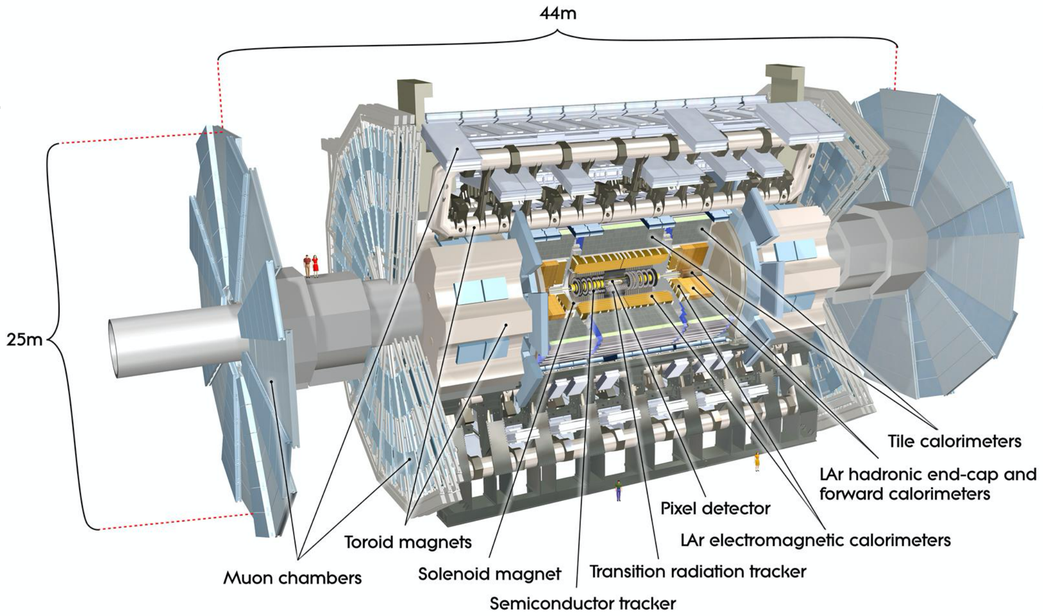
EDWARD MOYSE

SOFTWARE SUSTAINABILITY : ATLAS

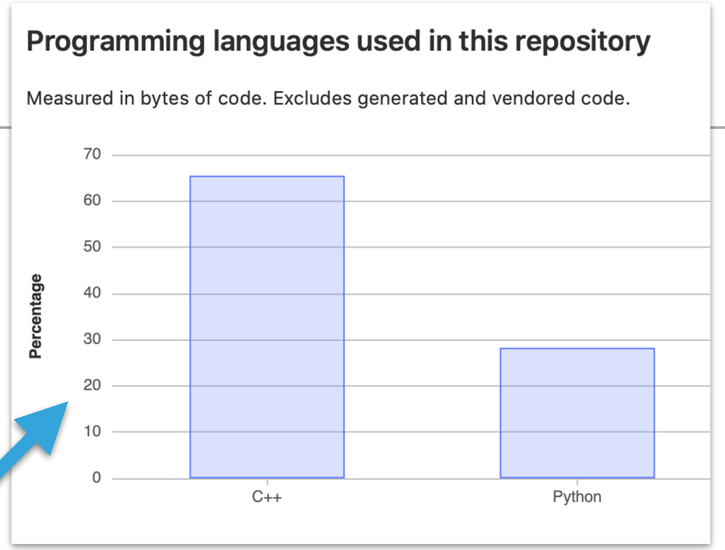
INTRODUCTION TO ATLAS

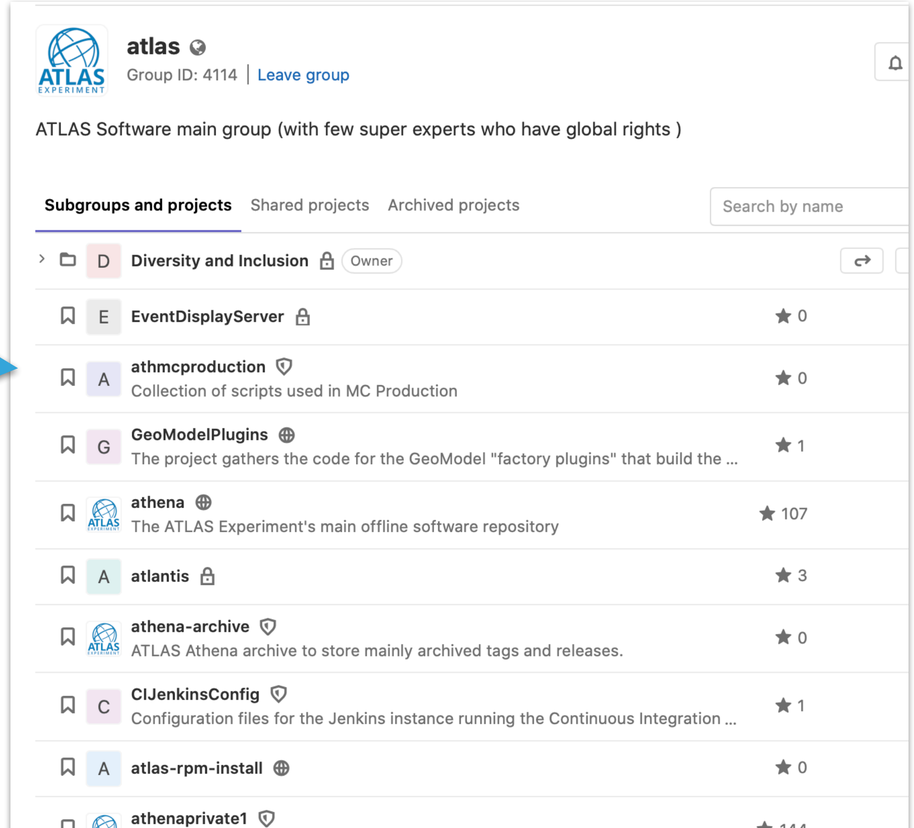
- Large international collaboration
  - ~ 2’900 Scientific Authors
  - ~ 1’200 Students
- Not unique to us (!), but poses some key challenges:
  - Complexity of managing large distributed teams of coders
  - We often have to convince collaborators to volunteer to help, because we have limited sticks and carrots
    - Much easier to get help with flashy new project, than maintaining an old piece of code



