Ten years analysing large code bases: a perspective

Roberto Di Cosmo
http://www.dicosmo.org

04/12/2015
EvoLille
Free Software, industrialised: distributions

Idea from FOSS in the 1990s: distributions are intermediate software vendors between FOSS developers and users, offering to share upstream tracking, integration, testing and QA among all of us.
Distributions: a “somehow” successful idea ...

Key notions: packages and package managers
Packages and their metadata

Package = \{ some files, some scripts, metadata \}

- Identification
- Inter-package rel.
  - Dependencies
  - Conflicts
- Feature declarations
- Other
  - Package maintainer
  - Textual descriptions
  - ...

Example (package metadata)

```plaintext
Package: aterm
Version: 0.4.2-11
Section: x11
Installed-Size: 280
Maintainer: Göran Weinholt ...
Architecture: i386
Depends: libc6 (>= 2.3.2.ds1-4),
         libice6 | xlibs (> 4.1.0), ...
Conflicts: suidmanager (< 0.50)
Provides: x-terminal-emulator
...```

A package is the elemental component of modern distribution systems (not FOSS-specific). A working system is deployed by installing a package set ($\approx 2'000+$ for modern FOSS distros).
Inter-package relationships get complex...

To play Backgammon...

- **Package:** gnubg
- **Version:** 0.14.3+20060923-4
- **Depends:** gnubg-data, ttf-bitstream-vera, libartsc0 (>= 1.5.0-1), ..., libgl1-mesa-gl | libgl1, ...
- **Conflicts:** ...

...pull a few strings

Distributions grow superlinearly

Using and maintaining such large software collections is becoming hard!

- manual package review
- semi-automated tools
Using and maintaining free software distributions is hard

We need advanced tools:

1. **end user side** – package managers
   - install packages and their dependencies, …
   - …according to the user needs and policies

2. **distribution editor side** – QA infrastructure (our focus today)
   - find “broken” packages (now easy!)
   - find packages which impact large parts of the distribution
   - predict repository update woes
   - identify compatibility issues
   - …

With a big boost from the Mancoosi project¹, we have made progress in both areas.

Let’s focus on the *distribution editor side*

---

¹http://www.mancoosi.org
Ensuring Quality throughout evolution

The Quality Assurance team and the release manager need to track tens of thousands of packages, their bugs, their incompatibilities, etc. that change every day.

It is important to catch as many installation-related errors as possible before they hit the user and the BTS, and this requires automation.

Static analysis of package dependencies: the state of the art

- find packages that cannot be installed at all, and...
  - spot the ones that surely need to be fixed (know who to blame)
  - provide advance warning for future problems
- find the incompatibilities between packages
- show how these incompatibilities evolve
- automate package migration
QA 101: find *individually* broken packages

The installability problem

In a repository $R$, decide whether a package $p$ can be installed in isolation.

Theorem

*The installability problem is NP complete* (Di Cosmo et al. ASE 2006)

Solving tens of thousands of NP-complete problems?

In practice: recent SAT solvers handle current instances easily

- few explicit conflicts (without conflicts, just dual Horn clauses)

A practical tool

- *rpmcheck*/*debcheck* (*Vouillon, 2006*)
  - finds all broken packages and provides short explanations
  - fast: analyses $\approx 40\,000$ (binary) packages in minutes

- In *dose3* library as *distcheck*, by Pietro Abate et al.

- Extensively used (*Abate, Di Cosmo et al. MSR 2015*)
QA 101, level 2: what can we say about the future?

**Definition (outdated packages)**

$p$ is outdated if $p$ is not installable, and it remains uninstallable no matter how the other packages evolve (*i.e. p’s maintainer has no excuses*).

**Definition (challengers)**

$p$ challenges $q$ if upgrading $p$ “forces” to upgrade $q$.

What they have in common:

- properties that hold in *all* installations of *any* future evolution of the repository.
- seems unfeasible, but *we can efficiently decide* some properties of this kind.

---

*Abate, Di Cosmo, Treinen, Zacchiroli*

*Learning from the Future of Component Repositories*

*CBSE 2012. Best paper award*

Tools in the *dose* library now used in qa.debian.org/dose
Next step: interaction between packages

Example: is there any package which cannot be installed together with *iceweasel*? with *kde-full*?

*Definition*: a set of packages are *co-installable* if they can be installed together.

- all packages *should be* installable (individually!)
- *some* package incompatibilities *are expected*

Can we *summarise* all incompatibility issues, and avoid browsing through hundreds of hyperlinked pages?
Coinst

A simplification theory for repositories, based on the extraction of a co-installability kernel, i.e. a repository much smaller than the original but equivalent wrt co-installability.

Highlights:

- reflexive/transitive dependency closure
- equivalent classes and quotients
- machine-checked proofs (in Coq!)

In a word: tough maths at work!

A tool: coinst (packaged in Debian)

---

Vouillon, Di Cosmo

*On Software Components Co-Installability*

Co-Installability Graphs

