CWP WG - Software Development, Deployment and Validation/Verification

Future software repositories and build platforms that leverage advances in these areas and improved software modularity and quality control that will allow a broader community of people to effectively contribute to software in the HL-LHC era.

Participants:

ALICE: Giulio Eulisse
ATLAS: Graeme Stewart, Emil Obreshkov, Tony Limosani, Paolo Calafiura
Daniel S. Katz, Nan Niu, Benedikt Hegner
Ohio Supercomputer Center: Karen Tomko

Scope:

This working group is concerned about:

- Software development **process and methodology**, that is, the way different activities and tasks (e.g., architecting, planning, coding, building, testing, issue tracking, reusing, packaging etc.) are structured and organized;
- **Quality** of software in ecosystem, for example, functional correctness of a code module, performance and throughput of a set of code modules, modularity and dependency in an ecosystem of APIs, codes developed in-house and adapted from open source, etc.;
- Relationship (interaction, feedback…) between software developers and operation teams, including feeding back operation monitoring data to developers
- **Tools** and best practices that support the above, for example, Git, code review, architectural violation warning, software synthesis and reuse, etc.
- **Validation and verification** of the software
- **Software Development** (both technical and social) **training and support for users**
- **Licensing** and impact on software interoperability

Challenges over the next 5-10 years:

- Paradigm shift from serial to parallel architectures. Tools needed. Dissemination of best practices.
- Tools to support performance improvements
- Tools to support heterogeneous architectures
- Adoption and integration in the software process and of new tools, languages.
- Multithreading support and tools to do so.

- Improve communication (and collaborative tools) between developers
- Improve efficiency of development workflow and management of user expectations (keep them inline with what can be provided)
- Codes: dependencies on OS, compiler → DevOps, dependencies on other codes (reuse, change)
- Ownership and certification ⇒ QC
- Planning for joint development and software reuse
- Actual software reuse
- Evaluating similar technologies (?) / technology watch
- Software deprecation: sustaining legacy software, methodology that can help working around it and evolving deprecated SW (e.g. test infrastructure)...
- Automation of procedures to enforce improved software engineering and education of contributors to testing good practices (including the requirement to write tests); includes changes to the community culture that makes these changes expected
    - A challenge is to ensure that the checks on contributions are done very quickly; likely automated
    - Human feedback should be both honest and constructive
    - In the end, the challenge is to raise the software's quality without increasing the human cost too much
- Define different classes of software that have different requirements
- Improve the quality of the software without increasing the manpower costs through process automation.

Charge/Questions for the WG:

- Define metrics
- Examples, anecdotals
- Cross-experiment experiences
- What did we learn from the past: experience and requirements [Peter Hristov]? What can we learn from each other (inside the HEP community) [Liz]? What can we learn from the outside (i.e. Google, Twitter, Facebook) [Giulio]?
- Open source and HEP
- Share, agree and document best practices
- Exchange forum about new tools, technology and methodologies

WG workplan

- Write a strawman of the whitepaper based on the existing document and notes (Benedikt, Giulio, ???)
    - Introduction mentioning that SW development process has changed significantly in the last years with large adoption of new tools (e.g. GitHub, GitLab) that allow a more agile approach of SW development.
- Email discussions on the strawman + Vidyo meetings

Comments:

Jim Pivarski: Modularizing software deployment, how to detangle the parts, make them independent?

IB: Need for validation in an automated validation in a heterogeneous environment. How do we build that into our software development process and QC

Michel Jouvin/Peter Elmer: Not sure it is a WG in the same way as the others. We have some work to do in this area for sure but not necessarily challenging. Risk of speaking about today technologies.

Pere Mato/Torre Wenaus: Quality Assurance/control is a big department in any big company, so probably good to speak about it specifically to say what we want. Will require manpower so it's worth mentioning it.

Peter Hristov: What did we learn from the past: experience and requirements?

Liz Sexton-Kennedy: This is clearly an area where the community could learn from each other's efforts and perhaps evolve to more common solutions thus reducing the amount of manpower to sustain it. That will be the challenging part.

Oxana Smirnova: regardless of technologies, we have to make sure that the software:

- Is independent on the operating system or compiler and such (to a reasonable extent)
- Has clearly expressed dependencies (or is fully self-consistent)
- Well documented
- Sound from the security perspective (formal security reviews might be needed)
- Has clear ownership and clear license
    - Open Source or Public Domain
- Has a release cycle (at least clear versioning)
- Has well-defined repositories for source code and eventual binary builds
- Has functioning support system
    - Unsupported software gets deprecated

Rob Gardner: Software - or service deployment - methods to incorporate best practices in DevOps --- so much of our software/services require HEP/VO/grid specific expertise and knowledge not transferrable elsewhere. Having easily deployable software increases potential for adoption by new communities outside HEP.

