Innovation Diffusion in Open Source Software

Preliminary Analysis of Dependency Changes in the Gentoo Portage Package Database

Remco Bloemen, Chintan Amrit, Stefan Kuhlmann, Gonzalo Ordóñez–Matamoros
University of Twente
PO Box 217, 7500 AE
Enschede, The Nethelands
<remco@coblue.eu> <c.amrit@utwente.nl>, <s.kuhlmann@utwente.nl>,
<h.g.ordonezmatamoros@utwente.nl>

ABSTRACT

In this paper we make the case that software dependencies are a form of innovation adoption. We then test this on the time-evolution of the Gentoo package dependency graph. We find that the Bass model of innovation diffusion fits the growth of the number of packages depending on a given library. Interestingly, we also find that low-level packages have a primarily *imitation* driven adoption and multimedia libraries have primarily *innovation* driven growth.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Measurement

Keywords

Innovation, dependencies, graph, Gentoo

1. INTRODUCTION

Diffusion is the process of market uptake of an innovation, the users of a particular innovation are called *adopters* [8]. Taking the open source projects as a 'market', these concepts can be applied to libraries and dependencies. For example, consider the projects with a graphical user interface (GUI). These have a demand for a GUI toolkit, and there are several competing implementations available (`Qt`, `GTK`, `Wx`, etc..). The introduction and uptake of a new GUI toolkit is a process of innovation diffusion and the projects that use a particular toolkit can be considered adopters of that toolkit. When talking about software projects such a relation is often called a *dependency*. In the next section we describe the Bass [1] diffusion model, we then describe the results of our analysis of fitting the Bass diffusion model on the Gentoo portage package dependency graph. This is followed by a discussion of the results of our analysis and finally we end the paper with the conclusions from our analysis and some discussion of possible future work.

MSR'14, May 31 – June 1, 2014, Hyderabad, India
ACM 978-1-4503-2863-0/14/05
http://dx.doi.org/10.1145/2597073.2597079

2. THE BASS DIFFUSION MODEL

To model the process of innovation diffusion, Bass [1] introduces two processes that propagate an innovation. The first processes involves individuals that decide to use an innovation based on their perception of its merits. The second process involves the word-of-mouth effect or the bandwagon effect: individuals adopt the innovation because they hear of the experiences of previous adopters. In reality however, everyone will be somewhere in between these two extreme types, but for the sake of modelling it suffices to consider the relative contribution of both types. It should be noted that for historical reasons Bass (1969) [1] and all later authors use the following terms; the first type are called "innovators", not to be confused with those actually inventing the innovation and the second type are called "imitators", not to be confused with those developing imitating offerings. Taking M to stand for the total market size and A to stand for the total number of adopters the Bass diffusion can be modelled with two simultaneous processes:

Innovators: Market participants not using use the innovation yet might decide to adopt the innovation. The rate at which this happens is p, the coefficient of innovation. The number of user that do not use the innovation is $M - A$, so the inflow of adopters is $p(M - A)$.

Imitators: Users of the innovation can express their fondness to market participants who do not yet use the innovation. This can influence them to adopt the innovation at a rate q, the rate of imitation. The number of user that do not use the innovation is again $M - A$, the chance of meeting someone that does use the innovation is proportional to $\frac{A}{M}$ so the inflow of imitators can be modelled as $q\frac{A}{M}(M - A)$.

When these two effects are combined the net inflow of adopters represented by the time derivative of A can be modelled as

$$\frac{\mathrm{d}A}{\mathrm{d}t} = p(M - A) + q\frac{A}{M}(M - A). \tag{1}$$

This first order homogeneous ordinary differential equation can be solved for $A(t)$ to give

$$A(t) = M\frac{1-\mathrm{e}^{-(p+q)t}}{1+\frac{q}{p}\mathrm{e}^{-(p+q)t}}. \quad (2)$$

The solution assumes the innovation is introduced at time zero, to account for this the substitution $t \to t - t_0$ is made. This results in four parameters, t_0, M, p and q that can be fit to the empirical data.

