A Historical Analysis of Debian Package Incompatibilities

Maelick Claes*, Tom Mens*, Roberto Di Cosmo† and Jérôme Vouillon†
* Software Engineering Lab, COMPLEXYS Research Institute, University of Mons, Belgium
Email: firstname.lastname@umons.ac.be
† Univ Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126 CNRS and INRIA, F-75205 Paris, France
Email: roberto@dicosmo.org, Jerome.Vouillon@pps.univ-paris-diderot.fr

***Abstract*—Users and developers of software distributions are often confronted with installation problems due to conflicting packages. A prototypical example of this are the Linux distributions such as Debian. Conflicts between packages have been studied under different points of view in the literature, in particular for the Debian operating system, but little is known about how these package conflicts evolve over time.**

This article presents an extensive analysis of the evolution of package incompatibilities, spanning a decade of the life of the Debian stable and testing distributions for its most popular architecture, i386. Using the technique of survival analysis, this empirical study sheds some light on the origin and evolution of package incompatibilities, and provides the basis for building indicators that may be used to improve the quality of package-based distributions.

I. Introduction

Many free software distributions (e.g., the Linux-based OS distributions RedHat, Debian, OpenSuse or Ubuntu) are highly successful repositories based on the central notion of a *package* management system. By providing precise metadata for each package, these distributions allow to compose highly flexible systems tailored to their user's needs.

An important part of this medatata are the *declared dependencies* of each package, which describe the other packages immediately necessary for its installation and execution. Another important part of the metadata are *declared conflicts* of each package, which describe the immediate incompatibilities with other packages. In principle one would like all packages to be installable together, and the packaging guidelines for Debian clearly suggest that conflicts should be used sparingly [1]. Nevertheless, there are still many why conflicts may arise [2]: two packages may want to control the same common resource (a library, a configuration file, a network port), two or more packages may provide incompatible implementations of the same functionality, and one can even find special packages (such as Debian package harden-servers) that are used to implement security policies by declaring conflicts with all other packages whose functionality may be abused.

Unfortunately, the interplay between *declared dependencies* and *declared conflicts* that, taken in isolation, make pefect sense, may end up preventing a user from installing together a set of software packages that he needs to use simultaneously [3], creating a defect in the repository. Identifying and resolving these issues is very important when maintaining a package repository, but unfortunately detecting such incompatibilities due to the interplay between declared dependencies and conflicts is algorithmically hard.

Only recently, efficient algorithms and tools have been proposed for detecting these incompatibilities [4], and one of these tools, known as comigrate has been specifically developed to prevent to a large extent the introduction of such incompatibilities [5]. Nonetheless, after a set of incompatible packages has been spotted, a distribution maintainer is still left with the complex and time-consuming task of finding the right course of action to resolve it: which of the hundreds of dependencies and conflict relations involved in the incompatibility needs to be modified? In which package metadata should one look to find it?

To provide help in this difficult and crucial task, we decided to perform an extensive analysis of a large package-based repository over a significant period of time, and study how package incompatibilities are introduced, evolve, and may get removed. Mining the *history* of the repository, and comparing some of the results with known issues, we are able to provide insight in the characteristics that are statistically significant to pinpoint the packages that are most likely to be problematic.

With our study, we aim to provide a basis for building future indicators and tools that may be used to improve the quality of package-based distributions. To this extent, we focus on the following questions. How can we identify potentially problematic packages in the distribution? When are incompatibilities introduced in, or removed from, packages? What causes (dis)appearance of package incompatibilities?

The case study that we have chosen to carry out such an empirical analysis contains two Debian Linux distributions (stable and testing) for the i386 architecture, over a 10-year time period (starting from January 2005). To the best of our knowledge, this is the first study focusing on the long-term evolution of package incompatibilities in the Debian distribution.

A replication package containing the data, scripts and results of our analysis is available online via the following URL: http://www.dicosmo.org/replication/msr2015-coinst.

[1]This work has been partially performed at IRILL, http://www.irill.org.

The remainder of this paper is structured as follows. Section II presents the context of the Debian package management system and introduces the necessary terminology. Section III provides some overall characterisation of the evolution of Debian package conflicts over time. Section IV presents the research questions and methodology, Section V reports on our empirical analysis, and Section VI discusses the results. Section VII presents some threats to validity of our research. Section VIII discusses related work, Section IX explores future work, and Section X concludes.

II. Context

A. About Debian

The Debian distribution is a coherent collection of free software, initially announced in 1993, with a first stable release in 1996. To facilitate maintenance and collaborative work, Debian is built out of individual *packages* maintained by independent developers. Over time, Debian has undergone an impressive growth, and today it contains tens of thousands of different packages, with over a thousand developers. While it has been ported to a multitude of architectures (see `www.debian.org/ports`), and supports several kernels, this article focuses on the GNU/Linux distribution for the i386 architecture only. This architecture is historically the first one for which Debian has been made available, and the most popular over time.

The development process of the Debian distribution is mainly organised around three collections of packages, called *releases*: stable, testing and unstable. stable corresponds to the latest official production release (see Table I), and only contains stable, well-tested packages. The testing distribution contains package versions that should be considered for inclusion of the next stable Debian release. A stable release is made by *freezing* the testing release for a few months to fix bugs and to remove packages containing too many bugs. unstable contains packages that are not thoroughly tested and that may still suffer from stability and security problems. This release contains the most recent packages but also the most unstable ones.

TABLE I
Stable production releases of Debian

Version	Name	Freeze date	Release date	# packages
3.0	woody		2002-07-19	
3.1	sarge		2005-06-06	about 15K
4.0	etch		2007-04-08	about 18K
5.0	lenny	2008-07-27	2009-02-15	about 23K
6.0	squeeze	2010-08-06	2011-02-06	about 28K
7.0	wheezy	2012-06-30	2013-03-04	about 36K
8.0	jessie	2014-11-05	in 2015	N/A

Because we are interested in studying the evolution of Debian development activity, our empirical study will primarily consider the testing release, as well as its impact on the stable release that is derived from it. The testing release corresponds most closely to a development version: package versions contained in it are candidates for the next stable production release.

B. Terminology

The Debian package management system relies on metadata stored in control files. Among others, the control file of each package P describes the *direct* relationships with other packages: *dependencies* indicate which other packages are directly needed to perform the installation of P, and *declared conflicts* indicate the packages for which it is explicitly known that they cannot be installed together with P.

However, this explicit declaration of dependencies and conflicts is only the beginning of the story. If a package P depends on Q, and Q depends on R, then installing P requires both Q and R, so the package manager needs to follow these declared dependencies transitively. Even if two packages P and Q do not declare a conflict, it may very well happen that they cannot be installed together. For example, P may depend on some P_2 and Q on some Q_2, with P_2 and Q_2 in *declared conflict*.

This is why in the literature, as well as in this paper, the term *strong conflict* is used to indicate that two (or more) packages can never be installed together, independently from what is explicitly declared as a conflict in their metadata [3]. In addition, we use the term *conflicting package* to refer to a package that has at least one *strong conflict* with another package.

It is important to stress that *strong conflicts* are not necessarily "bad": many packages may not be installable together "by design". But if such conflicts are not reported explicitly as *declared conflicts*, they should still be considered as "problematic": a user may be unaware of the impossibility to install both packages together, and during package evolution new and unexpected indirect *strong conflicts* may arise without the package maintainers being aware of them.

