5th Coding Sprint

1. Short overview over GammaLib and ctools
2. New features since last coding sprint
3. New development workflow
4. Goals of this sprint

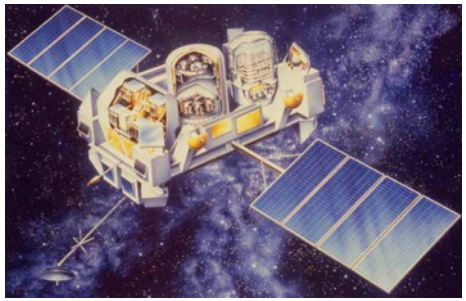
Jürgen Knödlseder (IRAP)

1. Short overview over GammaLib and ctools

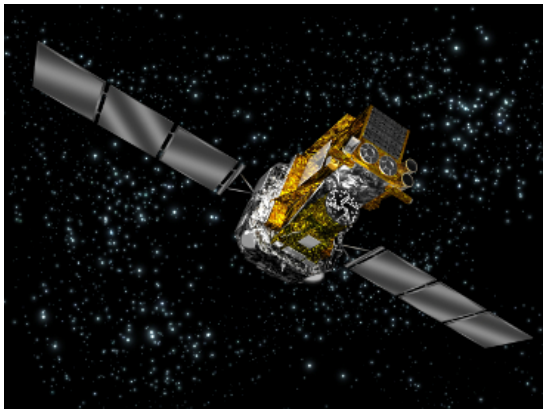
The advent of VHE astronomy

All these telescope produce the same kind of data, but for every telescope a different analysis software is used

The span of gamma-ray astronomy



CGRO

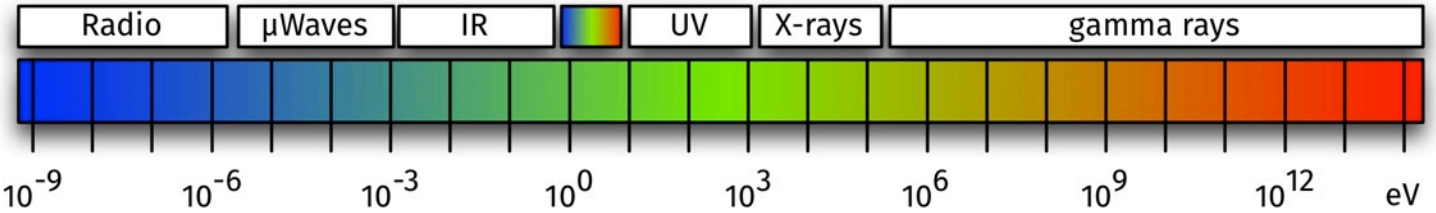


INTEGRAL

Fermi

HAWC

ASTROGAM



Gamma-ray astronomy commons

All instruments produce events

Events are characterised by an instrument direction, an energy estimate, and a trigger time

FITS is becoming the standard for delivering event data

Space-based telescopes deliver event data in FITS format. Efforts are underway to transform current IACT data into FITS. CTA will deliver event data in FITS. Dedicated workshop on 6-8 April 2016 in Meudon (Paris, France).

Maximum likelihood fitting is widely used for data analysis

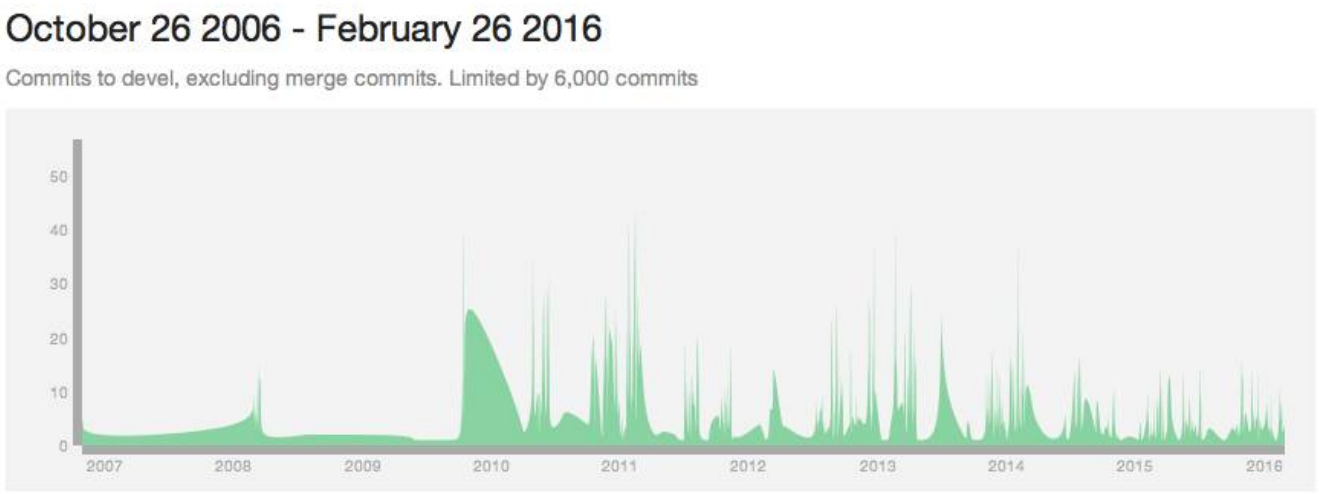
Fermi-LAT, INTEGRAL, COMPTEL, EGRET, OSSE, … and there is no reason not to use it also for VHE astronomy

Many analysis packages implement a modular ftools-style software

Let's do it ...

All these considerations led to the idea to develop a common software framework for the analysis of gamma-ray event data

Development started (slowly) in 2006 when I need a sparse matrix implementation to fit more than 10 000 parameters in a maximum likelihood fit for INTEGRAL/SPI data ...
Got with Fermi-LAT the first instrument support in 2008 ...
And took off in 2010 with the implementation of CTA support



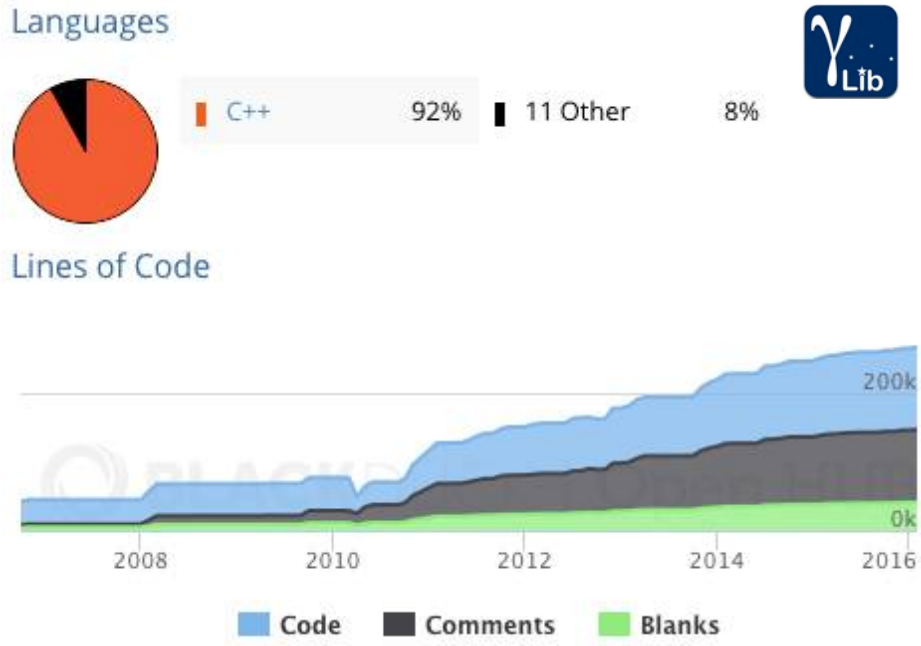
Let's do it …

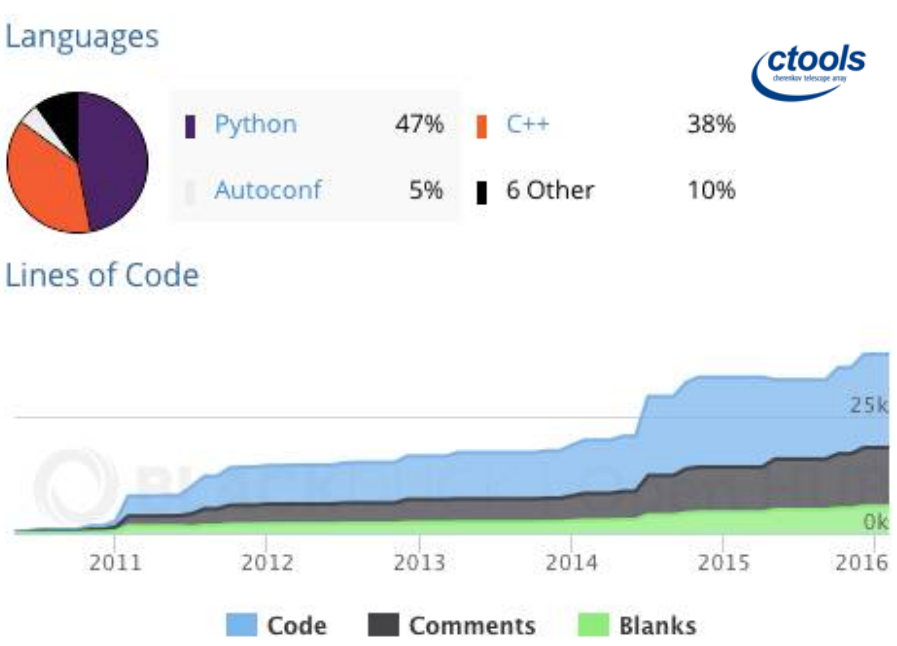
And once all the bricks for managing CTA data were there it was time to create …

ctools
cherenkov telescope array

Development started in 2010 as ctatools
First release in 2011 comprises ctobssim, ctselect, ctbin and ctlike (pure C++ package)
Named ctools in 2011 (with Python support)
Today comprises ~30 tools

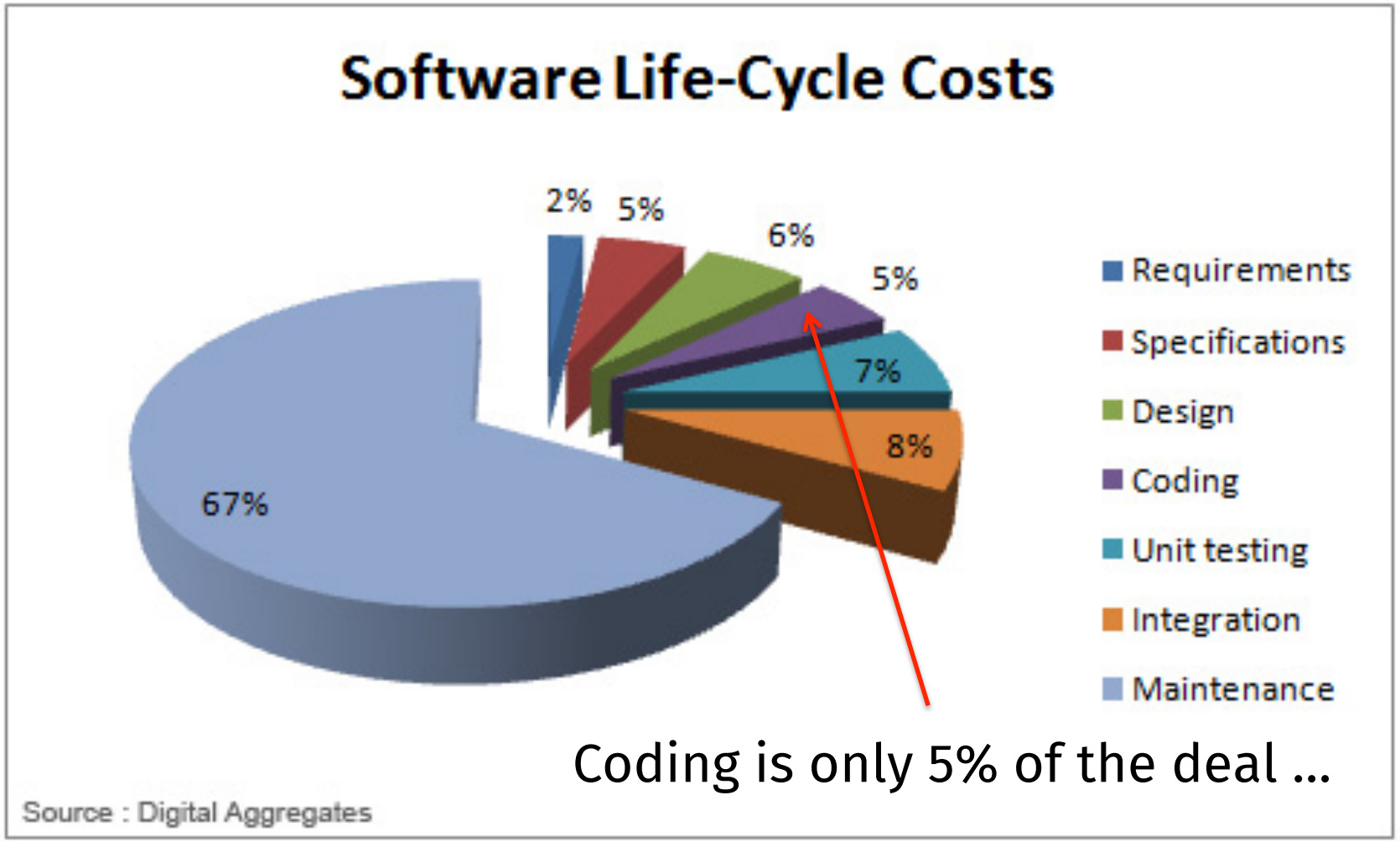
Code evolution





ctools first language is now Python

Developing sustainable software



Developing sustainable software

Some choices I made:

Define and enforce coding rules (code quality)

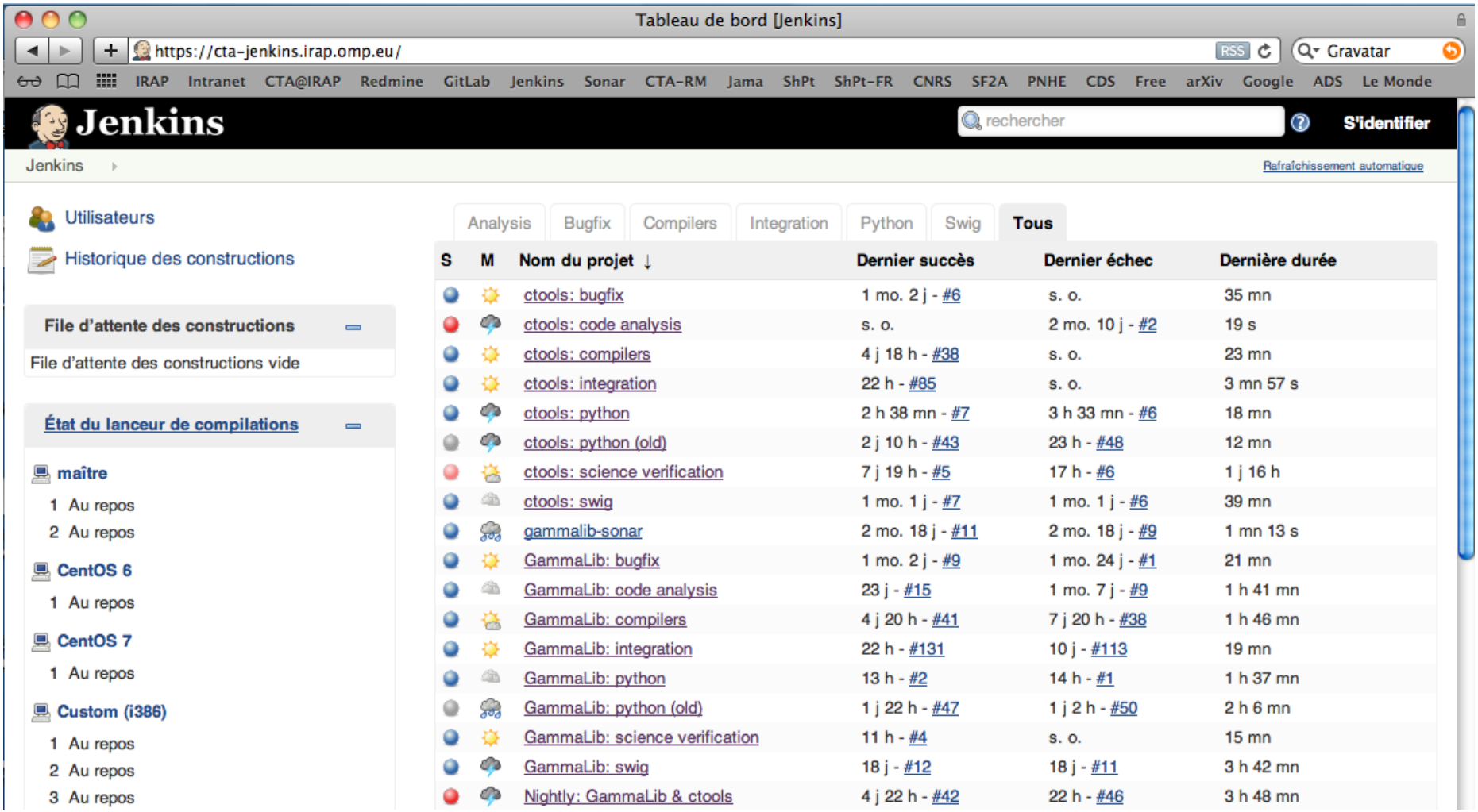
Avoid external dependencies (full control over the software)

Do not tie the software to a reference platform

Use continuous integration (building, testing, analysing) and modern code development tools (forge, version control)

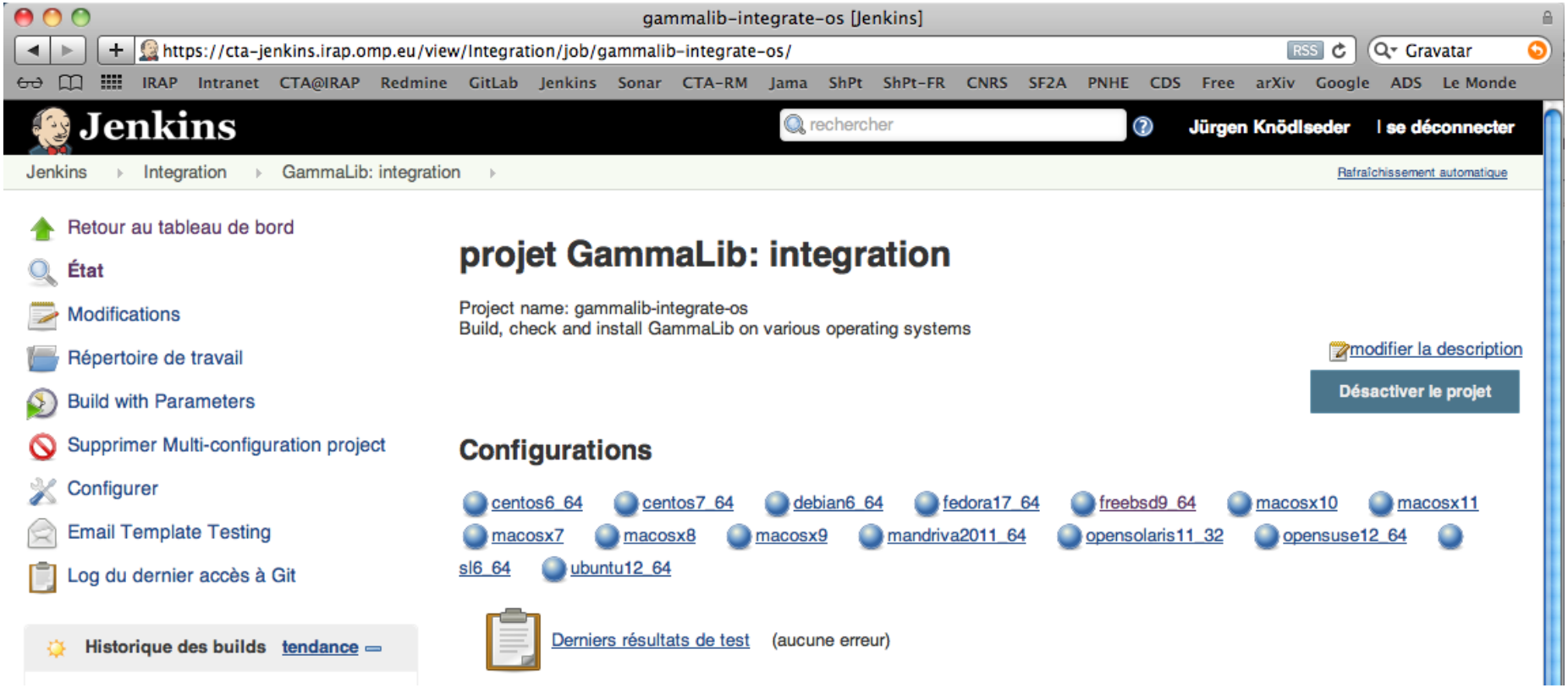
Continuous Integration

https://cta-jenkins.irap.omp.eu



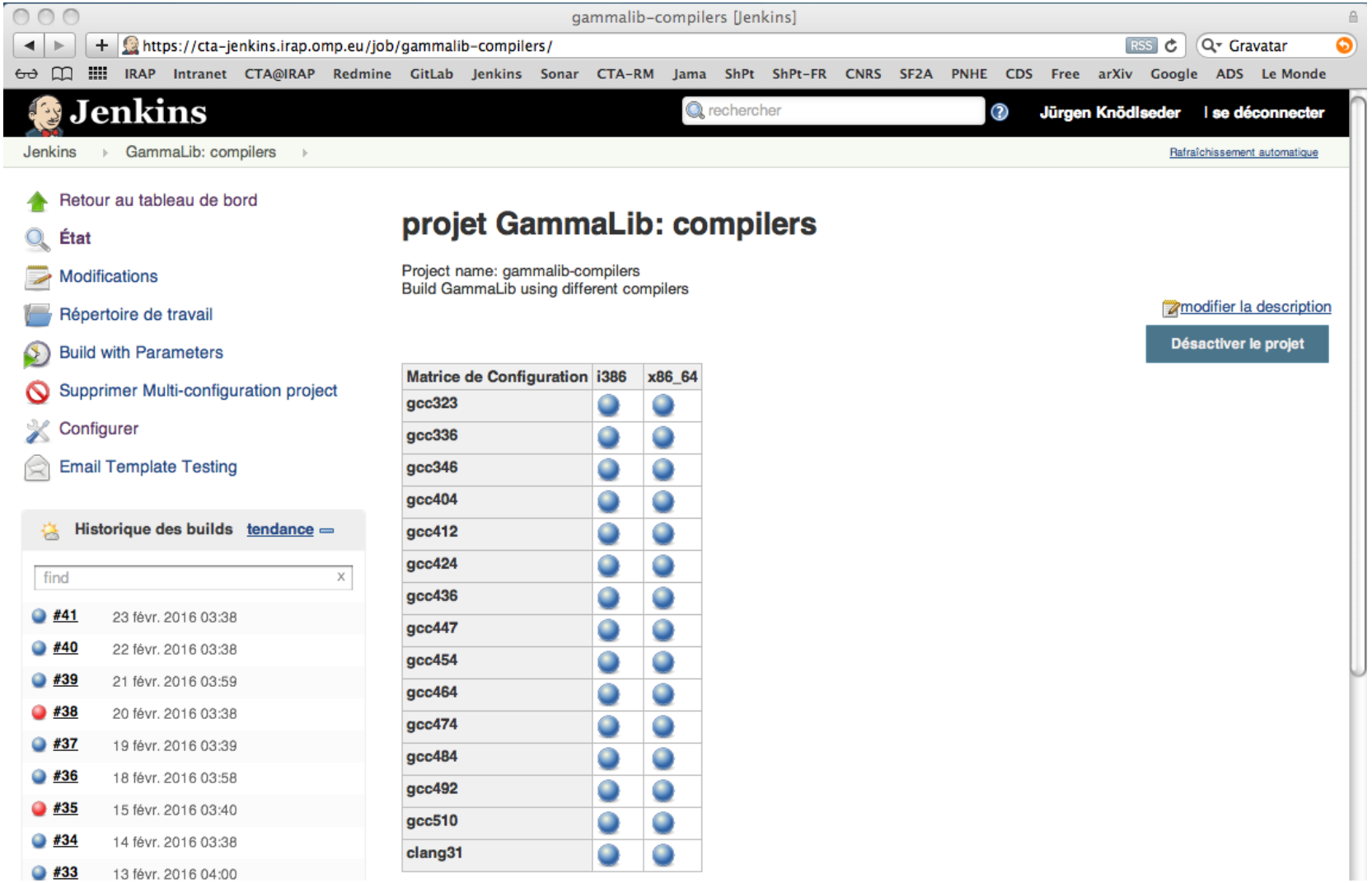
Continuous Integration

Multi-platform (Mac OS X, Linux, FreeBDS, OpenSolaris)



