Software Quality Control at Belle II

Martin Ritter ¹, Thomas Kuhr ¹, Thomas Hauth ², Timothy Gebhard ², Michal Kristof ³, Christian Pulvermacher ⁴ for the Belle II Software Group

¹Ludwig-Maximilians-University, Munich
²Institute for Experimental Nuclear Physics (IEKP), KIT
³Charles University, Prague
⁴High Energy Accelerator Research Organization (KEK), Japan

22nd International Conference on Computing in High-Energy and Nuclear Physics
October 12, 2016
Asymmetric $e^+e^-$ experiment mainly at the $\Upsilon(4S)$ resonance (10.58 GeV)

Focus on B, charm and $\tau$ physics

<table>
<thead>
<tr>
<th></th>
<th>KEKB/Belle</th>
<th>SuperKEKB/Belle II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>operation</strong></td>
<td>1999–2010</td>
<td>2018–2024</td>
</tr>
<tr>
<td><strong>peak luminosity</strong></td>
<td>$2.11 \times 10^{34}$ cm$^{-2}$s$^{-1}$</td>
<td>$8 \times 10^{35}$ cm$^{-2}$s$^{-1}$</td>
</tr>
<tr>
<td><strong>integrated luminosity</strong></td>
<td>$1023$ fb$^{-1}$ (772 million BB pairs)</td>
<td>$50$ ab$^{-1}$</td>
</tr>
</tbody>
</table>
**Electromagnetic Calorimeter**
16 X_0 CsI Crystals

**Time of Propagation counter**
20 mm quartz radiators for time of flight and Cherenkov-based PID

**K^0_L, \mu system**
RPC and Scintillator+SiPM between iron plates

**Magnetic Field**
1.5 T superconducting magnet

**Electromagnetic Calorimeter**
8000 CsI Crystals, 16 X_0

**Pixel Vertex Detector**
2 layer pixel detector (8MP)
DEPFET technology

**Silicon Vertex Detector**
4 layer double-sided strips
20–50 ns shaping time

**Central Drift Chamber**
Proportional wire drift chamber
15000 sense wires in 58 layers

**Aerogel RICH**
Proximity focusing RICH with silica aerogel
Mainly written from scratch using experiences from Belle and other experiments

- modular approach
- utilize new technologies: C++11 (GCC 5.2), ROOT 6, Geant 4.10, Python 3.5
- Python as steering/scripting language
- ROOT for input/output (also raw data)
- parallel processing support using `fork`
- approximately one million lines of code organized in 30 packages
- one librarian responsible for each package
- decides on commit permissions in package
- organized in central git repository

**Git Version Control**

- switched from SVN this summer
- hosted at DESY using Atlassian tools (Stash)
- works smoothly in most cases, still some tuning to be done

- librarians free to decide on development workflow
- monthly integration build to avoid divergence
Used very linear history in SVN
  ▶ cross-package changes required lots of coordination
  ▶ patches by email

After switching to Git/Stash
  ▶ commits to master branch still possible
  ▶ users quickly adopting to using branches
  ▶ pull-requests to simplify coordination of larger changes

Extensive Style and Permission Checks
  ▶ enforce common code style using artistic-style, pycodestyle
  ▶ check package permissions on master branch
  ▶ implemented in local pre-commit and server pre-receive hook
Issue tracking

- switched from Redmine to Atlassian JIRA
- old issues converted
- good user acceptance so far

Build Services

- using buildbot and Atlassian Bamboo
- incremental build on commits for all branches

The number of issues created vs. the number of issues resolved in the last 60 days.
Nightly build creates reports on
- compiler warnings (gcc, clang, icc)
- cppcheck, clang static analyzer
- missing doxygen documentation
- valgrind memcheck
- geometry overlaps
- missing/extra library dependencies
- unittest results
Three levels of testing and validation

**Unittests**
- using Google Test
- cover large part of the core framework, sparse use in other packages
- limited in complexity of setup/teardown

**Framework tests**
- run steering files in special folders
- check exit code
- optionally check output
- allows for larger complexity of tests

Run automatically after commits to master, user notification on fail
Testing and Validation

Three levels of testing and validation (cont.)

Physics Validation

- run steering files in special folders honoring dependencies
- create comparison plots from output root files
- compare between revisions and against known template
- run once per day

▶ all three can be run by the developers, either for selected packages or everything
▶ tools to simplify git bisect usage for finding responsible commits
Currently migrating to GCC 6.2

- ROOT version 6.06/00 showed problems in unit tests
- updated ROOT and ran full physics validation
- degradation in performance between GCC 5.2 and GCC 6.2
- erroneous \texttt{abs(double)} which gets compiled as \texttt{::abs(int)} or \texttt{std::abs(double)}

✓ problem could be fixed before migration
Meetings/Shifts

Weekly developer meetings
- informal, mostly oral
- short reports by librarians
- questions by users/developers
- quality shift report

Monthly software meeting
- more formal, fixed agenda
- presentations of new features
- discussion of release cycle

Software Quality Shifter
- weekly shifts to check for problems
- inquire about old issues, check style guides and documentation, review changes...
- counts as service task
Conclusions

Code Management
- switched from SVN to Git this year
- utilize branches/pull-requests to simplify collaboration
- sophisticated build service and monthly integration builds

Code Quality
- enforced unified code style
- different compilers (gcc, clang, icc)
- cppcheck, doxygen, memcheck, clang static-analyzer

Validation and Testing
- three levels of testing: Unit tests, framework tests and physics validation
- all runnable by the user

Others
- short weekly meetings, detailed monthly meetings
- software quality shift
Thank you for your attention