Nathalie Rauschmayr: Are we taking the most out of compiler optimizations e.g. Auto-Vectorization, devirtualization, PGO, LTO? Are there any comparisons between clang,

gcc? Could a repackaging of libraries help to improve optimization passes during compilation? Experiment software has become really large, but how much of the code is actually really used and could a manual dead code elimination be useful?

Giulio (WIP):

- Whole picture to be kept into account, not limiting to tools.
- How do we foresee people developing on their laptop?
    - Modify the experiment software
    - Modify the “external” software
    - How do we reuse and validate software already present on a laptop? Why do I need to recompile gcc if I already have a compatible version?
- How do we foresee larger scale deployments?
    - How do we shorten the development - deployment - production cycle?
    - How do we make sure that only the parts which have been modified get tested?
    - How do we provide more resources for everyday development?
    - Is this really scope of this working group?
- Reusing central installations
    - Deployment of supercomputers. I know a few places in the US are now experimenting with Docker containers. Is that a trend?
    - Deployment at big university / research centers.
    - Grid deployments
        - CVMFS. Do we miss anything there? Is support for docker volumes mature enough?
        - How to evolve CVMFS to be less centrally controlled so that it can be used also for semi-private patch releases? Is the coming storage API what we need?
    - Is this really scope of this working group?
- What do other big players do when they develop software?
    - Monorepos seems to be a common approach of big companies, tools like bazel (Google) , pants (Twitter), buck (Facebook) are all designed to make a single repo with all the sources scale. Is this an anti-pattern or is there some value in it? Wouldn’t projects like ROOT, or experiment software benefit from such an approach?
    - Real code reviews: why we always agree on them and then never enforce?
    - Pull requests
    - Monitoring / Analytics infrastructure like the one offered by Google Analytics for crash reports, stacktraces and so on. How can we use them in a sensible and privacy friendly manner? E.g. aliBuild optionally uses it to track what is being built, what errors happen during the build and what platforms are used.
- What other “Github-like” opportunities are there?
    - Github is becoming the de-facto standard for hosting opensource software repositories. Are there other third party services which we could exploit? Which

parts are we willing to outsource in order to make sure we can redirect resources to more fundamental tasks? What are the risks associated to usage of those external services?
    - Travis CI
    - Codecov
    - Bintray
    - Free coverity
    - Profiling (visualization) as a Service? How about a common interchange format for profiles which different visualization apps can reuse?
    - Others?
- Is the model “download sources, build on laptop, commit code to repository, have a central build deployed” only way to go?
    - Editing the sources online?
    - Smaller deployable units for middle scale productions?
    - Serverless / infrastructureless computing: is there part of our tools and infrastructure which could be completely hosted on external cloud providers via the “serverless” (e.g. Amazon Lambda) paradigm?

Nan: (1) Are HEP SW requirements defined? Clearly defined? (2) Are the codes tested? Sufficiently tested? (3) Where did programmers spend/waste their (most) time at? Most frustrated about?

Interesting reads:

- Scaling mercurial at Facebook
- Why Google Stores Billions of Lines of Code in a Single Repository

Antonio Limosani (Software Performance Monitoring at ATLAS):

Currently I’m upgrading ATLAS’ software infrastructure for the reporting of the computing resources used by benchmark jobs in development releases. Global job statistics are also in the process of being integrated into a Kibana dashboard sourcing an Elastic Search database. Currently this infrastructure cannot support monitoring of the individual software components that make up a job because of fine granularity or rather large number of software components. However it could be foreseen that most resource expensive components could be reported and put into elastic search. The upgrade will switch to using Jenkins to schedule jobs, implement a sqlite database, and will use a simpler sustainable visualisation web page.This is a common problem I imagine among all experiments. I imagine “CERN dashboard” (http://dashboard.cern.ch and https://twiki.cern.ch/twiki/bin/view/LCG/WLCGMonDataAnalytics) be used to support this use-case? However I don’t see an example of specific job performance

metrics made available on a CERN dashboard. Currently our reporting is focused on software changes that increase resource use. The developers are then asked to justify the change. Often the resource usage change measurement is seen more as a software group responsibility
What are other experiments doing to track software performance in their development releases? How is this information fed back to developers?

These benchmark jobs will also need to be adapted to support new platforms and compilers, and will provide a means of showing whether jobs/workflows are suited to a given platform/compiler configuration.

San Diego Meeting notes

- Whitepaper for this WG is probably not about identifying challenging issues or R&D project but more to identify tools and way of working together that could help addressing the challenges we have in our code (supporting the other WGs)
- What is the role of new tools that emerged and has been adopted to a certain extent, like GitHub by CMS
- Needs for training: not only technical topics but also SW development processes (e.g. Agile methodology)

Most notes have been taken updating the scope, challenges and WG working plan.