

Continuous Delivery of Debian packages

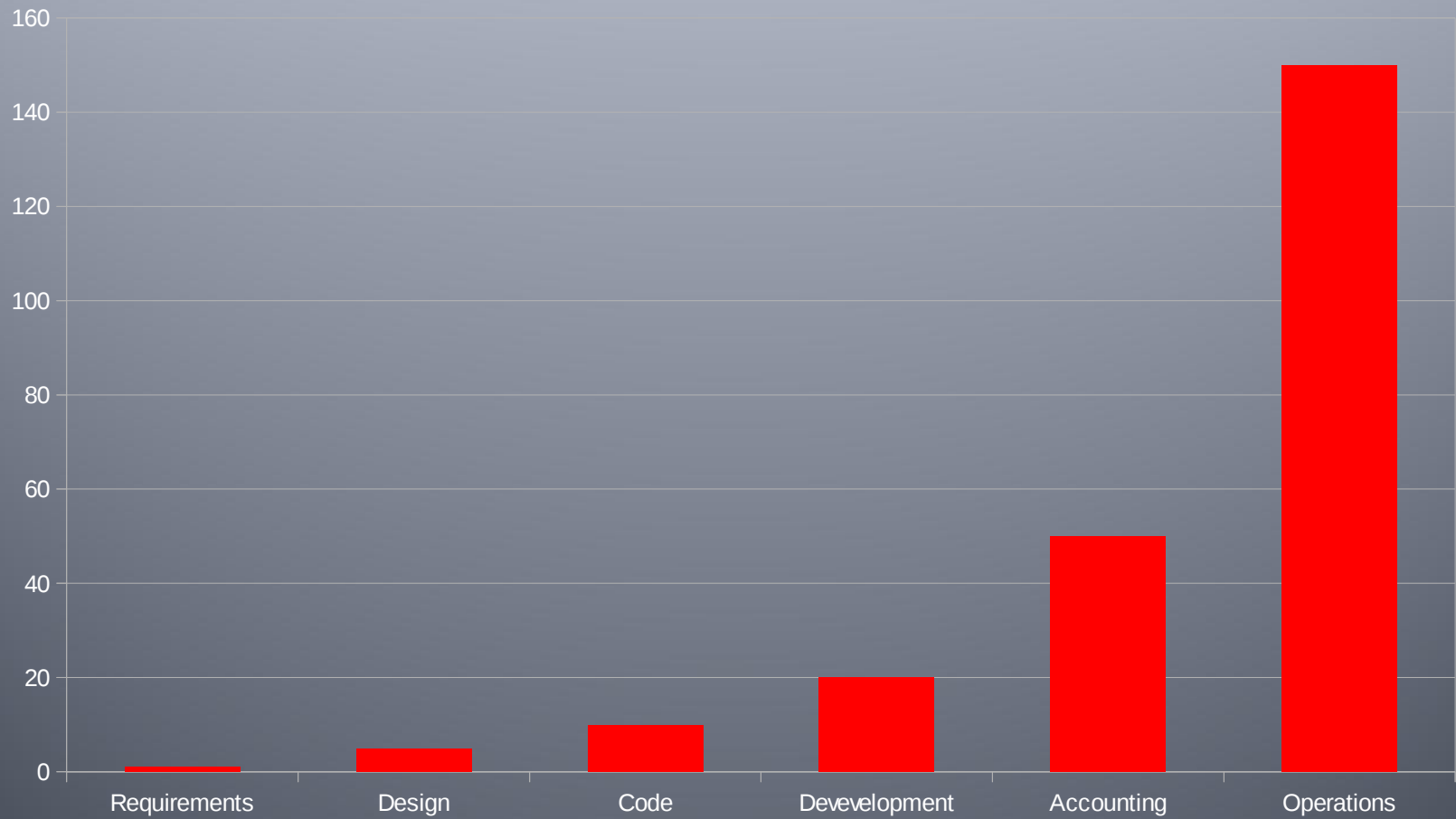
Michael Prokop

Terminology

- Continuous Integration
 - well known from software development
- Continuous Deployment
 - Q/A criteria says OK? Ship/deploy!
- Continuous Delivery
 - release whenever *you* decide it's useful to do so (= business decision!)

Why?

