Managing Libre Software Distributions under a Product Line Approach*

Israel Herraiz, Gregorio Robles, Rafael Capilla, Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos (Madrid, Spain)
{israel.herraiz, gregorio.robles, rafael.capilla, jesus.gonzalez.barahona}@urjc.es

Abstract

Software product lines have already proven to be a successful methodology for building and maintaining a collection of similar software products, based on a common architecture. However, when the base system is heterogeneous and extremely large in size, an extra level of complexity is introduced that should be addressed with appropriate methods and techniques. A good example of this kind of systems is the product family composed by the software distributions composed by libre (free, open source) software, and based on Linux or BSD kernels. All of them can be considered as a part of a product line, based on a large collection of thousands of packages. One of the main problems faced by these distributions is the increasingly growing number of dependencies among packages, which is already caused problems, with a high risk of rendering the management of such large distributions impossible. In this paper we address some of challenges and main problems of Linux distributions when adopting a product line approach, with special focus to the maintenance and evolution of such systems.

1. Introduction

Large software distributions based on libre software[1] have become widespread in recent years, mostly because of the success and popularity of those based on Linux (and to some extent, BSD) kernels. Many different groups have put together their own distribution. Some of them are volunteers (such as Debian), while some others fall under the umbrella of a company, usually with a business model based on providing services around it. The most well-known example of the latter approach is Red Hat Linux, but other products such as Ubuntu (promoted by Canonical), Mandriva or SuSE (now owned by Novell) feature also a considerable end-user attention.

In recent years, specialized and customized distributions are also increasingly popular, ranging from those promoted by public administrations to those used by companies to satisfy specialized needs. For instance, there are several Spanish regional governments switching to libre software in educational environments, most of them using a slightly modified version of the volunteer-driven Debian GNU/Linux distribution. With respect to companies, Maemo (the application development platform for the Nokia 770 Internet Tablet) can be a good example. Nokia has been able of completing a Debian-based distribution with some in-house developed software, shipping the result as the base software of the 770. With these antecedents, it would not come as a surprise that the use of end-user or product specific software distributions, based on libre software distributions, become a significant trend over the next years.

However, these collections of software are usually large (in the range of tens, or even hundreds of millions of lines of code [9]), composed of hundreds or thousands of heterogeneous packages developed by independent teams, and with an increasingly number of interdependencies [7]. Several problems arise from this size and complexity, which are been faced by the teams putting together and maintaining those distributions [4].

To better understand those problems, and find solutions to them, we propose to apply the notion of product lines to the construction of libre software distributions. Distributions, and the packages composing them, can be considered as a huge product line with hundreds

*The work of Israel Herraiz, Gregorio Robles and Jesus M. Gonzalez-Barahona has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033547), QUALOSS (FP6-IST-5-033547) and QUALIPSO (FP6-IST-034763) projects, and by the Spanish CICyT, project SobreSalto (TIN2007-66172). The work of Rafael Capilla is partially funded by the PILOH project of the Spanish Ministry of Education and Research programme under grant number URJC-CM-2006-CET-0603.

[1] In this paper we will use the term "libre software" to refer to any software licensed under terms compliant with the FSF definition of "free software", and the OSI definition of "open source software".

of dependencies and configurable options [5]. From this point of view, we analyze the evolution of the software packages in some releases of a well known distribution (Debian GNU/Linux), as well as the number of dependencies among them. This will serve to study how dependencies can be dealt with in a novel way, in order to better support their evolution and scalability when new releases are distributed. The remainder of the paper is as follows. In section 2 we discuss the relationship of Linux distributions to software product lines and we highlight the problem for managing dependencies of Linux packages. Section 3 sketches the challenges and an agenda for future research work and section 4 discusses related work. We finally summarize our work in section 5 as well as future work.

2. Linux distributions as product lines

Distributions are usually organized in software packages (usual applications or libraries). Each package often depend on the functionality offered by some other packages. In order to work together, these dependencies have to be met at installation time or run time. Hence, in order to release new distributions under a product line approach, we need to define which of these packages can be considered as core components in the Product Line Architecture (PLA) and how they should be composed. Selecting the right configuration options during package installation involves also the definition of the binding times when releasing a particular distribution.

As a distribution grows and becomes more complex, the number of packages, the possible system configurations, and the dependencies among packages grow exponentially over time. Different types of dependencies are possible and thus the number of alternatives are high. In general, the number of dependencies does not grow at the same pace as the number packages, but superlinearly. Therefore, a higher level of complexity is introduced when configuring and installing new releases. In next subsection we analyze the complexity of these dependencies with a case study.