3. THE GENTOO PORTAGE DEPENDENCY GRAPH

The empirical data used is the time evolution of the Gentoo Portage package dependency graph [2]. This dataset contains the full dependency graph for every month since the project was initiated in 2000, the resulting graphs have a combined total of 1.3 million packages and 6.9 million dependency relations, with the largest graph having 15 thousand packages and 80 thousand dependency relations.

Special tools where developed to extract the time series of the number of adopters A_t for a given package. This time series was then fit to eq. (2) using Mathematica's `NonLinearModelFit`. The goodness-of-fit was analysed using an ANOVA table and calculated using the adjust coefficient of determination $\bar{R}^2$. The parameters were extracted from the fit, and confidence intervals were calculated under assumptions of normality. In table 1 the relevant parameters are presented with their mean value and a 95% confidence interval. Since normality was assumed, the confidence intervals ignore the $p \geq 0$ and $q \geq 0$ constraints.

The plots were drawn using a thick red line for the model and shades of red for the prediction bands. The thick red line was drawn according to eq. (2), with the mean values used as parameters. Then single value predictions bands were calculated for 90%, 95%, 99% and 99.9% confidence and were drawn in progressively darker shades of pink. They represent a prediction for where a single additional value would likely fall. According to the model and the uncertainty introduced by the fit, there is a 90% chance that it will fall in the innermost band, 95% chance that it would falil in the second band, etc. If the data fits the model properly, one would expect to see 90% of the points in the inner band. Finally, the empirical data points were plotted as black dots, connected with a thin vertical line - the residual error of the model.

4. RESULTS

From the entire list of packages, a few well-known (at least according to the author) packages where selected. The selection criteria was that the package must not have existed before (around) 2004, because the Gentoo Portage database was still too immature then, and the package must have gained a considerable number of dependers since its introduction. In table 1, we list some selected packages along with their parameters. In all cases the adjusted coefficient of determination $\bar{R}^2$ was more than 99%.

4.1 Imitator Driven Growth

The first package we consider is `git`, a modern revision control system that shows an imitator driven adoption. It's growth can be seen in fig. 1, the corresponding statistics are in table 1. The package first appeared just before 2005, it had around ten packages depending on it in 2006, twenty in 2008 and is currently used by almost three hundred packages. According to the Bass model, it will continue to grow to approximately 750 users. The innovator inflow is only 0.2% of the potential market per year, so one would expect $0.002 \cdot (750 - 300) = 1$ user to adopt `git` out of sheer innovation. Taking the analogy with persons, if someone from the 450 current non-users where to meet a random person from the entire 750 market, there is a $\frac{300}{750} = 40\%$ chance of meeting a user which can convince him/her to start using `git`. The chance of this happening is the imitator inflow $q = 0.73$. Therefore, the total number of users `git` can expect to gain from imitation this year is $450 \cdot 40\% \cdot 0.73 = 131$. Very much imitator driven!

Table 1: Results of fitting the Bass model.

Package	p	q	M
`git`	0.00 ±0.01	0.73 ±0.13	746 ±394
`libnotify`	0.05 ±0.08	0.72 ±0.30	103 ± 9
`udev`	0.01 ±0.01	0.50 ±0.12	200 ± 65
`cairo`	0.01 ±0.01	0.43 ±0.09	249 ± 44
`libmad`	0.18 ±0.14	1.13 ± 0.3	55 ± 1
`libtheora`	0.11 ±0.09	0.63 ±0.21	32 ± 1
`taglib`	0.22 ±0.03	0.04 ±0.06	62 ± 12

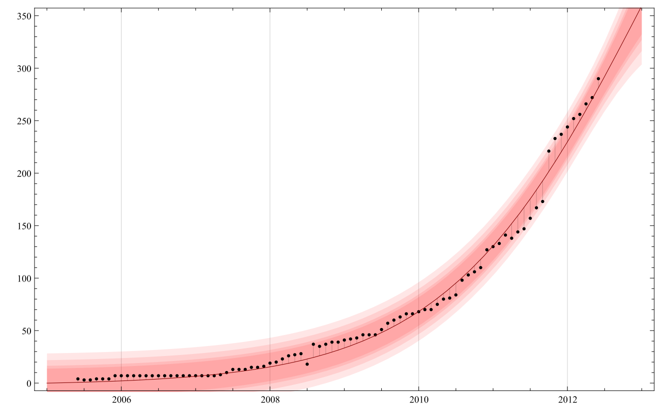

Figure 1: The imitation driven adoption of `git`.

