Software Maintaince & QA

Mini workshop: Trig & Reco Input for European Strategy for Particle Physics 2025

Dmitry Kalinkin

University of Kentucky

About speaker

» Medium Energy Nuclear Physicist with experience of the RHIC collider experiments

» Deputy Software and Computing Coordinator for Development at the ePIC experiment at EIC collider

» Validation WG Convener in the ePIC experiment

Expect some ePIC/EIC references, but this is not an overview of any existing experiment's practices.

Software maintainability

$$(\textbf{Maintainability}) = 171 - 5.2 \cdot log(\text{HV}) - 0.23 \cdot (\text{CC}) - 16.2 \cdot \log(\text{LOC}) + 50 \cdot \sqrt{2.46 \cdot (\text{\% comments})},$$

where HV – Halstead's Volume, CC – Cyclomatic Complexity

↰ Over-reliant on LOCs. Could use a term for domain knowledge?

Maintenance: A process in which facilitates

» incorporation of new features
» improval of performance
» adaptation to changing conditions

As a side effect we conserve resources and build trust in software.

Maintenance

1. Correction
   - addressing bugs preemptively and from reports
2. Prevention
   - changes to ensure that desired behaviour is preserved after future changes
3. Perfectioning
   - aligning with use cases of the long term future
   - removal of ineffective functionality
4. Adapting
   - changing environments (dependencies, operating systems, hardware)
   - new technologies
   - new operation processes

Dependency management

Sustainable development of dependencies:

- Identifiable group of active **core developers**
- Project matches or capable of matching the **technical requirements**
- **Open development** model (repo on a public Git forge, public developer meetings)
- That software itself is **maintainable**

Clear **performance indicators** for experiments to pick adoptable technologies and developers to strive for.

Succesful high-level projects such as `DD4hep` and `Acts` currenly adopted across different communities.

Testing

Primary interest – automated testing (propagation of assumptions through software evolution)

General classification:

» Unit testing (e.g. for individual algorithms in reconstruction framework)

» Integration tests (e.g. connection to an external DB works, file formats match)

» Functional tests

- Smoke tests
- End-to-end run (e.g. full reconstruction with benchmarks)

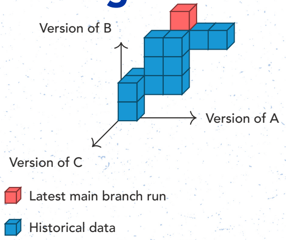
» Computing performance benchmarks (memory use and CPU time)

Pareto rule apply:

» 20% of code causes 80% of bugs

» 80% of bugs are uncovered by 20% of tests

Tracking for CI artifacts



Comparative testing is powerful if we look at a single change at a time.
Benchmarks can be as simple and universal as histogramming every branch in short output files to look for possible changes in value distributions.

- Need for a tool that extends traditional CI dashboard to include universal metrics such as named profiles and histograms, like rivet-mkhtml
- Need metadata integrated with
  - package managers
  - git histories
  - calibration databases
  - configuration registries
- Possibly a CI-agnostic tracker, like MLflow
- User interface to help navigate multi-dimensional nature of recorded points
- Synergies with "online" data QA

Workflow management

Tools that provide binding of individual components to achieve automation towards specific tasks

» Common Workflow Language
» DAGMan
» Makeflow
» Nextflow
» Snakemake
» Workflow Description Language

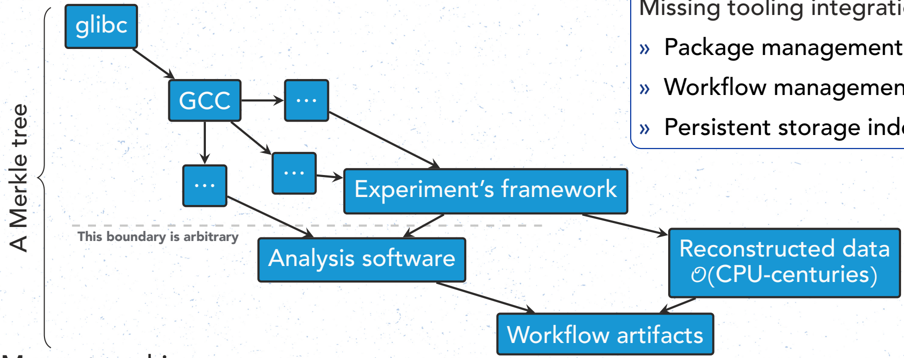
A lot of focus on getting the DSL or file format part right, but, unfortunately, **high-level descriptions are not as efficient**. Low-level are **not portable between sites**.