C. Mining Strong Conflicts

For the Debian i386 testing and stable distributions we have extracted daily snapshots during the almost 10-year period from 12 March 2005 (>14K packages) until 6 January 2015 (>42K packages). For each daily snapshot, we only considered packages included in the official Debian distribution. We excluded from our analysis those packages that belong to the contrib or non-free category due to a restrictive license or legal issues.

A major problem when analysing package *strong conflicts* is the sheer size of the package dependency graph: there are literally thousands of different packages with implicit or explicit dependencies to many other packages. As an example, the full graph for the Debian i386 testing distribution on 1 January 2014 contained 38,411 packages, 181,265 dependencies, 1,490 *declared conflicts* and 49,026 *strong conflicts*.

In [4] we addressed this problem by proposing an algorithm and theoretical framework to compress such a dependency graph to a much smaller one with a simpler structure, but with equivalent co-installability properties, which is called a *co-installability kernel*. The idea is that sets of packages are bundled together into a equivalence classes if all packages in the set do not have a *strong conflict* with one another, while the collection of other packages with which they have *strong*

conflicts is the same. Applying this algorithm to the Debian i386 testing distribution on 1 January 2014 results in 994 equivalence classes, and 4,336 incompatibilities between these equivalence classes.

The coinst tool (coinst.irill.org) was developed specifically for extracting and visualizing coinstallability kernels for GNU/Linux distributions. We used the output of this tool as the basis of our analysis.

For each daily snapshot, we used R scripts to browse and extract all names of packages contained in the *main* archive area (i.e., belonging to the official Debian distribution)[1]. To retrieve the information about the co-installation conflicts of these packages we used JSON output files generated by coinst with the command

```
coinst -conflicts conflicts -stats -o graph.dot Packages.bz2 >& log
```

Previous research used *strong conflict* graphs to determine appropriate solutions to package co-installation problems. These solutions, however, did not take into account the evolution over time of these *strong conflicts*. In our current work, we aim to determine to which extent this historical data provides additional information to understand and predict how *strong conflicts* evolve over time, and to improve support for addressing package co-installation problems.

III. Overall Characterisation

Let us start by presenting some plots and descriptive statistics characterising the evolution of *strong conflicting* packages belonging to the Debian stable and testing distributions.

Fig. 1 compares the daily evolution of the total number of packages (in blue) against the number of *strong conflicting* packages (in red). The evolution of the stable distribution (solid lines) clearly shows "plateaus" that start at the moment of a major public release of a new Debian version. This is quite normal, as the stable version of Debian is only allowed to incorporate security-critical changes after a release.

The testing distribution (dotted lines) is more interesting: the development process leads to a general linearly increasing trend, with some periods of stability or light decrease that start at the official freeze date of the testing distribution (dotted vertical lines), and end at the official date of the next stable public release (solid vertical lines). During these freeze periods only bug fixes are allowed or packages can be removed, while it is generally forbidden to add any new package or package version to the testing distribution.

Fig. 2 shows the evolution over time of the ratio of the number of *strong conflicting* packages in a snapshot over all packages in that snapshot. For the testing distribution (dotted blue lines) we observe that, starting from 2007 and with only a few exceptions, this ratio remains between 25% and 15%. We also observe a slight decrease over time, despite the fact that the number of packages continues to increase with

[1]The information for a given snapshot date <*DATE*> (using the format *YYYYMMDD*) is available on http://snapshot.debian.org/archive/debian/<*DATE*>T060000Z/dists/testing/main/binary-i386/Packages.bz2

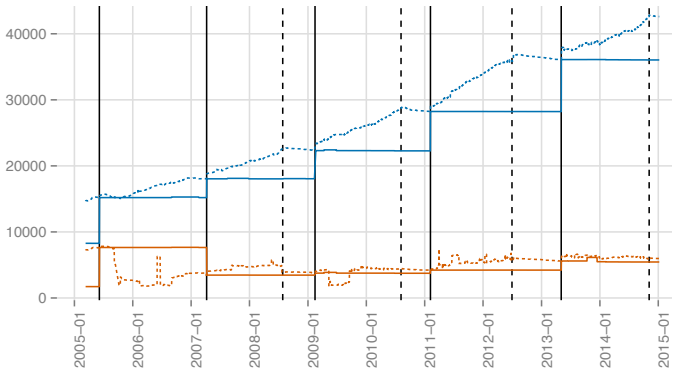

Fig. 1. Daily evolution of the total number of packages (in blue) and *strong conflicting* packages (in red) for the testing distribution (dotted coloured lines) and stable distribution (solid coloured lines) of Debian. Solid vertical black lines correspond to official dates of a stable public release. Dotted vertical black lines correspond to the freeze dates of the testing distribution preceding the stable release.

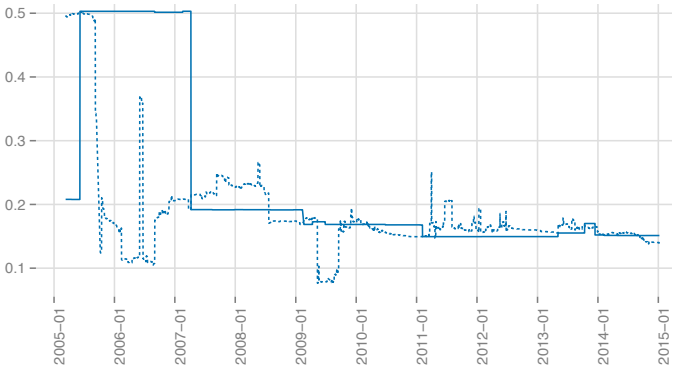

Fig. 2. Ratio of *strong conflicting* packages in snapshots of the testing distribution (dotted blue lines) and the stable distribution (solid blue lines).

each new major release: this corresponds to the fact that the Debian community actively works to keep *strong conflicts* at a minimum. For the stable distribution (solid blue lines) we observe the same evolutionary behaviour, combined with the presence of the "plateaus" corresponding to different public releases of Debian that were also found in Fig. 1. Finally, for the testing distribution we observe quite a number of "trend breaks", i.e., sudden increases in the number or ration of *strong conflicts* that appear suddenly and disappear after some time. This will be the subject of deeper investigation in Section V.

Fig. 3 displays, per daily snapshot of the testing distribution, the relative number of *strong conflicts* per package. Most of the time there are between 2,000 and 3,000 packages with exactly one *strong conflict*. This corresponds to a ratio of about 50% of all *strong conflicting* packages. There are much less packages having two *strong conflicts*, and even less with three *strong conflicts* or more.

Fig. 4 displays the same information but for the stable distribution. Again we observe the familiar "plateaus" and a ratio of between 50% and 70% of all conflicting packages that had only one *strong conflict* for the considered daily snapshots.

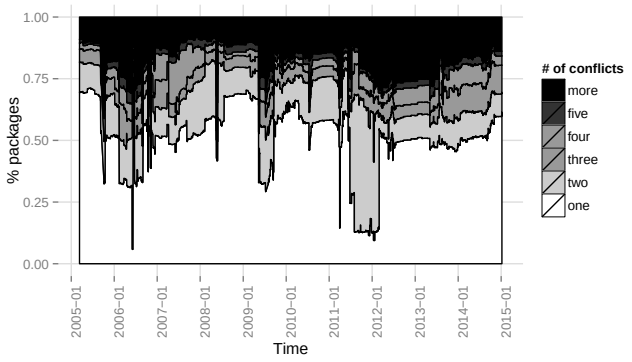

Fig. 3. Daily evolution of the number of packages in the testing distribution having a *strong conflict* with 1, 2, 3, 4, 5 or >5 packages.