2.1. Dependency Analysis

To explore the dependency maintenance problems, we have selected the Debian GNU/Linux distribution as case study. Debian is one of the largest (and maybe the largest) software systems in the world. The size of its latest stable distribution (Debian 3.1) contains well over 200 millions of lines of code, and is a healthy system with good reputation of stability and matureness [9, 10]. Debian 3.1 contains around 19,300 packages, which are maintained by 1380 developers. These packages are not developed by Debian developers themselves: their work consists basically on packaging and integrating the upstream software into a coherent GNU/Linux distribution. Furthermore, Debian is the base distribution for many other Linux distributions, as was already mentioned in the introduction.

For this study we have considered all Debian releases from its beginnings in the mid 1990s. We have used the standard Debian tool `apt-cache` for all the considered releases to obtain the necessary information for our analysis. We have counted the number of binary packages ready to be installed including the number of packages with different names[2], and the number of dependencies among those packages. Table 1 shows the results of our analysis. The first column refers to the release number while in the second column the release codename is given. The third column contains the release date, and the fourth column specifies the total number of binary packages. In the fifth column we provide the number of packages with different names (i.e. only one version per package is shown). The sixth column of table 1 represents the number of dependencies found among packages. The seventh and eighth columns show the ratio between the number of packages (RP) and dependencies (RD) of the last two releases. The last column gives the difference between the previous two columns (RP-RD), as a measurement of the difference between speed of growth of dependencies and packages.

Table 1 clearly shows the increasing growth rate of the Debian system. The number of packages has doubled in the last two releases, and the number of dependencies has exploded. This fact increases the complexity of the system, and makes it more difficult to maintain. The scalability and the customization activities for future distributions are also affected by this growth. Another empirical evidence of this rising complexity is that the periods elapsed between the latest releases is becoming larger when compared to older ones. In other words, as the number of packages and dependencies involved increases, releasing a new version becomes more complex, resulting in a delay.

When considered in detail, we can see that both the number of dependencies and packages is growing exponentially. The logarithm of the number of dependencies over the releases can be linearly approximated, with a coefficient of $r = 0.9850$. If we repeat this pro-

[2]Some packages may have different versions present in the distribution. In this metric we have counted each of those packages just once, regardless of the actual number of binary packages present because of the number of different versions.

Release	Codename	Date	# Packages	Distinct Versions	# Dependencies	RP	RD	RP-RD
1.3	Bo	1997-06-02	2088	1402	5111	N/A	N/A	N/A
2.0	Hamm	1998-07-24	2757	1946	7379	1.32	1.44	0.12
2.1	Slink	1999-03-09	3601	2691	10841	1.31	1.47	0.16
2.2	Potato	2000-08-14	5583	4311	19284	1.55	1.78	0.23
3.0	Woody	2002-07-19	10771	8693	44164	1.93	2.29	0.36
3.1	Sarge	2005-06-05	19300	15305	96981	1.79	2.20	0.40

Table 1. Number of packages and their dependencies in several of the stable releases of Debian. RP is the ratio between the number of packages of the last two releases. RD is the ratio between the number of dependencies of the last two releases.

cedure with the number of packages, the coefficient is $r = 0.9830$. Figures 1 and 2 show clearly this exponential growth.

In addition, the last column of table 1 shows the relative growth of the ratio of packages to dependencies. Therefore, it is clear that the relative growth of dependencies is higher than the relative growth of the number of packages, which means that dependencies are growing even faster than the number of packages over time: adding new packages increases the number of dependencies superlinearly.

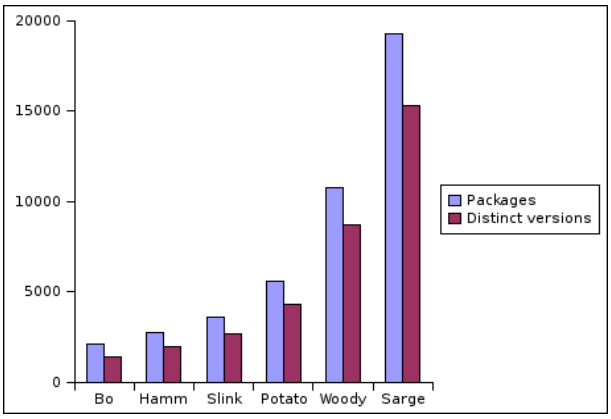

Figure 1. Number of packages in the distribution over releases.

