Learning from the future of component based repositories

Pietro Abate,
Ralf Treinen, Roberto Di Cosmo, Stefano Zacchiroli

PPS, Université Paris Diderot

CBSE 2012 - Bertinoro, Italy

Software Distributions as Components

A software distribution is a collection of packages.

- Packages are **reusable** software units which can be **combined** together in distributions

Software Distributions as Components

A software distribution is a collection of packages.

- Packages are **reusable** software units which can be **combined** together in distributions
- Packages are **independent** units and follow their own decentralized development and versioning

Software Distributions as Components

A software distribution is a collection of packages.

- Packages are **reusable** software units which can be **combined** together in distributions
- Packages are **independent** units and follow their own decentralized development and versioning
- The main difference with component based systems it that packages cannot be composed together to form larger components

Software Distributions as Components

A software distribution is a collection of packages.

- Packages are **reusable** software units which can be **combined** together in distributions
- Packages are **independent** units and follow their own decentralized development and versioning
- The main difference with component based systems it that packages cannot be composed together to form larger components
- Packages are used in different community as FOSS Linux distributions, BSD, Eclipse plugins, etc

Software Distributions

- Packages are described by a rich set of meta-data (dependencies, conflicts, . . .)

Software Distributions

- Packages are described by a rich set of meta-data (dependencies, conflicts, . . .)
- Repository: set of packages

Software Distributions

- Packages are described by a rich set of meta-data (dependencies, conflicts, . . .)
- Repository: set of packages
- Installation: *healthy* subset of a repository

Software Distributions

- Packages are described by a rich set of meta-data (dependencies, conflicts, . . .)
- Repository: set of packages
- Installation: *healthy* subset of a repository
- Installability problem: given a repository R and a package $p \in R$, does there exist an installation $I \subseteq R$ with $p \in I$?

Distribution evolution

quality vs freshness

FOSS distributions are constantly under pressure

- strict time based release cycle
- provide a rock solid platform and satisfying user experience.

During the release cycle a distribution is in continuous state of flux.

Sometimes packages are broken (not installable)

- Transient problems
  - Packages have to be recompiled on all architectures : packages can be broken while waiting for a package to be available.
  - Moreover, the compilation of a package can be delayed because it may depend on other packages that are not yet available. . .

Distribution evolution

quality vs freshness

FOSS distributions are constantly under pressure

- strict time based release cycle
- provide a rock solid platform and satisfying user experience.

During the release cycle a distribution is in continuous state of flux.

Sometimes packages are broken (not installable)

- Transient problems
  - Packages have to be recompiled on all architectures : packages can be broken while waiting for a package to be available.
  - Moreover, the compilation of a package can be delayed because it may depend on other packages that are not yet available. . .
- Non Transient problems : There are Package that needs update because there is a problem in the metadata of a package.

More Formally, Our questions

The present : checking the health of a distribution

Given a repository R and a package $(p, n) \in R$, is (p, n) uninstallable w.r.t R (distcheck) ?

More Formally, Our questions

The present : checking the health of a distribution

Given a repository R and a package $(p, n) \in R$, is (p, n) uninstallable w.r.t R (distcheck) ?

Looking at the future

Outdated	Given a repository R and a package $(p, n) \in R$, is (p, n) uninstallable w.r.t *all possible futures of* R?
Challenged	Given a repository R and a packages $(p, v) \in R$, how many package will be broken in a future repository W containing (p, w) and $v < w$.

More Formally, Our questions

The present : checking the health of a distribution

Given a repository R and a package $(p, n) \in R$, is (p, n) uninstallable w.r.t R (distcheck) ?

Looking at the future

Outdated Given a repository R and a package $(p, n) \in R$, is (p, n) uninstallable w.r.t *all possible futures of* R?

Challenged Given a repository R and a packages $(p, v) \in R$, how many package will be broken in a future repository W containing (p, w) and $v < w$.

