ACAT 2021

A new Software Training Model at Belle II

Kilian Lieret, Alejandro Mora, Moritz Bauer, Michel Bertemes, Sviatoslav Bilokin, Giulia Casarosa, Racha Cheaib, Samuel Cunliffe, Angelo Di Canto, Giacomo De Pietro, Nathalie Eberlein, Michael Eliachevitch, Marcela Garcia Hernandez, Philip Grace, Daniel Greenwald, Oskar Hartbrich, Michel Hernandez Villanueva, Chia-Ling Hsu, Ilya Komarov, Thomas Kuhr, Yaroslav Kulii, Frank Meier, Marco Milesi, Gian Luca Pinna Angioni, Markus Tobias Prim, Martin Ritter, Pascal Schmolz, Umberto Tamponi, Francesco Tenchini, Hannah Wakeling, Xing-Yu Zhou

Contact: kilian.lieret@posteo.de, software-doc@belle2.org

Abstract

The physics output of modern experimental HEP collaborations hinges not only on the quality of its software but also on the ability of the collaborators to make the best possible use of it.

With the COVID-19 pandemic making in-person training impossible, the training paradigm at Belle II was shifted towards one of guided self-study.

To that end, the study material was rebuilt from scratch as a series of modular and hands-on lessons tightly integrated with the software documentation on Sphinx. Each lesson contains multiple exercises that are supplemented with hints and complete solutions. Rather than duplicating information, students are systematically taught to work with the API documentation to find the important sections for themselves. Unit tests ensure that all examples work with different software versions, and feedback buttons make it easy to submit comments for improvements.

On this poster, we detail our experiences with the new setup and training model.

Challenges

The training model at Belle II faces the following challenges:

- **Suitability for both self-study and in-person events**
- **Versioning:** Analyst-facing API of the Belle II software is still evolving ⇒ different versions of lessons for different versions of software
- **Testability:** Code snippets in the lessons should be tested to ensure that they are working
- **Maintainability and sustainability:** Lessons should be consistent, stable, and easy to update.
- **Interactivity:** Exercises improve the learning experience and keep students engaged

Some **trade-offs** between these points are necessary. For example, self-study-ready material requires a level of verbosity that may be overwhelming when presented in traditional in-person events.

General principles

- Lessons are available as **web pages built with Sphinx**[1]
- Verbose text. Didactical style inspired by work of software carpentries[2] / HSF training[3] / LHCb StarterKit[4]
- **Source code is hosted in git together with the main software** (basf2, now open source[5]!)

  ⇒ Natural correspondence of software versions and lesson versions

  ⇒ Good for unit testing
- **Do not replicate existing documentation!** Either link to it or make it an exercise for students to find the information.
  - Helps with maintainability
  - Teaches students to work with "normal" documentation
- **Refer to outside training** for some software prerequisites (e.g., python). Currently using material of sw carpentries.
- **Events:** Guided self-study with Q & A sessions, mentoring sessions, and self-study time in between. Chat channels for asynchronous help.

