Strong Dependencies between Software Components

Paper by Pietro Abate, Jaap Boender, Robert Di Cosmo and Stefano Zacchirol

Presentation by Malte Schwerhoff

ETH, Software Engineering Seminar 2010
The paper

• **Domain:** Large component-based systems such as Debian (Debian 5 has about 22,000 packages)

• **Motivating problem:**
  – Component-based system has to be updated
  – Admin needs to know the criticality of the components to schedule the update

• **Contributions:**
  – Toolset for the **identification of important components** and their **relationships** in such systems
  – An **algorithm** computing the necessary information in a reasonable amount of time
### Debian package meta-data

<table>
<thead>
<tr>
<th>Package:</th>
<th>postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version:</td>
<td>2.5.5-1.1</td>
</tr>
<tr>
<td>Depends:</td>
<td>libc6 (&gt;= 2.7), libdb4.6, ssl-cert, libsasl2-2, libssl0.9.8 (&gt;= 0.9.8f-5), debconf (&gt;= 0.5)</td>
</tr>
<tr>
<td>Conflicts:</td>
<td>libnss-db (&lt;&lt; 2.2-3), small, mail-transport-agent, postfix-tls</td>
</tr>
</tbody>
</table>

- Requirements are **boolean formulas**: commas represent conjunctions, bars disjunctions

  ⇒ Package installability is a **SAT problem** (\(\mathcal{NP}\)-complete)
Existing: direct dependency

- **Package** $p$ *directly depends* on package $q$
  
  $$p \rightarrow q$$

  if

  $p$’s requirements contain $q$

- Introduced in 2004 by LaBelle and Wallingford

- **Ignores semantics** of the requirement formula
  
  - $<p; q, r; \rightarrow>, <q, -, r>, <r, -, q>$  ✗
  
  - $<p; q|r; \rightarrow>, <q, -, r>, <r, -, q>$  ✓

  \[
  \Rightarrow \text{Too weak to derive meaningful information}
  \]
New: strong dependency

• Package $p$ strongly depends on package $q$
  \[ p \Rightarrow q \]
  if
  every possible installation containing $p$ also contains $q$

• The definition is relative to a given *repository*: a set of packages with meta-data

• An *installation* is a subset of such a repository
Direct vs. strong dependency

• No difference in this case:

| Package: p | direct: \{p \rightarrow q, p \rightarrow r\} |
| Depends: q, r | strong: \{p \Rightarrow q, \Rightarrow r\} |

• Big difference in this case:

| Package: p | direct: \{p \rightarrow q, p \rightarrow r\} |
| Depends: q | strong: {} |

| Package: p | direct: \{p \rightarrow q, p \rightarrow r\} |
| Depends: r | strong: {} |

We can install \{p,q\} or \{p,q\}, hence \(p\) has no strong dependencies.
Direct vs. strong dependency

| Package: p | Package: r |
| Depends: q | Depends: s |
| Conflicts: p |

Direct dependencies: \{p \rightarrow q, p \rightarrow r, r \rightarrow s\}
Strong dependencies: \{p \Rightarrow r, p \Rightarrow s, r \Rightarrow s\}

Notice that the strong dependency relation is transitive.
Strong dependency computation

- **Naïve approach** requires checking $n^2$ SAT instances, unfeasible with $n = 22,000$ (packages in Debian 5)

- Possibilities to **reduce the search space**:
  1. All conjunctive dependencies of $p$ are strong dependencies.
     
     In Debian, most dependencies are conjunctive.
  2. All strong dependencies of $p$ are included in any installation of $p$.
     
     Hence, only regard the smallest installation.

$\Rightarrow$ Strong dependency computation took only about 5 minutes on a modern computer
Impact set

• The *impact set* of a package $q$
  \[ Is(q) := \{ p \mid p \Rightarrow q \} \]
  are the packages $p$ strongly depending on $q$.

• Note: this is a **under-approximation** w.r.t a given installation!
  