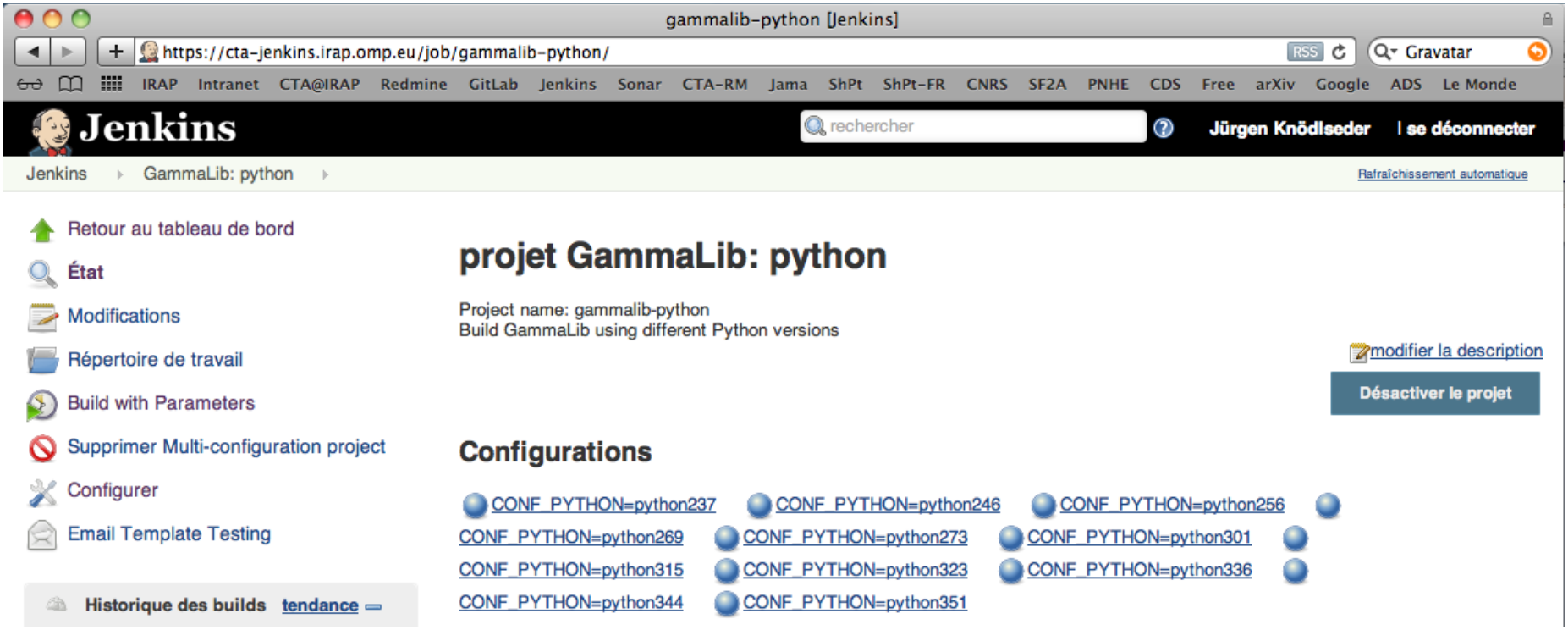
Continuous Integration

Compiler versions



Continuous Integration

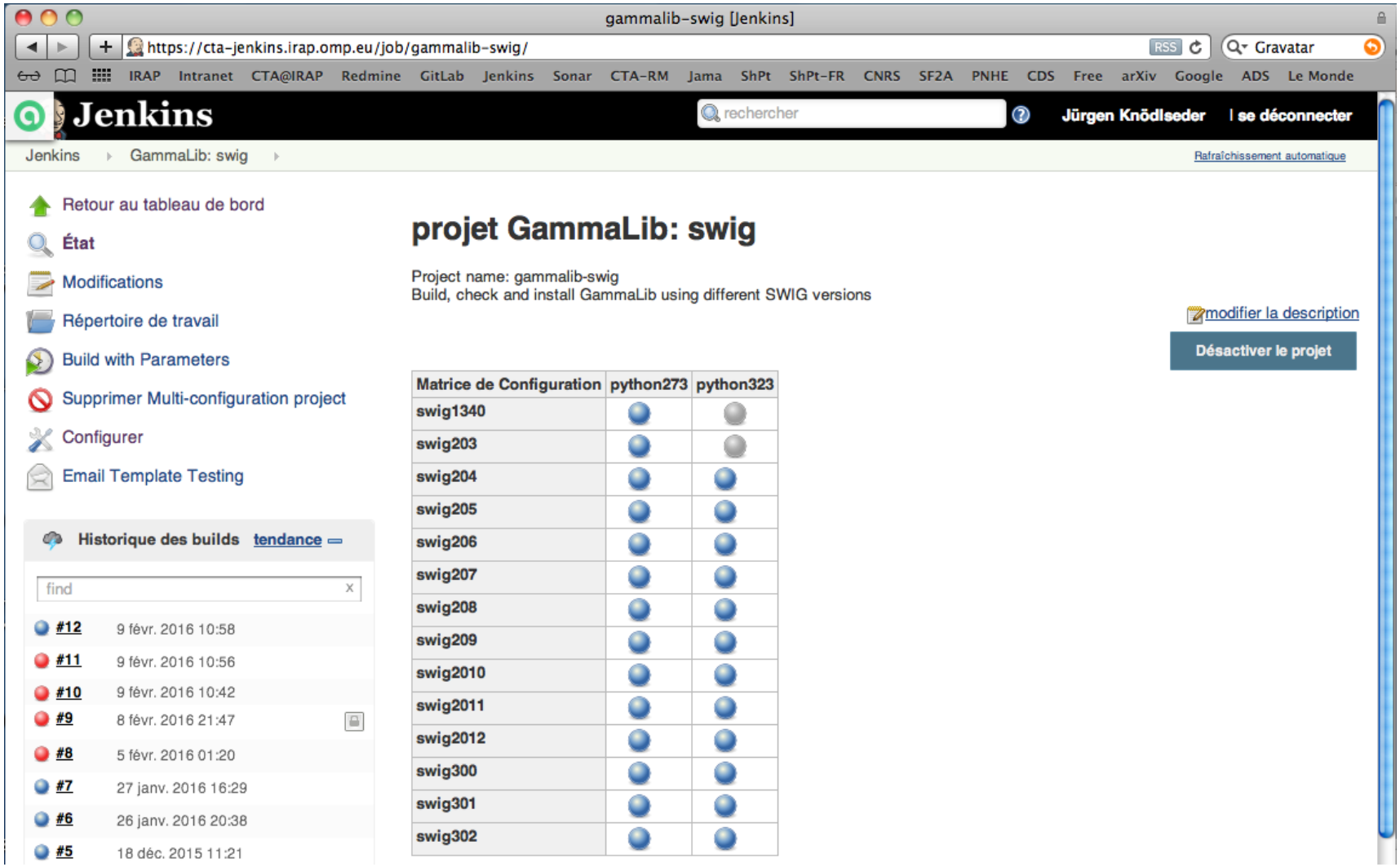
Python versions



Note that the code works with Python 2.3 – Python 3.5
(remember: we don't know the environment of the user)

Continuous Integration

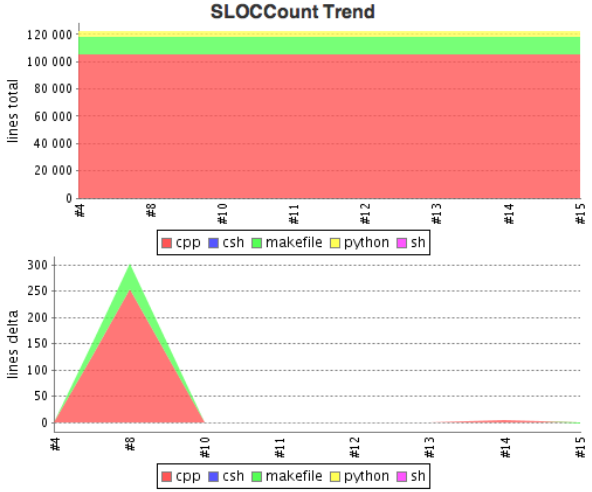
swig versions



Continuous Integration

static and dynamic code analysis

lines of code

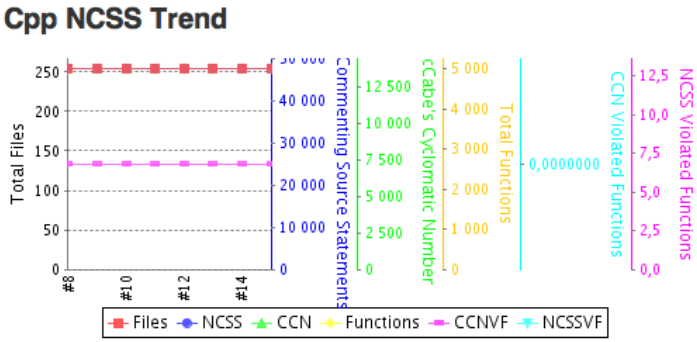


possible errors

Cppcheck Results

Severity	Count	Delta
Error	1	
Warning	0	
Style	103	
Performance	0	
Portability	0	
Information	0	
No category	0	
Total	104	

code complexity



test coverage

Project Coverage summary

Name	Packages	Files	Classes	Lines	Conditionals
Cobertura Coverage Report	100% 3/3	100% 286/286	100% 286/286	58% 28464/49334	40% 10626/26624

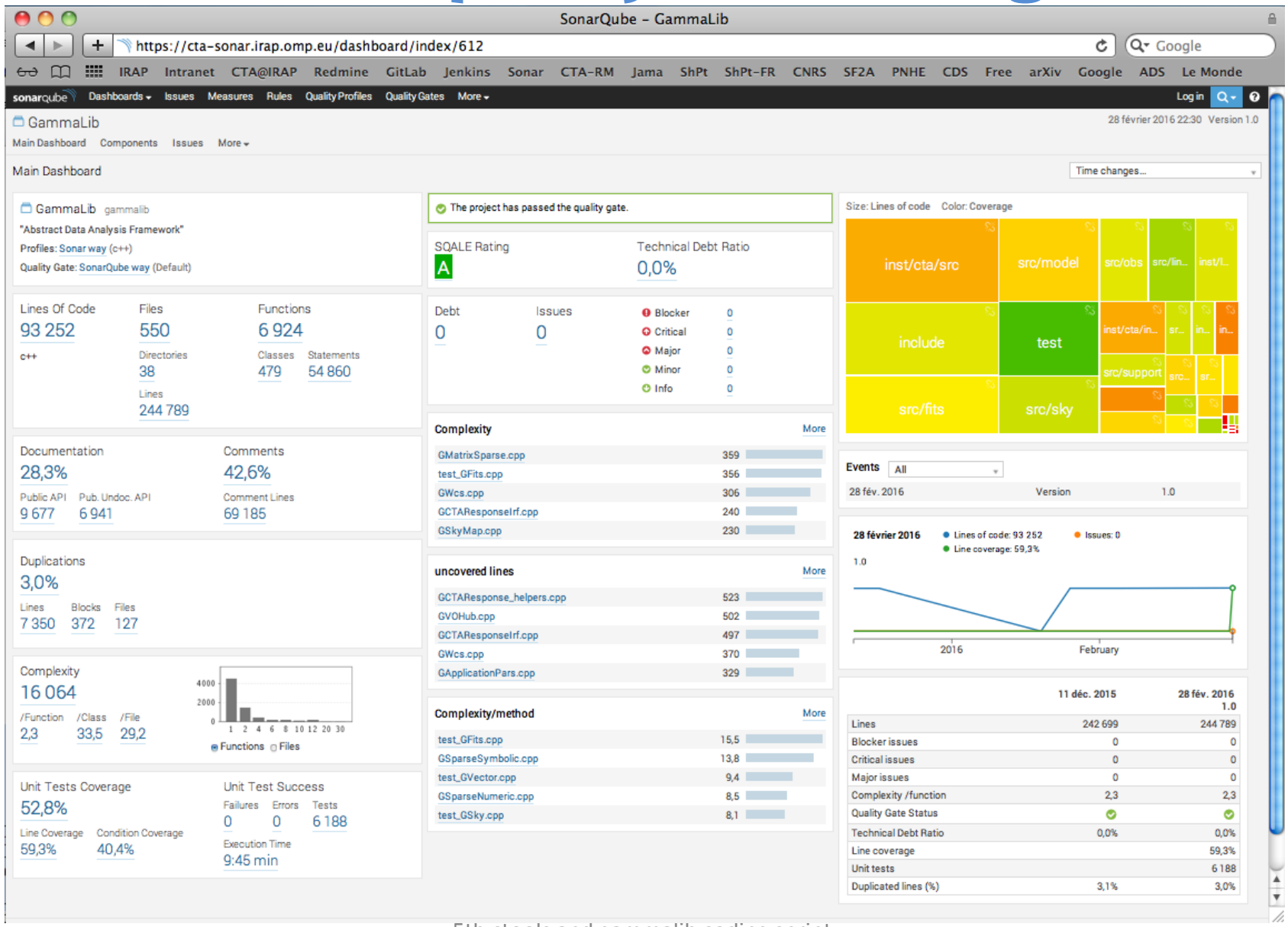
Coverage Breakdown by Package

Name	Files	Classes	Lines	Conditionals
<default>	100% 194/194	100% 194/194	63% 22298/35532	45% 8632/19362
src	100% 83/83	100% 83/83	44% 5909/13316	27% 1971/7184
testinst	100% 9/9	100% 9/9	53% 257/486	29% 23/78