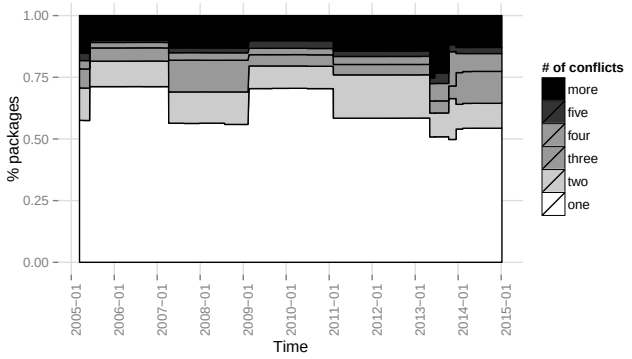

Fig. 4. Daily evolution of the number of packages in the stable distribution having a *strong conflict* with 1, 2, 3, 4, 5 or >5 packages.

Fig. 5 visualises the age of the packages present in the Debian testing distribution on 6 January 2015. There are in total 42,603 such packages (out of a total of 67,748 packages that existed at some time during the entire considered period). Gaps in the histogram are caused by the freeze periods during which addition of new packages is not allowed. The peak on the right represents all packages that have been there since the beginning of the considered period. It corresponds to 15.8% of all packages in the distribution of 6 January 2015.

Among all packages considered in Fig. 5, let us focus on only those 16,101 packages that had a *strong conflict* at least once in their lifetime. Fig. 6 visualises the number of conflicting days for these packages as a percentage of their total lifetime. We observe that 6,063 (i.e., 37.66%) packages were almost never conflicting (<5% of the time). Another peak is observed at the other side of the spectrum, were we find 21.28% of all packages (3,427 in total) that had at least one *strong conflict* >95% of the time. More specifically, 18.7% of all considered packages (3,009 in total) had *strong conflicts* during their entire lifetime.

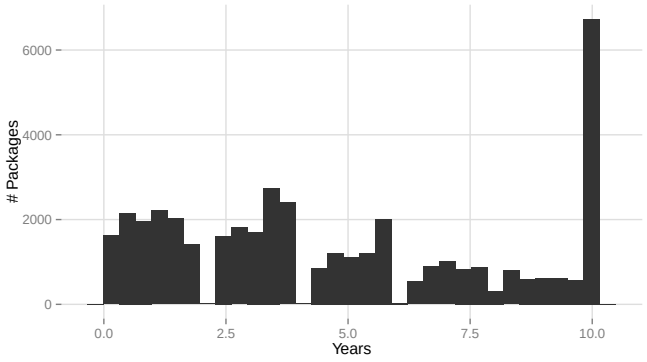

Fig. 5. Age (in years) of packages that were present in the Debian testing distribution on 2015-01-06.

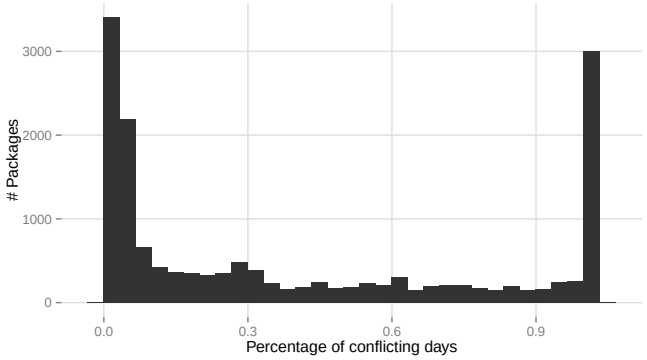

Fig. 6. Ratio of days that *strong conflicting* packages in the Debian testing distribution on 2015-01-06 were in conflict previously.

Fig. 7 shows the same information as Fig. 6, but for the stable distribution. Unsurprisingly, because packages in the stable distribution tend to be stable, *strong conflicting* packages in this distribution tend to remain in conflict during their entire lifetime.

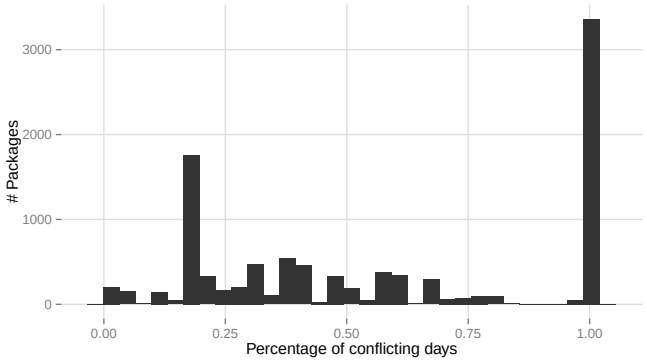

Fig. 7. Ratio of days that *strong conflicting* packages in the Debian stable distribution on 2015-01-06 were in conflict previously.

IV. Research Method

We will address the questions announced in the introduction by empirically analysing of the testing package distribution evolution of Debian's i386 architecture. More specifically, we will address each of the following research questions, in separate subsections. Answers to these questions will allow us, at the longer term, to come up with quality indicators and tool support for dealing with *strong conflicting* packages.

RQ_1 How can we identify problematic packages in the distribution?

RQ_2 How long does it take before a *strong conflict* is introduced in a package?

RQ_3 What is the effect of *strong conflicts* on the longevity of packages?

RQ_4 How long does it take before all conflicts get removed from a *strong conflicting* package?

RQ_5 What causes frequent appearance and disappearance of *strong conflicts*?

Because many of these research questions require us to study time-dependent data, we need to resort to the statistical technique of *survival analysis* [6], [7] to be able to answer research questions related to the introduction and survival of *strong conflicts* in packages (RQ_2 and RQ_4), as well as the survival of *strong conflicting* packages in the Debian distribution (RQ_3).

Survival analysis models the time it takes for events to occur and allows to take into account so-called *right-censored* data, for which it may be unknown whether the event occurred or not because it has not yet occurred or the subject has "disappeared". For example, if we study the survival of all packages during the period from January 2010 till December 2014, we do not know which of these packages may have become inactive after the end of the period of study.

Since we cannot assume a particular distribution of survival times, we need to resort to non-parametric survival analysis methods such as the Kaplan-Meier estimator [8]. The *survival function* models the probability of an arbitrary subject in the dataset to survive t units of time after the start of the study. Kaplan-Meier curves visualise the cumulative probability to survive from time zero. As a result, these curves start at value 1 (100% probability of survival at time zero) and continue to decrease monotonically over time.

All survival analysis results produced in this paper were obtained using R scripts that relied on the R package *survival* for computation and on the R package *ggplot2* for visualisation.[2]

V. Empirical Analysis

RQ_1 How can we identify potentially problematic packages?

