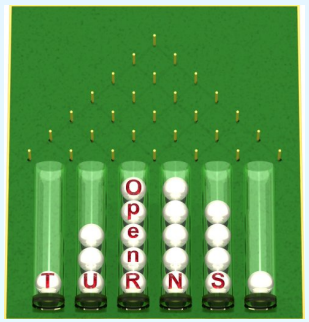
OpenTURNS Developer Training: the platform overview

Trainer : Régis LEBRUN
Airbus
regis.lebrun@airbus.com

Developers training



Platform overview

Introduction

Objectives

The objectives of this course is to give a broad overview of the OpenTURNS project and the resulting platform. We will cover the following points:

- The history of the project,
- The global organization of the platform,
- The multi-layer view of the library,
- The several usages of the platform.

History

2005-2025: 20 years of partnership

2005 Conception

2007 First release the 10th of May. Python bindings.

2009 Multithreaded wrappers, polynomial chaos expansion.

2010 First windows port, parallelization.

2011 Sparse chaos implementation.

2012 v1.0, Stochastic processes

2013 Bayesian updating, matplotlib viewer

2014 Kriging, native windows support

2015 Vectorial kriging, HMat support

2016 Karhunen-Loeve process decomposition, NLopt bindings

2017 Canonical format low-rank tensor approximation, field functions

2018 Domains arithmetic, asymptotic Sobol' estimators, new simulation algorithms

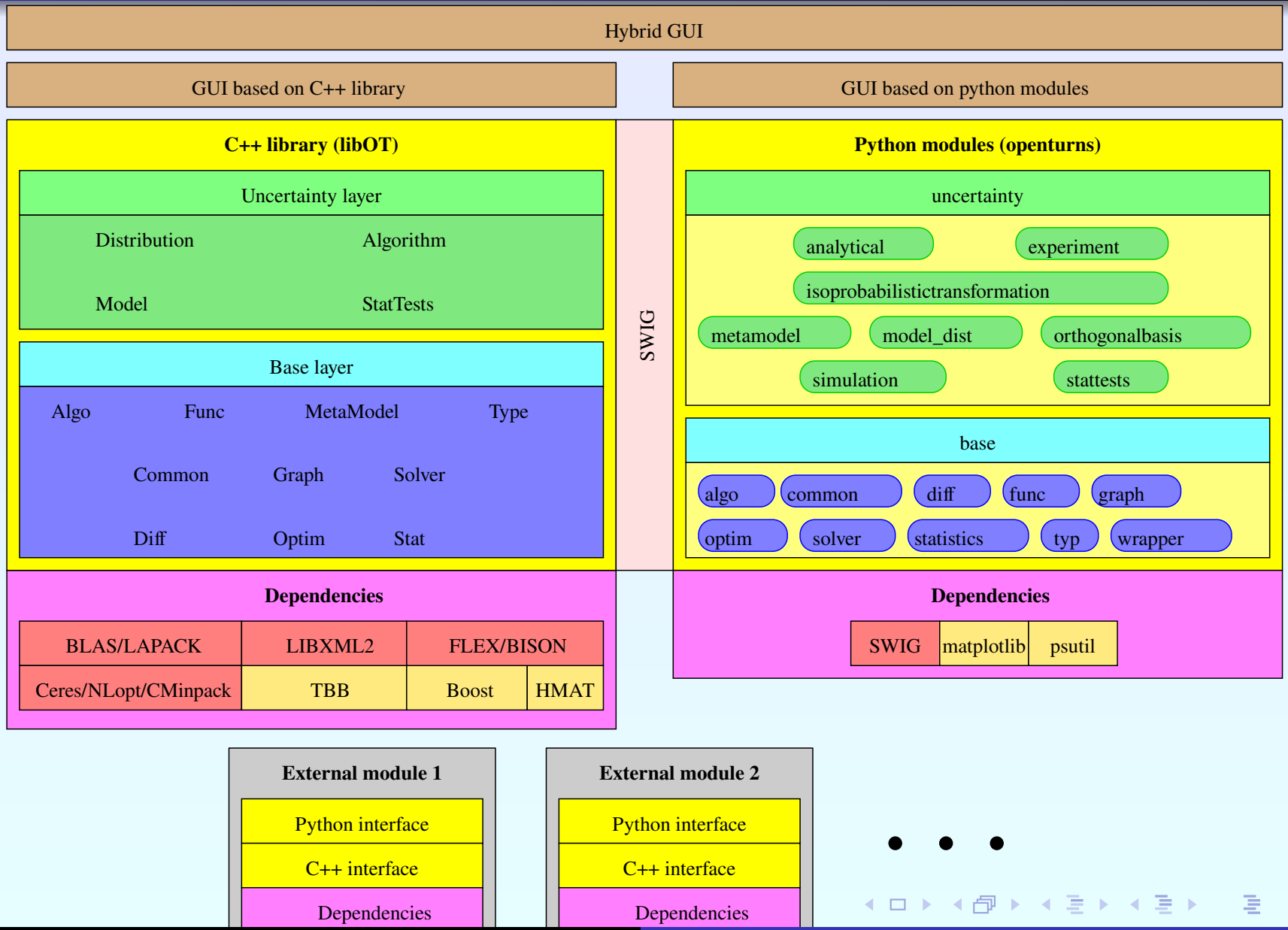
2019 Calibration, new optimization algorithms, system events