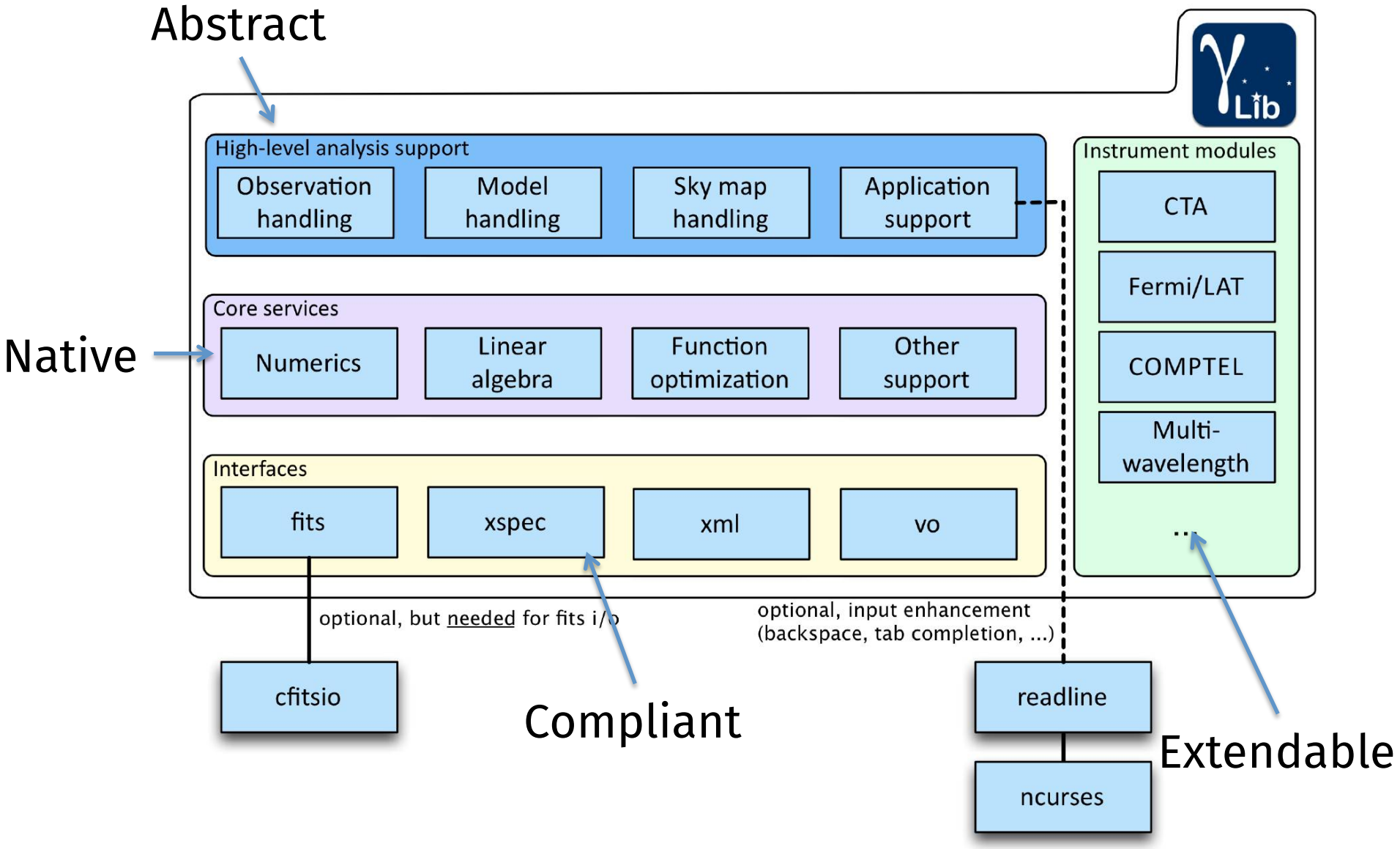
memory leaks

Valgrind Results

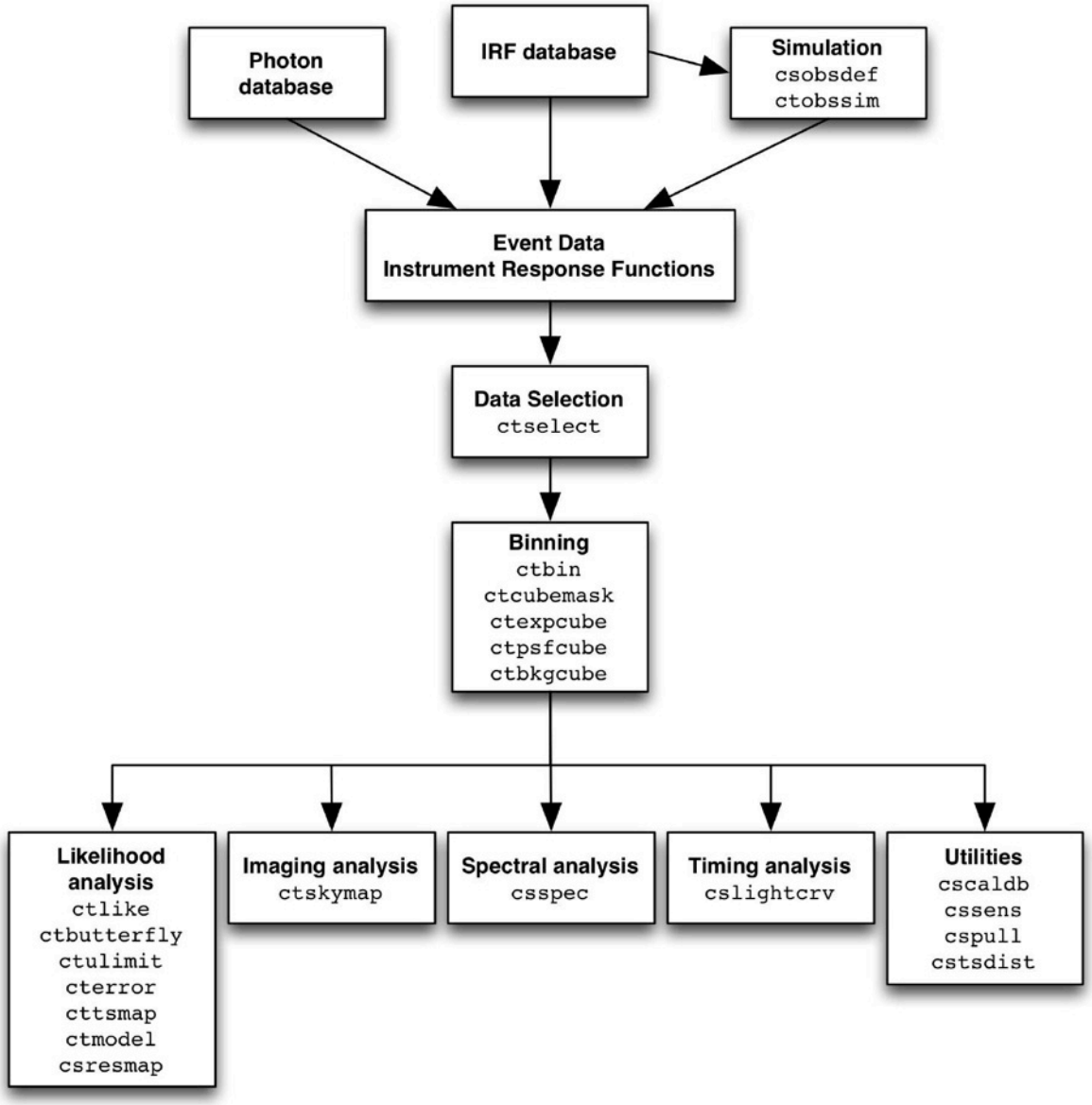
Code quality monitoring



GammaLib overview



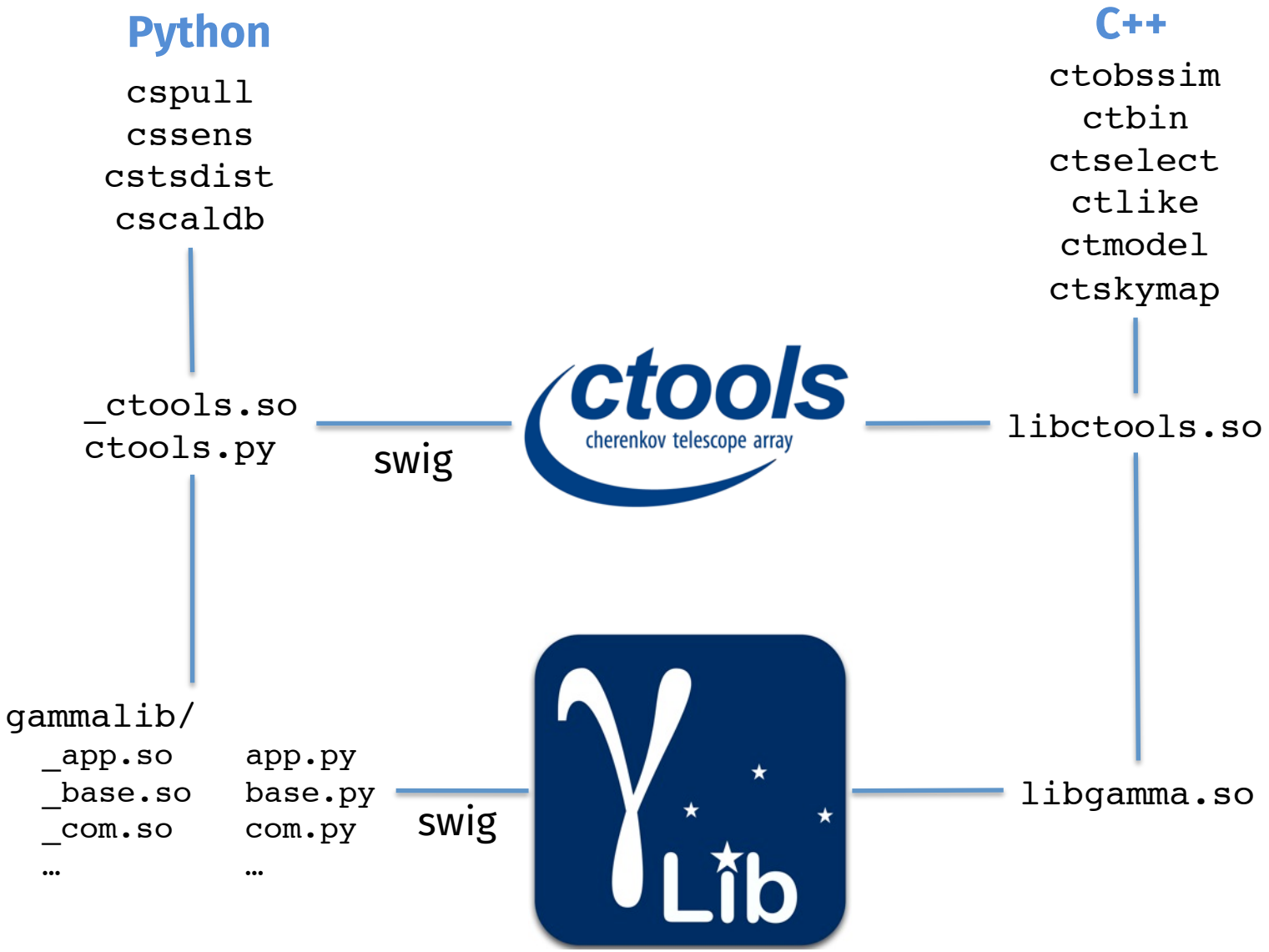
ctools overview



From http://www.swig.org/:

SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. SWIG is used with different types of target languages including common scripting languages such as Javascript, Perl, PHP, Python, Tcl and Ruby …

The overall picture



Getting informed

Sign-in in Redmine, watch anything you're interested in, participate in discussions

Send an e-mail to ctools-subscribe@irap.omp.eu to join the mailing list (news, release info, discussions)

Read the documentation (http://cta.irap.omp.eu/gammalib/ and http://cta.irap.omp.eu/ctools/)

Follow us on Twitter
@gammalib
@ctools_software
#ctools5

Documentation

Code documentation generated using Doxygen

User documentation written using Sphinx

Trunk (devel) documentation
http://cta.irap.omp.eu/gammalib-devel/
http://cta.irap.omp.eu/ctools-devel/

Release documentation
http://cta.irap.omp.eu/gammalib/
http://cta.irap.omp.eu/ctools/

Publications

Acknowledgment statement

This research made use of ctools, a community-developed analysis package for Imaging Air Cherenkov Telescope data. ctools is based on GammaLib, a community-developed toolbox for the high-level analysis of astronomical gamma-ray data.

You can cite GammaLib & ctools, it's in the Astrophysics Source Code Library (ASCL) that is indexed by ADS (and soon also Web of Science)

GammaLib: ascl:1110.007

ctools: ascl:1601.005

Release 1.0 paper in advanced stage (all committers are co-authors)

GammaLib and ctools

A software framework for the analysis of astronomical gamma-ray data

J. Knödlseder[1], M. Mayer[2], C. Deil[3], J.-B. Cayrou[1], E. Owen[3], N. Kelley-Hoskins[4], C.-C. Lu[3], R. Buehler[4], F. Forest[1], T. Louge[1], H. Siejkowski[5] K. Kosack[6], L. Gerard[4], A. Schulz[4], P. Martin[1], D. Sanchez[7], S. Ohm[4], T. Hassan[8], and S.Brau-Nogué[1]

2. New features since last coding sprint

New GammaLib classes
New cscripts
--help option
User defined HDU names
Virtual Observatory support

New GammaLib classes

GFilename

- Replaces now all string arguments for filenames
- Handles transparently FITS extensions
- Includes methods to check whether a file exists or whether it is a FITS file

CCTAModelAeffBackground

- Uses CTA effective area as background template

New cscripts

csfindobs
csiactcopy
csiactdata
csiactobs
csobs2caldb

} existing IACT analysis

csobsinfo
csmodelinfo
csmodelmerge

} support scripts

Current IACTs analysis support

csiactcopy

Download IACT data from a remote machine

csiactdata

Explore local IACT database

csfindobs

Creates a run list based on user criteria (pointing direction, zenith angle, etc.)

csiactobs

Creates an observation definition XML file and a model definition XML file containing background models based on a run list

csobs2caldb

Creates a CALDB entry for an observation container (useful for simulations for current IACTs)

Support scripts

csobsinfo

Dumps content of an observation definition XML file

csmodelinfo

Dumps content of a model definition XML file

csmodelmerge

Merge several model definition XML file

Read more on
http://cta.irap.omp.eu/ctools-devel/reference_manual/

--help option

All tools and scripts now accept the –help option to display the reference manual text

```
$ ctobssim --help

ctobssim
========

Simulate event list(s).


Synopsis
--------

This tool simulates event list(s) using the instrument characteristics
specified by the instrument response function(s) and an input model. The
simulation includes photon events from astrophysical sources and background
events from an instrumental background model.

By default, ctobssim creates a single event list. ctobssim queries a
pointing direction, the radius of the simulation region, a time interval,
an energy interval, an instrumental response function, and an input model.
ctobssim uses a numerical random number generator for the simulations with
a seed value provided by the hidden seed parameter. Changing this parameter
```