As previously discussed, some of the conflicts present in the repository are there by design, but others are unjustified and harmful. Distinguishing the good from the bad ones is a complex task that has traditionally required a lot of manual investigation, with many issues going unnoticed for quite an extensive amount of time. In this research question, we look for a way of automating the detection of potentially problematic packages, and reduce the amount of effort needed to nail down real issues.

a) Aggregate Analysis

A natural approach to identify potentially problematic packages is to look for trend breaks in the evolution of the absolute or relative number of *strong conflicting* packages in the distribution: sudden increases in their number is a clear hint that some problematic package has appeared, and sudden decreases indicate that some problematic package has been fixed. Many discontinuities are clearly visible in Fig. 1 and 2, with peaks ranging from a few hundreds to over 4000 *strong conflicts*.

We retrieved all trend breaks that added at least 500 *strong conflicts*, using the coinst-upgrade tool described in [5] that is able to identify the root causes for the changes in conflicts between two repositories. We then manually inspected each trend break, and checked it against the information available from the Debian project, to determine the nature of the problematic packages and the degree of seriousness of the problem, and paired the events where each problematic package was first introduced and then removed.

The result of this analysis is summarised in Table II. For each problematic trend break, we report the date of the trend break, the number of new *strong conflicts* that were introduced at that date, the main root cause of the problem, the number of days it took to fix the problem, and the number of *strong conflicts* that were resolved by the fix. We also report whether the root cause of the problem would have been prevented by using one of the more recent tools comigrate [9] and challenged [10] that have been developed to improve the quality assurance process.

From Table II we observe that a few trend breaks were *dayflies* that were fixed the day after their introduction, while several took a few weeks, three took hundreds of days to fix, two have been fixed in several phases, and two still remain unfixed today. Most of these issues would have been captured by the comigrate tool if it would have been available at that time, and one issue could have been anticipated using the challenged tool.

Interestingly, a few relevant trend breaks *are not identifiable by any of the existing tools*, while a check for trend breaks in the aggregate analysis (as done here) would have drawn attention to them. This provides evidence of the added value of our approach.

b) Individual Analysis

Once a trend break has been spotted, one still needs to identify manually what are the potentially problematic packages. This process can be automated by studying their characteristics related to *strong conflicts* by resorting to three simple metrics for each package:

[2]See cran.r-project.org/web/packages/survival and cran.r-project.org/web/packages/ggplot2.

Trend breaks	Start date	Days to fix	Main root cause (manually identified)	Tool able to detect	Relevance
+4379/-4201	2006-06-02	19	updated x11-common conflicts with videogen	comigrate	medium
+2364/-2371	2011-03-30	1	new libgdk-pixbuf* conflicts with libgtk2.0-0	*this paper*	medium
+1658	2009-09-16	*not fixed yet*	new liboss-salsa-asound2 conflicts with all alsa tools	*this paper*	minor
+1279/-809	2005-10-15	120	reinstallable cdebconf conflicts with debconf	*this paper*	**serious**
+1268/-1270	2012-01-12	10	updated initscripts conflicts with sysklogd	comigrate	**serious**
+1188/-2442	2006-09-01	984	updated python conflicts with ppmtofb	challenged	minor
+1025/-1282	2011-06-19	45	updated initscripts conflicts with selinux-policy-default	comigrate	**serious**
+859/-1126	2012-06-23	1	new libopenblas-base conflicts with libatlas3gf-*	*this paper*	medium
+763	2011-04-26	*not fixed yet*	updated libsdl1.2debian conflicts with liboss-salsa-asound2	comigrate	minor
+758/-756	2012-05-18	1	updated netbase conflicts with ifupdown	comigrate	**serious**
+727	2013-05-05	*multiple dates*	new libopenmpi1.6 conflicts with libopenmpi1.3	comigrate	medium
same	*same*	*multiple dates*	less conflicts with man	comigrate	**serious**
+706/-732	2008-05-17	11	updated libldap-2.4-2 conflicts with libldap2	comigrate	minor
+682/-1074	2007-09-10	316	updated libpam-modules conflicts with libpam-umask	comigrate	minor
+633/-577	2013-07-26	19	updated initscripts conflicts with bootchart	comigrate	minor
+632	2007-04-08	*multiple dates*	new package libgif4 conflicts with libungif4g	*this paper*	minor
+536/-558	2011-03-21	31	new packages libhttp-* conflicts with libwww-perl	*this paper*	medium

TABLE II

AGGREGATE ANALYSIS OF TREND BREAKS AND THEIR MANUALLY IDENTIFIED ROOT CAUSE. THE FIRST COLUMN DISPLAYS +N/-M WHERE N IS THE NUMBER OF CONFLICTS INTRODUCED BY THE TREND BREAK AND M THE NUMBER OF RESOLVED CONFLICTS WHEN THE ROOT CAUSE IS FIXED.

- *minimum* number of *strong conflicts*
- *maximum* number of *strong conflicts*
- *conflicting days over mean*, i.e., number of days the package has more *strong conflicts* than $\frac{maximum+minimum}{2}$

The motivation for choosing these metrics is that one should focus on packages with a significant amount of *strong conflicts*, while at the same time ignoring those packages that have such a large number of conflicts only for a short period of time. Indeed, the latter case usually corresponds to transient problems, like the *dayflies* that we were able to identify in the previous aggregate analysis.

Potentially problematic package	minimum conflicts	maximum conflicts	conflicting days over mean
libgdk-pixbuf2.0-0	0	675	1349
libgdk-pixbuf2.0-dev	0	3320	915
liboss4-salsa-asound2	2963	3252	891
liboss-salsa-asound2	1741	2664	862
klogd	3	502	709
sysklogd	3	719	639
ppmtofb	0	719	639
selinux-policy-default	0	719	633
aide	0	719	633
libpam-umask	0	720	546
libldap2	0	719	546
libaws2.2	0	719	546
libaws-bin	0	2247	315
libhugs-ldap	0	2620	44
bootchart	0	598	31
libopenblas-base	0	1171	28

TABLE III

TOP 16 OF POTENTIALLY PROBLEMATIC PACKAGES IDENTIFIED BY THREE SIMPLE METRICS. PACKAGES LISTED IN **BOLDFACE** ALSO APPEAR AS A ROOT CAUSE IN TABLE II.

After ordering the packages with respect to our three metrics, we obtain a list of potentially problematic packages, of which we presented the first lines in Table III. Interestingly, we find back most of the packages that were already identified during the aggregate analysis (see Table II), with the important advantage that the proposed metrics can be computed fully automatically, and do not require any manual inspection.

RQ_2 How long does it take before a strong conflict *is introduced in a package?*

For our second research question, we are interested in the first time a *strong conflict* appears in a package. We hypothesise that newly introduced packages have a high likelihood of introducing *strong conflicts*.

To analyse this, we have to exclude all packages that are present at the first day of the considered period for which we have data, since we have no way of knowing when a *strong conflict* first appeared in them. This leaves us with 54,988 packages that are introduced somewhere during the considered timeframe.

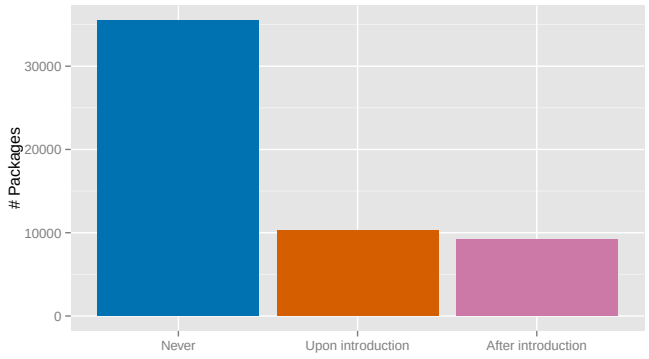

Fig. 8. Number of newly introduced Debian packages, classified according to when the first *strong conflict* was introduced for that package: never, upon package introduction, or after package introduction.