SOFTWARE IN ATLAS

- Athena is ATLAS's event processing framework
  - >1 million lines of python and ~4 million lines of C++
  - Largest & most active repository in CERN GitLab (by far)
- We also have many smaller repositories
  - I don't know the total, and am not sure it's possible to find out
    - (Will come back to this!)





SOFTWARE SUSTAINABILITY

- Will now go through a few topics I think are relevant to sustainability, covering what I think works and what doesn't
- Includes
  - Standard tools
  - Open-sourcing our code
  - Social coding
  - Validation and testing, static analysis
  - Documentation and training, minimising expertise loss
- (Large overlap in topics in some cases, so there will be some repetition)

USE STANDARD TOOLS (& LIBRARIES)

- **Initially, ATLAS used many home-grown tools (or exclusive to HEP)**
  - Examples: CMT build system, TagCollector to manage releases, CLHEP etc etc
  - Some advantage of handwritten solutions:
    - We get a tool that (hopefully!) perfectly fits our use case
  - Some problems:
    - Ongoing maintenance load (which take resources from elsewhere)
    - Dedicated training required
- **Since then, made many efforts to move to industry standards:**
  - e.g. Git + CMake (2016/2017), Eigen (2014) etc
  - Some advantages of industry standards:
    - Generally better functionality (have real experts writing the code, rather than physicists with some fraction of their time)
    - Many fantastic tutorials online beginners can use (and means they learn transferrable skills)
  - Some problems
    - Some older developers struggled to move e.g. to git, and in some cases, migration was a lot of work

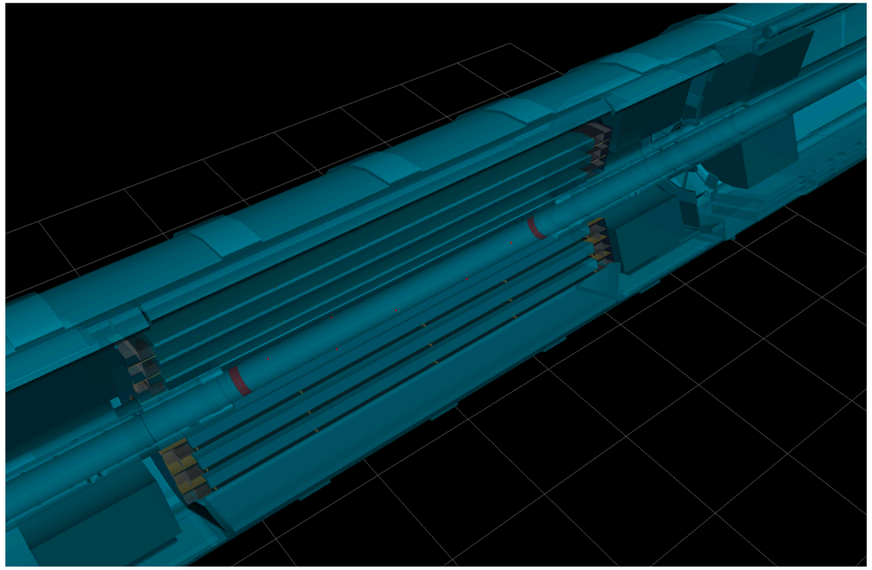
USE STANDARD TOOLS (& LIBRARIES)

- **Initially, ATLAS used many home-grown tools (or exclusive to HEP)**
  - Examples: CMT build system, TagCollector to manage releases, CLHEP etc etc
  - Some advantage of handwritten solutions:
    - We get a tool that (hopefully!) perfectly fits our use case
  - Some problems:
    - Ongoing maintenance load (which take resources from elsewhere)
    - Dedicated training
- **Since then, made many eff**
  - e.g. Git + CMake (2016/2017), Eigen (2014) etc
  - Some advantages of industry standards:
    - Generally better functionality (have real experts writing the code, rather than physicists with some fraction of their time)
    - Many fantastic tutorials online beginners can use
    - Learn transferable skills
  - Some problems
    - … in some cases, migration was a lot of work