Doing HTC effectively is hard. In case of testing, support for **data locality** for intermediate products is desirable.

» central storage is always an option, but comes with a bottleneck for distributed computing
» some level of short term caching between pipelines is needed

Declarative specificaitons for CI allow every user to contribute to – transparent deployment procedures, avoid gatekeeping maintenance behind access.

Reproducibility



Missing tooling integration between:

- Package management (spack)
- Workflow management
- Persistent storage index and backend

Means to achieve:

- Reproducible environments/workflows

  Immutable environments are not a solution to reproducibility! Modifying arbitrary components at different level is a key functionality to extract utility out of reproducibility.
- Software designed with care about how PRNG sequences are consumed

  N.B. This is not a call for a bit-by-bit reproducibility
- Sandboxing

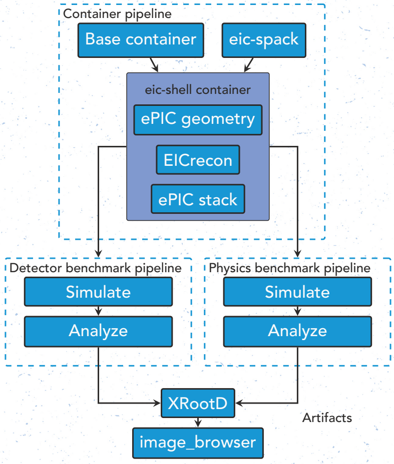
Infrastructure for running tests

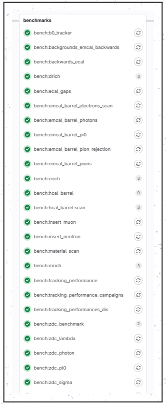
Testing with elements of full simulation/reconstruction becomes extremely resource-hungry

- Testing software is much like doing an analysis
- Instead of fixed dedicated CI resource, an ideal deployment would **reuse existing HTC** computing and scale its use on demand
- **No interoperability between industry CI tools** (GitHub Actions, GitLab CI, …) and WMS (Slurm, HTCondor, …) due to requirement of Docker
- Time for CI functionality in REANA?

"Detector" and "physics" benchmarks

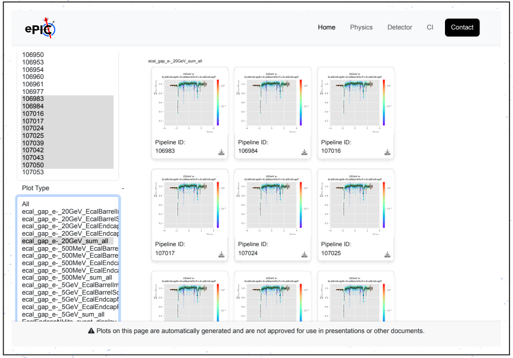
Validation effort for the ePIC detector design





Running in a dedicated GitLab CI (AMD EPYC 7H12 – 256 threads, 512 GB)

`image_browser`, a prototype



University of Kentucky

Application of AI

» long term projections are subject to a large uncertainty

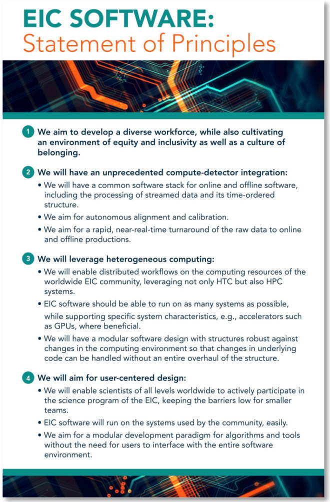
» expect a major **change in cost function** for routine development tasks

» software as a knowledge database with **value in its connections to material world**
unambiguous documentation for experiment's hardware will be most wanted?