```bash
coinst -root iceweasel -o graph.dot Packages_i386
```

```
xul-ext-greasemonkey
\hspace{0.5cm} xul-ext-gnome-keyring
\hspace{0.5cm} xul-ext-sage
iceweasel
\hspace{0.5cm} libasound2
\hspace{2.5cm} liboss4-salsa-asound2
```

```bash
coinst -root kde-full -o graph.dot Packages_i386
```

```
kde-full
\hspace{2.5cm} kde-window-manager
\hspace{2.5cm} libasound2
\hspace{2.5cm} libgps20
\hspace{2.5cm} lib phonon4
\hspace{2.5cm} libqt4-phonon
\hspace{2.5cm} oxygen-icon-theme
\hspace{2.5cm} libboss4-salsa-asound2
```

Roberto Di Cosmo (INRIA/Paris Diderot)

Analysing large code bases

EvoLille 2015
Interactive graph viewers on http://coinst.irill.org/
Ubuntu main: 7’000 packages, 31’000 dependencies

http://coinst.irill.org

Running time: less than 10 seconds!
QA 301: New Co-Installability Issues

Compare two versions of a repository

- New issues are more likely to be bugs
- Can report *precisely* what changes caused an issue

Example

```
 a   b
|    |
p   q
```

*Many* new issues between packages p and q due to a *single* new conflict between packages a and b.

Vouillon, Di Cosmo

*Broken Sets in Software Repository Evolution*

ICSE 2013
Finding New Co-Installability Issues

Tool coint-upgrades
http://coinst.irill.org/upgrades
- graphs illustrating each new issue
- context: other packages involved, package popularities (popcon)

The new version of unoconv depends on any version of python3-uno

The new version of tdsodbc conflicts with any version of libiodbc2

Package libiodbc2 had been unmaintained for years
Should not be a big issue if it gets removed, right?
Context is Crucial

QA 501: package migration

Conflicting goals

- package should reach testing rapidly
- keep testing as stable as possible
The *comigrate* tool

- Supplement/Replace Britney
  Generate hints that can be fed to Britney
- Interactively investigate migration issues
  Run it repeatedly, studying different scenarios
- Report of issues preventing package migration
Start with simple constraints

The Boolean solver generates a tentative migration

Check for (co-)installability issues; analyse these issues to generate new constraints (“package A cannot migrate”, or “package A cannot migrate without package B”)

Repeat until no more issue is found
Data collected twice a day from 2013-06-24 to 2013-09-09

Missing datapoints: Britney can take more than 24 hours (Perl transition)....
Bibliography and tools excerpts

Di Cosmo, Leroy, Treinen, Vouillon et al
Managing the complexity of large free and open source package-based software distributions.

Di Cosmo and J. Vouillon.
On software component co-installability.
In ESEC/FSE 2011.

Abate, Di Cosmo, Treinen, Zacchioli
Learning from the Future of Component Repositories
CBSE 2012: Component Based Software Engineering.

Vouillon, Dogguy, Di Cosmo.
Easing software component repository evolution.
ICSE 2014.

Abate, Di Cosmo, Gesbert, Le Fessant, Treinen, and Zacchioli.
Mining component repositories for installability issues.
MSR 2015

Claes, Mens, Di Cosmo, and Vouillon.
A historical analysis of debian package incompatibilities.
MSR 2015

Tools

- Cudf library: http://gforge.inria.fr/projects/cudf/
- Dose library: http://gforge.inria.fr/projects/dose/
- Coinst suite: http://coinst.irill.org
- Debian QA: http://qa.debian.org/dose
A recurring pattern

In all examples above

- identify a real world problem whose solution requires a research effort
- work hard to find a solution
- implement a tool, validate it on real world cases
- publish a research article
- foster adoption (the hardest part!)

In a picture

Under the hood

Question:

What were the technical prerequisites that made this work possible?
Technical prerequisites

### Availability
- all the *(history of)* Debian packages (after 2005)
- no *technical* restrictions
- no *legal* restrictions on content or metadata

### Traceability
Debian packages have
- *unique identifier*
- *reference central repository*

### Uniformity
Debian packages: a central catalog with
- *uniform metadata structure*
- *uniform naming and versioning schema*

These are all essential features for *reproducibility* and for *preservation*... ... we need them for *all* software!
Availability: software is *fragile*

An example is worth a thousand words... let’s see a few
Inconsiderate or malicious loss of code

The Year 2000 Bug … uncovered an inconvenient truth

in 1999, an estimated 40% of companies had either lost, or thrown away the original source code for their systems!


Murder in the Amazon cloud

The demise of Code Spaces at the hands of an attacker shows that, in the cloud, off-site backups and separation of services could be key to survival

Yes, for seven years all seemed ok. No, they did not recover the data.
Business-driven loss of code support: Google