2020 hmat AcaRandom compression, Spectra iterative SVD, examples gallery, XML/H5

2021 Karhunen-Loeve validation, new covariance models

2022 HSIC indices, iterative statistics, field chaos/sensitivity

The platform at a glance, developer's view



The platform at a glance, developer's view

The big parts

- The core of the OpenTURNS platform is a C++ library, made of about 500 classes of various size. The library has a multi-layered architecture that is materialized by both the namespace hierarchy and the source tree.
- The main user interface is the python module, automatically generated from the C++ library using the wrapping software SWIG. It allows for a usage of OpenTURNS through python scripts of any level of complexity.
- The library relies on relatively few dependencies and most of them are optional.
- A service of modules is provided in order to extend the capabilities of the platform from the outside.
- Several GUIs have already been built on top of the C++ library or the Python module.

The C++ library

A multilayered library

The two main layers in the C++ library are the Base layer and the Uncertainty layer.

- Base layer: it contains all the classes not related to the probabilistic concepts. It covers the elementary data types (vectors as Point, samples as Sample), the concept of models (Function), the linear algebra (Matrix, Tensor) and the general interest classes (memory management, resource management);
- Uncertainty layer: it contains all the classes that deal with probabilistic concepts. It covers the probabilistic modelling (Distribution, RandomVector), the stochastic algorithms (MonteCarlo, FORM), the statistical estimation (DistributionFactory), the statistical testing (FittingTest)

A class in the Uncertainty layer can use any class in the Base or the Uncertainty layer.
A class in the Base layer can ONLY USE classes in the Base layer.

The C++ library

A monolythic library?

The C++ library is provided as a unique object file (libOT.so) created using the libtool technology. As such, it is a monolythic library (of about 8Mo stripped), but internally it is made of numerous sub-libraries, one for each folder in the source tree. A future objective is to modularize this library in order to speed-up both the compilation time and the loading time.

A parallel library

Some of the most time-consuming algorithms have been parallelized using the Thread Building Block technology (INTEL), a C++ library that allows for a parallelization of C++ code in a shared memory model. One of the objectives of OpenTURNS is the ability to execute external simulation softwares on large simulation models for a large amount of independent data sets. As such, OpenTURNS provides basic functionalities to distribute the executions of these simulations on a multiprocessors(cores) infrastructure using the low-level pthread technology or the TBBs.

The python module

Interfacing python and C++ libraries: SWIG

In order to provide a convenient interface to the user, the C++ library can be manipulated as a set of Python modules (18) that are organized through a hierarchy, on top of which stands the openturns module. The binding of the library is done almost automatically by SWIG (Simplified Wrapper Interface Generator) through a set of SWIG interface files.
An additional significant work has been made in order to ease the interation between the OpenTURNS objects and the native Python objects.
A unique feature of the Python interface is the ability to wrap a Python function into an OpenTURNS concept of mathematical function, namely the Function, using a specific class: the OpenTURNSPythonFunction. Using this mechanism, it is possible to adress virtually all the scenarios of coupling with an external simulation software.

The target OSes

A linux platform that works on windows

The historic platform of OpenTURNS is Linux. The first stage of developments have been made on 32 bits Intel Linux platforms (debian, mandriva), then the (minor) adaptations have been made in order to run on 64 bits Intel platforms as well. Currently, the platform works on the following platforms:

- Linux;
- macOS;
- Windows

The development platform remains Linux, the Windows version is obtained by cross compilation under Linux (using the MinGW tools). Native compilation is possible too.

