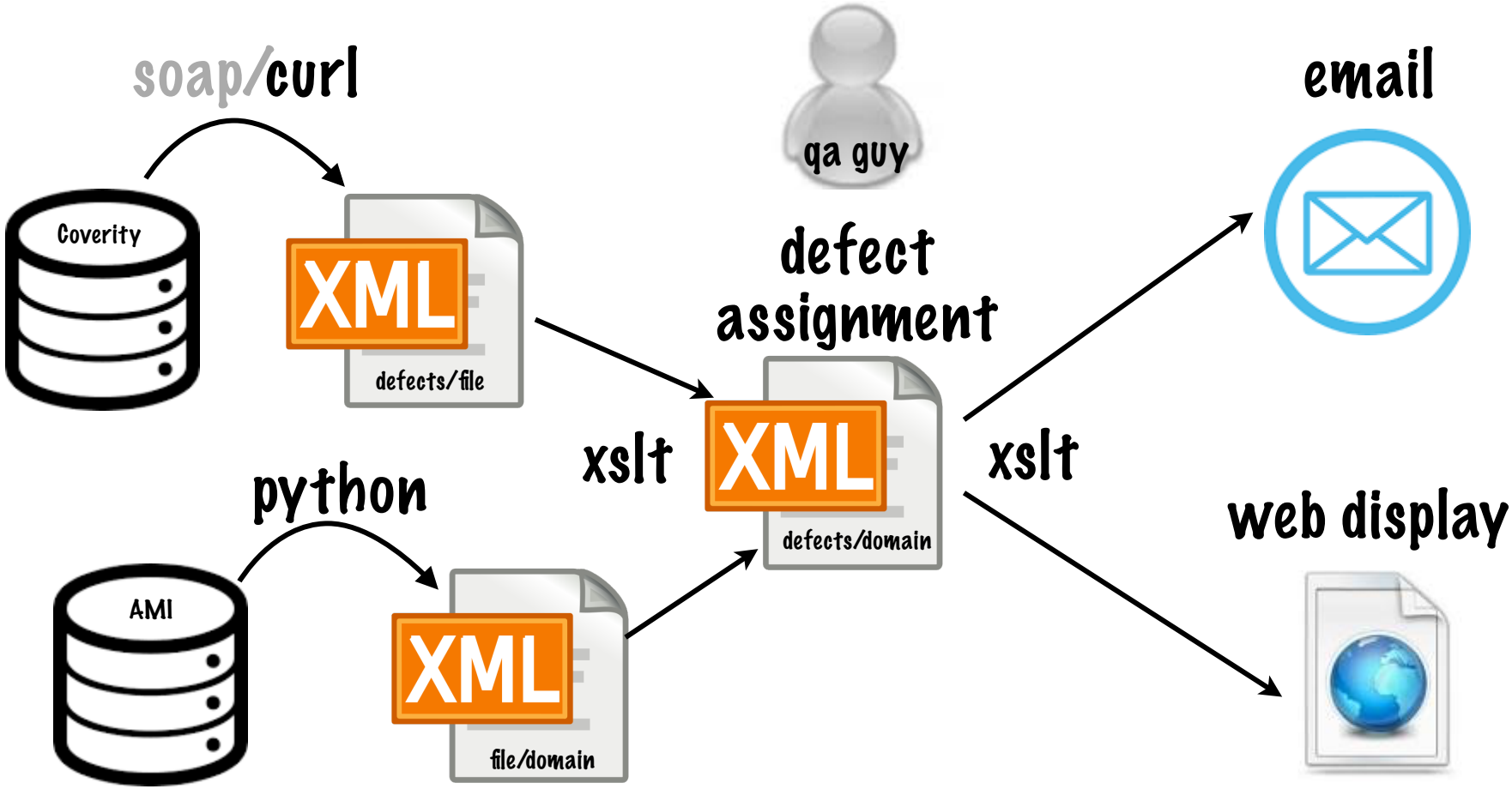
- Mismatched struct/class; Unused private class members; Mismatched header guard.
- `std::abs(eta < 2.5)`
- `if ((bitfield && 0x0011) != 0) …`
- `!val==10`

gcc plug-ins

- Check for inheritance structure already in place
- Considering: naming convention checks