Costs of a Bugfix



Source: Barry Boehm's „EQUITY Keynote Address“

Independence



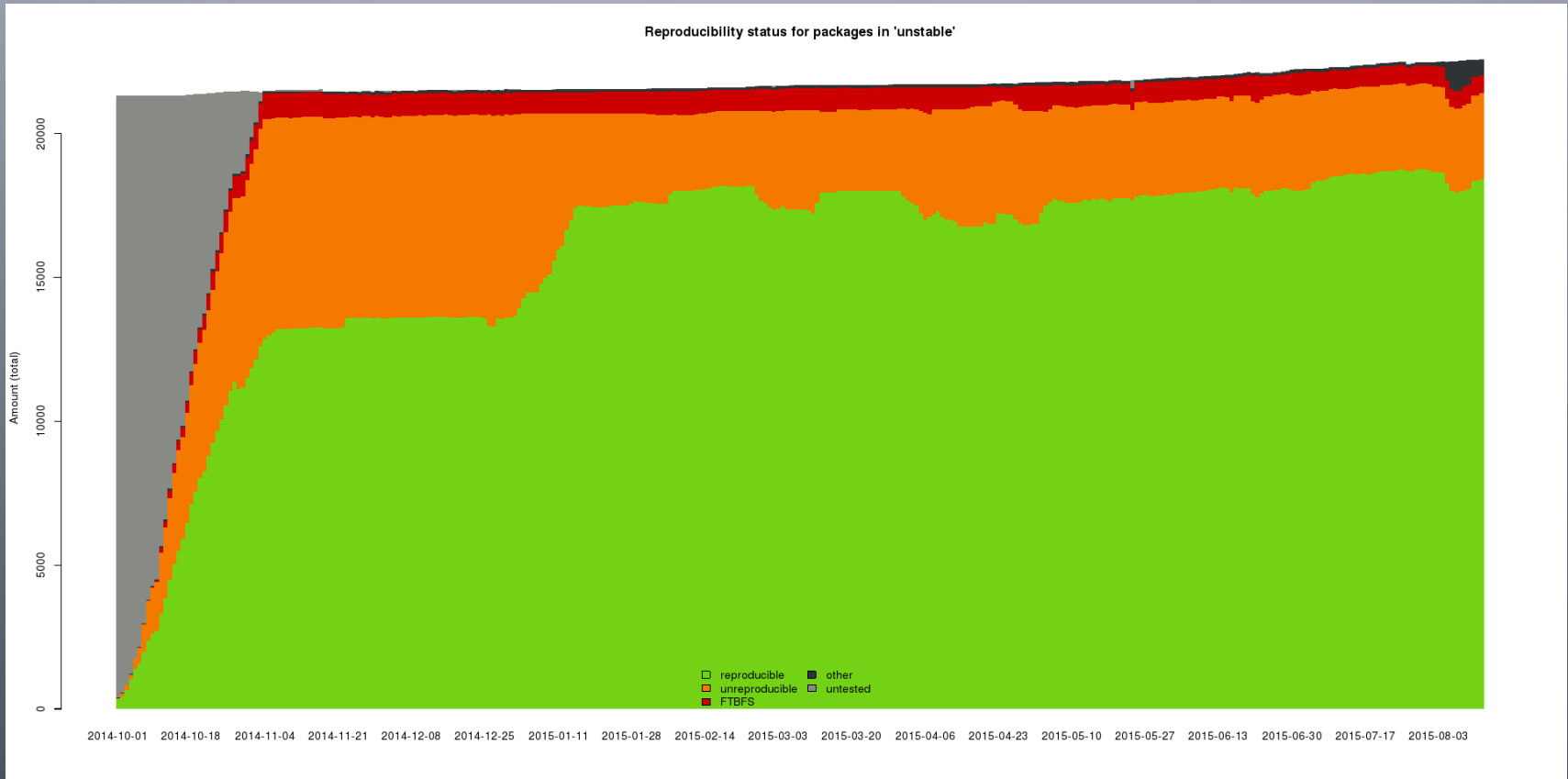
Source: <http://decarabia.soup.io/post/241926962/Image>

