The certification process of the LHCb distributed computing software

F. Stagni, C. Haen

on behalf of the LHCbDIRAC team

Today's agenda

- What's DIRAC
- What's LHCbDIRAC
- How we do QA for LHCbDIRAC
  - And when

What's DIRAC

diracgrid.org github.com/DIRACGrid

"A software framework for distributed computing"

- open source
- used by 20+ VOs
- developed mostly by LHCb
- python 2
- ~200K lines only its core
- has few extensions, e.g. WebDIRAC
- 2 or 3 releases per year

What's LHCbDIRAC

lhcbdirac svn/lhcbdirac

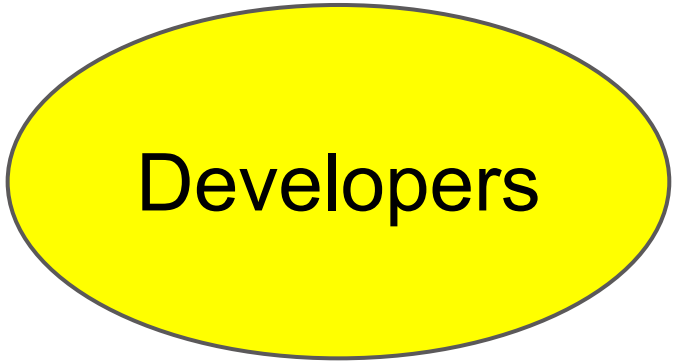
"The LHCb extension of DIRAC"

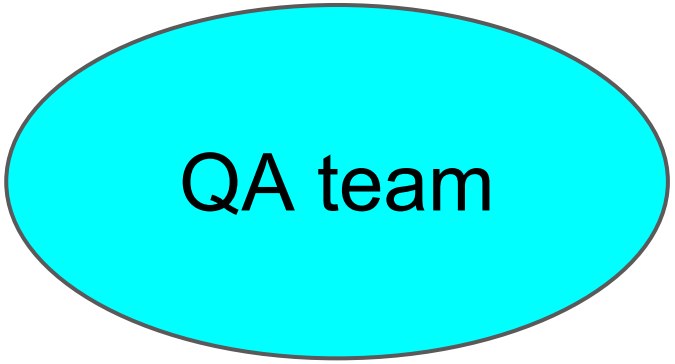
- ~120K lines of code
- everything that is LHCb specific
- each release has a strict dependency from a DIRAC release
- LHCbWebDIRAC extends WebDIRAC

The LHCbDIRAC dev team

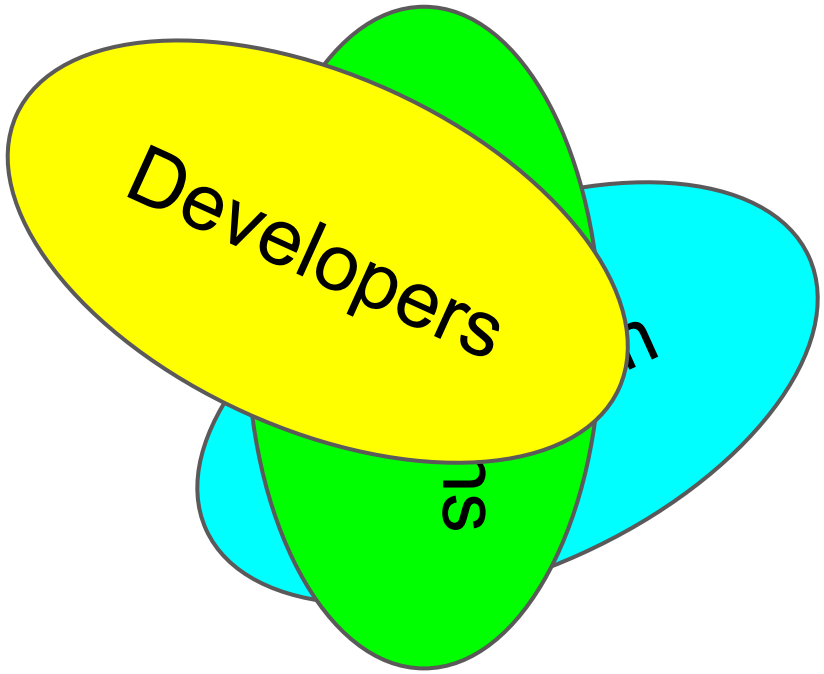
- ~12 FTE, high variance
- Mostly based at CERN
  - but not all
- Many are developers for a fraction of their time
- We all develop for DIRAC and for LHCbDIRAC
  - And for WebDIRAC, LHCbWebDIRAC, ...
- SCRUM is almost impossible
  - we evaluated it, then gave up

Theory





Practice



QA

- For us it mostly means releasing without fear
- We also have coding guidelines
  - And we use static code analyzers
- We don’t aim at “industry standards”

So, how to test it all?

There are some approaches to consider

One approach

This practice has actually a name: “**TiP**”

And it is often combined with “Exposure control”, i.e. exposing new features to few Guinea pigs

TiP

TiP has actually been the ~~main~~ only way for testing LHCbDIRAC for years!