  – Given $<p; q|r; \rightarrow>$, package $p$ has no strong dependencies
  
  – But in an installation $I = \{ p, q \}$, updating $q$ can nevertheless have an impact on $p$. 
Results: Impact set correlation

• The authors
  1. Computed the strong and direct impact sets of all packages in all stable Debian releases from 1994 until 2009
  2. Computed the correlation between both impact sets using two different correlation indices
     • $\rho \approx 0.92$ (Spearman, insensible to exceptional values)
     • $r \approx 0.55$ (unknown, assumably sensible)

⇒ Large discrepancy between $\rho$ and $r$ hints at a small cluster of packages with large strong impact sets but only a small direct impact sets.
Results: Important packages

• Excerpt from the computed table for Debian 5

| Package p    | |strong Is(p)| | |direct Is(p)| |
|--------------|-----------------|--------|
| gcc-4.3-base | 20128           | 43     |
| libgcc1      | 20126           | 3011   |
| libc6        | 20126           | 10442  |
| ncurses-bin  | 7721            | 1      |
| libx11-data  | 6693            | 1      |

• Just stating the size of the impact sets hides dependency relationships between important packages
We say $p$ strongly dominates $q$

$$p \succeq q$$

if $p \Rightarrow q$ and if all packages in $\text{Is}(q)$ are also related to $p$
We say $p$ strongly dominates $q$

$p \succeq q$

if $p \Rightarrow q$ and if all packages in $\text{Is}(q)$ are also related to $p$

$\Rightarrow$ Strong dominance is too restrictive, we again lose valuable information.
Relative strong dominance

We say \( p \) strongly dominates \( q \) up to \( z \) percent

\[ p \succeq^z q \]

if \( p \Rightarrow q \) and where \( z \) is the fraction of packages depending on \( q \) but being unrelated to \( p \).

Here:

\[ p \succeq^{1/3} q \]

because of 3 packages depending on \( q \) only one is not coming over \( p \).
Authors identified 4 such clusters amongst the 20 most important packages.
Perspective applications

• **Quality assurance**
  
  – Efficiently detect if (deficient) packages can be removed from a repository.
  
  – Identify sensitive packages that are to be frozen earlier.

• **Upgrade risk evaluation**

  Plan timing of and safety measures for a system upgrade according to package sensitivity.

• **Release upgrades**

  Facing major version updates (e.g. Python 2.x to 3.x) current systems suggest to update all involved packages to their latest version.

  Strong dependency can distinguish between mandatory and optional updates.
Threats to validity

• Authors gave none

• Mine are
  – Only a single system has been studied.
  – Search space reduction possibilities might not always be given (conjunctive dependencies, small installations).
  – Definition of an “installation” as required by the efficient algorithm isn’t given.
    (However, it seems to be computable efficiently.)
Questions?
Efficient computation: algorithm

S ← {} /* strong dependencies */

for all p ∈ R do
    S ← S U conj_deps(p, R)
end for

for all p ∈ R do
    I ← install(p, R)
    for all q ∈ I do
        if (p,q) /∈ S ∧ strong_dep(p,q,R) then
            S ← S U {p,q}
        end if
    end for
end for
Future work (authors)

- **Impact sets** are an under-approximation, refine it to be more precise with respect to a given installation

- Debian’s **staged releases**
  - Debian maintains two development repositories
    - Packages are initially uploaded to **unstable**
    - If QA demands are met, moved to **testing**
  - Use strong dependency clusters to identify packages that should be moved to **testing** as a group
Future work (mine)

• **Formula caching**
  
  – Check if certain (sub-)dependency formulas occur repeatedly
  
  – If so, cache them to speed up the SAT solving
Some numbers

• From a 2006 paper by Mancinelli et al.
  – Over 200,000 inter-package relationships are defined in the Debian repository
  – Average dependency formula (transitive closure) has 400 literals
  – KDE’s dependency formula has 32,000 literals