Scaling



Source: <https://www.flickr.com/photos/scobleizer/4870003098/>

Reproducible



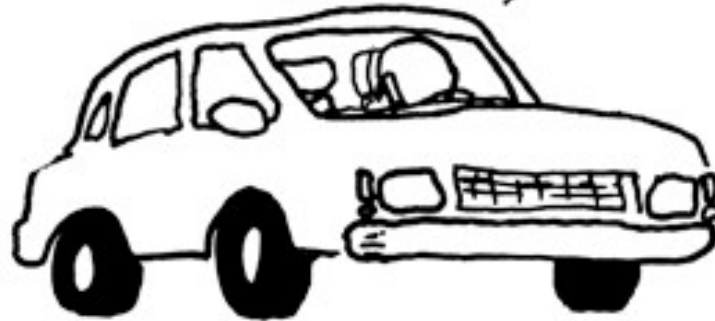
Source: <https://wiki.debian.org/ReproducibleBuilds>

Predictable

I'M JUST OUTSIDE TOWN, SO I SHOULD
BE THERE IN FIFTEEN MINUTES.

ACTUALLY, IT'S LOOKING
MORE LIKE SIX DAYS.

NO, WAIT, THIRTY SECONDS.



THE AUTHOR OF THE WINDOWS FILE
COPY DIALOG VISITS SOME FRIENDS.

Problems

Problems \$company experienced 1/2

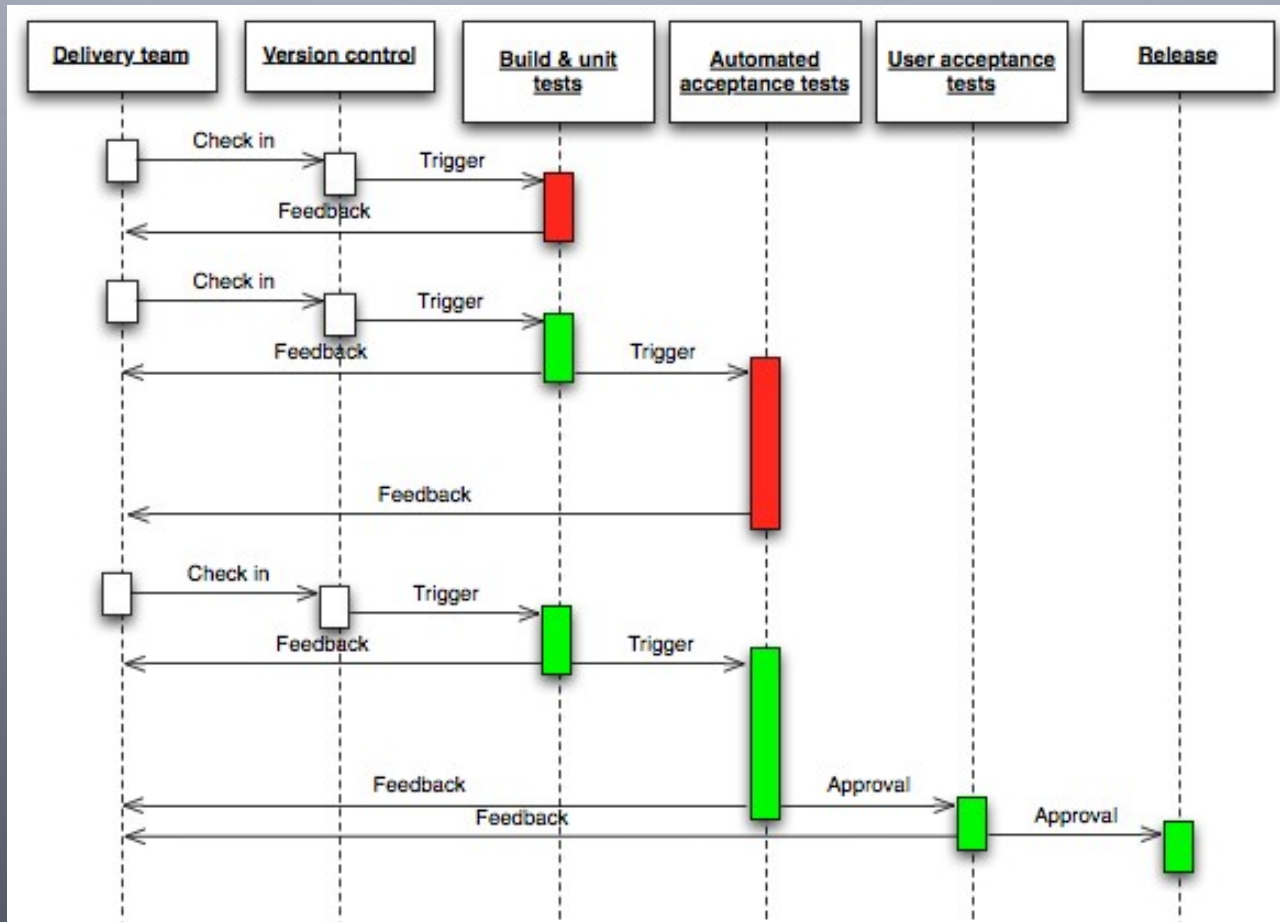
- Mess with golden images (to ship a custom software stack to customers)
- Long build times (e.g. single change → rebuild full image, upload to customer,...)
- Builds non-reproducible (unmanaged build infrastructure, devs can build + include their own packages,...)