Coverity®


1x
24
result
email,
result
Mon
Tue
Thu
Fri
Sun
start
start


soap/curl
qa guy
email
Coverity
XML
defects/file
defect
assignment
xslt
XML
defects/domain
xslt
python
web display
AMI
XML
file/domain

-
-

12598 09/07/2014 (High) Resource leak :/ ForwardDetectors/ALFA/ALFA_CLinkAlg/src/ALFA_CLinkAlg.cxx in function "execute"

-

devs

coverity | AtlasEvent | Configuration | Help | Shaun Roe | Enter CID(s)

DASHBOARDS
Quality Advisor
Security Advisor
Test Advisor

ISSUES: BY SNAPSHOT
High Impact Outstanding
My Outstanding
Outstanding Defects
Outstanding Security ...
Outstanding Test Rule...
Outstanding Untriaged
Recently Viewed

ISSUES: PROJECT SCOPE
All In Project
Recently Viewed

FILES
In Latest Snapshot
Uncovered By Tests

FUNCTIONS
High CCM (>15)
In Latest Snapshot
Uncovered By Tests

COMPONENTS
All In Project
High Issue Density (>1)
With Outstanding Issues
With Untriaged Issues

CHECKERS
All In Project

OWNERS
All In Project

SNAPSHOTS

12598	Resource leak	High	New	1	09/07/2014	Unassigned

All 1 issue selected | Page 1 of 1

/ForwardDetectors/ALFA/ALFA_CLinkAlg/src/ALFA_CLinkAlg.cxx

```
51 }
52
53 StatusCode ALFA_CLinkAlg::execute()
54 {
     1. Condition !!this->msgLvl(MSG::DEBUG), taking false branch
     2. Condition !!this->msgLvl(MSG::DEBUG), taking false branch
55       ATH_MSG_DEBUG ("ALFA_CLinkAlg::execute()");
56
     3. alloc_fn: Storage is returned from allocation function operator new.
     4. var_assign: Assigning: pDataEvent = storage returned from new ALFA_CLinkEvent.
57       ALFA_CLinkEvent* pDataEvent=new ALFA_CLinkEvent();
     5. noescape: Resource pDataEvent is not freed or pointed-to in LoadAllEventData. [show details]
     6. Condition !sc__.isSuccess(), taking true branch
     CID 12598 (#1 of 1): Resource leak (RESOURCE_LEAK)
     7. leaked_storage: Variable pDataEvent going out of scope leaks the storage it points to.
58       CHECK(LoadAllEventData(pDataEvent));
59       if (m_nDataType==1) pDataEvent->SetDCSFolderIDs(&m_CurrentDCSId);
60       CHECK(evtStore()->record(pDataEvent, "ALFA_CLinkEvent"));
61
62       CHECK(GenerateXAOD());
63
64       return StatusCode::SUCCESS;
65 }
66
67 StatusCode ALFA_CLinkAlg::finalize()
68 {
```

12598 Resource leak

The system resource will not be reclaimed and reused, reducing the future availability of the resource.

In ALFA_CLinkAlg::execute(): Leak of memory or pointers to system resources (CWE-404)

Triage

Classification: Unclassified
Severity: Unspecified
Action: Undecided
Ext. Reference: Type attribute text
Owner: Unassigned

Enter comments (See the History section below for previous comments)