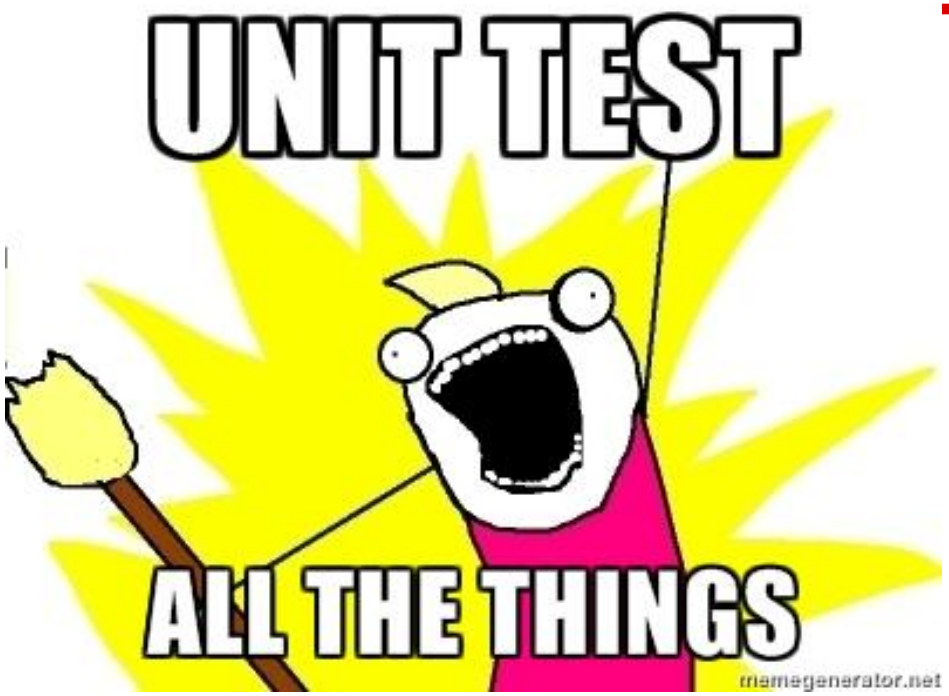
Pros:

- No need for QA team!
- Easy way to test your luck factor

Cons:

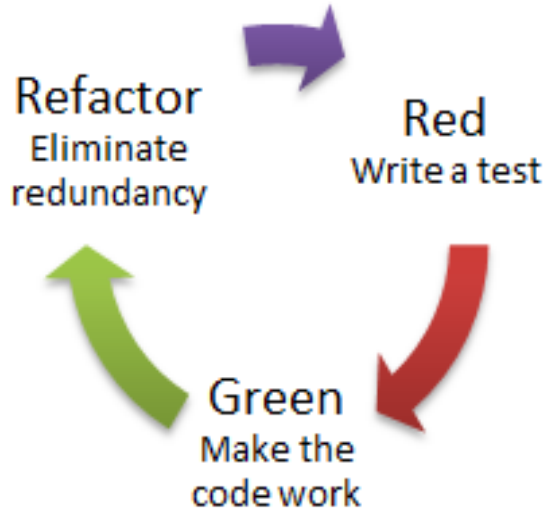
- Disasters might be around the corner

The other approach



Writing unit tests:

- Looks like a good thing to do
- And it is good to do
- And it's true that you'll end doing a lot of this



What's wrong with unit tests

You can't write unit tests for everything

- The backbone of DIRAC is in its configuration
- There are services, and agents, and DBs
- And there is the Grid
  - Several CE types, and several SEs
  - And external services
- Mocks become too complex

The third approach

- Do Unit tests for what *can be* unit tested
- Do Integration tests for what *can be* integration tested
- Do Regression tests for what *can be*… regression tested
- Do System tests for...
- TiP for everything else

This is what we call the **certification** ***process***

Testing becomes a certification *process*

- We automate what we can automate
  - Static code analysis
  - Unit tests
  - Integration tests
  - Regression tests
- Then, there are guidelines
  - For system tests
- And the rest is art

Who's responsible for what

- Unit tests are written by the code developers
  - They are part of the released code
- Most of the Integration tests are also written by the developers
  - In a separate repository
- Regression tests are written by the QA
  - In the same separate repository
- System tests are defined by the QA
  - On paper

Automation with Jenkins

- Static code analysis with pylint
- Python nose for finding and running the UT
  - Cobertura for coverage of the UT
- Integration tests:
  - Install (LHCb)DIRAC, install all the DBs on a separate instance, run all the services, try them out
  - Run a pilot, run user and production jobs locally
  - Run a pilot, match a test job
- Regression tests
  - Run a pilot, run old user and production jobs locally

System tests in a sandbox

- We install the pre-release (AKA release candidate) in a separate “setup”
- We do things that we would normally do in production:
  - Send pilots, run jobs, run productions
  - Write files, replicate them, remove them
  - …
  - We have guidelines, commands set, etc...

The problem of impossible isolation

- There's not such a thing like a "test Grid"
  - If we submit (pilot) jobs in certification, they go to the same CEs and WNs as our production jobs
  - The SEs used in production are the same as the ones used in certification
  - And the same FTS server
  - ...and so on
- And the Configuration Server for all "setups" is the same

Yay, all tests passing!

Credits:
@dave1010

Can we test it all?

No, we can't test everything

(obviously)

- But we considerably increased the percentage of tested code
- And boosted our confidence in doing (also) big changes
- Every time we find some problems in production (and this always happens) we think if and how we could have spotted the problem during certification
  - and if we could, code a test

Problems encountered

- We started writing tests for code that was (almost never) tested
  - It was “exercised”...
  - So, lots of refactoring (still) going on
- Biggest problem: convincing all developers that’s actually useful

ToDo list

- DIRAC Pull Request -> Jenkins
- Not yet moved LHCbDIRAC to GIT
  - https://gitlab.cern.ch/
    - And then we can use Jenkins tests as automated review system

What we don't (yet) plan to do

- Performance tests
- Scaling tests
- Strict code reviews

We are here to learn

In summary

- Testing is a lot of encapsulation
- Run tests in the sandbox when you can
- Be consistent, don’t give up
- TiP can’t be completely avoided

?