Problems \$company experienced 2/2

- Release process holding back ongoing development work (VCS freezes are preventing ongoing work)
- Getting more and more customers → not scaling (golden images → even worse)
- Tried Debian source package uploads to custom build service → many pitfalls + developers still needed to manually build/release packages (some of them not even using Debian/Ubuntu → tools like git-dch, debuild,... unavailable)

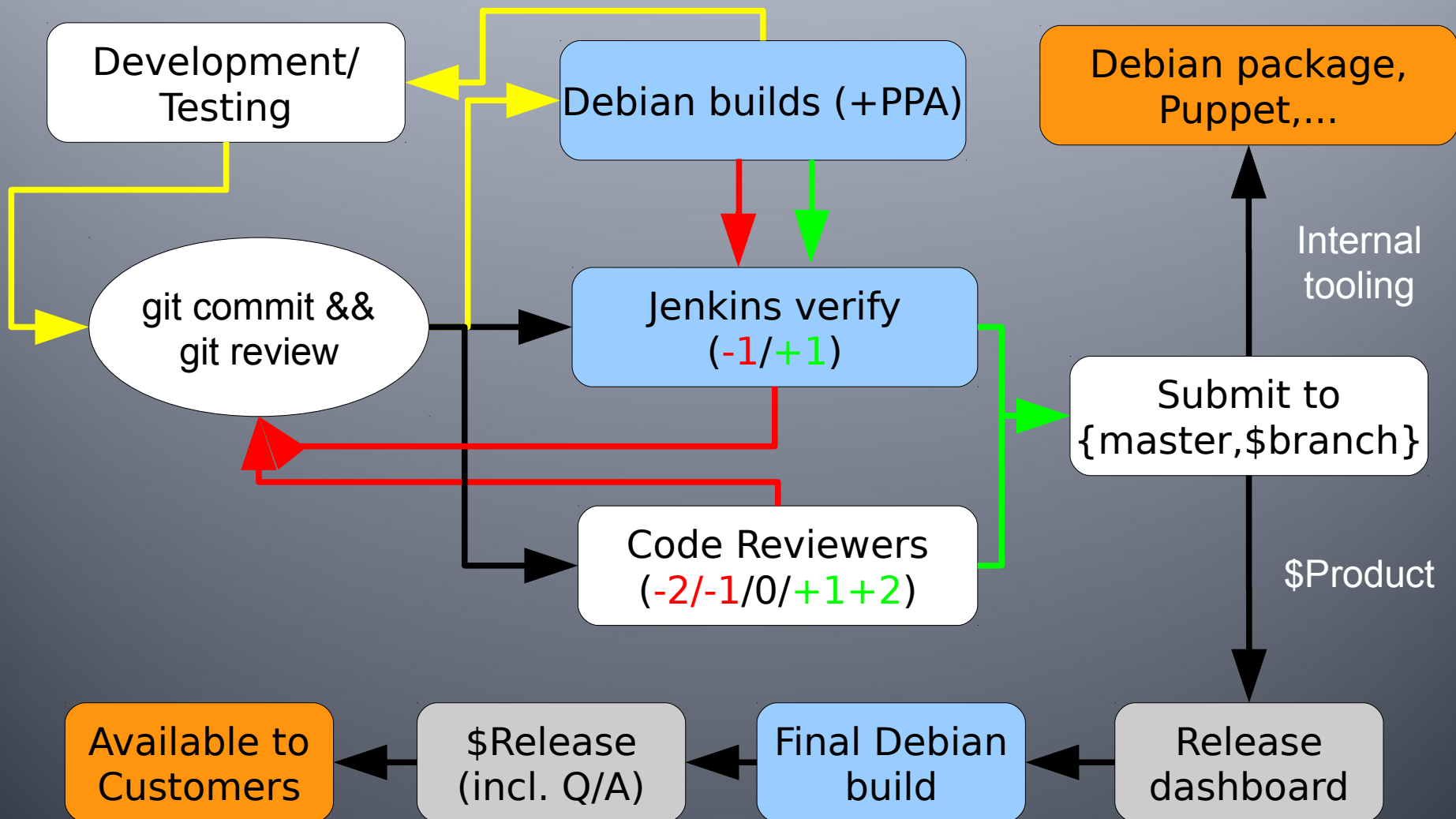
What do
we want?