To be made more precise

Define "possible futures of R"

Example : Will (*foo*,1) ever be installable?

```
Package: foo
Version: 1
Depends: (baz (=2.5) | bar (=2.3)),
        (baz (<2.3) | bar (>2.6))

Package: baz
Version: 2.5
Conflicts: bar (> 2.4)

Package: bar
Version: 2.3
```

Upgrading *baz* alone will not work since *baz* can only be upgraded to versions greater than the current version 2.5, hence *baz*(< 2.3) can never be satisfied. Upgrading *bar* alone will not work either since when we upgrade it to a version greater than 2.6 then we will get a conflict with *baz* in its current version.

Example : Challenged Packages

```
Package: foo
Version: 1.0
Depends: bar (<= 3.0) | bar (>= 5.0)

Package: bar
Version: 1.0

Package: baz
Version: 1.0
Depends: foo (>= 1.0)
```

What if we upgrade package *bar* ? Package *bar* challenges package *foo* for versions between $<= 3.0$ and $>= 5.0$

Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete : the problem can be reduced to 3-SAT

Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete : the problem can be reduced to 3-SAT

Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete : the problem can be reduced to 3-SAT
- Not-installability of a package w.r.t. all possible futures:

Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete : the problem can be reduced to 3-SAT
- Not-installability of a package w.r.t. all possible futures:
    - co-NP hard, since it allows to encode the original non-installability problem.

Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete : the problem can be reduced to 3-SAT
- Not-installability of a package w.r.t. all possible futures:
  - co-NP hard, since it allows to encode the original non-installability problem.
  - however, there are infinitely many possible futures of a repository!

What are possible futures of *R*?

First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)

What are possible futures of *R*?

First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)
- Newer versions of packages my change their relations in any way (quite pessimistic approximation)

What are possible futures of *R*?

First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)
- Newer versions of packages my change their relations in any way (quite pessimistic approximation)
- Packages may be removed.

What are possible futures of *R*?

First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)
- Newer versions of packages my change their relations in any way (quite pessimistic approximation)
- Packages may be removed.
- New packages may pop up.

What are possible futures of *R*?

First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)
- Newer versions of packages my change their relations in any way (quite pessimistic approximation)
- Packages may be removed.
- New packages may pop up.
- There are infinitely many possible futures.

Formalization of futures : Optimistic future

Definition

A repository F is an *optimistic future* of a repository R if any package in $F - R$ has empty dependency and conflicts.

Optimistic futures : if we advance a package q to a newer version then we may assume that this new version behaves as nicely as possible, that is it does not depend on any other packages and does not conflict with any packages.

Formalization of futures : Conservative future

Definition

depnames(R): names of packages used in dependencies in R. Let $R \rightsquigarrow F$. F is a *conservative* future of R if

$$names(F) = names(R) \cup depnames(R)$$

Conservative future : a future F of R is *conservative* iff F contains all packages of R, possibly in a newer version, and if F contains only packages whose names occur in R, either as names of existing packages or in dependencies.

Formalization of futures : Observational equivalence

- **open ended version space**: the number of possible future version being infinite, it is not possible to test them all extensively.

Formalization of futures : Observational equivalence

- **open ended version space**: the number of possible future version being infinite, it is not possible to test them all extensively.
- We notice that it is sufficient to consider a finite number of versions representative: for each package name we consider all version numbers that are explicitly mentioned, plus one intermediate, plus one that is beyond.

Formalization of futures : Observational equivalence

- **open ended version space**: the number of possible future version being infinite, it is not possible to test them all extensively.
- We notice that it is sufficient to consider a finite number of versions representative: for each package name we consider all version numbers that are explicitly mentioned, plus one intermediate, plus one that is beyond.
- Build quotient under observational equivalence: identifying versions that behave the same on all these unary predicates.

Formalization of futures : Observational equivalence

- **open ended version space**: the number of possible future version being infinite, it is not possible to test them all extensively.
- We notice that it is sufficient to consider a finite number of versions representative: for each package name we consider all version numbers that are explicitly mentioned, plus one intermediate, plus one that is beyond.
- Build quotient under observational equivalence: identifying versions that behave the same on all these unary predicates.
- Example : $(p, 5) \in R$
  Dependencies and conflicts in R on $(p, \diamond 9)$, $(p, \diamond 12)$, where $\diamond$ is any comparison.