REDUCING THE AMOUNT OF UNNECESSARY CODE WE NEED TO MAINTAIN IS A HUGE STEP TOWARDS SOFTWARE SUSTAINABILITY

OPEN SOURCE

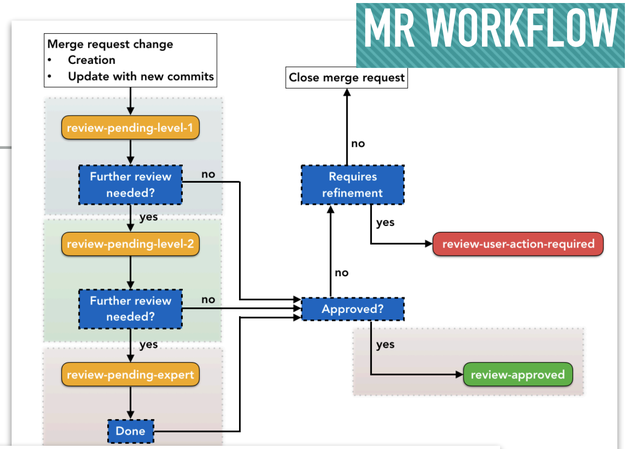
- Athena was open-sourced at the end of run-2, under an Apache 2.0 licence
  - ATLAS is committed to opening all of its software (some exceptions e.g. analysis)
  - Our experience has been very positive - much easier to share with interested outsiders
  - **One issue**: CERN lightweight account is not enough to contribute to GitLab.
- Examples of other open-source projects originating from ATLAS
  - Rucio - data management
  - GeoModel - geometry description language and tools
  - ACTS - experiment agnostic tracking software
  - Phoenix - experiment agnostic event display
- **Open sourcing software allows us to share effort with other experiments, and facilitates help from e.g. industry**
  - (It is also the right thing to do, IMO, with publicly funded projects)

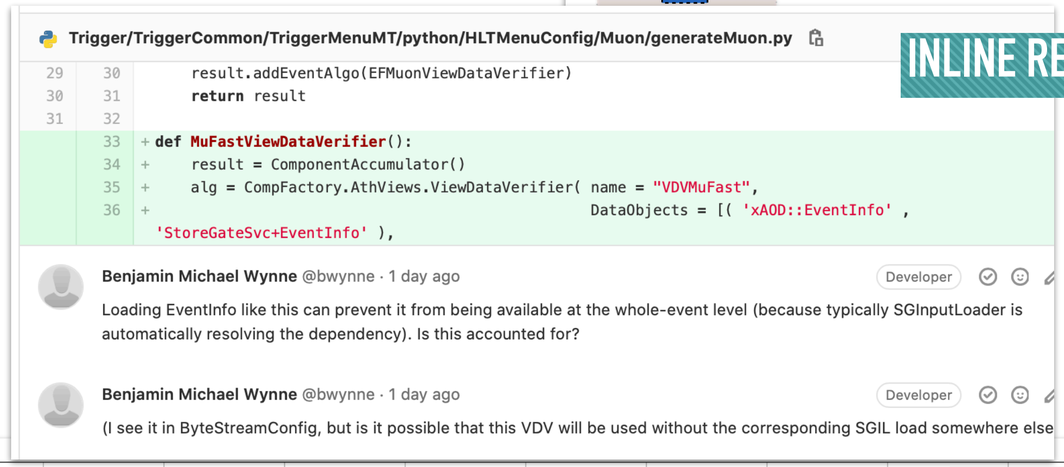


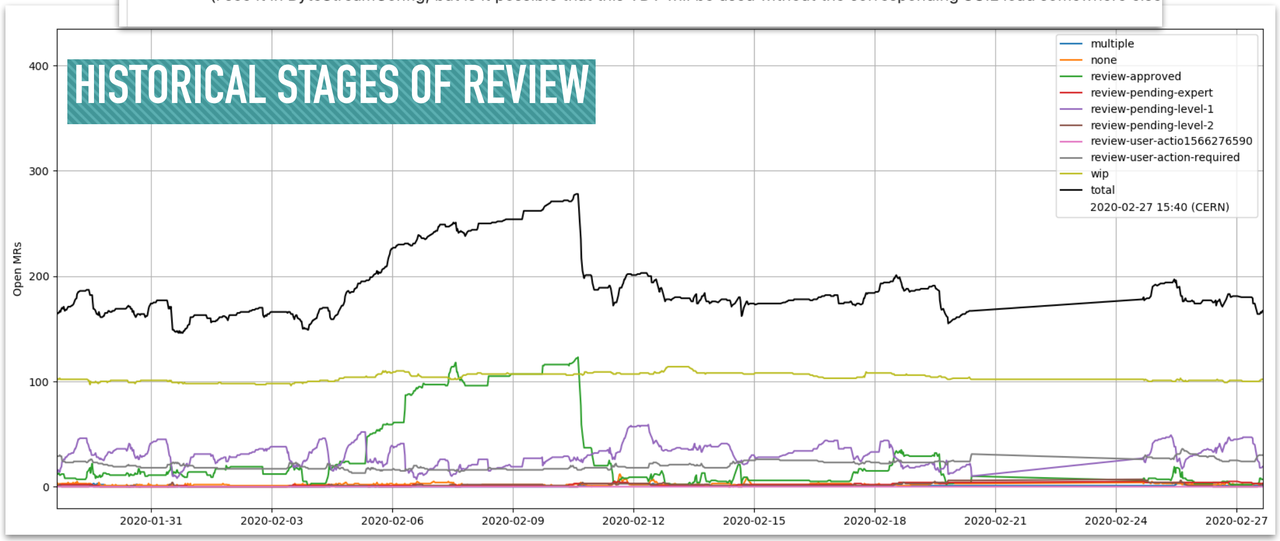
GeoModelClash

SOCIAL CODING / MERGE REVIEWS

- Within ATLAS, we make extensive use of gitlab (and github) & make extensive use of social coding feature, in particular: Merge(Pull) Reviews
- For Athena in particular this is very organised:
  - Two levels of shifters, working morning and afternoon
  - Review approximately 40 MRs per day (14k in 19 months)
  - Check for :
    - CI failures (see next slide)
    - Known gotcha (e.g. memory leaks)
    - Good code documentation
    - Following ATLAS coding conventions (writing "good" code, but also trying for some level of conformity)
  - Comments added inline to code - can trigger many rounds of updates
- IMO this is one of the most important improvements towards sustainability we've made
  - ***Continuous code review***
  - (Though balance between moving to latest and greatest feature, and rapidly fixing important bugs)