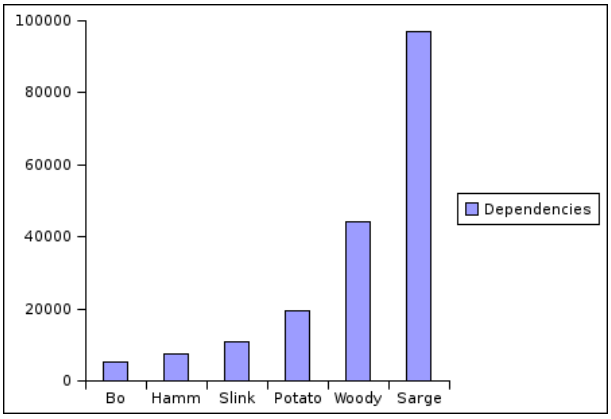

Figure 2. Number of dependencies in the distribution over releases.

2.2. Adopting a Product Line Approach

In the previous subsection, we have shown empirical evidence of the dependency growth problem. In such an scenario, the maintenance and evolution of those systems becomes more difficult. Package requirements, dependencies and configurable options may render the network of intertwined packages unmanageable. The difficulty of modeling and managing the overall set of dependencies is therefore large in many modern distributions.

To deal with this problem, we encourage the use of software product lines as a well-known software engineering approach for producing similar software products while taking advantage of hundreds of configurable options. Product lines exploit the amount of variability specified both in the architecture and in the software components that can be customized as late as possible (i.e. binding time).

Product lines have been widely discussed in the literature. Bosch describes in [1] the adoption of a product-line approach and the applicability to product family development. Product-line scoping is a key issue before initiating our product line to identify and select the products and features that meet the goals specified in the business case. Product-line architectures include the specification of a set of appropriate variation points for producing and configuring different products. This variability constitutes a powerful mechanism before product

derivation which makes easy the evolution of the products over time. The definition of the variation points, variants and dependencies must be carefully planned in order to avoid incompatible versions at the end of the product-line.

Classic feature models can be used for establishing a category of dependencies for distributions (similar to the way described in [6]. After that, some of those dependencies can be isolated and broken, in order to reduce the problem space and keep the increasing number of dependencies more stable. An analysis of existing dependency graphs should be carried out to analyze which links can be broken. In addition, variability must be introduced at a high level to reduce the dependencies during compilation or installation procedures.

3. Challenges for libre Product-Lines

In this position paper we have presented some of the problems of Linux-based distributions that may be tempered by adopting a product line approach. At present time there are many problems to be tackled and many solutions that can be explored. Improving the quality of Linux distributions implies certain changes at the architecture level to make them more flexible and adaptable. In addition, the aim of these new challenges is to reduce the enormous effort spent in their maintenance and evolution. In this work we identify, as a future research agenda, some key problems and research issues when adopting a product-line architecture-centric approach.

- Problem/Issue 1: Organizational issues of Linux product lines.
- Research goal 1: It seems interesting to study how typical organizational issues of product lines affect to Linux OSS contributors, that is how domain and application engineering teams should work.
- Problem/Issue 2: Scoping a Linux product line.
- Research goal 2: We need to re-examine the number and type of packages in Linux distributions to re-organize large product-lines into smaller ones as a way to make them more manageable and maintainable. Smaller but related product-lines add a multi-dimensional facet to the development process. This issue will impact on organizational and costs problems that have to be treated accordingly to the definition of the product-line.
- Problem/Issue 3: Linux distributions as a product-line architecture (PLA).
- Research goal 3: The design of a suitable PLA for Linux-based systems is a key goal in which we need to specify the variability (where and how) in the architecture. The definition of good PLAs will have a strong impact in maintenance and evolution. Also, the flexibility of the future architecture has to be analyzed to predict better future changes or new product configurations. In addition, the separation of concerns of the Linux software architecture should be analyzed to decouple the complexity of such distributions.
- Problem/Issue 4: Binding time.
- Research goal 4: An analysis of which binding times would be better to deploy and to re-configure a Linux system is required to provide flexible and configurable design choices and to increase the customer satisfaction.
- Problem/Issue 5: Manage and visualize dependencies among packages.
- Research goal 5: The management and visualization of the many dependencies among packages is not a trivial task, as the dependency graph may have to consider multiple packages with transverse dependencies or even loops. Some efforts towards solving this problem can be found in [5]. After visualizing the dependency graph, we need to analyze how and where to break such huge dependency clusters in order to decrease the complexity of large Linux-based distributions. The type of the packages to install can be used to categorize and visualize multi-faceted dimensions of the variability and use these types to re-organize the dependency network and to decouple the inter-dependencies among packages.