User defined HDU names

Previously the FITS extension names were hard coded (e.g. “GTI” for Good Time Intervals, “EVENTS” for CTA events list, etc.). Now the user can specify the FITS extension name in the filename

```
gti.save("gti.fits[Good Time Interval]", true)
obs.save("cta_events.fits[EVENTS3;GTI3]", true)
```

Syntax to specify events and GTI HDU name

Virtual Observatory Support

From http://www.ivoa.net/

The Virtual Observatory (VO) is the vision that astronomical datasets and other resources should work as a seamless whole

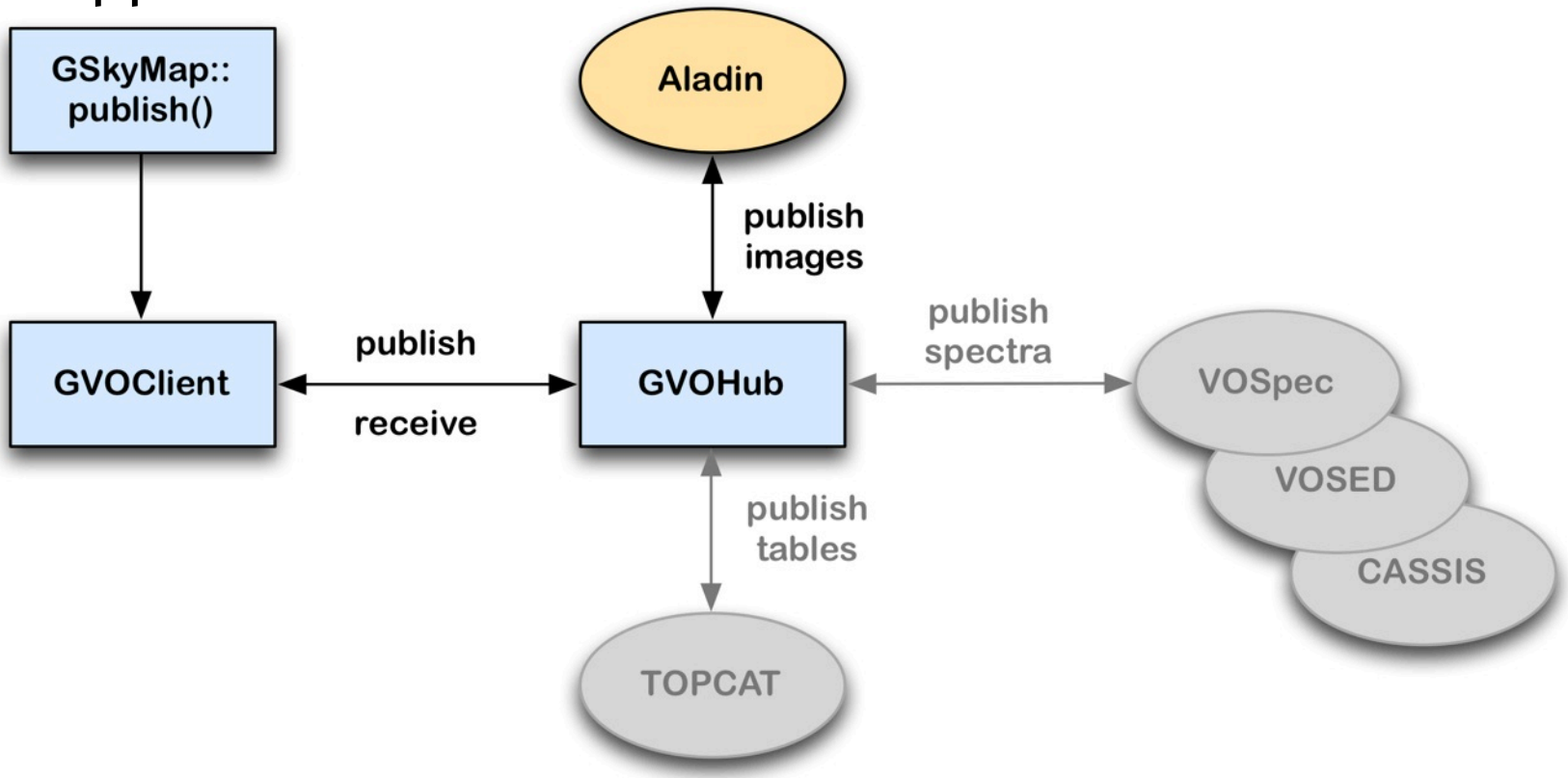
The VO is not a concrete thing, like a data warehouse. Rather, it is more like an ecosystem of mutually compatible datasets, resources, services, and software tools which use a common set of technologies and a common set of standards. The idea is to make all these things *inter-operable* - i.e. to make them work nicely together.

From myself

The VO is great if you are not aware that you are using it

Virtual Observatory Support

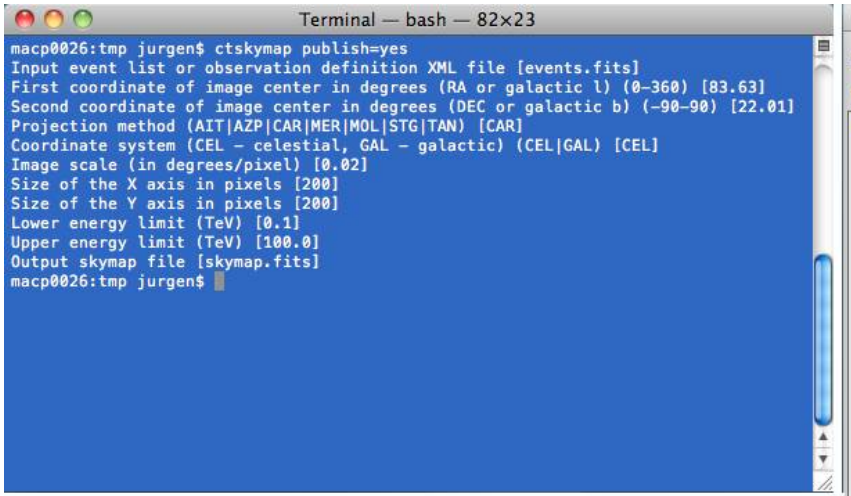
VO support in a nutshell

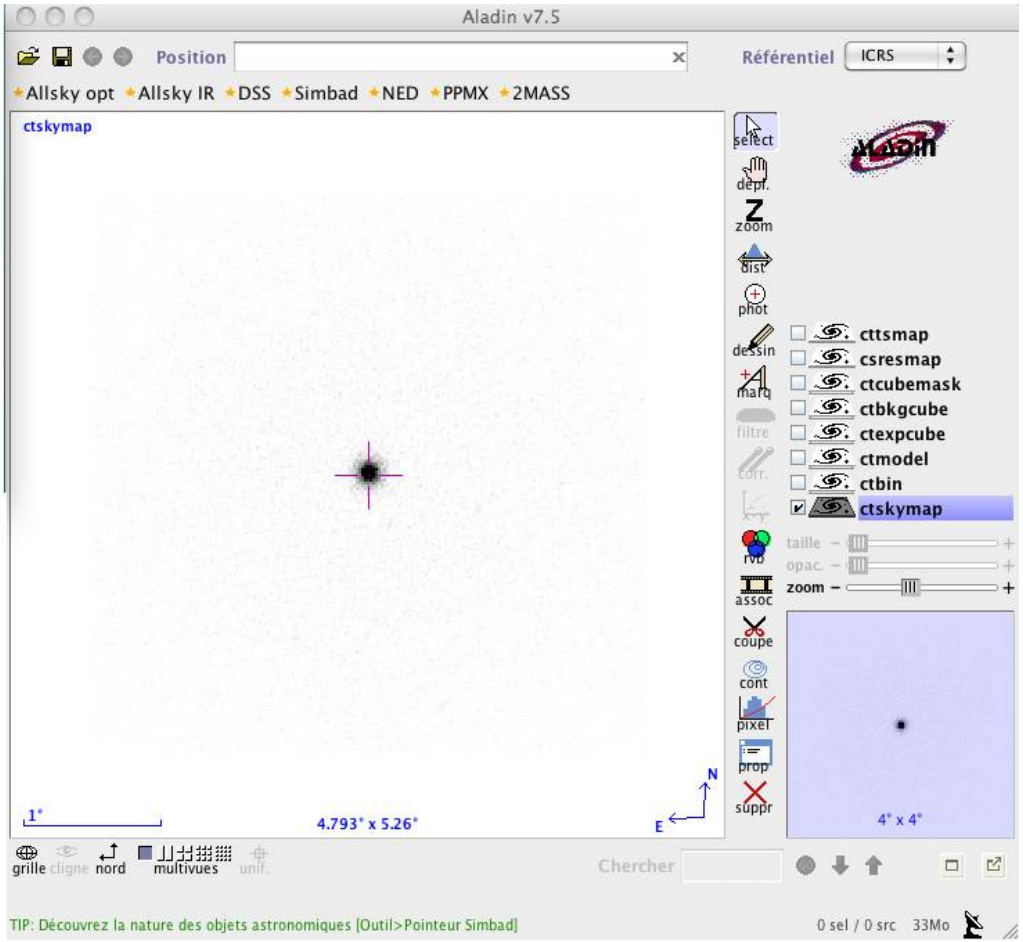


All ctools generating images have now a hidden publish parameter, e.g.

```
$ ctskymap publish=yes
```

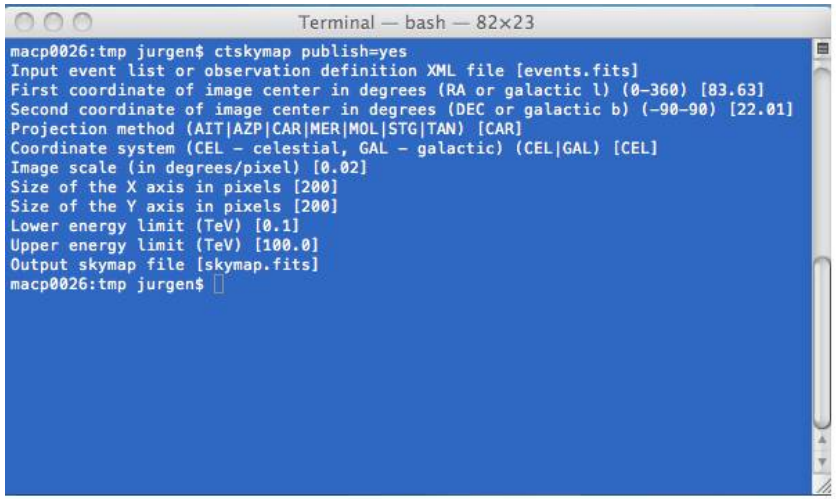
Virtual Observatory Support

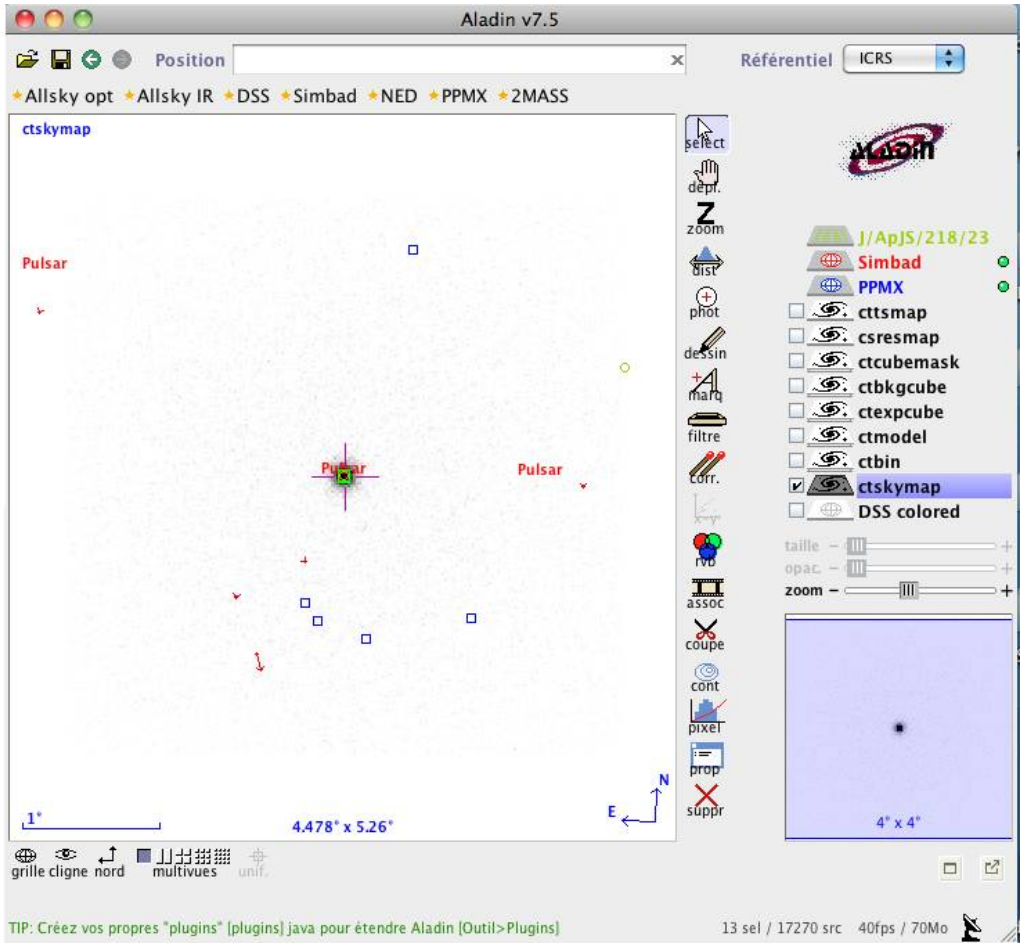




ctskymap sends a sky map via the Hub to Aladin (open Aladin before sending the sky map)