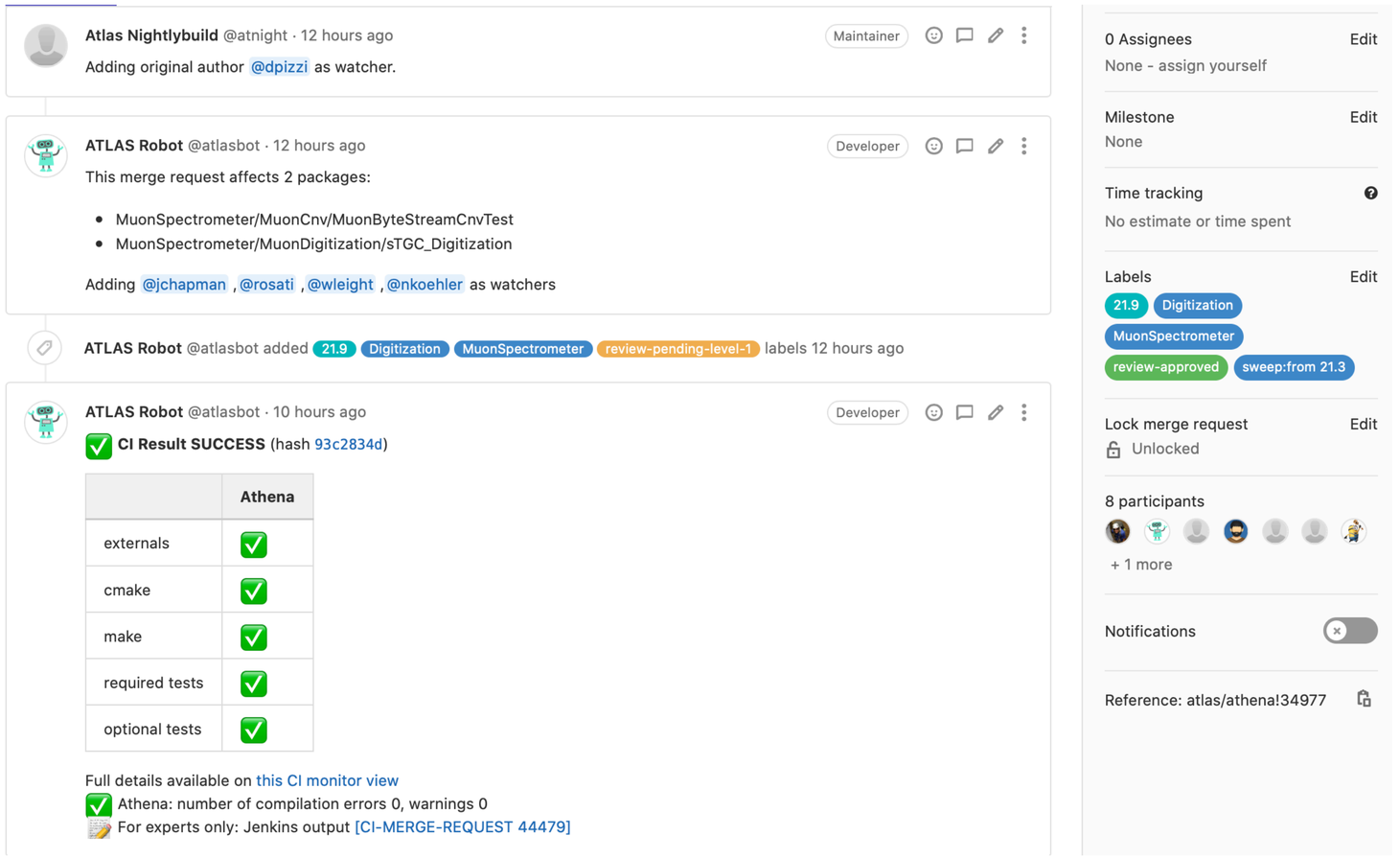






CONTINUOUS INTEGRATION

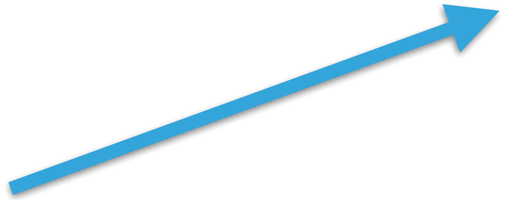
- On every MR, we run continuous integration
  - Label MRs with software domain, so correct experts are notified
  - Compile code
  - Run unit tests (ctests) for affected packages (or all, if developer sets relevant gitlab label)
  - Runs some simple jobs to test Athena
  - Optionally: check to see if physics objects are changed
- Currently runs on Jenkins, but investigating moving to Gitlab CI.
- **Vitally important** to ensuring sustainability of our code
  - It is incredibly rare to lose a nightly because of a coding mistake



