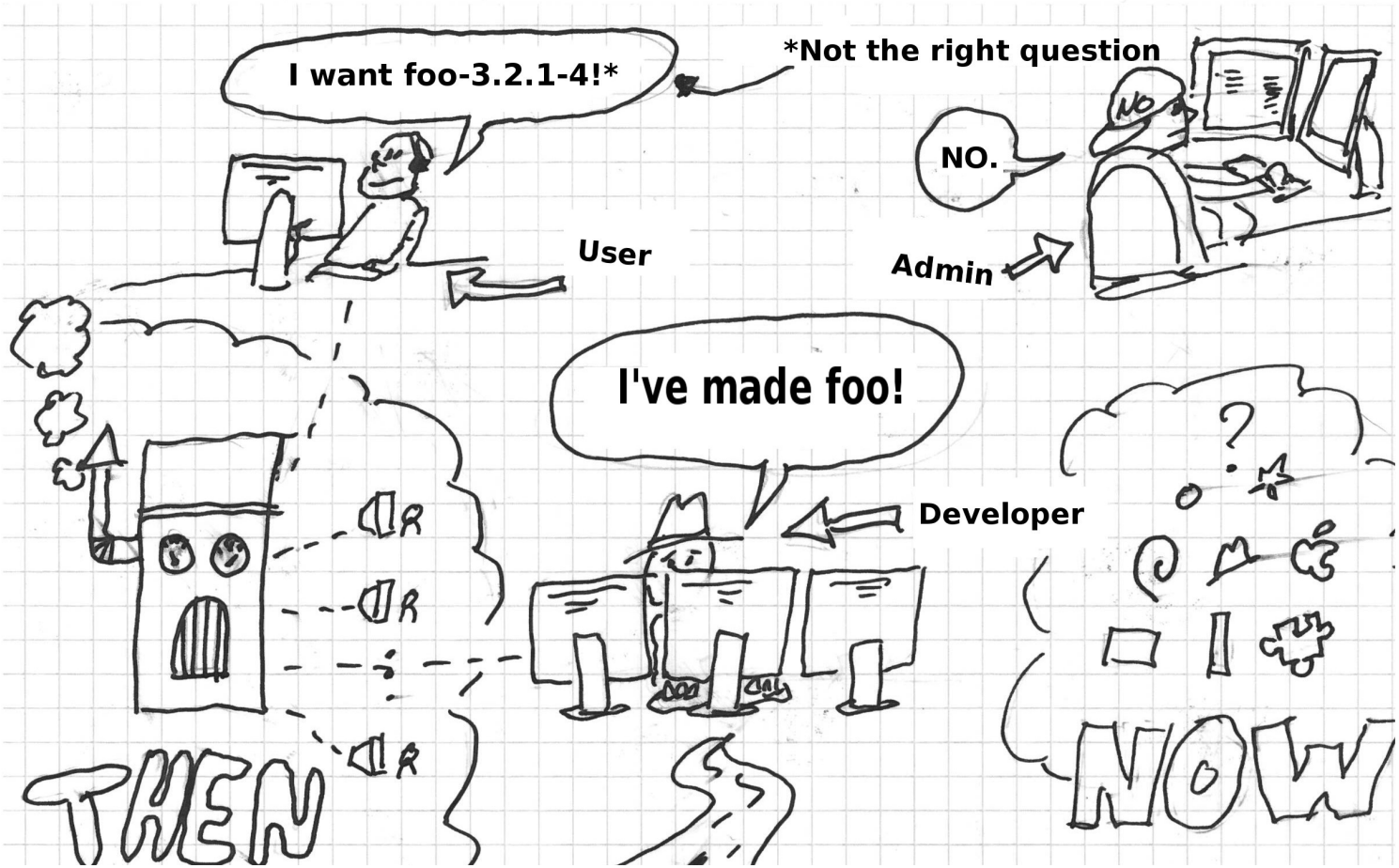
Experiences with moving to open source standards for building and packaging