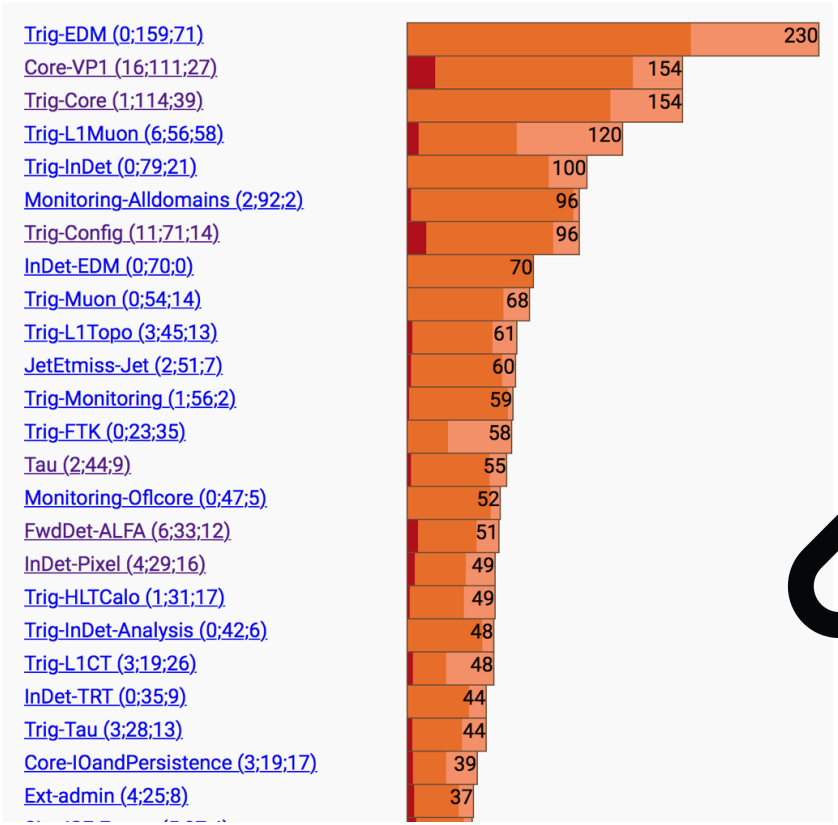
Apply + Next | Apply

Projects and Streams
Detection History
Triage History
Occurrences

1: AtlasEvent

Events contributing to defect:

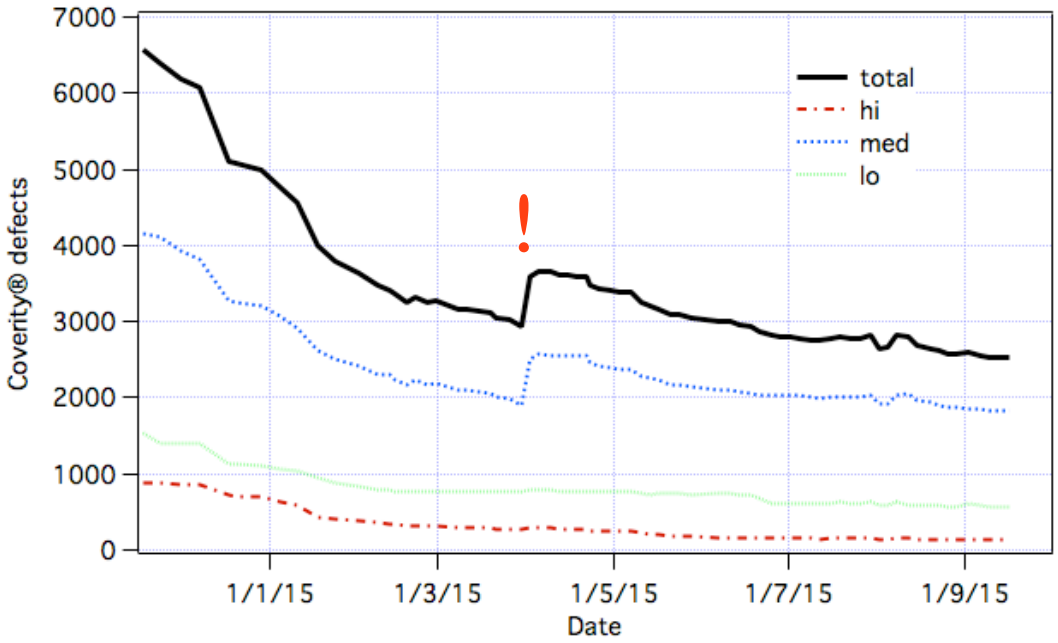
3 **alloc_fn**	ALFA_CLinkAlg.cxx:57
4 **var_assign**	ALFA_CLinkAlg.cxx:57
5 **noescape**	ALFA_CLinkAlg.cxx:58