These packages can be classified into three different categories, summarised in Fig. 8 and discussed below.

1) Most new packages (64.59%, corresponding to 35,516 packages) **never** encounter a *strong conflict*.
2) For the 19,472 packages (i.e., 35.41%) that do encounter a *strong conflict*, in the majority of the cases (52.91%, corresponding to 10,302 out of 19,472 packages) a *strong*

conflict is already present **at the moment of introduction of the package**.

3) For the remaining 9,170 *strong conflicting* packages, a *strong conflict* was introduced at least one day (but often much longer) **after package introduction**. The distribution of the number of days before the first *strong conflict* is introduced has a median value of slightly below one year (326 days to be precise) and follows a decreasing trend (see Fig. 9).

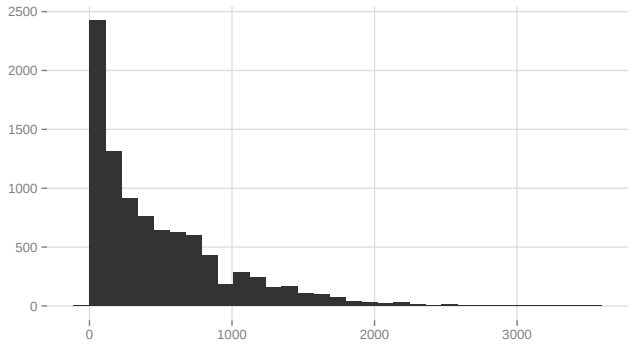

Fig. 9. Frequency distribution of the number of days (x-axis) before *strong conflicts* arise in newly introduced packages. Packages without *strong conflicts* or containing *strong conflicts* at the day of their creation are excluded.

It is important to note that the results in Fig. 9 are an underapproximation, since packages that have not encountered a *strong conflict* during the considered period may still become *strong conflicting* in the future. Survival analysis takes into account this probability. Fig. 10 shows the Kaplan-Meier curve. It shows the cumulative probability $S(t)$ that a package stays without conflicts for at least t years. The curve shows that a package has around 80% of chance of never gaining any conflicts in its first 10 years of existence. Moreover, as the curve appears to converge and because of its shape, the longer a package has survived without *strong conflicts*, the less likely it becomes that a *strong conflict* will appear.

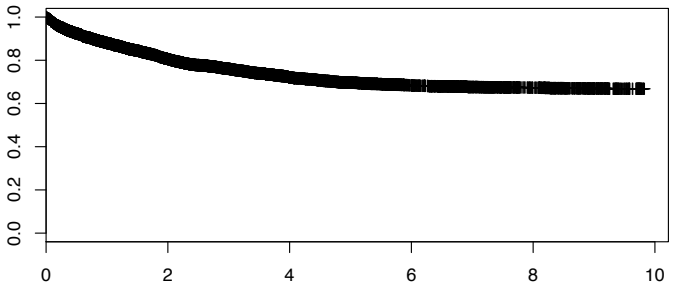

Fig. 10. Kaplan-Meier curve for the introduction of *strong conflicts* in non-conflicting packages. The time scale on the x-axis is expressed in number of years.

RQ_3 What is the effect of strong conflicts *on the longevity of packages?*

First, we study whether the *absence* of *strong conflicts* upon *introduction* of a package has an effect on its longevity. For the same reason as in RQ_2 we use survival analysis to answer this question. We analyse only those 54,988 packages that are newly introduced after the beginning of the considered period, because we cannot know the age of the other packages. Fig. 11 shows the Kaplan-Meier curves for the cumulative probability of the survival function. The green curve shows the survival probability for packages without *strong conflicts* upon introduction, the red curve shows the probability for packages containing *strong conflicts* at the time of package introduction.

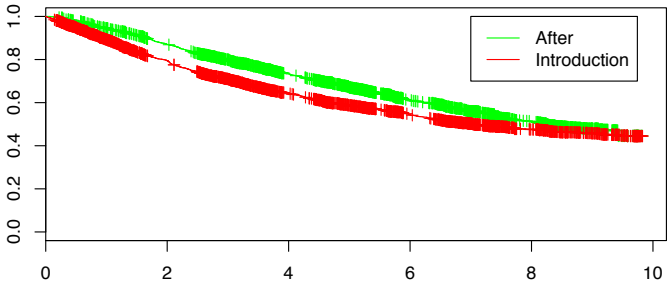

Fig. 11. Kaplan-Meier curves of the longevity (in years) of Debian testing packages with *strong conflicts* upon (in red) or after (in green) the time the package got introduced.

We used the survdiff function from the R package survival to test for difference with statistical significance between two survival distributions. This function implements the G^ρ family of nonparametric tests [11]. If $\rho = 0$ (as in our case), this becomes a log-rank test, also known as a Mantel-Haenszel test [12], [13]. Using this test, we found that packages for which a *strong conflict* has been introduced after introduction of the package live longer than packages that already had a *strong conflict* upon introduction. When looking at the figure, however, the difference is fairly small, and becomes smaller as the package survives longer.

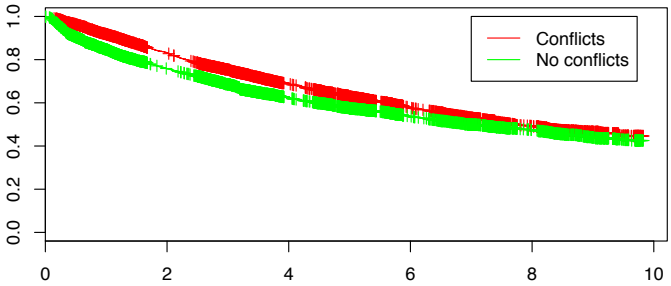

Fig. 12. Kaplan-Meier curves of the longevity (in years) of Debian testing packages without (in green) or with at least one *strong conflict* (in red) during their lifetime.

Secondly, we study whether the *absence* of *strong conflicts* during the entire observed lifetime of a package has an effect on its longevity. Fig. 12 shows the Kaplan-Meier curve

for the survival probabilities. Again, a log rank test reveals a difference with statistical significance: packages suffering from *strong conflicts* during their lifetime tend to live longer than packages without *strong conflicts*. This difference is in the opposite direction of what one would intuitively expect. When looking at the figure, however, the observed difference appears to be negligible.

Thirdly, we compare the longevity of packages that were *strong conflicting* during their *entire* lifetime (i.e., 100% of the time) with packages that only had *strong conflicts* occasionally (<100% of the time). Fig. 13 shows the Kaplan-Meier curve for the survival probability. Again, a log rank test reveals a difference with statistical significance: packages that are in *strong conflict* occasionally tend to live longer than packages that are in *strong conflict* during their entire lifetime. In this case, the difference is much more pronounced. Nevertheless, a package which is in conflict its entire lifetime has still more than 25% probability to survive more than 10 years.