The relative slowness of `git`'s growth and its dependence on imitator can be explained. Open source projects, and software project in general, consist of numerous large textual files containing source code. Changes made in one place can hugely and unpredictably affect other places. To complicate matters further, usually more than one developer works on the source code at the same time. There are competing systems such as `cvs`, `subversion`, `mercurial`, etcetera., but the basic functionality of maintaining version is provided by all of them. Thus two explanations can be derived for `git`'s growth: First, the revision control system is not a part that affects the products delivered by the open source project and second, there is little incentive to switch unless the new revision control system is proved to be superior.

`libnotify` is a library for notifications. In modern desktop environments applications may want to notify the user of certain events, for example a battery that is about to go empty, a new email or an incoming phone call. The adoption is relatively slow, despite its usefulness. A possible explanation is that the target applications all have their own custom solutions, which the developers are keen to keep.

`udev` is a device manager. Its task is to communicate

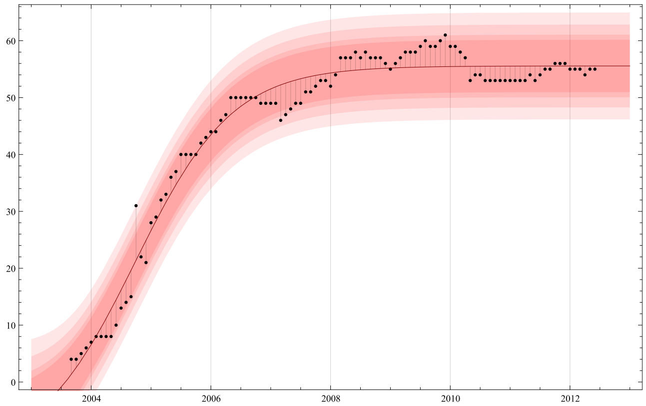

Figure 2: The innovation driven adoption of `libmad`.

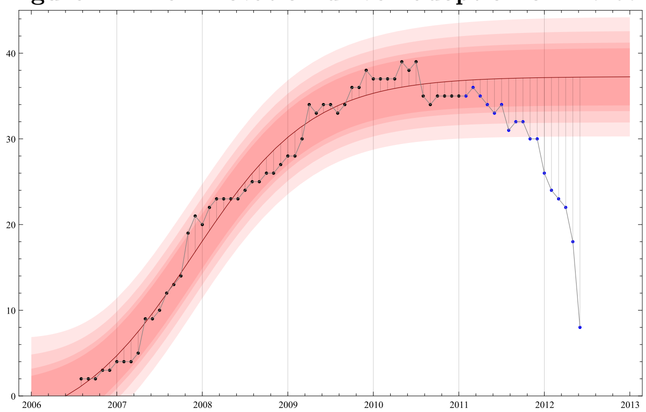

Figure 3: The rise and decline of `xulrunner`.

closely with the hardware drivers in Linux kernel to monitor any changes in the hardware configuration. It represents an architectural change in a very low level component, this might explain its slow imitator driven growth.

`cairo` is a graphics library. It provides facilities for drawing lines, circles, text and other graphics primitives and is used by user graphics-heavy projects such as user interface libraries. Much like `udev` it is an architectural change at a low level, this might explain its similar growth pattern.

4.2 Innovator Driven Growth

A typical example of innovator driven growth is given by `libmad`. The model is fitted resulting in fig. 2. Again, the data is neatly explained by a Bass diffusion process, in particular the rapid steep growth and the stable user base afterwards. The name is an acronym for "library for MPEG Audio Decoding" and the package provides a high quality mp3 decoder for use in multimedia applications. This might also explain the rapid growth of its adoption: multimedia applications can benefit a lot from good quality mp3 support.