NIGHTLIES

entries

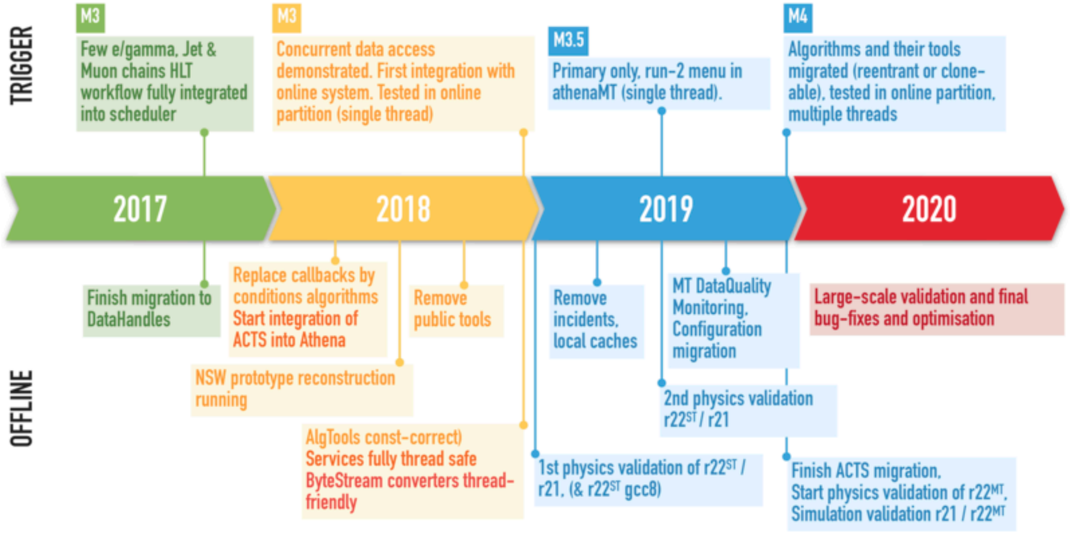
Release	Job time stamp	git clone	Extern. build	CMake config	Build time	Comp. errors (w/warn)	Test time	CTest errors (w/warn)	ART LOCAL	ART GRID	CVMFS (on server)	CVMFS (on client)
2020-03-04T2134	2020/03/05 04:37	✔	✔	✔	2020/03/05 04:37	0 (0)	2020/03/05 05:52	2 (2)	145,28	43,162,84,35	2020/03/05 06:30 ✔	2020/03/05 06:41
2020-03-03T2133	2020/03/04 04:34	✔	✔	✔	2020/03/04 04:34	0 (0)	2020/03/04 05:46	3 (3)	147,26	2,196,84,41	2020/03/04 06:21 ✔	2020/03/04 06:31
2020-03-02T2133	2020/03/03 04:29	✔	✔	✔	2020/03/03 04:29	0 (0)	2020/03/03 05:44	1 (1)	151,28	0,197,86,34	2020/03/03 06:24 ✔	2020/03/03 06:32
2020-03-01T2140	2020/03/02 04:32	✔	✔	✔	2020/03/02 04:32	0 (0)	2020/03/02 05:41	0 (0)	151,28	0,197,84,39	2020/03/02 06:13 ✔	2020/03/02 06:21
2020-02-29T2133	2020/03/01 04:32	✔	✔	✔	2020/03/01 04:32	0 (0)	2020/03/01 05:46	0 (0)	151,28	0,195,83,41	2020/03/01 06:20 ✔	2020/03/01 06:31
2020-02-28T2133	2020/02/29 04:31	✔	✔	✔	2020/02/29 04:31	0 (0)	2020/02/29 05:42	0 (0)	151,28	0,179,100,38	2020/02/29 06:28 ✔	2020/02/29 06:31
2020-02-27T2133	2020/02/28 04:35	✔	✔	✔	2020/02/28 04:35	0 (0)	2020/02/28 05:44	2 (2)	145,34	0,183,94,40	2020/02/28 06:18 ✔	2020/02/28 06:31
2020-02-26T2201	2020/02/27 05:06	✔	✔	✔	2020/02/27 05:06	0 (0)	2020/02/27 06:12	55 (55)	150,29	0,173,69,73	2020/02/27 06:49 ✔	2020/02/27 07:01
2020-02-25T2133	2020/02/26 04:43	✔	✔	✔	2020/02/26 04:43	0 (0)	2020/02/26 05:49	2 (2)	139,40	0,191,92,32	2020/02/26 07:05 ✔	2020/02/26 07:11
2020-02-24T2133	2020/02/25 04:37	✔	✔	✔	2020/02/25 04:37	0 (2)	2020/02/25 05:42	2 (2)	144,35	0,188,93,34	2020/02/25 06:15 ✔	2020/02/25 06:21
2020-02-23T2132	2020/02/24 04:39	✔	✔	✔	2020/02/24 04:39	0 (2)	2020/02/24 05:45	2 (2)	129,42	N/A	2020/02/24 06:21 ✔	2020/02/24 06:32