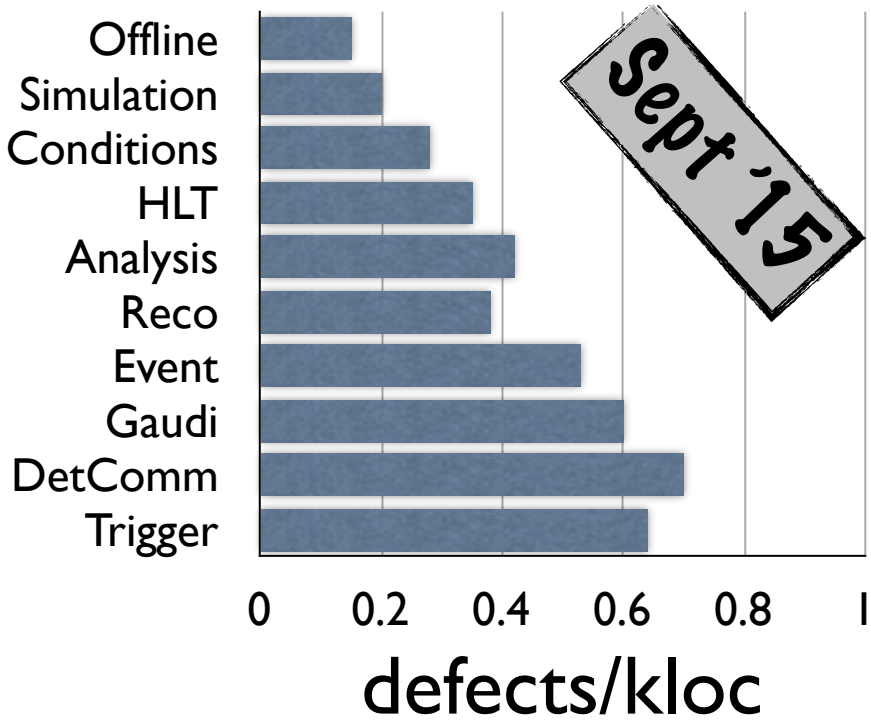


A command line tool, '`issues`' will query the web page and list issues for a specified file

Pitfalls: Coverity's classification may not correspond to your own; e.g. Uninitialised members are classified 'high', as are obvious resource leaks; however a faulty assignment operator which can also leak resources will be 'low'. Parse errors can occur and are classified 'low' but may mask many other defects.

Coverity® progress





Industry report 2012

"Coverity's analysis found an average defect density of .69 for open source software projects that leverage the Coverity Scan service, and an average defect density of .68 for proprietary code developed by Coverity enterprise customers. Both have better quality as compared to the **accepted industry standard defect density for good quality software of 1.0** [defects/kloc]."

Coverity® pleasures and pitfalls

"spending hours with a memory profiler could save you 5 minutes looking at the coverity report" ☺

False positives (*very few*): ☹

e.g. "restoring ostream format"

"parse errors" (particularly complicated templates)

Near misses:

```
792     //set the foreign key
6. return_constant: Function call variableType() returns 4.
CID 11595 (#2-1 of 2): Out-of-bounds read (OVERRUN)
7. overrun-local: Overrunning array of 4 bytes at byte offset 4 by dereferencing pointer &"_FK"[variableType()].
793     pixel_columns.createForeignKey(variableType<T>()+"_FK","FK",m_pixeltable,"FK");
794     // create indices
```