Dennis van Dok, Mischa Sallé and Oscar Koeroo

Nikhef Amsterdam

SIGSOFTENG 2013, Nikhef, Amsterdam

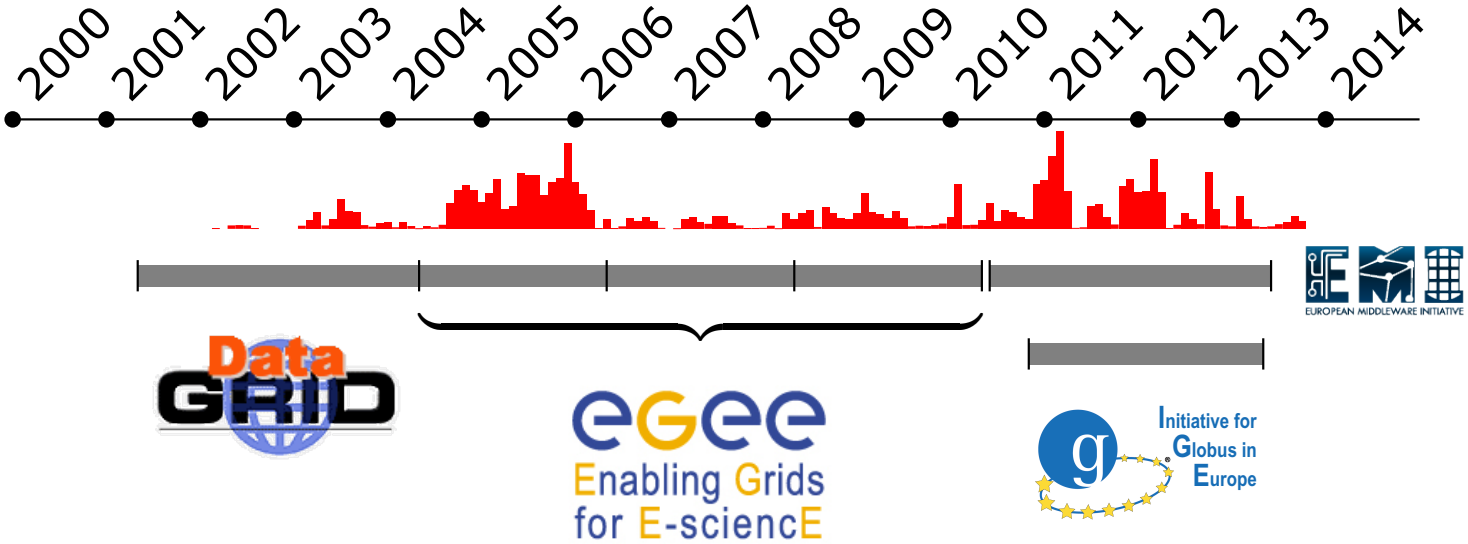
Software midwifery



Software midwifery II

Where we come from

- Middleware contributions in a series of EU funded grid projects: DataGrid, EGEE (I, II and III), EMI and IGE.
- current sustained 'maintenance mode' through SURF e-infrastructure.



The ETICS era

The EGEE era saw increased scale in every way:

- the EGEE Grid
- the middleware stack
- the code complexity and interdependencies

EGEE II had to deliver more reliable and stable software. To manage this, a build system was introduced called **ETICS**.
This system had a few shortcomings.

ETICS' shortcomings

- It was not easy to build outside of ETICS
- Building in ETICS was too slow for debugging/test cycles
- The system required specific `m4` macros to build
- There was little or no focus on portability
- The output consisted of binary products and no rebuildable source material.
- The ETICS project stopped in 2013 and the portal went off-line.

We had to do things ourselves.