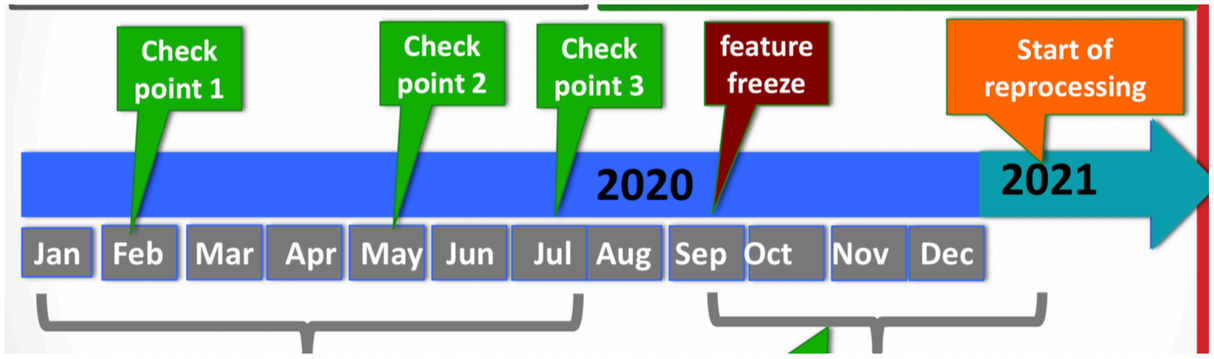


- We currently build ~20 branches per night
- On these we:
  - run unit tests (as with CI)
  - local longer tests, and
  - grid-based large statistics test (from which plots can be made and compared with references)
- (Non-unit tests are controlled by ART, unit tests run by ctest)
- Longer tests check for regressions, subtle bugs missed by CI
- Aside: **Compilers & heterogeneous platforms**
  - Our workhorse right now is gcc8 but we also compile nightly with clang8 (and developers run local builds with more exotic choices
  - More compilers & platforms = more chances to find bugs in older code

REGULAR BUILDS AND VALIDATION CAMPAIGNS

- So, as demonstrated we build our software every night, and run a battery of tests on it
- We also run larger validation campaigns
  - Typically ~1 million events
  - Primarily to measure physics performance
  - Also to find (very) rare bugs, and measure technical performance





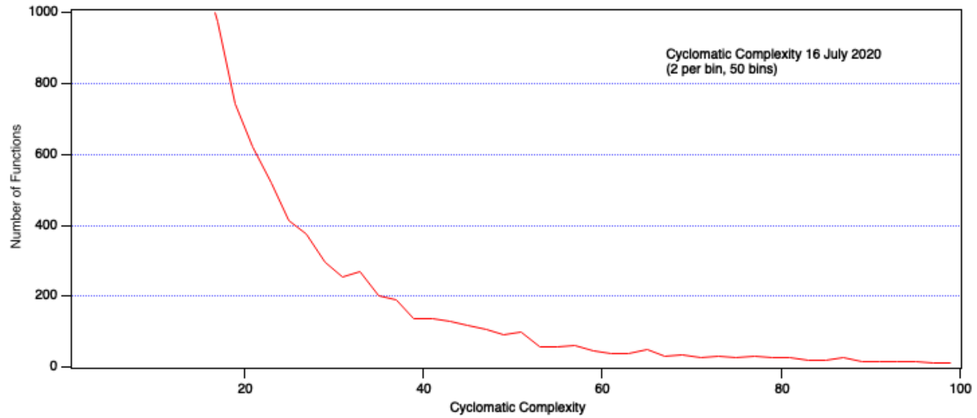
STATIC ANALYSIS TOOLS

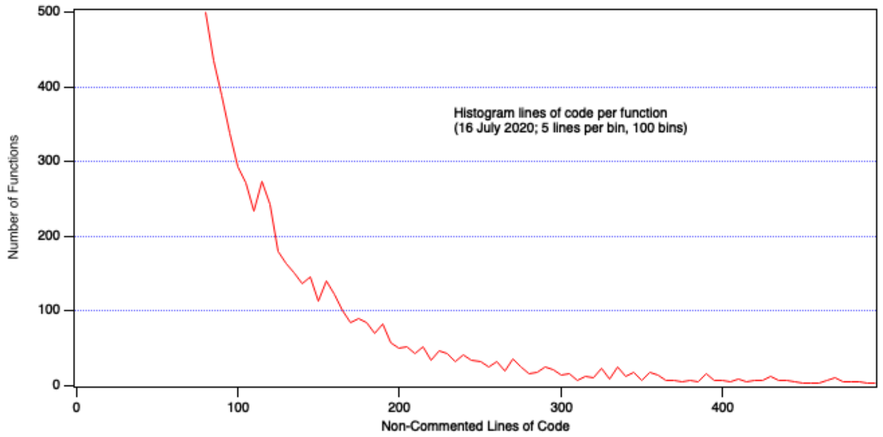
- For large codebase in particular, running tools to look for problems is very important
  - Not feasible to do complete code review of our software
- Cppcheck
  - Very static analysis useful tool
    - >100 MRs to Athena handling cppcheck warnings
    - Some false positives
- Coverity
  - Used to use this. Struggled to get it working recently.
  - Very slow (scans entire repository), complex licensing, requires dedicated server … but probably better than cppcheck
- Lizard
  - Cyclomatic Complexity Analyzer
    - Potentially gives some interesting clues to ‘hotspots’

InDetVKalVxInJetTool+TrkVKalVrtFitter: Fix cppcheck warnings.