Deployment Pipeline



Source: <http://continuousdelivery.com/2010/02/continuous-delivery/>

Workflow



How did we
get there?

Principles

- Rely on Debian packages + Debian repositories for everything (no exceptions)
- Only what's under version control matters (no option to build something manually on your own system)
- Automate infrastructure handling (Puppet/Ansible)

Automation

- Automated debian/changelog handling to simplify releasing of new package versions (devs don't need Debian/Ubuntu at all)
- Automated release branch handling (release 0.42 is available as such a branch)
- VMs for testing/development (via Vagrant → run `vagrant up \$product-\$version`, automated box builds at least once per day)
- PPAs for development (no VCS freezes, fast-forward + release branches only)

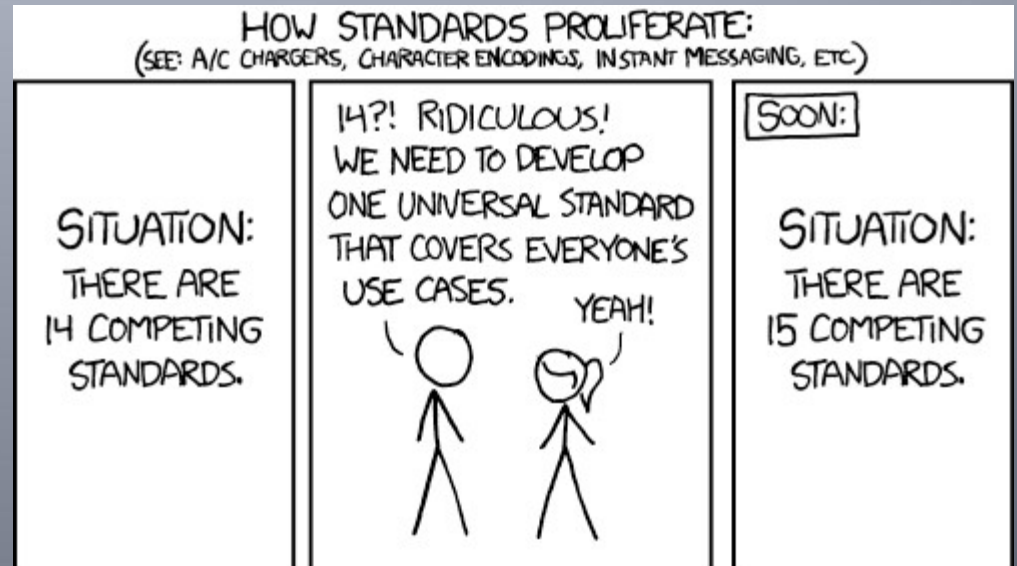
Improvements

- Usage of tmpfs/eatmydata, ccache,... for build speedups
- Dashboards for abstraction + let people focus on their tasks instead of tools
- Code review system (improves code quality but also sharing knowledge + introducing new people)

Jenkins-
debian-
glue

Standards

„The nice thing about standards is that there are so many of them to choose from.”



Source: <https://xkcd.com/927/>

Jenkins?