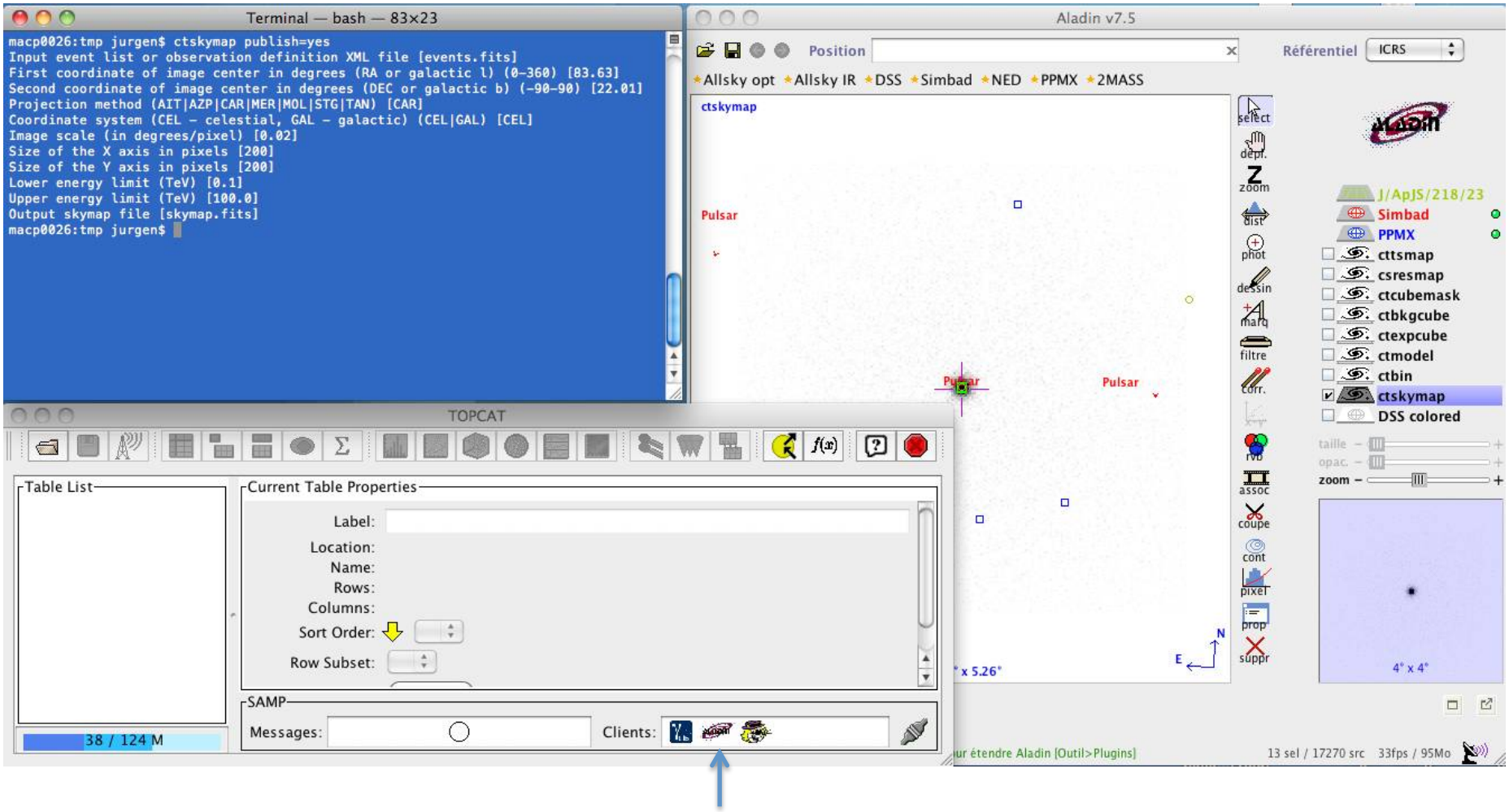
Virtual Observatory Support





Aladin can be used to search for catalogues, sky maps obtained at other wavelengths, etc.

Virtual Observatory Support



TOPCAT shows who's connected to the Hub (GVOHub)

Virtual Observatory Support

Next steps

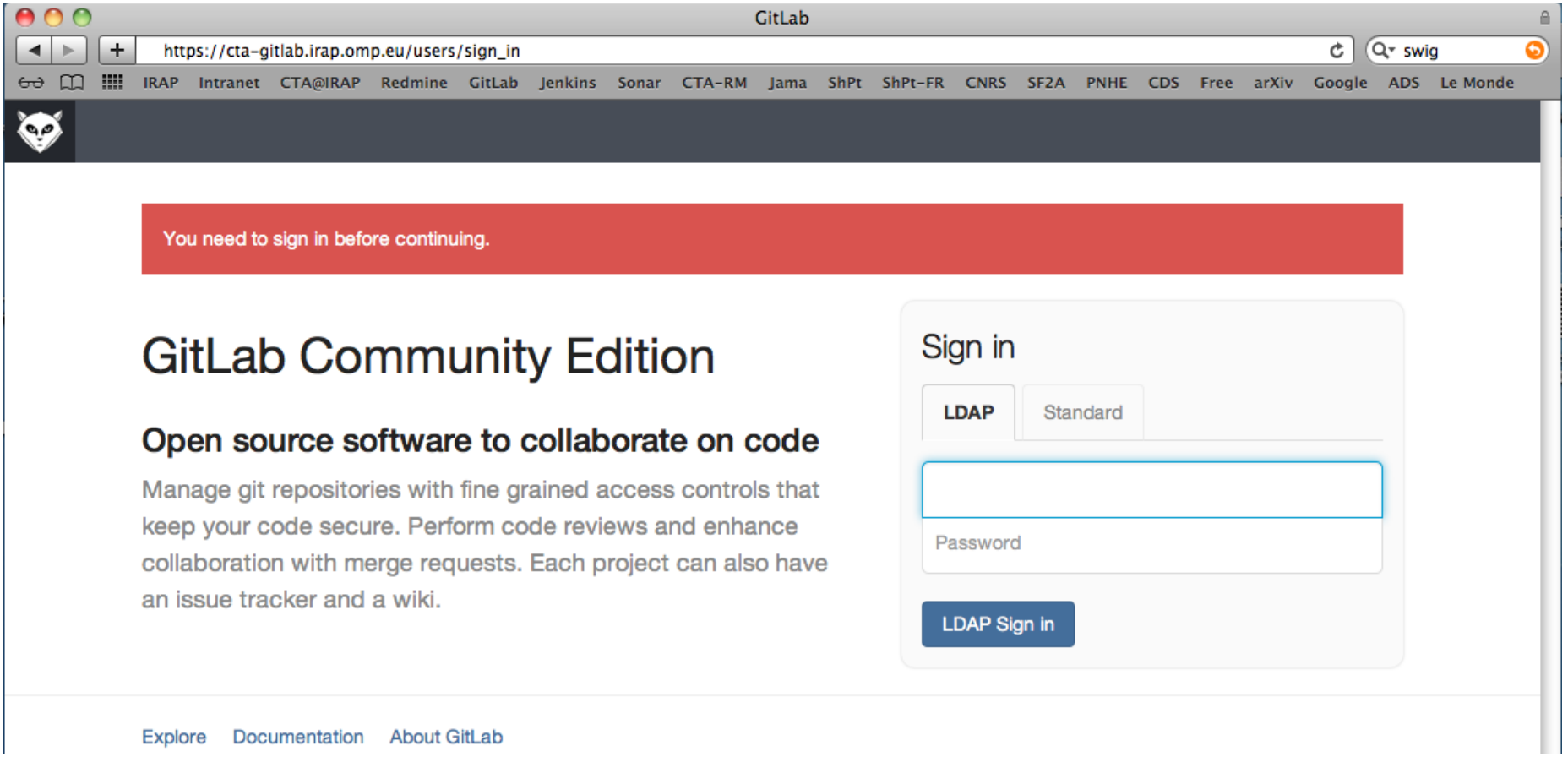
publishing of spectra
publishing of table (e.g. events lists)
feedback of VO tools to ctools (e.g. source selection)

Note: VO support is still exploratory, but you can now try it out and provide your feedback

3. New Development Workflow

GitLab

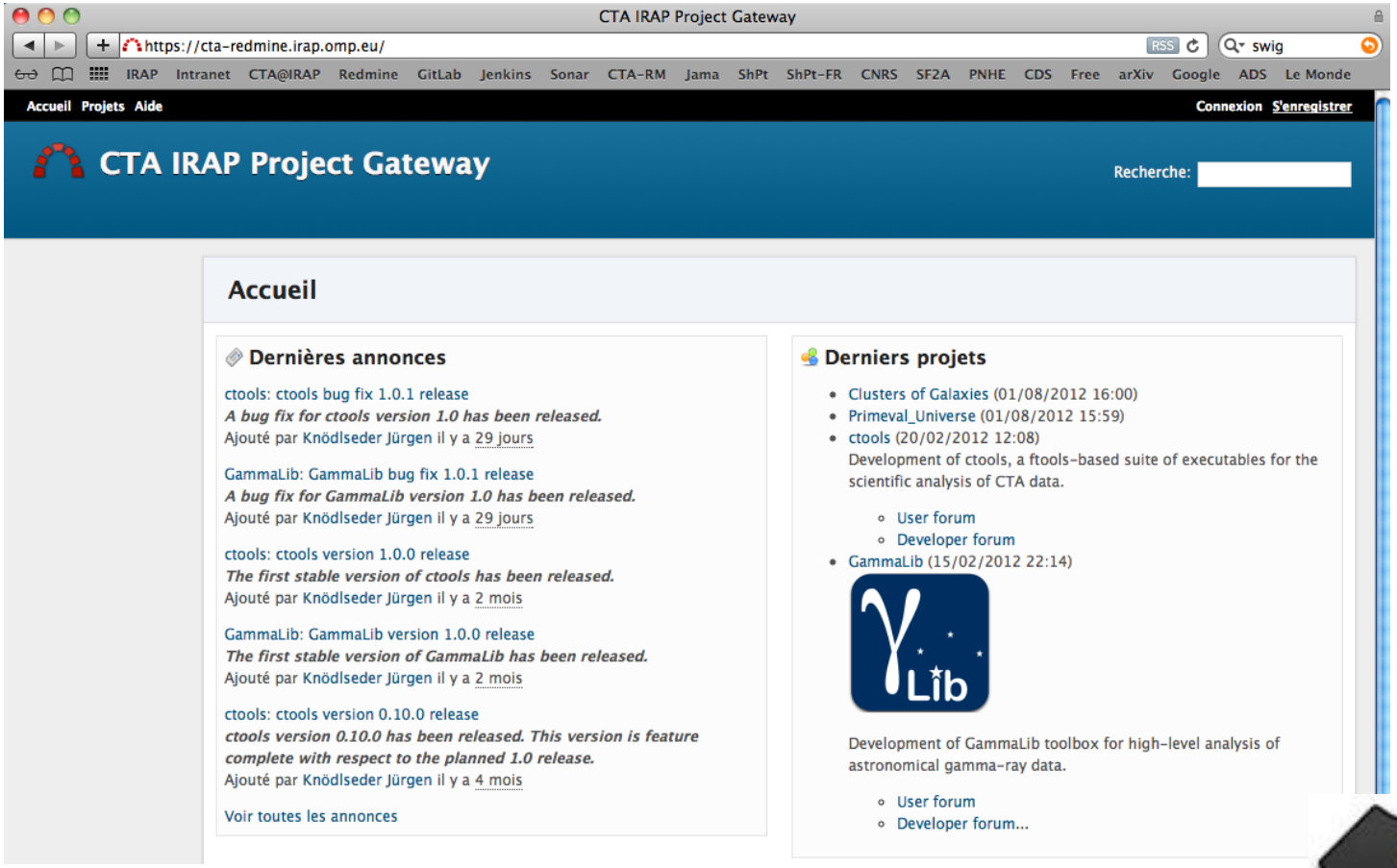
https://cta-gitlab.irap.omp.eu



A web based Git front-end (à la GitHub)