Misses: `if (m_Analogue[1][strip]<0.0000001 || m_Analogue[1][strip]>-0.0000001){ //`

Other Static Analysers

Available, not yet integrated into reporting system:

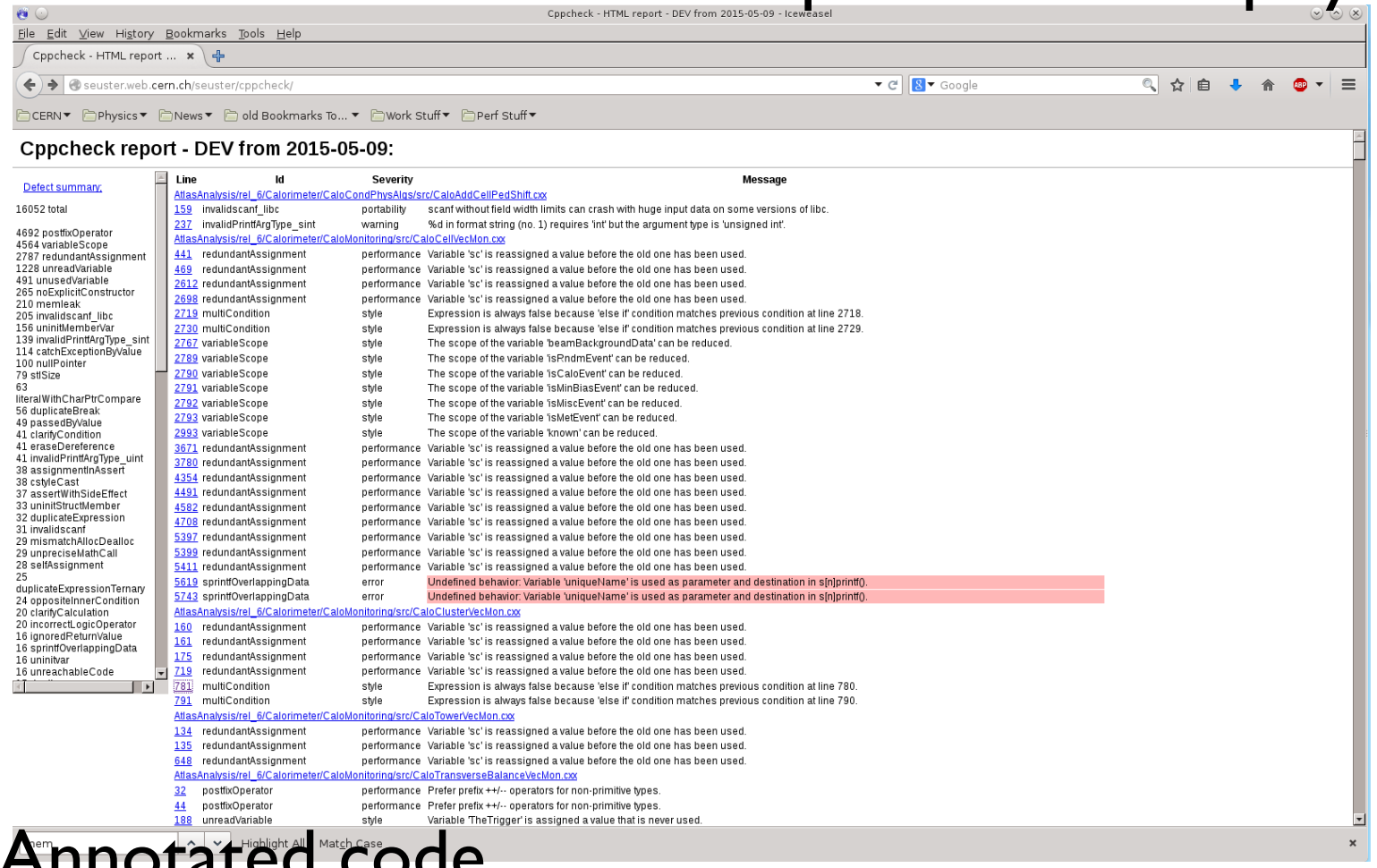
- **cppcheck**
  Open source, easily configurable, quick (~1 hr)
  somewhat noisy, more false positives, but *does find additional defects*. Output to XML/static web page. Options for performance (e.g. "use pre-increment"), style.
- **Include-what-you-use**
  Easy to set up; scope is limited to tidying up #includes; web interface available for reports (R. Seuster).