The development infrastructure

Compilation infrastructure

The present compilation infrastructure relies on CMake. In the past the autotools were used. It covers:

- The detection and configuration aspects of the platform;
- The dependency management of the sources;
- The generation of parallel makefiles;
- The regression tests;
- The library packaging.

The use of CMake provides a way to compile the Windows version using Microsoft compilers, in order to easily reuse the C++ library in native Windows projects.

The development infrastructure

Versioning system

The versioning system used for the development of the whole platform is Git, the project is currently hosted by GitHub.

- Git is a software versioning and a revision control system started by Linus Torvalds. It is used for the sources of the platform, the documentation, as well as for the development of modules.
- GitHub is a web-based hosting service for version control using Git. GitHub allows hyperlinking information between a bug database, revision control and wiki content.

Repositories

Several repositories are used for the development of the platform, its documentation and its modules. This choice has been made for the following reasons:

- The time scale is not the same for these three activities;
- The teams are different partly in term of people, but mainly in term of expertise.

The development infrastructure

Platform repository

The openturns Git repository is in charge of both the C++ library source code and the Python interface. It has the following layout, which is quite standard:

- A master branch that stores the source code of the upcoming version of the platform. The rule is to have only source code that pass with success all the tests embedded with both the library and the python module.
- Maintenance branches that contain stable versions.
- The development branches are part of contributors forks and integrated into master via pull requests.
- Several tags for each release candidates or official releases. 20 releases have been tagged so far in Git.

The usage of this infrastructure is described in the developer documentation, one of the several documents that come with the platform. In particular, a specific role is assigned to an integrator, in charge of the merges from the different branches into master.