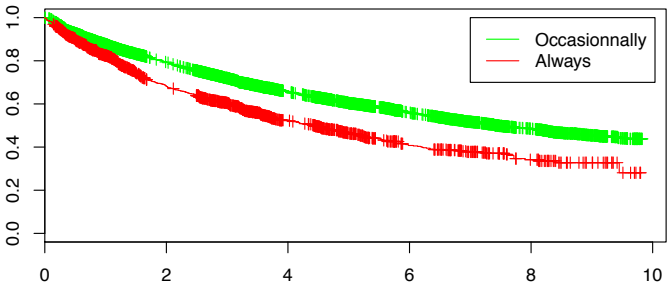

Fig. 13. Kaplan-Meier curves of the longevity (in years) of Debian testing packages with occasional *strong conflicts* (green) versus packages with *strong conflicts* during their entire lifetime (red).

RQ_4 How long does it take before all conflicts get removed from a strong conflicting *package?*

This question is the counterpart of question RQ_3 where we studied how long packages survive. With RQ_4 we analyse how long *strong conflicts* survive. For this analysis, we do not include those packages that were already in *strong conflict* at the beginning of the considered period. We therefore exclude 220 packages that already existed at the beginning of the studied period, that still existed at the end of the considered period, and that contained *strong conflicts* all their lifetime. Because of this, we might slightly underestimate the probability for a *strong conflict* to be long-lived.

Fig. 14 presents the Kaplan-Meier curve of the probability $S(t)$ of a package to stay in *strong conflict* at least t years. We make the distinction between *strong conflicts* that were introduced *upon* package introduction and those that were introduced *after* package introduction. The survival probability for the latter starts with a steep descent. Indeed, most *strong conflicts* introduced after package introduction do not last very long: 50% of them stay less than 24 days. In contrast, 50% of the *strong conflicts* that were already present upon package introduction stay more than 11 months! Similarly,

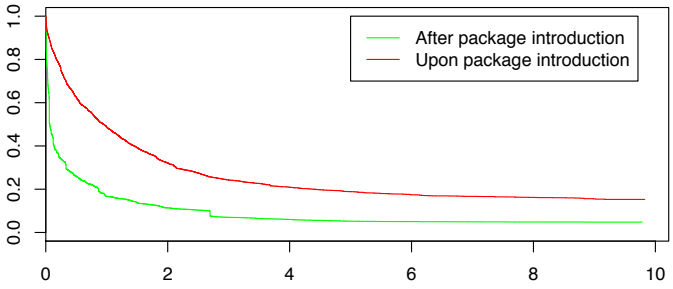

Fig. 14. Kaplan-Meier curve of the probability (over time) for all *strong conflicts* to get removed from packages.

strong conflicts added upon package introduction have a 15% probability to survive at least 10 years, while those added after package introduction have less than 5% probability of surviving 10 years or more.

Even if most *strong conflicts* are short-lived, some packages will continue to have *strong conflicts* for a long time, and it may not be possible to remove these conflicts. An example of such a package is courier-imap, which provides an IMAP mail server and which is in conflict with any other package providing an IMAP server.

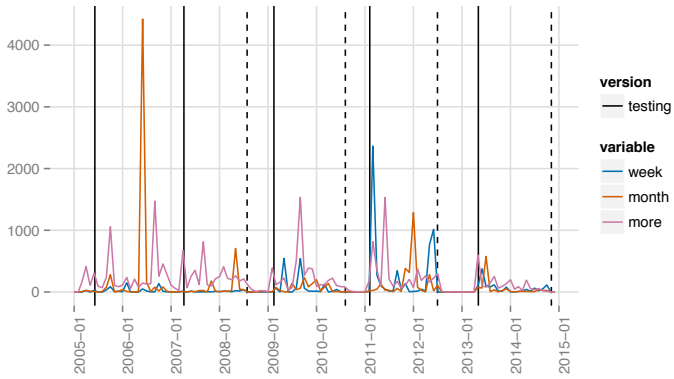

Fig. 15. Number of Debian testing packages for which at least one *strong conflict* got introduced and for which all *strong conflicts* were removed after, respectively: less than one week (blue); between one week and a month (red); more than a month (purple).

Because of the short-lived nature of *strong conflicts*, we analysed the history of the conflict resolution times in Fig. 15. As in Fig. 1, vertical lines indicate the start date and end date of each *freeze* period. Regardless of the resolution time, we observe that *strong conflicts* do not get introduced during freeze periods. This is indeed what we expected, since the freeze periods are meant to fix bugs and resolve problems, rather than introducing new problems. When comparing the dates of *strong conflict* introduction for those packages with short resolution times (less than a week) to those packages with longer resolution times (more than a week), we cannot reveal any specific pattern. Except perhaps for the fact that, since 2011, the introduction of *strong conflicts* in packages with short resolution times tends to be concentrated just before or just after a *freeze* period.

RQ_5 *What causes frequent appearance and disappearance of* strong conflicts*?*

We now focus on the events that cause a package to become *strong conflicting* or to loose all its *strong conflicts*.

During the considered period, there were 26,266 packages that became *strong conflicting* 49,768 times. Similarly, there were 25,178 packages that lost all their *strong conflicts* 51,248 times.

TABLE IV
DISTRIBUTION OF THE NUMBER OF TIMES EACH PACKAGE BECAME *strong conflicting*.

$< 50\%$	60%	70%	80%	90%	100%
1	2	2	3	4	20

TABLE V
DISTRIBUTION OF THE NUMBER OF TIMES EACH PACKAGE LOST ALL ITS *strong conflicts*.

$< 50\%$	60%	70%	80%	90%	100%
1	2	2	3	4	21

Tables IV and V show that most packages became *strong conflicting* or lost all their *strong conflicts* only once, while for only very few packages this happened many times (up to respectively 20 and 21 times). We manually analysed the packages with most repeated *strong conflict* additions and removals: erlang, openoffice.org-thesaurus-en-us and a few related packages. The explanations we found for these frequent state changes are twofold.

A first reason is that new versions of related packages can get introduced in the testing distribution at slightly different times. This introduces temporary incompatibilities because there is no explicit dependency between the involved related packages. The old Debian migration tools could not cope with these situations, while the more recent comigrate tool would prevent this. This happened twelve times for the packages erlang and erlang-doc-html, and four times for the packages openoffice.org-thesaurus-en-us and openclipart-openoffice.org (later renamed openclipart-libreoffice).

A second reason for repeated addition and removal of *strong conflicts* is that some packages have a large number of dependencies, and are hence more likely to be impacted. This was especially the case for OpenOffice packages, but also happened for erlang that depends on iniscripts which got transient *strong conflicts* three times.

VI. DISCUSSION

With RQ_1 we have shown that a simple approach based on monitoring trend breaks in the number of *strong conflicts* present in the distribution is able to identify several significant disruptions in the past history of Debian packages. Manual inspection of these issues revealed that most of them uncover medium to serious issues in the quality of the repository, as summarised in Table II. Many of these issues would have been prevented by using recent tools like `comigrate` [9] and `challenged` [10], which are now being gradually introduced in the Debian QA process. This constitutes strong evidence of the relevance of these tools, which may be adapted to other kinds of repositories. We also showed that some of the uncovered issues would have not been captured by any of the existing tools, while a simple check for sudden increases in the number of *strong conflicts* would spot them. This provides strong motivation for adding such a check in Debian's QA process, and more generally to the QA process for all GNU/Linux distributions.