Investigated:

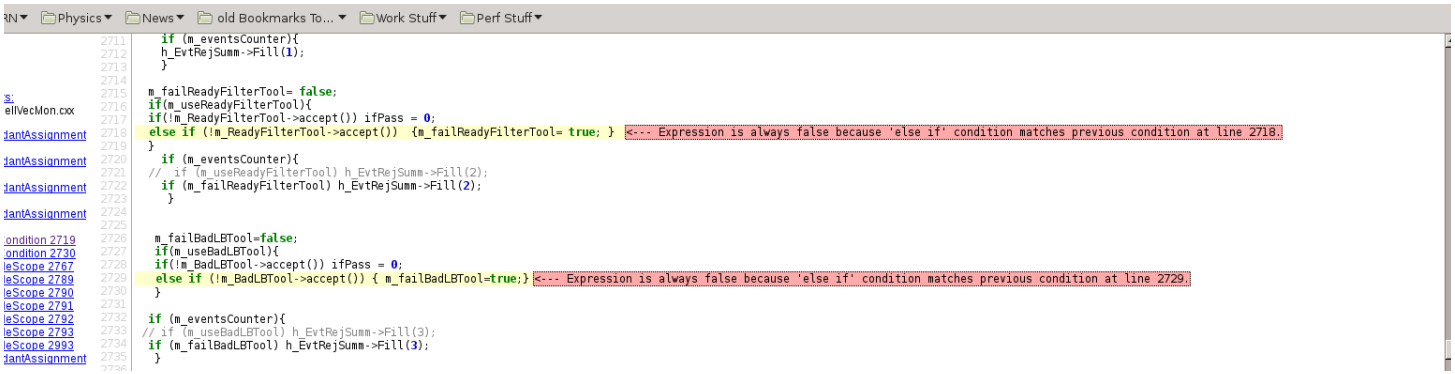
PRQA: Nice system, flexible, industry standards, can enforce naming conventions; $$, difficult to integrate in our build system