Formalization of futures : Observational equivalence

- **open ended version space**: the number of possible future version being infinite, it is not possible to test them all extensively.
- We notice that it is sufficient to consider a finite number of versions representative: for each package name we consider all version numbers that are explicitly mentioned, plus one intermediate, plus one that is beyond.
- Build quotient under observational equivalence: identifying versions that behave the same on all these unary predicates.
- Example : $(p, 5) \in R$
  Dependencies and conflicts in R on $(p, \diamond 9)$, $(p, \diamond 12)$, where $\diamond$ is any comparison.
- Representatives of future versions of p can be found in three equivalence classes:

$$5, 6(\in]5, 9[), 9, 10(\in]9, 12[), 12, 13(> 12)$$

Wrapping up : Outdated Packages

- So far we have a finite set (but huge) set F of repositories.

Wrapping up : Outdated Packages

- So far we have a finite set (but huge) set F of repositories.
- Since Packages (p, n) in any repository in F are unique

Wrapping up : Outdated Packages

- So far we have a finite set (but huge) set F of repositories.
- Since Packages (p, n) in any repository in F are unique
- We can build a new repository ($\bigcup F$) by lumping together all possible futures into one big repository

Wrapping up : Outdated Packages

- So far we have a finite set (but huge) set F of repositories.
- Since Packages (p, n) in any repository in F are unique
- We can build a new repository ($\bigcup F$) by lumping together all possible futures into one big repository
- ☺Any F-installation is a $\bigcup F$-installation.

Wrapping up : Outdated Packages

- So far we have a finite set (but huge) set F of repositories.
- Since Packages (p, n) in any repository in F are unique
- We can build a new repository ($\bigcup F$) by lumping together all possible futures into one big repository
- ☺Any F-installation is a $\bigcup F$-installation.
- ☹There are $\bigcup F$ installations that aren't in any future repository.

The problem when lumping together all futures

- When considering $\bigcup F$: we have to exclude all installations that mix binary packages coming from the same source but different version.

The problem when lumping together all futures

- When considering $\bigcup F$: we have to exclude all installations that mix binary packages coming from the same source but different version.
- Key observation: Binary packages coming from the same source are synchronized !

The problem when lumping together all futures

- When considering $\bigcup F$: we have to exclude all installations that mix binary packages coming from the same source but different version.
- Key observation: Binary packages coming from the same source are synchronized !
- Solution: add (versioned!) provides and conflicts:

The problem when lumping together all futures

- When considering $\bigcup F$: we have to exclude all installations that mix binary packages coming from the same source but different version.
- Key observation: Binary packages coming from the same source are synchronized !
- Solution: add (versioned!) provides and conflicts:
- If (p, n) has source s: Add
  Provides: src:s $(= n)$
  Conflicts: src:s $(\neq n)$

The problem when lumping together all futures

- When considering $\bigcup F$: we have to exclude all installations that mix binary packages coming from the same source but different version.
- Key observation: Binary packages coming from the same source are synchronized !
- Solution: add (versioned!) provides and conflicts:
- If (p, n) has source s: Add
  Provides: src:s $(= n)$
  Conflicts: src:s $(\neq n)$
- Finally : One single `distcheck` run on a large repository to identify all outdated packages..

Outdated : Experimental results

- In October 2011 we run our tools on the testing distribution of Debian.

Outdated : Experimental results

- In October 2011 we run our tools on the testing distribution of Debian.
- We found 110 outdated packages :

Outdated : Experimental results

- In October 2011 we run our tools on the testing distribution of Debian.
- We found 110 outdated packages :
  - 60% of those packages were outdated because of ongoing transition from python 2.6 to python 2.7.