For questions RQ_2, RQ_3 and RQ_4 we studied the relation between the presence of *strong conflicts* on the longevity of packages. To this extent we made use of the statistical technique of survival analysis.

RQ_2 revealed that, for all packages in the Debian testing distribution that were newly introduced during the considered analysis period, *strong conflicts* only occurred in about one third of them (35.41%). We also observed that, the longer a package has survived without *strong conflicts*, the less likely it becomes that *strong conflicts* will appear.

With RQ_3 we assessed the effect of *strong conflicts* on the longevity of packages. Packages that were introduced conflict-free tend to live longer than packages that already had a conflict at the moment they were introduced, but the observed difference is quite small. For those packages where *strong conflicts* did occur, in roughly half of the cases *strong conflicts* were already present at the moment of package introduction.

Finally, packages that are in conflict occasionally tend to live longer than packages that are always in conflict, with a clear observed difference. Hence, it makes sense to focus on packages that are always conflicting, to detect as early as possible those which need to be dropped.

With RQ_4 we studied the time it takes for all *strong conflicts* in a package to disappear. We observed that for those packages that already had *strong conflicts* upon package introduction, it takes much longer (if at all) before all these *strong conflicts* get removed than for packages that started off without any *strong conflicts*. Although this may seem contradictory at first, it is consistent with the intuition that a *strong conflict* present at the moment of package introduction may be actually needed to express intended incompatibilities, and does not necessarily represent a real defect. This also explains why many *strong conflicts* never get removed.

We also observed that, if a previously existing package becomes *strong conflicting*, it often does not take a long time before these conflicts get removed (less than 24 days in half of the cases), which is strong evidence that these conflicts are not intended incompatibilities, but defects in the repository that need to be fixed. Their present is clear indication of the need of incorporating better tools in the QA process.

Finally, our analysis of the packages that most frequently switched from conflicting to non conflicting (RQ_5) showed

again clearly the need for modern tools like comigrate or an improved version thereof that are able to prevent the appearance of new incompatibilities. Without such tools, several packages get impacted and fixed over and over again with every new version coming in.

VII. Threats to Validity

The foremost threat to validity relates to *generalisability*. We have restricted ourselves to Debian in this paper, but the lessons learned from our study of the evolution of package incompatibilities could be applied to other package-based software distributions as well. Such insights, as well as the tools and best practices used for reducing the extent of the problem (e.g., comigrate in the context of Debian) could help maintainers of other distributions to improve upon their practices and increase the quality of their repositories.

In most of our analyses, we had to exclude those packages that already existed before the considered 10-year period, because earlier data is unfortunately no longer available, and those packages that continue to exist after the considered period. If we could include these packages, the obtained results might change. We are fairly confident, however, that the main conclusions of our analysis will remain the same, given the fact that the evolution history over time remained fairly stable.

Our analysis is based on the output produced by the coinst tool. The risk that possible bugs in this tool may affect the outcome of our results is quite limited because the algorithms underlying coinst have been formally verified in Coq [14], and this tool has been used repeatedly in the past by different researchers. Moreover, conflicts identified by coinst can be independently checked using other existing tools, like dose-deb-coinstall from the Dose suite used regularly on Debian repositories [15].

Finally, the scripts that we have developed for our empirical analysis may still contain some bugs, and the obtained results may be biased by some simplifying assumptions we have made during our analysis.

VIII. Related Work

The Debian free software distribution is one of the largest organised collections of software packages today, and the availability of the full history of its evolution has made it an ideal object of study over the last few years, to the point that several infrastructures have been built to ease the extraction of information from this historical data: the Ultimated Debian Database (UDD) described by Nussbaum et al. in [16], and the Debsources archive described by Zacchiroli et al. in [17].

At the macro level, several characteristics of the Debian package repositories have been discussed in the literature. The small-world structure of the repositories is shown in [18] and [19]. The growth of the distribution, according to its package size and programming language usage has been first analysed in [20] and more recently in [17]. Changes in package characteristics such as age, maintainers, bugs and popularity are charted in [21].

Another series of studies focused on identifying uninstallable packages in a repository: since the pioneering work of [22] we know that, despite the NP-compeleteness of the general problem, efficient tools can be developed for identifying them. Galindo et al. [23] even propose to use software product line tools for this task. Strong dependencies and conflicts have been studied in [24], [3]. Incompatibilities among sets of packages, whose origins have been classified in [2], can be efficiently computed [4] and used to guide the acceptance of new packages in the distribution [5]. The current paper builds on the above described tools to perform an extensive historical analysis of these incompatibilities.

With motivations similar to ours, Bavota et al. studied the 14-year evolution of project dependencies and their likely impact on upgrade problems in the Apache ecosystem, consisting of 147 projects [25]. A significant difference with our work is that they performed an in-depth *manual* investigation of the issues reported in the bug tracker to identify the origin of incompatibilities, while our work relies to a large extent on advanced *automated* tools such as coinst and comigrate, since manual inspection is unfeasible at the scale of a package repository as large as Debian.

The statistical technique of *survival analysis* that we have used to respond to research questions RQ_2 to RQ_4 has been used by other researchers as well in the context of empirical software engineering. Samoladas et al. [26] used it to predict the survivability of open source projects over time. Scanniello [27] analysed dead code in five open source Java software systems. Kyriakakis et al. [28] studied function usage and function removal in five large PHP applications.

IX. Future Work

The different techniques employed in this incompatibilities mining effort may be aggregated into a metrics-based dashboard targeted to Debian package maintainers and users, replicated for all supported architectures, besides the i386 we studied here, and integrated into a platform such as Debsources, that has been specifically created to analyse and reason about the evolution of the Debian distribution [17].

The current empirical analysis was only based on metrics related to packages and their *strong conflicts*. A natural future line of investigation is to augment this data taking into account informations related to the package maintainers (such as proportion and size of maintained packages, experience [29], territoriality, turnover [30] and focus [31]), or to user adoption, exploiting the data collected by the Debian Popularity Contest project (popcon.debian.org).

Finally, we plan to investigate to what extent the findings extracted from, and the tools used for, analysing the Debian history can be reused in the framework of other package-based software distributions, such as NPM and CRAN [32].

X. Conclusion

The incompatibilities among packages known as *strong conflicts* are an important problem in package-based distributions, and have been studied in a series of recent research works [4], [5], [9]. Leveraging the coinst, coinst-upgrade and comigrate tools issued from this research work, we empirically analysed the evolution of *strong conflicts* among packages for all the available history of the Debian package-based software distribution for the i386 architecture, which spans a decade.

While the number of packages in the Debian testing distribution increases linearly, the ratio of packages with *strong conflicts* stays more or less constant, with occasionally important decreases or increases in the number of *strong conflicts*. This reflects the fact that the Debian maintainer make a specific effort to reduce as much as possible *strong conflicts*, which must be accepted only when they describe component incompatibilities that cannot be otherwise eliminated.

We investigated the likely causes of introducing or removing *strong conflicts* by relating them to the presence of declared dependencies and declared conflicts stored in the control file metadata of each Debian package. We observed that the introduction or removal of *declared conflicts* in a limited number of packages tends to spread across thousands of other packages because of direct or indirect dependencies.