cppcheck

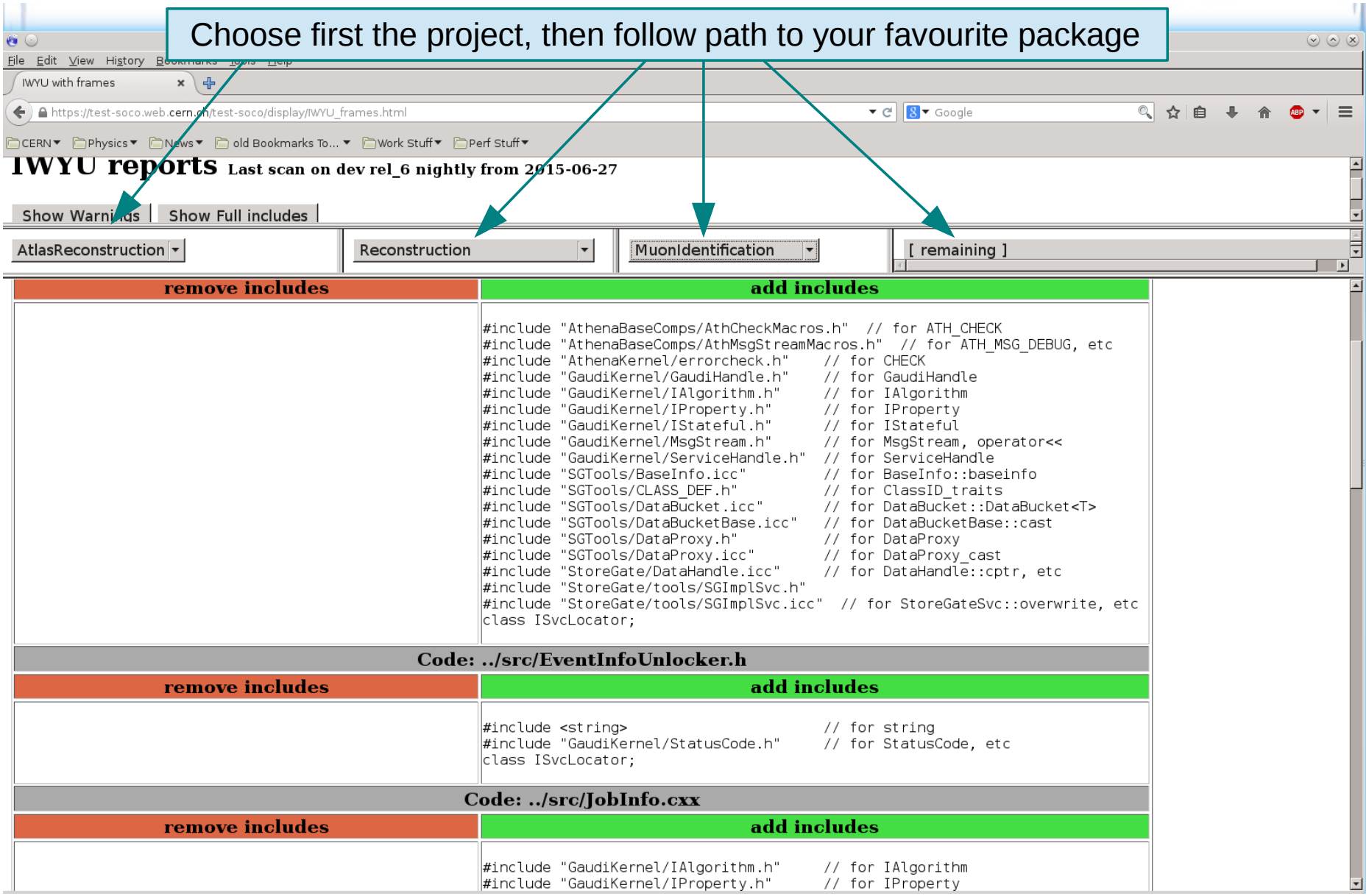
Top level web display



Annotated code



Include-what-you-use



Run-time Sanitizers

Implemented in nightly tests of DBG builds:

- **UBSan** (undefined behaviour)

Clear warnings; easy to implement :

```
HepMcParticleLink.h:72:51: runtime error: left shift of 200001 by 16 places cannot be
represented in type 'int'
```

Investigated:

- **ASan** (address)

Similar to Valgrind but much faster and catches more; output maybe a bit intimidating...

```
==7552==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x61400000ffd4 at
pc 0x400825 bp 0x7fff248f7d80 sp 0x7fff248f7d78...
```

ABI incompatible.

- **TSan** (thread)

Currently not useful 'out of the box' for Atlas

"If all you have is a hammer, everything looks like a nail"

- Psychology of Science (Maslow, 1966)

Coding to solve the line-by-line problems revealed by these tools *might* produce better code, but it won't forcibly produce **good** code.

"Code smell" detectors (tentative results so far):

- **TCToolkit**

~1 hr to scan release, produces >50Mb html.

- code duplication detector (looks useful)
- token tag cloud ("how noisy" the code is)
- class co-occurence matrix (interdependencies)

Summary

- All of these tools are useful as developer aids; Coverity in particular reveals many programming errors and maintainability issues.
- Public league tables help motivate groups.
- None of these will replace peer review or make up for a lack of education (another important topic) or experience.

ATLAS will continue to use these tools, integrating the reports from different analyses, and keep an eye on new tools as they become available.