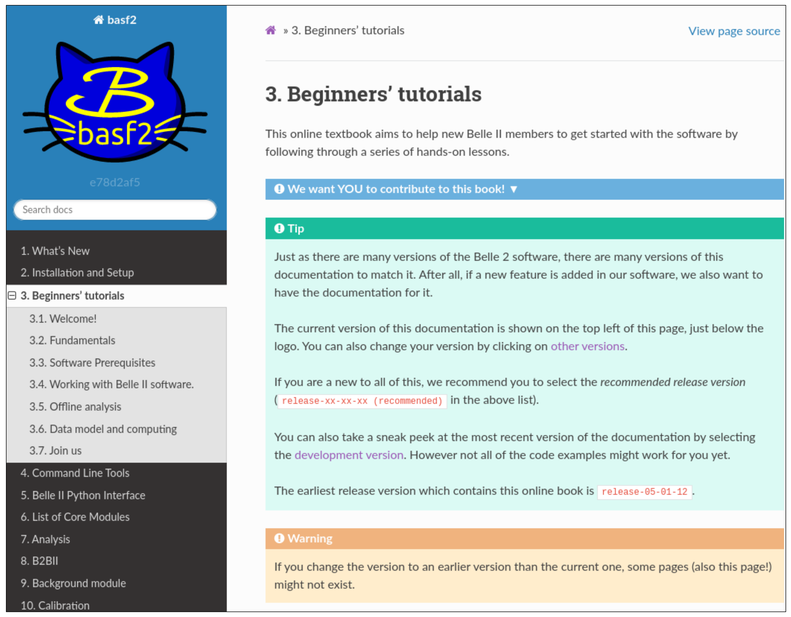

Figure 1. Home of the lessons at training.belle2.org

Figure 2. Features and statistics

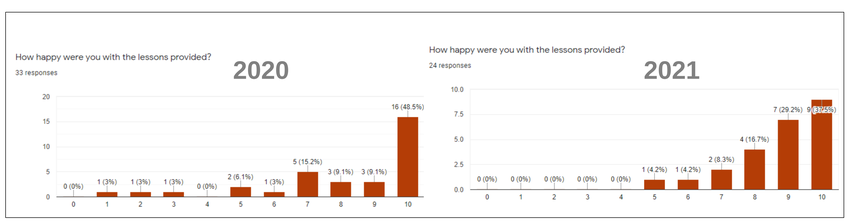

Figure 3. Customer satisfaction: "How happy were you with the lessons?"

Technical setup

- **Continuous roll out:** Developed on the main branch of the basf2 git repository (view package on github[6])
  - Recommended training versions live on release branches
    - E.g., training.belle2.org automatically points at lessons of the latest full release
    - For substantial improvements/hotfixes: Needs cherry-picking into minor or fix-releases
  - Generally, this means that changes propagate relatively slowly (but you can always point a student to the latest development build)
- **Code inclusion:**
  - Put code in **separate files** and use code inclusion
    - These files are then executed as unit tests
    - Allows to include complete "solutions" at the end
    - Allows using code-inspection/formatting tools for a uniform experience and fewer bugs
  - For step-by-step build-up of larger code examples: Use **partial code-inclusion**
    - Initial version: line-number based ⇒ very cumbersome with code changes as line numbers need to be updated
    - Now: include small marker comments in code and use start-after/end-before directives of Sphinx
      ⇒ Using explicit markers consistently solves ambiguity issues
      ⇒ makes it obvious where code is included
- Additional **admonition types for Sphinx**: e.g., exercise blocks, foldable hints and solutions (Fig. 2)
- Quick **feedback form** below every lesson for bug reports, however still rarely used (Fig. 2)

Experiences

- **Technical complexity:** Contributions require basic knowledge of git, PR workflows, reStructuredText (rst), and sphinx directives.
  - Automatic Sphinx builds in PRs for preview & checks, but take ~20 min. Local builds are faster but require initial compilation of (part of) main software + 1-3 min per Sphinx build.
  - Contributing to documentation teaches the entire software development workflow
  - Additional step-by-step tutorials and dedicated hackathons could help beginners to contribute, growing number of unique contributors
- **Very complete onboarding experience**, even for newcomers who join "off-season." Lessons cover everything from basic physics knowledge and software prerequisites to submitting grid jobs.
- Very **positive feedback** (Fig. 3), e.g., "Very solid work regarding the textbook! Congrats! Everything was very clear which significantly minimized the need for guidance (…) Software Prerequisites was the highlight of the workshop as it summarized all the necessary tools that no one really spends time on explaining thoroughly to newcomers."
- **Very few issues with broken code snippets** due to unit testing

[1] https://www.sphinx-doc.org/
[2] https://software-carpentry.org/
[3] https://hepsoftwarefoundation.org/training
[4] https://lhcb.github.io/starterkit-lessons/
[5] https://github.com/belle2/basf2
[6] https://github.com/belle2/basf2/tree/main/online_book