  These and other problems have to be addressed to facilitate the development of Linux-based systems as well as to estimate the effort with respect to other traditional development practices. The aforementioned research goals are proposed a basic research agenda to improve the development and evolution of libre software in the long term.

4. Other related work

There is some previous work that is related to our proposal besides the literature that has been referenced so far. Regarding the study of GNU/Linux distributions, it has been a field of study for some time now. For

instance, in [9], the releases of the Debian distribution have been studied for the last eight years. One of the findings of this research was that the size (in SLOC) of Debian is doubling with each new release, another evidence of its rising complexity.

The dependency problem in software distributions has been already addressed in [2, 8]. Here the approach is slightly different, as it is based on components and the goal is to have a tool to be used by *distribution editors*, who select components from a repository to create the distribution.

In [11], the authors discuss the notion of variability and they suggest a method for managing variability in software product lines. Variability can be realized at different binding times by means of different implementation mechanisms, such as suggested in [3]. The aforementioned method for managing variability [11] consists of four steps ans uses feature diagrams for describing the variation points. Different properties for variation points are defined. The feature graph defines the relationships and dependencies between variation points and variants. Related to this, in [6] the authors use a feature-oriented approach for modeling different types of dependencies for product line component design. Up to seven types of dependencies are defined for describing different types of relationships between the elements of the features tree. Feature dependencies are used for modeling different product configurations. Commonality and variability analysis becomes here a key technique for identifying common features among the assets we want to develop.

5. Summary and further work

In this paper we have stated that problems may arise in Linux-based distributions when the number of packages and dependencies grow exponentially and there are no mechanisms to lower the complexity. The scalability, maintenance and evolution for the future can be seriously compromised. We have identified this especially for dependencies among packages, as the growth of the number of dependencies is bigger than the one of software packages.

To address this problem we propose to use variability modeling techniques for a subset of packages and substitute some of the dependencies by appropriate variation points. A category of dependencies based on a feature model has to be defined and also, how this classification can be extended to the variety of packages. A high level description of the feature model with the variations points is planned to be introduced. This variability should be defined according to a product-line context under which products are delivered.

Finally, one of the expected benefits of this approach is to decrease the time to market for distributions by making them more maintainable.

References

[1] J. Boch. *Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[2] J. Boender, R. D. Cosmo, B. Durak, X. Leroy, F. Mancinelli, M. Morgado, D. Pinheiro, R. Treinen, P. Trezentos, and J. Vouillon. News from the EDOS project: Improving the maintenance of free software distributions. In *Proceedings from Intl track of WFS 2006 Porto Alegre*, Porto Alegre, Brazil, May 2006.

[3] C. Fritsch, A. Lehn, and T. Strohm. Evaluation variability implementation mechanims. *Proceedings of International Workshop on Product Line Engineering The Eraly Steps,PLEES2002*, pages 59–64, Nov. 2002.

[4] D. M. Germán. Using software distributions to understand the relationship among free and open source software projects. In *MSR*, page 24, 2007.

[5] D. M. Germán, J. M. González-Barahona, and G. Robles. A model to understand the building and running inter-dependencies of software. In *WCRE*, pages 140–149, 2007.

[6] K. Lee and K. Kang. Feature dependency analysis for product line component design. *Proceedings of the 8th International Conference on Software Reuse, ICSR2004*, LNCS 3107:69–85, 2004.

[7] F. Mancinelli, J. Boender, R. D. Cosmo, J. Vouillon, B. Durak, X. Leroy, and R. Treinen. Managing the complexity of large free and open source package-based software distributions. In *Proceedings of the Automated Software Engineering Conference*, pages 199–208, 2006.

[8] 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 *21st IEEE/ACM IASE*, pages 199–208, Tokyo, Japan, Sept. 2006.

[9] G. Robles, J. M. Gonzalez-Barahona, M. Michlmayr, and J. J. Amor. Mining large software compilations over time: Another perspective of software evolution. In *Third International Workshop on Mining Software Repositories*, pages 3–9, Shanghai, China, May 2006.

[10] G. Robles, J. M. González-Barahona, and M. Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy, July 2005.

[11] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. *Proceedings of the IEEE WICSA 2001*, pages 45–54, 2001.