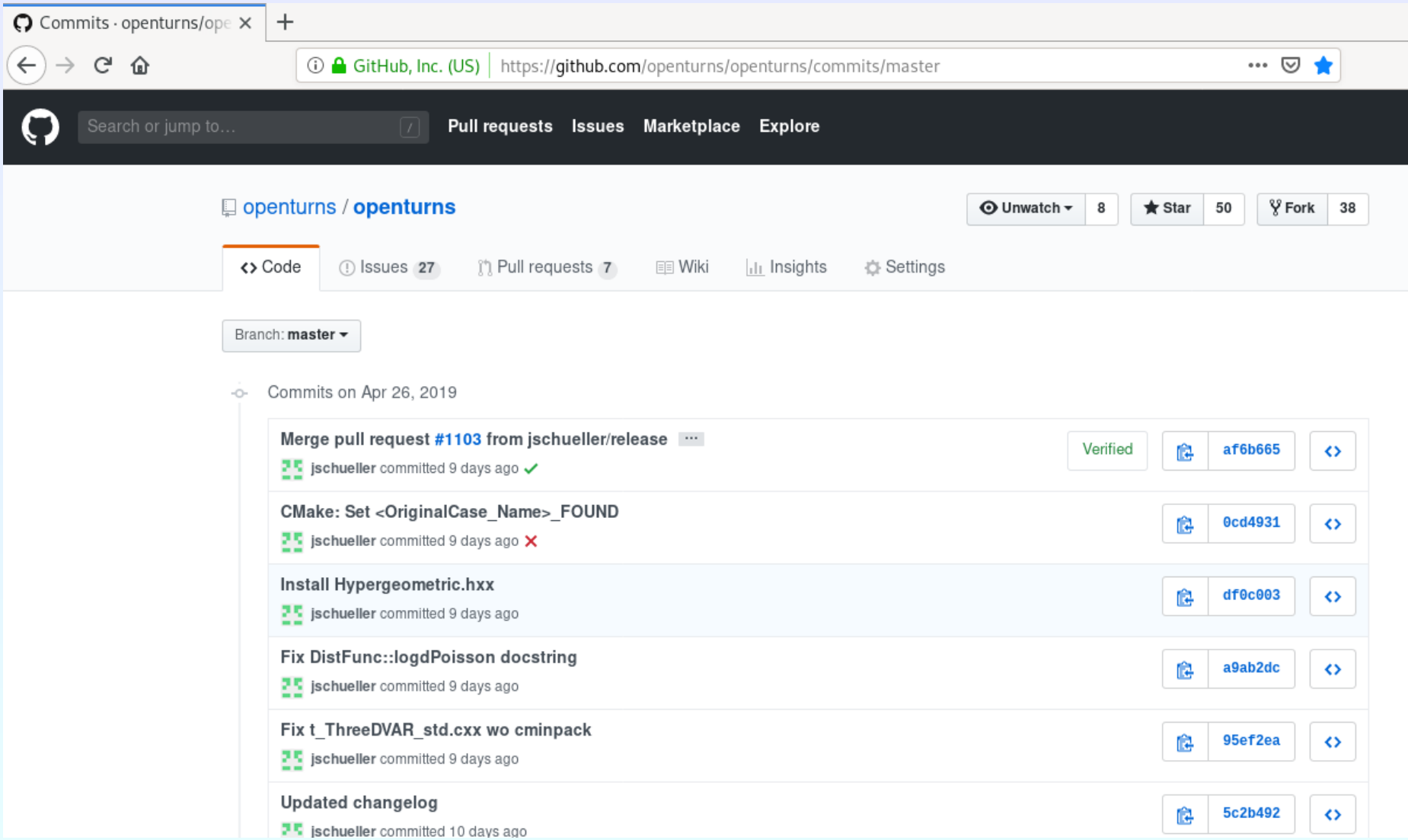
The development infrastructure

Continuous integration

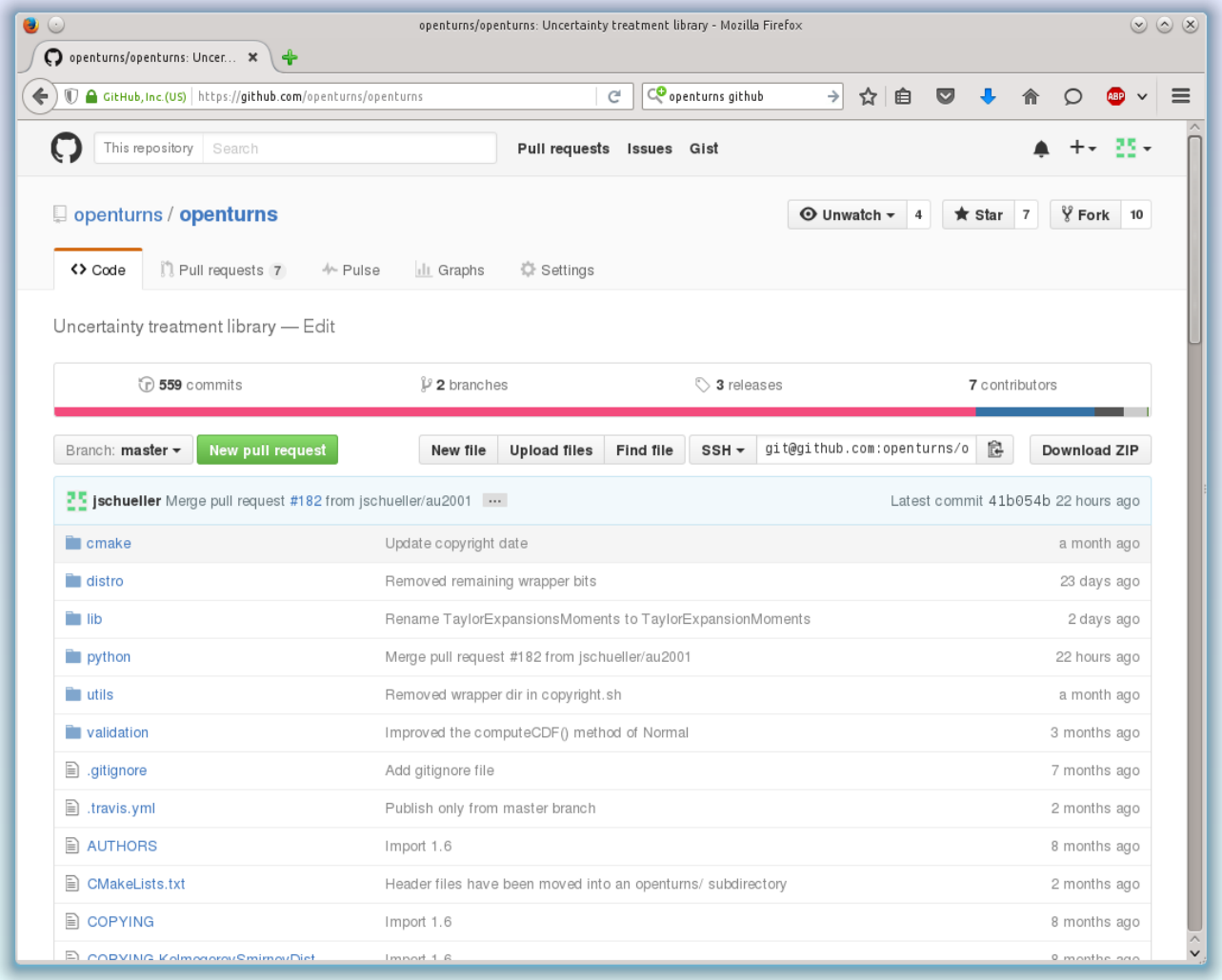
OpenTURNS makes extensive use of online continuous integration providers to test the library throughout the development process. Currently the following services are used to test the library under several platforms:

- CircleCI: Linux and MinGW
- Travis: macOS
- Appveyor: MSVC

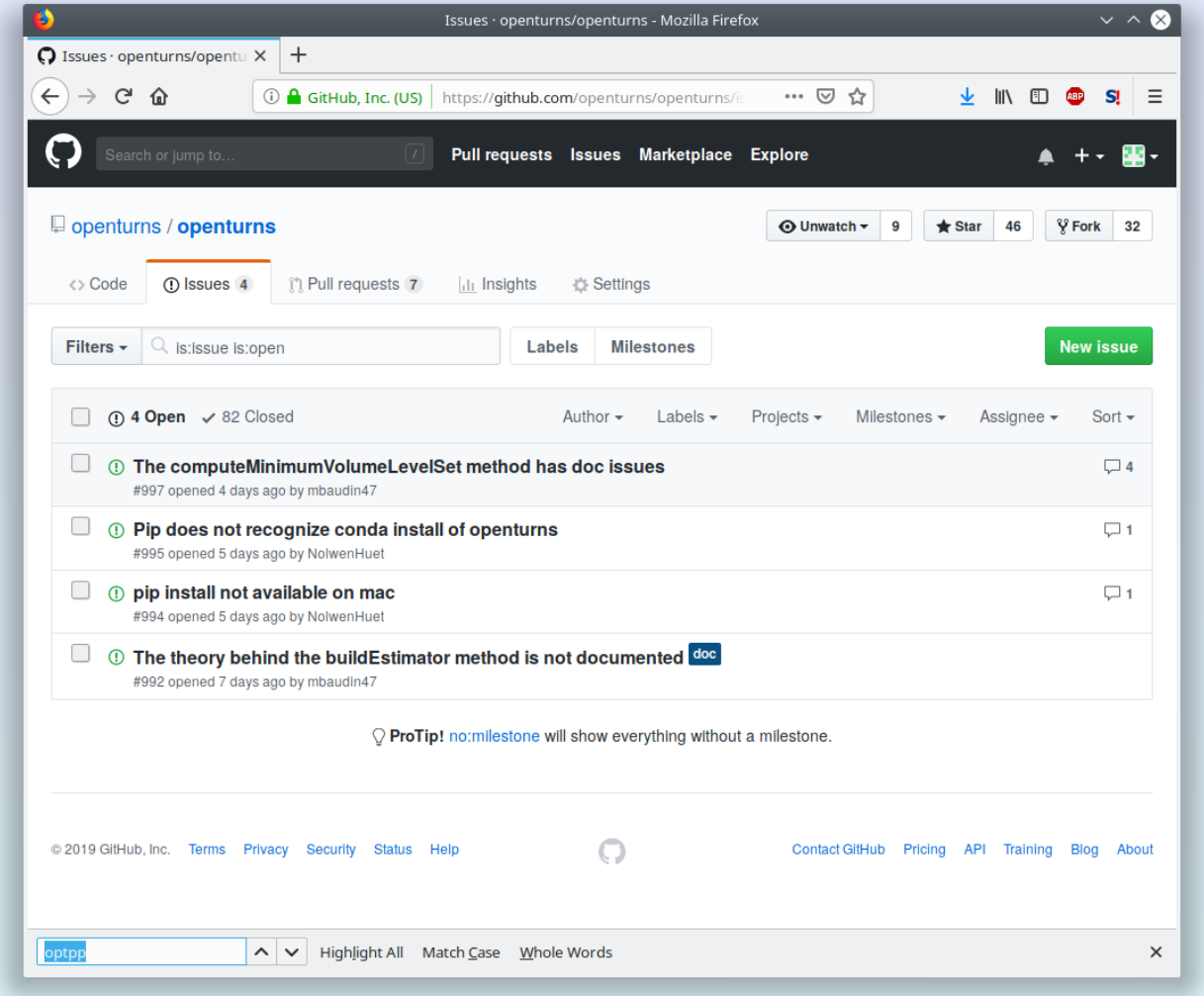
GitHub interface: the timeline



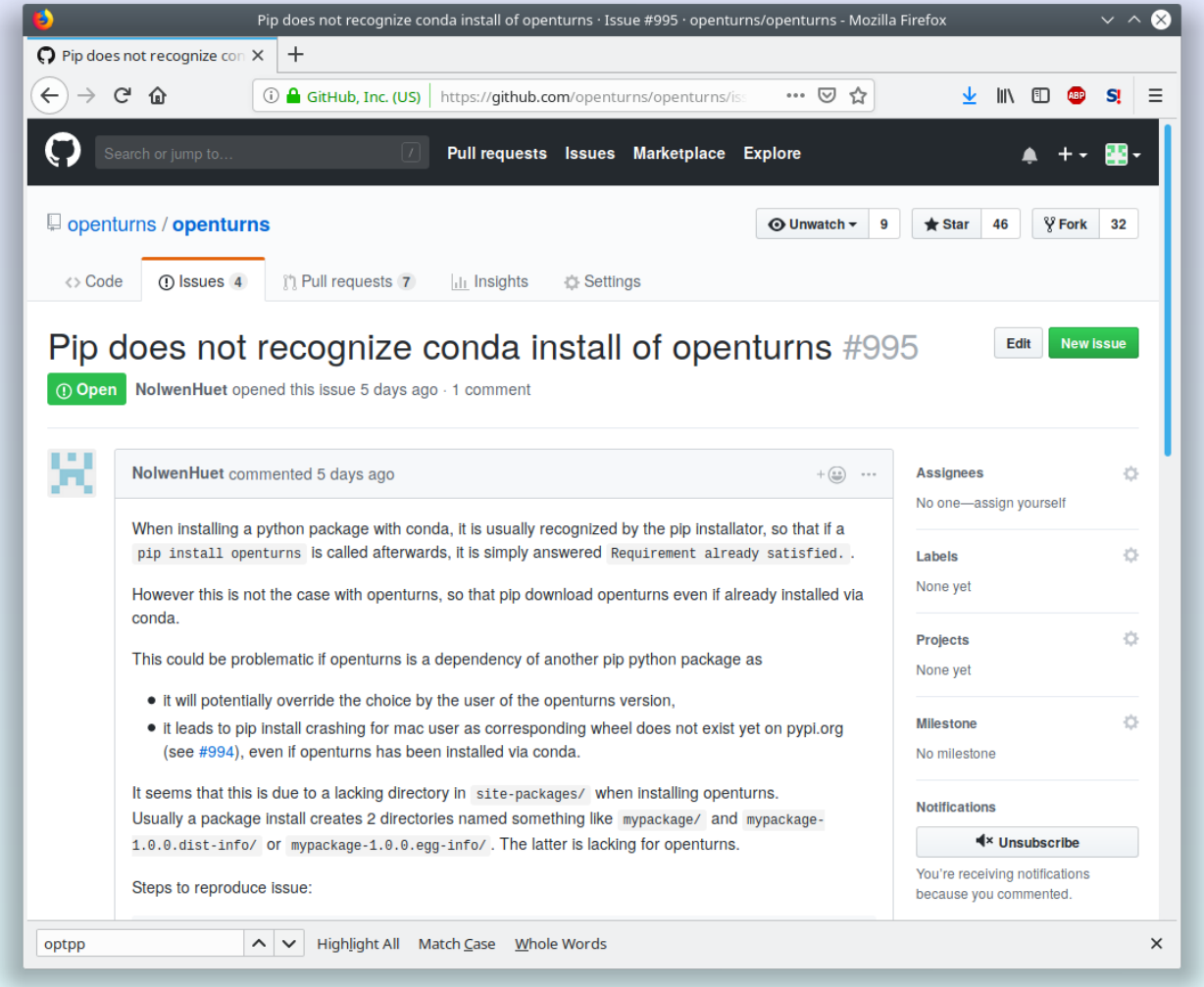
GitHub interface: the source navigator



GitHub interface: the bug tracking (1/2)



GitHub interface: the bug tracking (2/2)



Conclusion

- A quite mature project: 17 full years of development
- A structured (rigid ;-)?) development process
- A working infrastructure to help the developer in his/her hard job!