Using the statistical technique of survival analysis, we investigated the moment and cause of introduction and removal of *strong conflicts* in Debian packages, as well as the relation with the packages' longevity. We found limited evidence that packages containing *strong conflicts* live longer than those without. We also found evidence that:

- packages that are always in *strong conflict* have a smaller survival probability than those who are not;
- the longer a package has survived without *strong conflicts*, the less likely it is that a *strong conflict* will appear;
- *strong conflicts* that are already present upon package introduction tend to stay present much longer than *strong conflict* that are added later;
- half of the *strong conflicts* that appear after package introduction stay a short amount of time (< 1 month).

These findings confirm the importance of adopting tools and techniques that prevent the introduction of *strong conflicts*. Without these tools, the historical analysis reveals that a lot of defects get regularly reintroduced, with peaks reaching tens of times for the same package.

Using metrics related to the presence, amount and duration of *strong conflicts*, we could identify several packages that have been reported as problematic by the Debian community in the past. We have shown how various of these issues would have been prevented by using recently developed tools, but several issues spotted by our metrics would not be captured by any existing tool. This is a strong motivation for introducing these metrics in the future into the repository quality assurance process. As an added bonus, the simplicity of our metrics makes them easily transposable to other package repositories.

Acknowledgments

This research was carried out in the context of ARC research project AUWB-12/17-UMONS- 3. We thank S. Zacchiroli and A. Serebrenik for feedback on an earlier version of this paper.

References

[1] T. D. Project, "Debian policy manual, section 7," https://www.debian.org/doc/debian-policy/ch-relationships.html, March 2015, retrieved March 2015.

[2] C. Artho, K. Suzaki, R. Di Cosmo, R. Treinen, and S. Zacchiroli, "Why do software packages conflict?" in *Int'l Conf. Mining Software Repositories*, 2012, pp. 141–150.

[3] R. Di Cosmo and J. Boender, "Using strong conflicts to detect quality issues in component-based complex systems," in *Indian Software Engineering Conf.*, 2010, pp. 163–172.

[4] J. Vouillon and R. Di Cosmo, "On software component co-installability," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, p. 34, 2013.

[5] J. Vouillon and R. Di Cosmo, "Broken sets in software repository evolution," in *Int'l Conf. Software Engineering*, 2013, pp. 412–421.

[6] D. Kleinbaum, *Survival Analysis a Self Learning Text*, 2nd ed. Springer, 2005.

[7] J. P. Klein and M. L. Moeschberger, *Survival Analysis: Techniques for Censored and Truncated Data*, 2013.

[8] P. M. E.L. Kaplan, "Nonparametric estimation for incomplete observations," *J. American Statistical Association*, vol. 53, no. 282, pp. 457–481, 1958.

[9] J. Vouillon, M. Dogguy, and R. D. Cosmo, "Easing software component repository evolution," in *Int'l Conf. Software Engineering*, 2014, pp. 756–766.

[10] P. Abate, R. Di Cosmo, R. Treinen, and S. Zacchiroli, "Learning from the future of component repositories," in *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, ser. CBSE '12. New York, NY, USA: ACM, 2012, pp. 51–60. [Online]. Available: http://doi.acm.org/10.1145/2304736.2304747

[11] D. P. Harrington and T. R. Fleming, "A class of rank test procedures for censored survival data," *Biometrika*, vol. 69, pp. 553–566, 1982.

[12] N. Mantel, "Evaluation of survival data and two new rank order statistics arising in its consideration," *Cancer Chemother Rep.*, vol. 50, no. 3, pp. 163–170, 1966.

[13] R. Peto and J. Peto, "Asymptotically efficient rank invariant test procedures," *Journal of the Royal Statistical Society*, vol. 135, no. 2, pp. 185–207, 1972.

[14] The Coq Development Team, *The Coq Proof Assistant Reference Manual – Version V8.2*, 2008. [Online]. Available: http://coq.inria.fr

[15] P. Abate and R. Treinen, "The dose-debcheck primer," https://gforge.inria.fr/docman/view.php/4395/8241/debcheck-primer.html, October 2012, retrieved November 2012.

[16] L. Nussbaum and S. Zacchiroli, "The ultimate Debian database: Consolidating bazaar metadata for quality assurance and data mining," in *Working Conf. Mining Software Repositories (MSR)*, 2010, pp. 52–61.

[17] M. Caneill and S. Zacchiroli, "Debsources: Live and historical views on macro-level software evolution," in *Int'l Symp. Empirical Software Engineering and Measurement*, 2014, p. 28.

[18] N. LaBelle and E. Wallingford, "Inter-package dependency networks in open-source software," Tech. Rep., 2004.

[19] J. Boender and S. Fernandes, "Small world characteristics of FLOSS distributions," in *Software Engineering and Formal Methods Collocated Workshops – Revised Selected Papers*, 2013, pp. 417–429.

[20] J. M. González-Barahona, G. Robles, M. Michlmayr, J. J. Amor, and D. M. Germán, "Macro-level software evolution: a case study of a large software compilation," *Empirical Software Engineering*, vol. 14, no. 3, pp. 262–285, 2009.

[21] R. Nguyen and R. C. Holt, "Life and death of software packages: an evolutionary study of debian," in *Center for Advanced Studies on Collaborative Research (CASCON)*, 2012, pp. 192–204.

[22] F. Mancinelli, J. Boender, R. Di Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen, "Managing the complexity of large free and open source package-based software distributions." in *Int'l Conf. Automated Software Engineering (ASE)*, 2006, pp. 199–208.

[23] J. A. Galindo, D. Benavides, and S. Segura, "Debian packages repositories as software product line models. towards automated analysis." in *Int'l Workshop on Automated Configuration and Tailoring of Applications (ACoTA)*, 2010, pp. 29–34.

[24] P. Abate, R. Di Cosmo, J. Boender, and S. Zacchiroli, "Strong dependencies between software components," in *Int'l Symp. Empirical Software Engineering and Measurement*, 2009, pp. 89–99.

[25] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the Apache community upgrades dependencies: an evolutionary study," *J. Empirical Software Engineering*, Sept. 2014.

[26] I. Samoladas, L. Angelis, and I. Stamelos, "Survival analysis on the duration of open source projects," *Information & Software Technology*, vol. 52, no. 9, pp. 902–922, 2010.

[27] G. Scanniello, "Source code survival with the kaplan meier estimator," in *Int'l Conf. Software Maintenance*, 2011, pp. 524–527.

[28] P. Kyriakakis and A. Chatzigeorgiou, "Maintenance patterns of large-scale PHP web applications," in *Int'l Conf. Software Maintenance and Evolution*, 2014, pp. 381–390.

[29] D. Izquierdo-Cortazar, G. Robles, and J. M. González-Barahona, "Do more experienced developers introduce fewer bugs?" in *Int'l Conf. Open Source Systems*, 2012, pp. 268–273.

[30] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. M. González-Barahona, "Using software archaeology to measure knowledge loss in software projects due to developer turnover," in *Hawaii Int'l Conf. Systems Science*, 2009, pp. 1–10.

[31] D. Posnett, R. D'Souza, P. Devanbu, and V. Filkov, "Dual ecological measures of focus in software development," in *Int'l Conf. Software Engineering*. IEEE, 2013, pp. 452–461.

[32] M. Claes, T. Mens, and P. Grosjean, "On the maintainability of CRAN packages," in *Int'l Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, 2014, pp. 308–312.