What we did

code improvements

- better `m4` macros for dependency discovery
- stick to standards (ANSI C, POSIX, Autotools)
- Build without compiler warnings
- Implement secure coding standards
- active portability testing to many platforms (solving bugs along the way)
- documented software process

what we also did

version control: imported all CVS history for our components from CERN CVS, with full history

packaging and building: wrote Fedora and Debian compliant packaging sources, built native packages suitable for inclusion in mainstream distributions

Implementation of guidelines: adopt Fedora packaging guidelines and Debian Policy

automation: set up Koji for automated building

software delivery: deliver software through our own (signed) repositories

improved documentation in the form of manpages and Wiki pages.

What we (eventually) managed

Most (autotools based) open source software can do this out of the box:

```
./configure
make
make install
```

We actually stuck to the mantra:

```
make distcheck
```

which builds outside the source tree and tests if an install with `DESTDIR` works.

Fedora packaging

- Tagging in SVN (of the `SPEC` file) triggers a Koji build
- Koji does mock builds of the source and binary RPMs for all targeted platforms, i.e. the latest Fedora releases and EPEL5 and EPEL6.
- The builds are signed with a tool called *sigul.*
- The *mash* utility generates the repositories that can be installed through yum

koji, sigul and mash are also used by the Fedora project.

Debian packaging

For Debian, the procedure is slightly different. There is no equivalent of Koji for Debian ☹.

- a Debian source package is created for currently supported Ubuntu versions, Debian unstable, stable and oldstable,
- each source package is build with cowpoke/cowbuilder/pbuilder (equivalent to mock).
- the resulting packages are signed with the packager's GPG key
- The package is uploaded to a software repository from where it can be installed with apt-get.

Catching common errors

For Fedora, use `rpmlint`; for Debian, `lintian` to see if packages do not contain silly mistakes (lintian helped find several common spelling errors.)

The automated build logs revealed more warnings due to using different compiler settings.

This is not a substitute for real testing, of course.

Communi{ty,cation}

- Mailing lists

  We've set up a few mailing lists:

  - grid-mw-security-support@nikhef.nl for support questions and
  - grid-mw-security-announce@nikhef.nl for announcements of new versions. This list has an open subscription policy.

  No general discuss mailing list (yet)... no sizable community either (besides wLCG/EGI there is OSG).

Sources, binaries and bugs

- Version control in SVN
  https://ndpfsvn.nikhef.nl/viewvc/mwsec/
  https://ndpfsvn.nikhef.nl/ro/mwsec/
- Download sources
  http://software.nikhef.nl/security
- RPM/Deb distribution
  http://software.nikhef.nl/dist
- Bug tracking

  NEW: https://bugzilla.nikhef.nl/, moving away from CERN Savannah.

Why we did it

Changes were implemented gradually over time, with each step bringing new benefits.
Being good netizens and playing along with common open source practices is rewarding even without drawing a crowd.
Supporting OSG was much easier once we took control of the process.
We believe we have greatly improved sustainability of our software.

What we got in return

Some of the benefits our work rendered:

1. playing fair with package management avoids conflicts
2. installation of software becomes trivial
3. reproducing bugs becomes easier
4. pin-pointing bugs to source lines is easier
5. cycle time to deliver updates becomes shorter
6. uncovered some lurking bugs
7. improved portability
8. easier integration with third parties
9. using common technology makes it easier to pass the support to future staff members.

References

- Guide to setting up the Koji build system
  http://fedoraproject.org/wiki/Koji/ServerHowTo
- Nikhef Security Access Control sofware procedures
  https://wiki.nikhef.nl/grid/SAC_software_procedures
- Fedora packaging guidelines
  https://fedoraproject.org/wiki/Packaging:Guidelines
- Debian Policy
  http://www.debian.org/doc/debian-policy/
- Debian upstream guide
  https://wiki.debian.org/UpstreamGuide

Bonus item: How to tell if a FLOSS project is doomed to FAIL

Original by Tom “Spot” Callaway; inspired by Chromium.