Overview 3 Commits 2 Pipelines 1 Changes 10

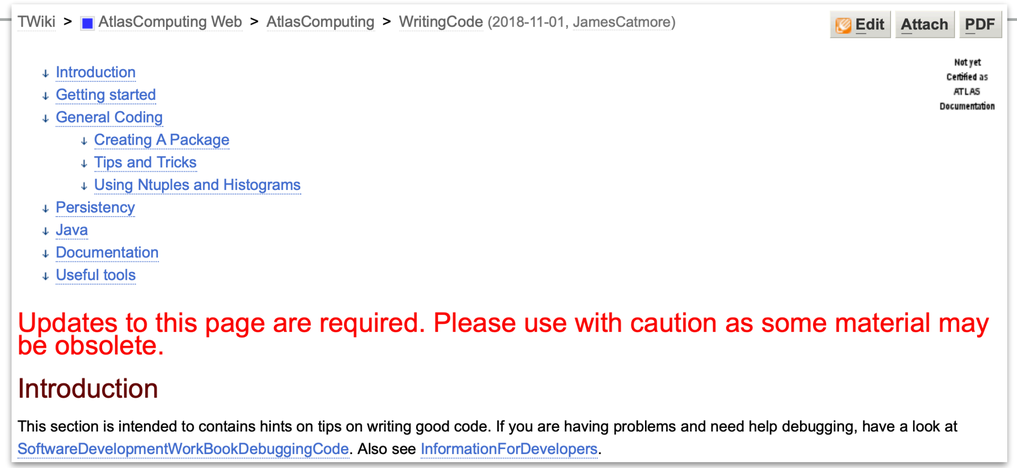
- Prefer preincrement to postincrement for user iterators.
- Pass large structures by const reference rather than by value.
- The MAT << 0,0,... notation confuses cppcheck. But setZero is anyway clearer, so just use that.
- Fix use of uninitialized variable.

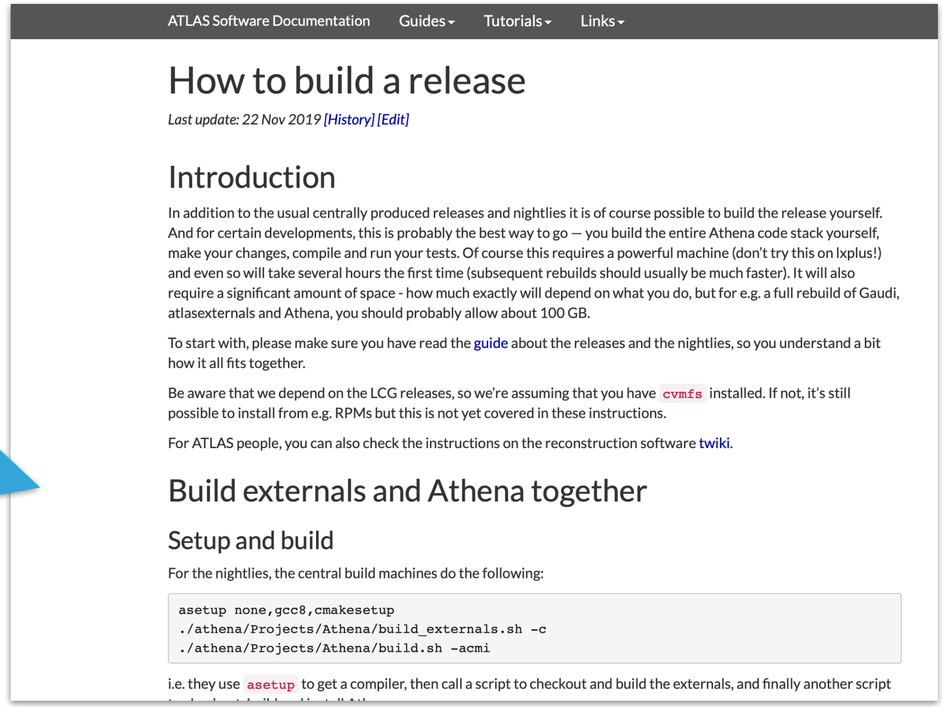




DOCUMENTATION

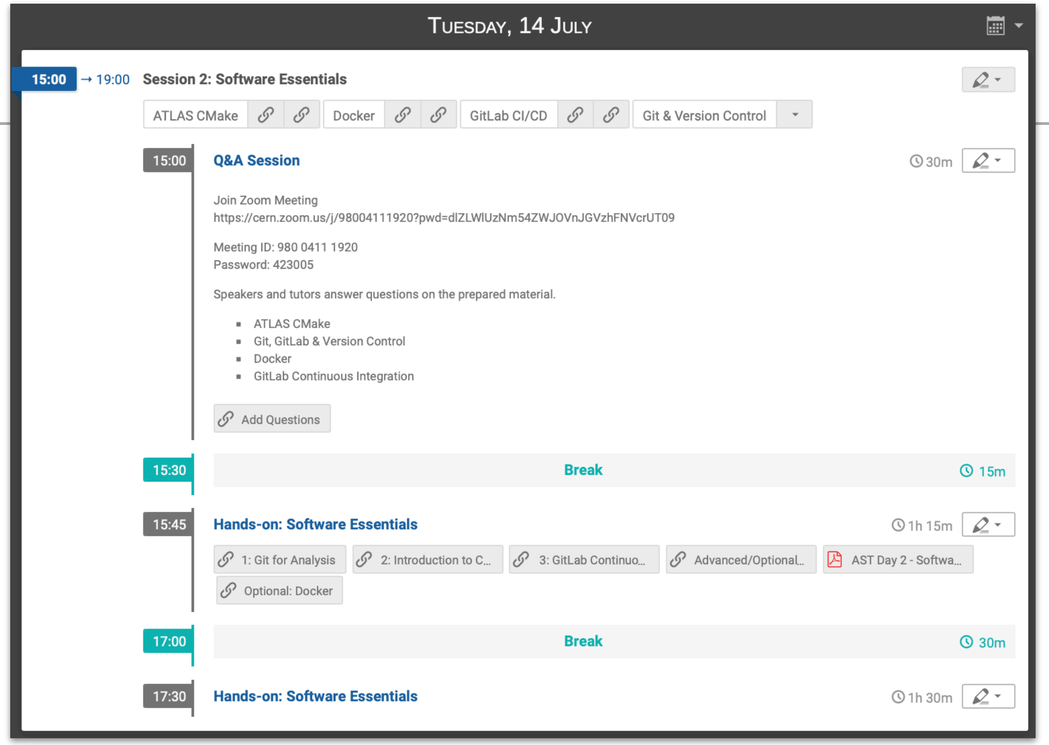
- ATLAS has historically documented software using Twiki
  - Search is awful, it decays fast, etc etc
  - Lots is restricted to atlas users
- We now have some documentation, atlassoftwaredocs, maintained by experts
  - Public, searchable by google, cleaner interface
  - Problem is it is a lot of work for over-burdened experts