- Hudson: 2004
- Jenkins: 2011
- Weekly releases + LTS versions
- MIT license
- >1000 plugins available
- >120k registered installations (07/15)
- Disclaimer: written *in* Java, but absolutely *not* restricted to Java projects

Why jenkins-debian-glue?

- Formalize existing knowledge into a customizable framework
- Provide a common ground to base (further) work on
- Gather feedback from what other users (might) need
- Community building
- Don't create new tools and standards, instead rely on existing and working ones
- Should be easy to use also for non-Debian folks

What's behind j-d-g?

- Open Source Project (MIT license)
 - started in 2011
 - >25 contributors
 - written mainly in shell, easy to adjust + extend
- CI server (Jenkins)
- Build environment (cowbuilder/pbuilder)
- VCS (git + svn OOTB)
- Repository management (reprepro + freight)
- Q/A tools: piuparts, lintian, autopkgtest, pep8, perlcritic, shellcheck, checkbashism

Who's using j-d-g?

- Grml (incl. dpkg, FAI, initramfs-tools,...)
 - <https://jenkins.grml.org/>
- PostgreSQL
 - <https://wiki.postgresql.org/wiki/Apt>
- LLVM
 - <http://llvm.org/apt/>
- Kamailio
 - <https://kamailio.sipwise.com>
- Wikimedia
 - <https://integration.wikimedia.org/ci/view/Ops-DebGlue/>
- ... and many more

Setup? Automatic deployment

```
% wget https://raw.githubusercontent.com/mika/\
jenkins-debian-glue/\
master/puppet/apply.sh
% sudo bash ./apply.sh $your_password
```

Setup

If you want to get all the work done for you automatically then please choose the automatic approach.

Notice: recommended if you are starting with a plain Debian/Ubuntu system from scratch.

 Choose Automatic

If you want to manually set up the system on your own please follow the documentation.

Notice: recommended if you already have a running Jenkins system on Debian/Ubuntu.

 Choose Manual

http://jenkins-debian-glue.org/getting_started/

What do I get?

A screenshot of the Jenkins web interface. The top navigation bar includes the Jenkins logo, a search box, and a 'log in' link. Below the navigation bar, there are links for 'People' and 'Build History'. The main content area is titled 'jenkins-debian-glue Continuous Integration labs'. It features a table with columns for 'S', 'W', 'Name', 'Last Success', 'Last Failure', and 'Last Duration'. The table lists three jobs: 'jenkins-debian-glue-binaries', 'jenkins-debian-glue-piuparts', and 'jenkins-debian-glue-source', all with 'N/A' for success and failure dates. On the left, there are sections for 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing two executors in 'Idle' status). At the bottom, there are links for 'Help us localize this page', 'Page generated: Feb 5, 2013 3:25:25 PM', 'REST API', and 'Jenkins ver. 1.480.2'.

Jenkins

search ? log in

Jenkins ENABLE AUTO REFRESH

[People](#) [Build History](#)

jenkins-debian-glue Continuous Integration labs

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
		jenkins-debian-glue-binaries	N/A	N/A	N/A
		jenkins-debian-glue-piuparts	N/A	N/A	N/A
		jenkins-debian-glue-source	N/A	N/A	N/A

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

[Help us localize this page](#) Page generated: Feb 5, 2013 3:25:25 PM [REST API](#) [Jenkins ver. 1.480.2](#)

`${project}-source`

- generate Debian source package using VCS
 - (Upstream) Source (orig.tar.xz)
 - Debian changes (debian.tar.xz) [optional]
 - Control file (.dsc)
- Script *generate-`{git,svn}`-snapshot*
- Automates changelog generation (git-dch ftw, thanks Guido!)
- Important: needs to be run only once per project (exception: multi distribution usage in one repository)

`${project}-binaries`

- Debian Binary Packages (*.deb)
- Script build-and-provide-package
 - Automates pbuilder/cowbuilder setup, usually nothing to do manually
- Important: build once per architecture/distribution (exception: “Architecture: all”)

`${project}-piuparts`

- Install/deinstall/upgrade testing (optional)
- Useful since you might forget about it otherwise