How to tell if a FLOSS project is doomed to FAIL

Count along the POFs (Points Of Fail) to see how well your project is doing. Our overall fail score: 10 points.

Size

- The source code is more than 100 MB. **+5**
- If the source code also exceeds 100 MB when it is compressed **+5**

Source Control

- There is no publicly available source control (e.g. cvs, svn, bzr, git) **+10**
- There is publicly available source control, but:
  - There is no web viewer for it **+5**
  - There is no documentation on how to use it for new users **+5**
  - You've written your own source control for this code **+30**
  - You don't actually use the existing source control **+50**

Building From Source

- There is no documentation on how to build from source **+20**
- If documentation exists on how to build from source, but it doesn't work **+10**
- Your source is configured with a handwritten shell script **+10**
- Your source is configured editing flat text config files **+20**
- Your source is configured by editing code header files manually **+30**
- Your source isn't configurable **+50**
- Your source builds using something that isn't GNU Make **+10**
- Your source only builds with third-party proprietary build tools **+50**
- You've written your own build tool for this code **+100**

Bundling

- Your source only comes with other code projects that it depends on **+20**
- If your source code cannot be built without first building the bundled code bits **+10**
- If you have modified those other bundled code bits **+40**

Libraries

- Your code only builds static libraries **+20**
- Your code can build shared libraries, but only unversioned ones **+20**
- Your source does not try to use system libraries if present **+20**

System Install

- Your code tries to install into /opt or /usr/local **+10**
- Your code has no “make install” **+20**
- Your code doesn’t work outside of the source directory **+30**

Code Oddities

- Your code uses Windows line breaks (“DOS format” files) **+5**
- Your code depends on specific compiler feature functionality **+20**
- Your code depends on specific compiler bugs **+50**
- Your code depends on Microsoft Visual Anything **+100**

Communication

- Your project does not announce releases on a mailing list **+5**
- Your project does not have a mailing list **+10**
- Your project does not have a bug tracker **+20**
- Your project does not have a website **+50**
- Your project is sourceforge vaporware **+100**

Releases

- Your project does not do sanely versioned releases (Major, Minor) **+10**
- Your project does not do versioned releases **+20**
- Your project does not do releases **+50**
- Your project only does releases as attachments in web forum posts **+100**
- Your releases are only in .zip format **+5**
- Your releases are only in OSX .zip format **+10**
- Your releases are only in .rar format **+20**
- Your releases are only in .arj format **+50**
- Your releases are only in an encapsulation format that you invented. **+100**
- Your release does not unpack into a versioned top-level directory (e.g. glibc-2.4.2/) **+10**
- Your release does not unpack into a top-level directory (e.g. glibc/) **+25**
- Your release unpacks into an absurd number of directories (e.g. home/johndoe/glibc-svn/tarball/glibc/src/) **+50**

History

- Your code is a fork of another project **+10**
- Your primary developers were not involved with the parent project **+50**
- Until open sourcing it, your code was proprietary for:
- 1-2 years **+10**
- 3-5 years **+20**
- 6-10 years **+30**
- 10+ years **+50**

Licensing

- Your code does not have per-file licensing **+10**
- Your code contains inherent license incompatibilities **+20**
- Your code does not have any notice of licensing intent **+30**
- Your code doesn’t include a copy of the license text **+50**
- Your code doesn’t have a license **+100**

Documentation

- Your code doesn’t have a changelog **+10**
- Your code doesn’t have any documentation **+20**
- Your website doesn’t have any documentation **+30**

FAIL METER

Points of FAIL	Verdict
0	Perfect! All signs point to success!
5–25	You're probably doing okay, but you could be better.
30–60	Babies cry when your code is downloaded
65–90	Kittens die when your code is downloaded
95–130	HONK HONK. THE FAILBOAT HAS ARRIVED!
135+	So much fail, your code should have its own reality TV show.