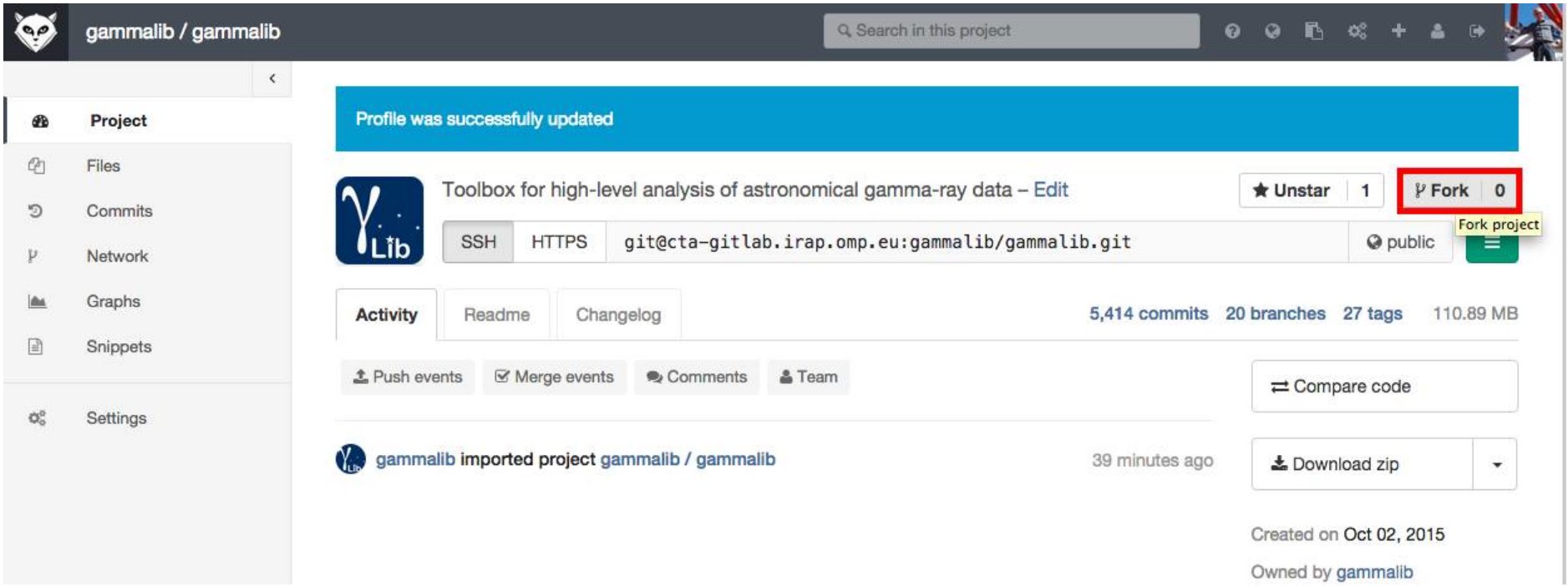
Redmine

https://cta-redmine.irap.omp.eu



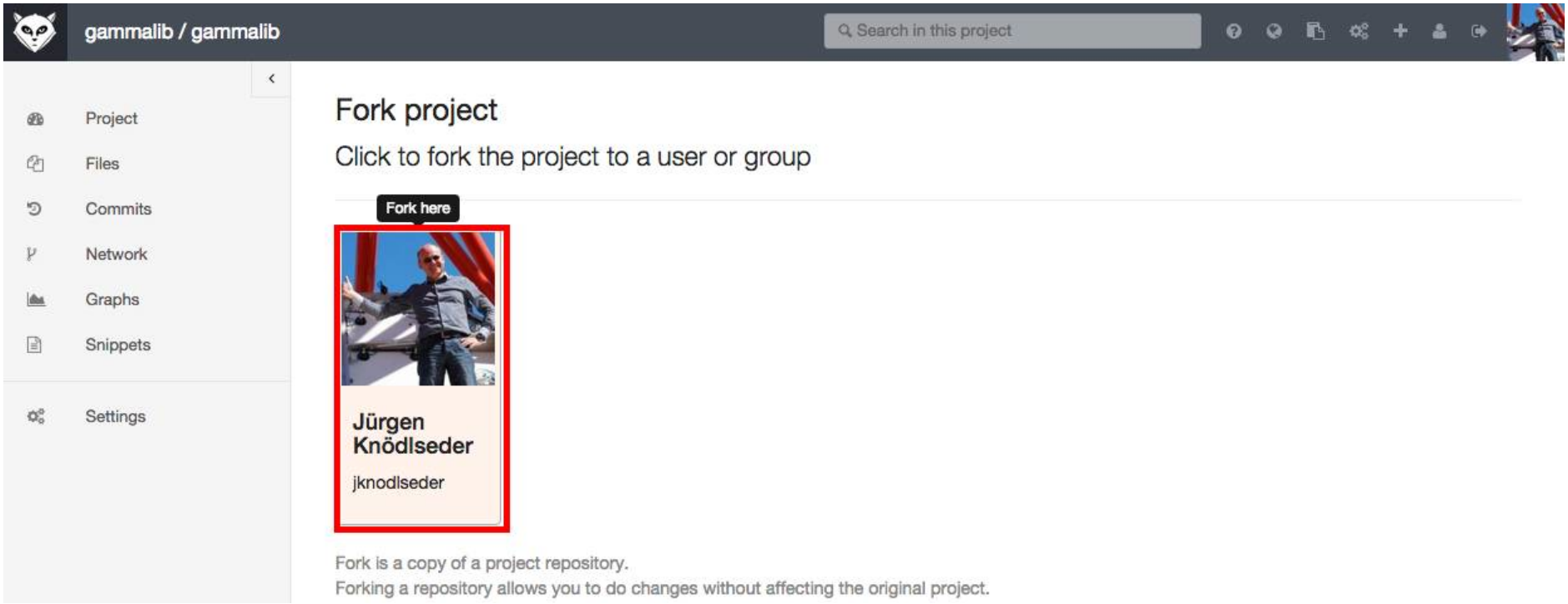
Handles account creation (also for GitLab)

Creating a fork on GitLab



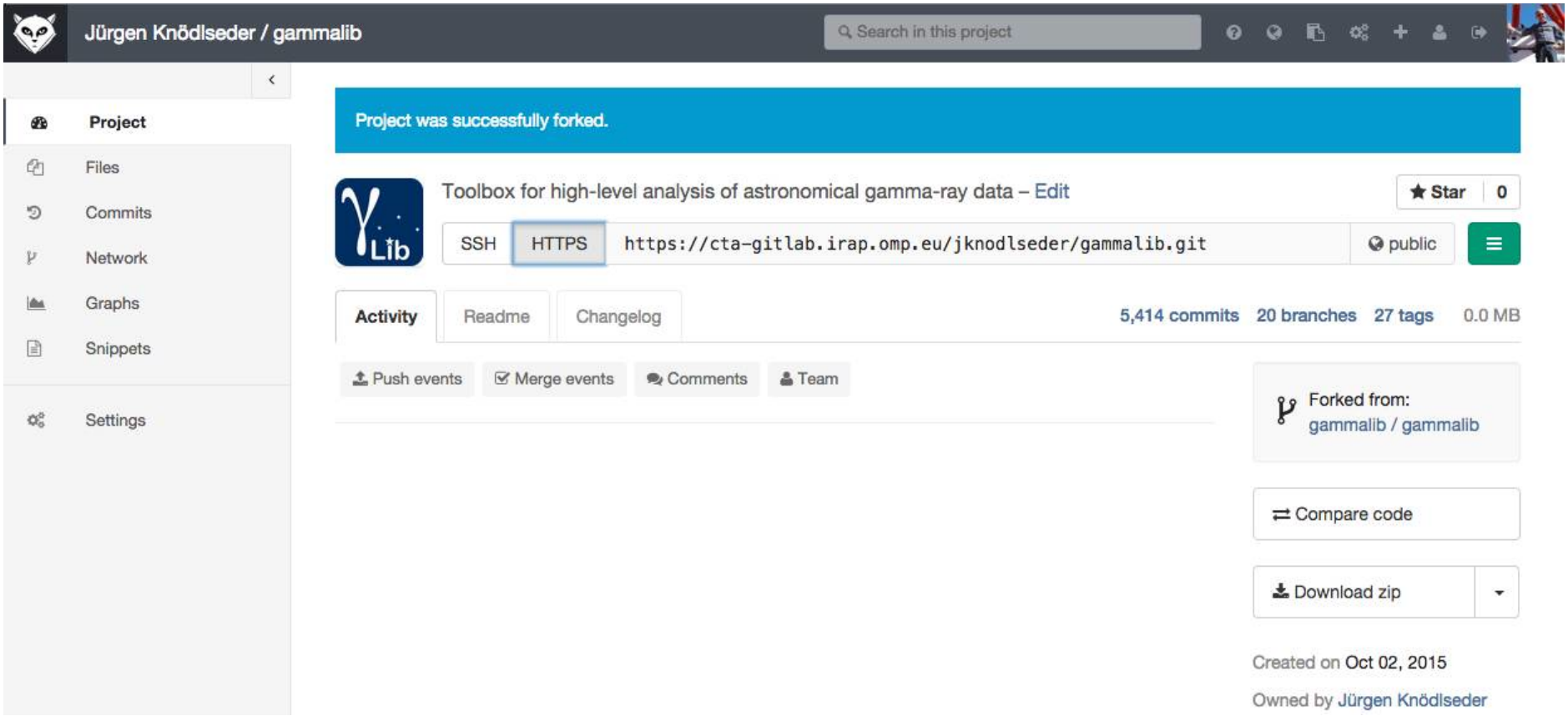
Connect to GitLab with your Redmine user name and password and select the GammaLib or ctools project and click on “Fork”

Creating a fork on GitLab



Now click on your user to fork (copy) the project into your user space (will result in <user>/gammalib)

Creating a fork on GitLab



Now you have a copy of the project under your user on GitLab

Working with GitLab

Make sure you can access https

$ export GIT_SSL_NO_VERIFY=true

or

$ git config --global http.sslverify "false"

Get a clone of the code on your machine

$ git clone https://cta-gitlab.irap.omp.eu/jknodlseder/gammalib.git

$ git init

Add main gammalib (or ctools) repository as remote repository to fetch any changes before starting to work on a new feature

$ git remote add upstream https://cta-gitlab.irap.omp.eu/gammalib/gammalib.git

$ git pull upstream devel

Contributing code

Create a new branch. Always pull upstream changes first. Always put the Redmine issue number at the beginning of your branch.

```
$ git pull upstream devel
$ git checkout -b 9101-correct-nasty-bug
```

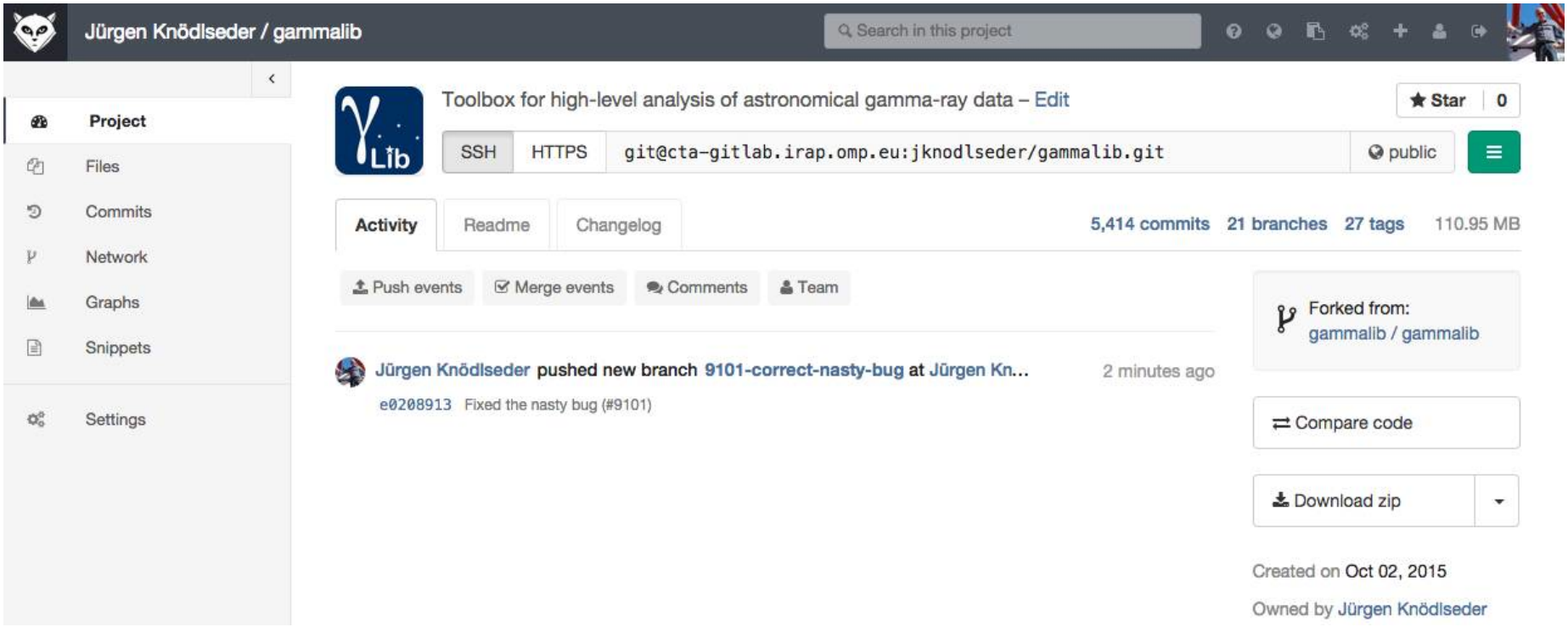
Code, stage and commit

```
$ git add inst/cta/src/GCTAModelIrfBackground.cpp
$ git commit -m "Fixed the nasty bug (#9101)”
```

Push commit(s) into your GitLab repository

```
$ git push origin 9101-correct-nasty-bug
```