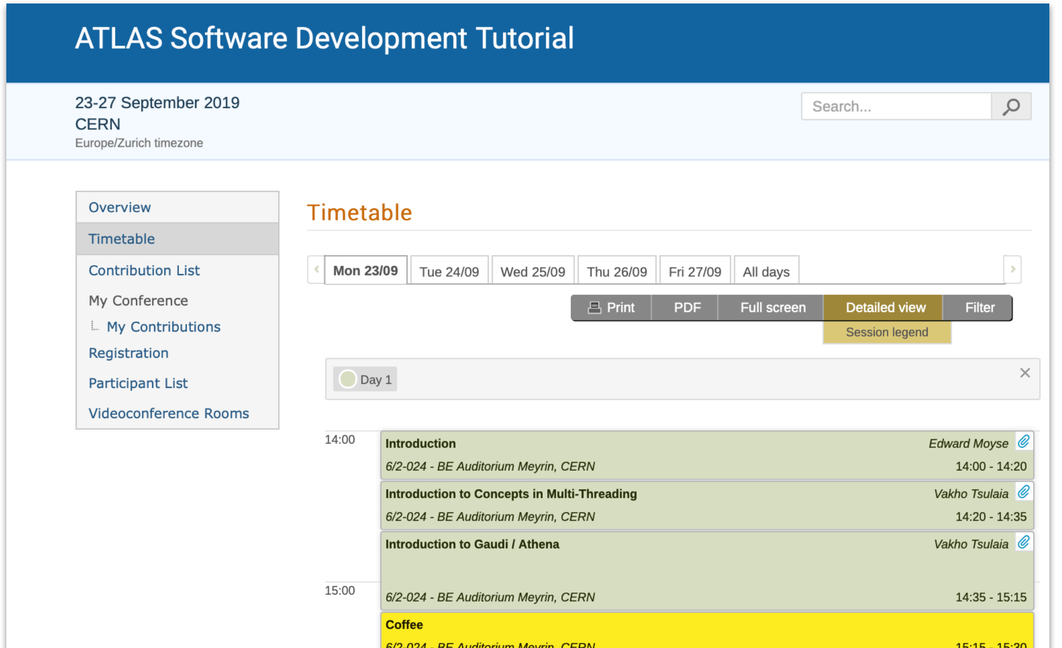




TRAINING

- Every person joining ATLAS is encouraged to go to week long induction
  - Happen multiple times a year
  - Software training is part of this
    - Primarily aimed at analysts, not software developers …
    - … but they do learn e.g. CMake and git!
- Have some infrequent software specific training
  - Merge request shifter training
  - ATLAS software developer training
  - Most recently GeoModelXML
- Have some very complete Coding guidelines





RETAINING OUR EXPERTS

- **Big problem for us:**
  - Without retaining people who understand the software, maintaining (and training the next generation of developers, is very hard)
  - Unfortunately for many, focusing on SW development is perceived to damage their career prospects
  - And indeed, we lost some key people because they could not get a job(!)
- We have tried to combat this with
  - Grants for SW development - paying people to become experts
  - Institutional commitments - i.e. institute takes responsibility for a core task)
- **Mixed results** - the core problem is (IMO) funding agencies
  - Some countries are MUCH worse than others

SUMMARY OF KNOWN SUSTAINABILITY ISSUES

- **(Dark) code rot**
  - We don’t (typically) review untouched code - but it might still run in production
- **Fractured codebase**
  - Used to be that almost all production code was in Athena
  - Nowadays we have many, many repositories (truthfully, I am not sure how many), not all of which are in gitlab
  - Some are not visible even to the SW coordinators
  - Best we can do is to try to educate people about best practices and concentrate on Athena
- **Documentation & Training**
  - Could always do better
- **Rapid turnover of personnel**
  - In some areas we have long term experts
  - In others, code is written by young physicists who move on
  - No surprise which makes code easier to sustain (though of course, longer term passing on knowledge is important too)
- **Size of codebase:**
  - Makes migrations e.g. Move to MT, Python 2 to Python 3 etc painful
  - (however forces us to review code!)

CONCLUSIONS

- ATLAS's experience with software sustainability:
  - A hard problem given our large codebase, distributed users (without clear hierarchy), and high turnover of experts
- What works:
  - Use industry-standard tools (don't re-invent the wheel)
  - Open source
  - Social coding: merge reviews
  - Documentation, training and coding guidelines
- Where we could do better
  - See previous slide!

BACKUP

NIGHTLY PERFORMANCE TESTS

- Track memory and CPU for each night
- Comparisons between nightlies