Outdated : Experimental results

- In October 2011 we run our tools on the testing distribution of Debian.
- We found 110 outdated packages :
    - 60% of those packages were outdated because of ongoing transition from python 2.6 to python 2.7.
    - 20% were outdated because of a single package package `kdebindings` that was broken as part of a larger refactoring process.

Outdated : Experimental results

- In October 2011 we run our tools on the testing distribution of Debian.
- We found 110 outdated packages :
  - 60% of those packages were outdated because of ongoing transition from python 2.6 to python 2.7.
  - 20% were outdated because of a single package package `kdebindings` that was broken as part of a larger refactoring process.
  - The remaining packages were broken because of outdated dependencies and these problems were not known to the developers and are now fixed also thanks to our contribution.

Wrapping up : Challenged Packages

- We have a finite set (but huge) set F of repositories

Wrapping up : Challenged Packages

- We have a finite set (but huge) set F of repositories
- For each package (p, v) we are interested to check those futures that contain the packages p with a version $w > v$.

Wrapping up : Challenged Packages

- We have a finite set (but huge) set F of repositories
- For each package (p, v) we are interested to check those futures that contain the packages p with a version $w > v$.
- We can check only one representative for observational equivalence class.

Wrapping up : Challenged Packages

- We have a finite set (but huge) set F of repositories
- For each package (p, v) we are interested to check those futures that contain the packages p with a version $w > v$.
- We can check only one representative for observational equivalence class.
- The number of versions to check is large and using a parallel algorithm takes up to 20 mins on a standard desktop machine.

Challenged Packaged : Results for Debian Lenny

Source	Version	Target Version	Breaks
python-defaults	2.5.2-3	≥ 3	1079
python-defaults	2.5.2-3	$2.6 \leq . < 3$	1075
e2fsprogs	1.41.3-1	any	139
ghc6	6.8.2dfsg1-1	$\geq$ 6.8.2+	136
libio-compress-base-perl	2.012-1	$\geq$ 2.012.	80
libcompress-raw-zlib-perl	2.012-1	$\geq$ 2.012.	80
libio-compress-zlib-perl	2.012-1	$\geq$ 2.012.	79
icedove	2.0.0.19-1	$>$ 2.1-0	78
iceweasel	3.0.6-1	$>$ 3.1	70
haskell-mtl	1.1.0.0-2	$\geq$ 1.1.0.0+	48
sip4-qt3	4.7.6-1	$>$ 4.8	47
ghc6	6.8.2dfsg1-1	6.8.2dfsg1+ $\leq . <$ 6.8.2+	36
haskell-parsec	2.1.0.0-2	$\geq$ 2.1.0.0+	29

Table: Top Challenged Packages in debian lenny

Conclusions

- Outdated Packages can give us an effective way to pinpoint those package that need manual intervention in order to be fixed. We routinely use our tools to analyse the debian testing distribution and signal problems when araise.

Conclusions

- Outdated Packages can give us an effective way to pinpoint those package that need manual intervention in order to be fixed. We routinely use our tools to analyse the debian testing distribution and signal problems when araise.
- Challenged package give us an effective way to predict the impact of the upgrade of a component in a repository.

Conclusions

- Outdated Packages can give us an effective way to pinpoint those package that need manual intervention in order to be fixed. We routinely use our tools to analyse the debian testing distribution and signal problems when araise.
- Challenged package give us an effective way to predict the impact of the upgrade of a component in a repository.
- Our analysis is general enough that it can be applied to other component base system with similar characteristic.

Conclusions

- Outdated Packages can give us an effective way to pinpoint those package that need manual intervention in order to be fixed. We routinely use our tools to analyse the debian testing distribution and signal problems when araise.
- Challenged package give us an effective way to predict the impact of the upgrade of a component in a repository.
- Our analysis is general enough that it can be applied to other component base system with similar characteristic.
- All our tools are free software, modular and available in major FOSS distributions : `http://mancoosi.org/software`

Questions?

Pietro Abate
pietro.abate@pps.jussieu.fr
http://mancoosi.org/~abate

http://www.mancoosi.org/