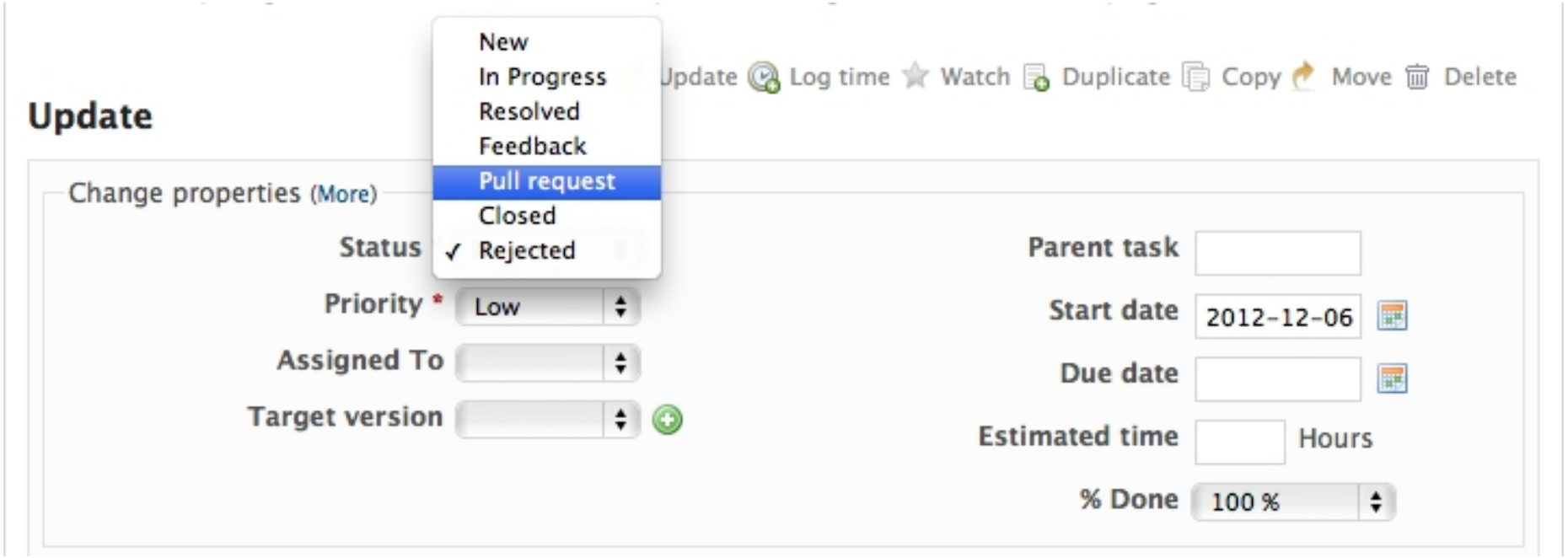
Contributing code



You should see your commit now on GitLab …

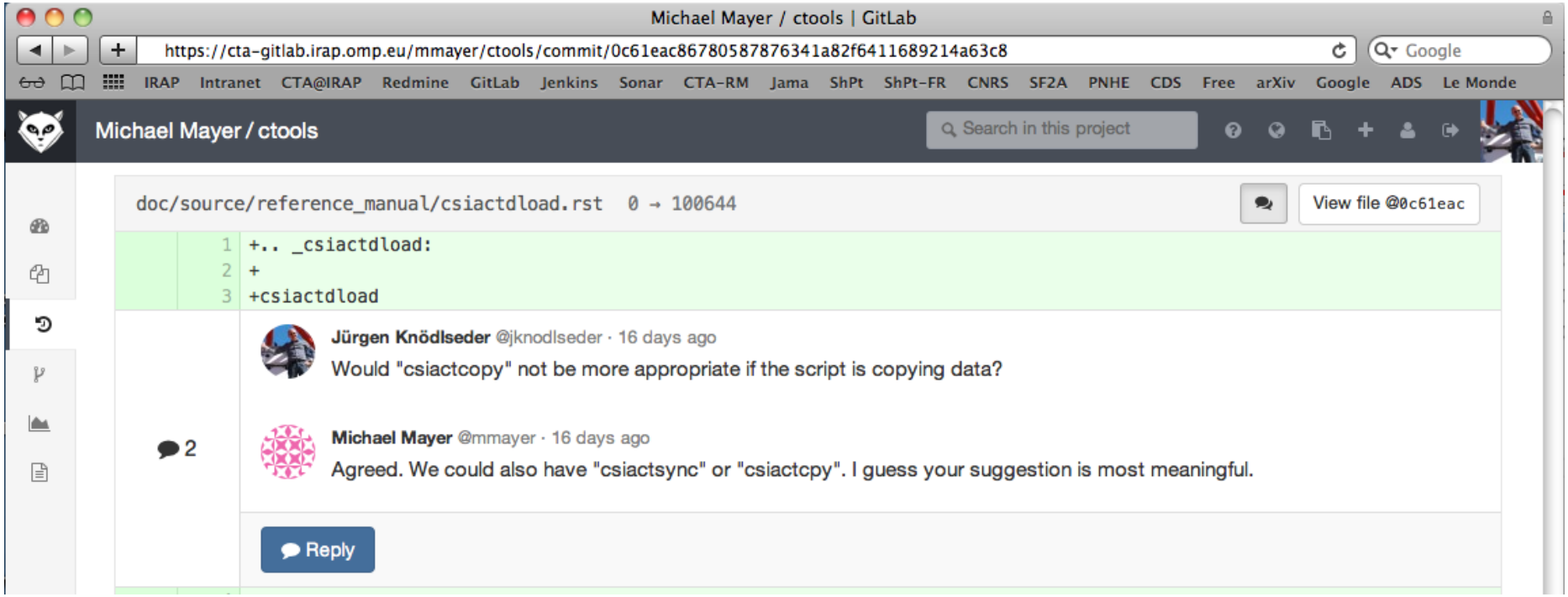
Now go to the Redmine issue to tell me that there is something to merge into the trunk …

Contributing code



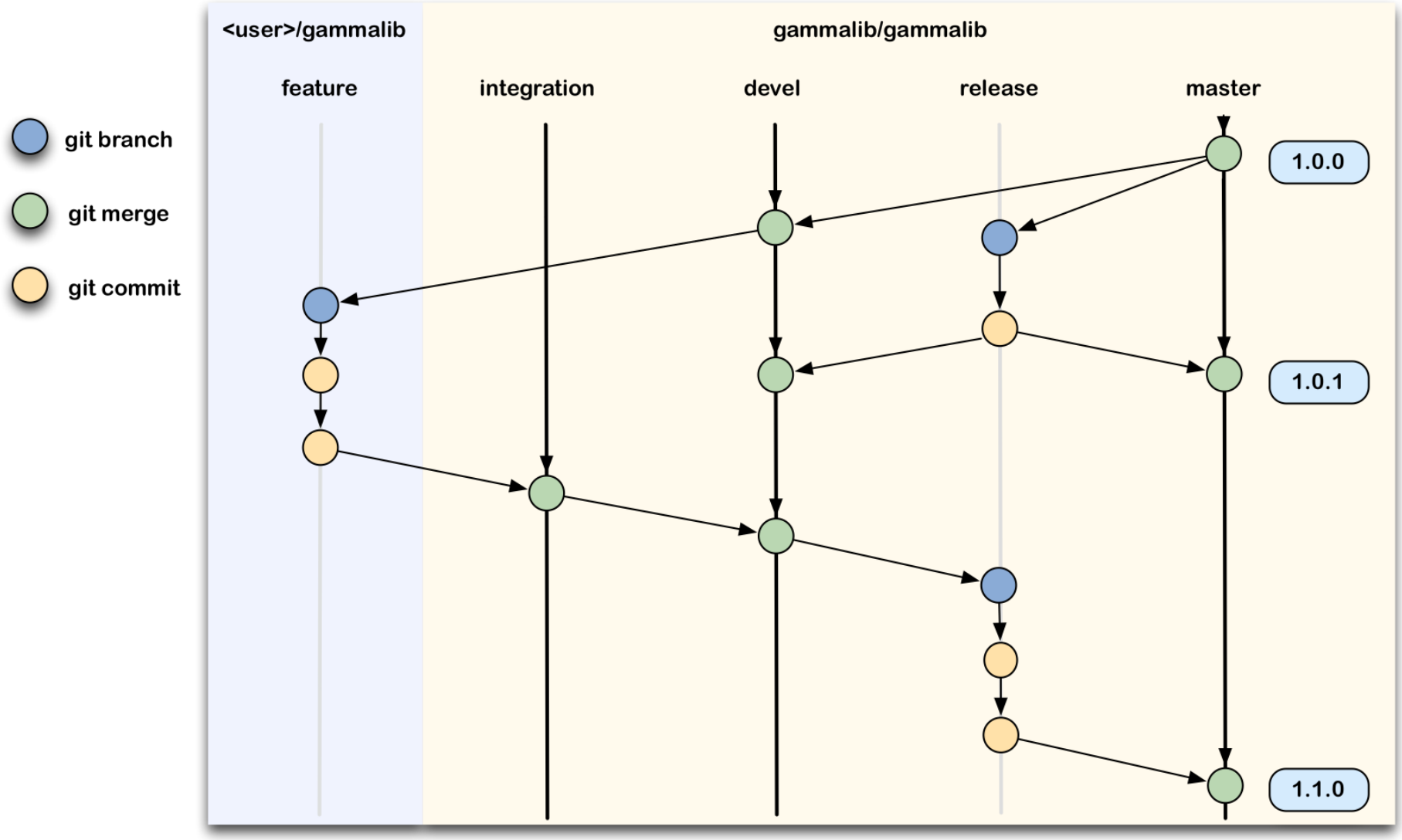
Please always explain in the Redmine issue what you did and what the name of the branch is that you want to get merged

Code review on GitLab



Once you issued a pull request, I will inspect your code on GitLab and eventually make some comments (you should get an e-mail notification of that). You can of course reply and engage a discussion ☺

Git workflow



4. Goals of this sprint

Goals of this sprint

Collection of issues to be addressed during the sprint

I hope by the time of the coding sprint we have a good draft of our release paper ready. We may work on analysis examples for the release paper. But we can also address new features needed and discuss the next steps.

Just list below what you would like to do during the sprint:

- Have fun
- Implement analysis workflows (#1508, see also https://cta-redmine.irap.omp.eu/boards/14/topics/237)
- Python Function to Convert GObservations to GCTABackground3D (#1530)
- Implement a PSF table format
- Finalize the classical analysis (I know, we have this pending since a long time, but we should terminate the work which is almost done)
- Implement smoothing, oversampling and denoising of images (and skymaps), mentioned in (#1530)
- tool to compute systematic errors (#1712)
- Finish a GModelSpatialRadial for dark matter halos (#1520)
- ...

Workflow implementation

To make a proof of principle for an XML-driven analysis workflow I pushed a branch `1508-implement-workflow` into the ctools repository. It contains a script `csworkflow.py` to execute a workflow described by the `test/data/workflow.xml` file.

The format of the XML file is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<workflow>
    <actor name="input">
        <output>
            <parameter name="inmodel" value="$CTOOLS/share/models/crab.xml" />
            <parameter name="caldb"   value="prod2" />
            <parameter name="irf"     value="South_50h" />
            <parameter name="edisp"   value="no" />
            <parameter name="deadc"   value="0.95" />
            <parameter name="ra"      value="83.63" />
            <parameter name="dec"     value="22.01" />
            <parameter name="target"  value="Crab" />
            <parameter name="emin"    value="0.1" />
            <parameter name="emax"    value="100.0" />
            <parameter name="tmin"    value="0.0" />
            <parameter name="tmax"    value="1800.0" />
            <parameter name="chatter" value="2" />
            <parameter name="clobber" value="yes" />
            <parameter name="debug"   value="no" />
            <parameter name="mode"    value="ql" />
        </output>
    </actor>
    <actor name="ctobssim" tool="ctobssim">
        <input>
            <parameter name="inobs"   value="NONE" />
            <parameter name="inmodel" value="inmodel" actor="input" />
            <parameter name="caldb"   value="caldb" actor="input" />
            <parameter name="irf"     value="irf"   actor="input" />
            <parameter name="edisp"   value="edisp" actor="input" />
            <parameter name="prefix"  value="sim_events_" />
            <parameter name="seed"    value="1" />
            ...
```

The file contains a single `<workflow>` element that is composed by a number of `<actor>` elements. Each `<actor>` has optional `<input>` and `<output>` elements. The syntax of using an output element of an actor as input of another actor is:

```
<parameter name="inmodel" value="inmodel" actor="input" />
```

where `value` is the name of the parameter of the actor, and `actor` is the name of the actor.

Agenda

Tentative agenda

- Monday, 29 February:
  - 14:00 – 16:00: Introduction, meeting goal, status of CTA developments & analysis (Jürgen)
  - 16:00 – 16:30: HESS analysis progress report (Michael)
  - 16:30 – 17:00: VERITAS analysis progress report (Nathan)
  - 17:00 – 18:00: Slots for more progress reports, analysis results, etc. (just enter your proposal)
- Tuesday, 1 March:
  - 9:00–18:00: Coding, Testing, Documenting
- Wednesday, 2 March:
  - 9:00–18:00: Coding, Testing, Documenting
- Thursday, 3 March:
  - 9:00–18:00: Coding, Testing, Documenting
- Friday, 4 March:
  - 9:00 – 12:00: Sprint wrap up

What day is best to organise a social dinner (add your wishes or thoughts)?