» regardless of how landscape will look, the necessary tooling will need to be deployed and adopted
...so **embrace automation** now!

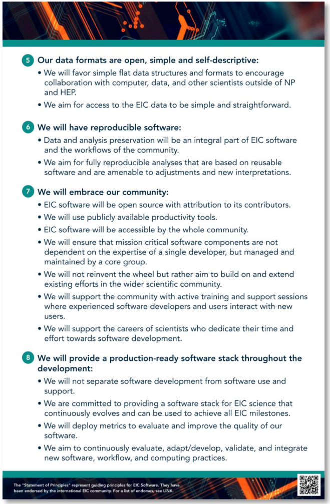
Guiding principles

Example case: "EIC Software: Statement of principles"



EIC SOFTWARE:
Statement of Principles

1. **We aim to develop a diverse workforce, while also cultivating an environment of equity and inclusivity as well as a culture of belonging.**
2. **We will have an unprecedented compute-detector integration:**
   - We will have a common software stack for online and offline software, including the processing of streamed data and its time-ordered structure.
   - We aim for autonomous alignment and calibration.
   - We aim for a rapid, near-real-time turnaround of the raw data to online and offline productions.
3. **We will leverage heterogeneous computing:**
   - We will enable distributed workflows on the computing resources of the worldwide EIC community, leveraging not only HTC but also HPC systems.
   - EIC software should be able to run on as many systems as possible, while supporting specific system characteristics, e.g., accelerators such as GPUs, where beneficial.
   - We will have a modular software design with structures robust against changes in the computing environment so that changes in underlying code can be handled without an entire overhaul of the structure.
4. **We will aim for user-centered design:**
   - We will enable scientists of all levels worldwide to actively participate in the science program of the EIC, keeping the barriers low for smaller teams.
   - EIC software will run on the systems used by the community, easily.
   - We aim for a modular development paradigm for algorithms and tools without the need for users to interface with the entire software environment.



5. **Our data formats are open, simple and self-descriptive:**
   - We will favor simple flat data structures and formats to encourage collaboration with computer, data, and other scientists outside of NP and HEP.
   - We aim for access to the EIC data to be simple and straightforward.
6. **We will have reproducible software:**
   - Data and analysis preservation will be an integral part of EIC software and the workflows of the community.
   - We aim for fully reproducible analyses that are based on reusable software and are amenable to adjustments and new interpretations.
7. **We will embrace our community:**
   - EIC software will be open source with attribution to its contributors.
   - We will use publicly available productivity tools.
   - EIC software will be accessible by the whole community.
   - We will ensure that mission critical software components are not dependent on the expertise of a single developer, but managed and maintained by a core group.
   - We will not reinvent the wheel but rather aim to build on and extend existing efforts in the wider scientific community.
   - We will support the community with active training and support sessions where experienced software developers and users interact with new users.
   - We will support the careers of scientists who dedicate their time and effort towards software development.
8. **We will provide a production-ready software stack throughout the development:**
   - We will not separate software development from software use and support.
   - We are committed to providing a software stack for EIC science that continuously evolves and can be used to achieve all EIC milestones.
   - We will deploy metrics to evaluate and improve the quality of our software.
   - We aim to continuously evaluate, adapt/develop, validate, and integrate new software, workflow, and computing practices.

The "Statement of Principles" represent guiding principles for EIC Software. They have been endorsed by the international EIC community. For a list of endorses, see LINK.

Advertised at
https://eic.github.io/activities/principles.html

- » agile development
- » production-ready software stack
- » meeting near-term needs of ePIC
- » timeline-based prioritization
- » user-centered design

Conclusion

Possible recommendations are:

» Consider a reference set of CI tools with support of running on existing farms

» Develop tooling for quantitative tracking of metrics with focus on HEP/NP experiment needs

» Embrace reproducibility in software design, delivery and workflows for data processing and analysis

» Expand user training to get them proficient and engaged with modern DevOps practices