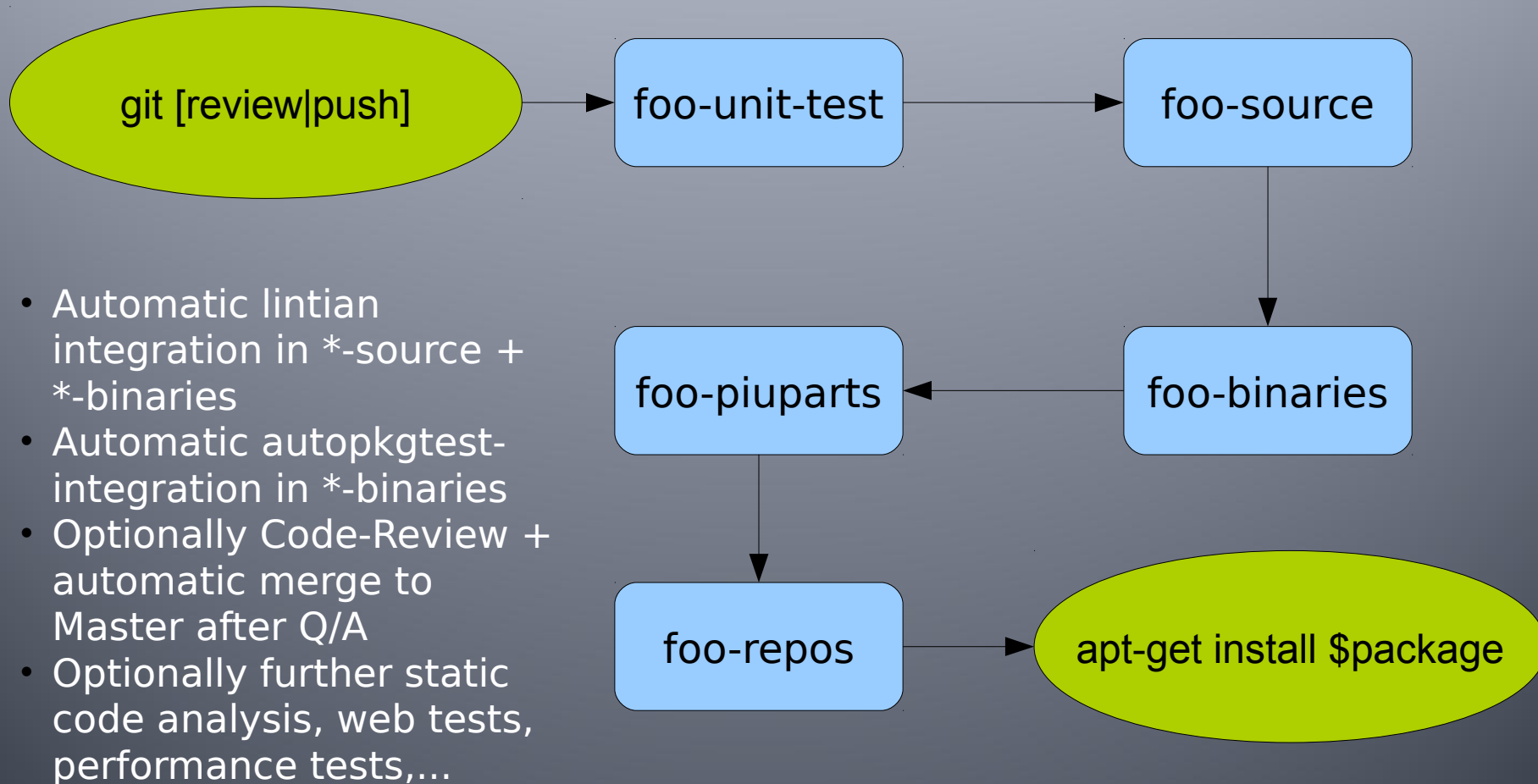
Repository Handling

- Automatic handling of repositories without manual interaction
 - reprepro
 - freight
- By default part of `project`-binaries job
- Separate usage via `project`-repos job:
 - BUILD_ONLY vs PROVIDE_ONLY

Further Q/A testing available OOTB

- Lintian
- Autopkgtest
- perlcritics/checkbashism/
shellcheck/pep8/...
- Results as TAP/jUnit/... reports in
Jenkins available

Example of a Build Pipeline



Managing many Jenkins jobs without driving nuts?