When we started the Google Code project hosting service in 2006, the world of project hosting was limited. We were worried about reliability and stagnation, so we took action by giving the open source community another option to choose from. Since then, we’ve seen a wide variety of better project hosting services such as GitHub and Bitbucket bloom. Many projects moved away from Google Code to those other systems. To meet developers where they are, we ourselves migrated nearly a thousand of our own open source projects from Google Code to GitHub.

As developers migrated away from Google Code, a growing share of the remaining projects were spam or abuse. Lately, the administrative load has consisted almost exclusively of abuse management. After profiling non-abusive activity on Google Code, it has become clear to us that the service simply isn’t needed anymore.

Beginning today, we have disabled new project creation on Google Code. We will be shutting down the service about 10 months from now on January 25th, 2016. Below, we provide links to migration tools designed to help you move your projects off of Google Code. We will also make ourselves available over the next three months to those projects that need help migrating from Google Code to other hosts.

- March 12, 2015 - New project creation disabled.
- August 24, 2015 - The site goes read-only. You can still checkout/view project source, issues, and wikis.
- January 25, 2016 - The project hosting service is closed. You will be able to download a tarball of project source, issues, and wikis. These tarballs will be available throughout the rest of 2016.

Google will continue to provide Git and Gerrit hosting for certain projects like Android and Chrome. We will also continue maintaining our mirrors of projects like Eclipse, kernel.org and others.
Hi zacchiro,

I’m Rolf Bjaanes, CEO of Gitorious, and you are receiving this email because you have a user on gitorious.org. As you may know, Gitorious was acquired by GitLab [1] about a month ago (NDLR: 3/3/2015), and we announced that Gitorious.org would be shutting down at the end of May, 2015.

... Rolf
Traceability: disruption of the web of reference

Web links are not permanent (even permalinks)

There is no general guarantee that a URL… which at one time points to a given object continues to do so


URLs used in articles decay!

Analysis of IEEE Computer (Computer), and the Communications of the ACM (CACM): 1995-1999

- The half-life of a referenced URL is approximately 4 years from its publication date

D. Spinellis. The Decay and Failures of URL References.

Uniformity: nowhere to be seen

Reference repository(ies)

- Bitbucket
- GitHub
- Gitorious (no, scratch this)
- Google Code (no, scratch this)
- Maven
- Sourceforge
- your institution’s forge
- your home page
- ...

And they are all different / incompatible
Our mission

Collect, organise, preserve and share all the software that lies at the heart of our culture and our society.

Provides exactly

- availability
- traceability
- uniformity
We are working on the foundations
one infrastructure to build them all

- Mankind's memory
- Long term preservation
- Unique reference
- Software Wikipedia

- Reference repository
- Provenance
- Certification
- Security

- Reproducibility
- Traceability
- Open Access
- Software studies

- Universal SourceBook
- Reference examples
- Enriched source code
- Code documentation

Cultural Heritage
Industry
Research
Education

Software Heritage
Fostering wider education to computing

A global source referencing all software

- a *SourceBook* for technological *education*
- intrinsic persistent identifiers for stable course materials
- extensive access to real-world documentation
Supporting more accessible and reproducible Science

A global library referencing all software used in all research fields

- completes the infrastructure for Open Access in Science
- provides intrinsic persistent identifiers needed for scientific reproducibility
- enables large scale, verifiable Software Studies

Roberto Di Cosmo (INRIA/Paris Diderot)

Analysing large code bases

EvoLille 2015
The Knowledge Conservancy Magic Triangle

Legenda (links are important!)
- articles: ArXiv, HAL, …
- data: Zenodo, …
- software: Software Heritage tackles this
Need your help

make it easy to integrate your work
- development workflow
- publication workflow
- contribute importers

make it ok to integrate, from the legal point of view
- make licences explicit
- make licences of dependencies explicit

make it useful for research
- contribute to the API

help us make Software Heritage sustainable
- support/sponsorship
- open process and collaboration
Focus on the legal issues

a plurality of concerns

Who owns the rights to your research?

- articles, data, software
- too often forgotten: metadata
  - You own the software, but who owns the metadata?

we need to recover our rights

- it is possible!
  - compulsory exclusive copyright transfer for free
    - is illegal in France (art L. 131-4 of CPI)
    - is debatable in all jurisdictions
- see Free Scientific Publication
- paying the editors (OpenAire) is not a solution
Questions?

Subscribe to

mailing list: swh-science@inria.fr
https://sympa.inria.fr/sympa/info/swh-science