- usage of jenkins-job-builder to create and manage Jenkins jobs
 - <http://docs.openstack.org/infra/system-config/jjb.html>
 - <https://github.com/sipwise/kamailio-deb-jenkins> (example)
- YAML file(s) for configuration
 - No webinterface clicking!
 - Version control!
 - Code review for job changes!

Lessons
learned

Lessons learned 1/3

- Developers needs vs operations/
distribution needs (\$package or \$version
not available)
 - Contribute back to Debian when
reasonable
- Diverse people improve overall quality
 - Homogeneous systems, diverse people
- Code review requires good remote working
culture
 - Open Source folks are used to remote
working :)

Lessons learned 2/3

- Avoid external dependencies
 - Github, CPAN, PyPI, RubyGems, Puppetlabs, Percona, \$local_debian_mirror... unreachable?
 - set up local mirrors
 - Speedup!
 - Staging options
- Configuration management (e.g. for setup of Jenkins slaves) is essential → infrastructure as code
- Consistent timezones (UTC) + time (NTP!)

Lessons learned 3/3

- Catch 22
 - build scripts broken but build infrastructure receives updates via build infrastructure/scripts? → recursion problem
 - upgrading from wheezy to jessie, deployment of configuration management depends on unit-tests which don't work on jessie yet
- Provide test infrastructure for setup, configuration, ... changes without breaking production
- Rebuild of a system might look different from currently running one, even with cfgmngmt → use testing also for cfgmngmt (serverspec, mspectator, Tests::Server, Test Kitchen, ...)

Tips 1/2

- Regular rebuilds of all packages → apply recent policies + package build infrastructure changes so packages are up2date
- mr/myrepos is very useful for dealing with large amounts of repositories (thanks joeyh!)
- Integrate CI/CD system into your monitoring environment

Tips 2/2

- Collect metrics independent from Jenkins & CO to be able to remove jobs/builds without losing metrics
- Use gerty cmdline tool if you don't like gerrit web interface
- Set up „jenkins-verify” job to ensure Jenkins works as needed

Antipatterns 1/3

- Manual SSH → provide debugging options instead
- Flaky Tests (fast vs slow hardware, „sleep X”, ...) → people don't trust + care any longer
- Polling/pull/cronjobs instead of triggering → get immediate actions + effects

Antipatterns 2/3

- Manual setup of machines/configs → snowflake pattern (AKA they look alike but are still different)
- No standardized output in tools → makes parsing hard[er]
- Checklists → use automation instead

Antipatterns 3/3

- Hardcoding (IP addresses, hostnames, port number, test system,...) instead of configurability
- Same thing gets built multiple times in the deployment pipeline → share artifact instead
- Lack of notifications for failing builds/tests/... → developer starts to wait + poll

Unresolved problems 1/2

- dependency management alla wanna-build to get package builds automatically in the right order (package *foo* Build-Depends on package *bar* → build *bar* before *foo*)
- Build-Depends vs Depends, but no „Test-Depends“ (bundler, carton,...)

Unresolved problems 2/2

- „High frequency“ (CI/CD) Debian repositories causing apt to often fail while mirror is updated (“Hashsum mismatch”)
- piuparts: successful runs even though there have been issues, e.g. package that gets tested has dependency issues though removing the package itself is considered a valid solution

Recap - projects possibly worth a look

- Debian :)
- Jenkins
- Jenkins-debian-glue
- Vagrant
- Gerrit + Gertty
- Jenkins-Job-Builder

Recap - tl;df

- Put everything under version control
- Automation (deployment, cfgmgmt, release process)
- Custom Dashboards
- Tests, tests, tests
- Rely on established workflows + tools
- PS: Once you're used to that working in non-CD environments feels bad

Jenkins-debian-glue BoF

- Date: 2015-08-21
- Time: 18:00-19:00
- Room: Helsinki
- Purpose: In this BoF session we provide an opportunity to meet developers + contributors of the jenkins-debian-glue project, discuss issues for improvements, upcoming new features and get your questions answered.

Questions || Wishes?

@mikagrml
mika (at) debian.org

<http://michael-prokop.at/blog/>
<http://jenkins-debian-glue.org/>

Thanks for feedback to Christian Hofstaedtler + Victor Seva