`libtheora` is a library for the Schroedinger video codec. It implements a multimedia standard for use by video players. Just as with `libmad` there is a strong innovator driven growth.

`taglib` is a library that processes metadata from multimedia files. The package allows media players to read and store information such as artist and title from multimedia files. Again, like the other multimedia packages we observe rapid innovator driven growth.

4.3 Growth and Demise

The previous examples are all about projects that start and undergo a growth phase that can be explained by a Bass diffusion process. So far, the Bass diffusion model has appeared to give a very accurate explanation of the adoption of an open source software library.

A Bass diffusion increases monotonically, and never declines. However, not all packages follow this behaviour. Project `libmad` (see fig. 2) is a good example of this contra-behaviour. The package has an innovator driver growth that brings it close to its maximum in about two years. After that, the package's usage remains almost flat for years, and will do so indefinitely if it is a perfect Bass diffusion process. This is called the "maturity stage" in product life-cycle parlance.

A real product life-cycle will also include a "decline stage" where the product begins to become obsolete. The Bass innovation diffusion model does not account for this. In a deep sense it would not have to, once ideas spread they become part of our collective knowledge and will continue to be used by the new products being developed. But the Bass model was not developed for the spreading of ideas, it was developed in the context of marketing to model the adoption of products. Extending the Bass model to include obsolescence would be an interesting extension for future research.

The package `xulrunner` in the dataset is a nice example of a short but complete life cycle, see fig. 3. When the Bass model is applied naively and a least mean squares best-fit is made, the result is a poor fit. If one looks at the dependency growth of the package, the cause is clear: the package becomes obsolete, which the Bass model as presented in section eq. (2) does not represent. The decline of the package from approximately 2011 onwards can be seen as blue dots in the figure.

Excluding the blue dots from the data results in the Bass model fit from figure fig. 3. The fitness increases to $\bar{R}^2 = 99.54\%$ and the parameters have tighter and reasonable confidence intervals. This is strong evidence that the initial adoption of the package is a Bass diffusion process. To explain the last part, the model should be extended with an obsolescence term.

5. RELATED WORK

In 2008 Crowston et al. [4] published a comprehensive overview of academical research on open source software development. Of the 184 articles they cite, the vast majority of articles are case studies or surveys, with 4% of the articles describing the development of empirical instruments and/or measurements. Also, most articles look at the level of a particular group or project, with 4% looking at the societal level of interacting projects. Crowston et al. find no articles that develop instruments or measurements on the level of OSS packages, which this paper aims to do.

In their 2008 article Zheng et al. [10] analyse the dependency graph of the Gentoo Portage package database as it was in February 2007. The global structure of the graph is analysed in graph theoretic terms of sparsity, clustering coefficient, degree distribution and degree growth rate. The authors conclude that the graph can not be explained well by existing models of network growth and they propose two new models instead. Our study differs in two ways: First, we analyse the changes to the dependency graph over time, whereas Zheng et al. look at a particular instance in 2007. Second, we analyse and model the development of a given software package in the dependency network, instead of the overall development of the network. It can be interesting to study how the Bass diffusion model for the development of individual nodes corresponds to Zheng et al. model for the development of the whole graph.

Haefliger et al. (2008) [6] study code re-use on six open source projects, they conclude that there is extensive code re-use in open source software. The article proceeds by identifying the process of code re-use, such as the drivers for re-using and the tools used to find relevant code. The study by Haefliger et al. looks at a given project and how it re-uses existing components, whereas this paper looks at a given project and how it is being re-used by other projects. Additionally, we performed our analysis using a large set of automatically collected and processed empirical data.

Dedrick and West (2004) [7] and Chen (2006) [3] study the adoption process of open source software by (commercial) end users. Their focus is on the competitive economic strengths of open source software versus commercial software. The conclusion is that cost is the most important driver for open source adoption and freedom and extensibility plays a lesser role. The articles do not provide empirical data on the adoption process itself, which makes it hard to compare it to our present finding of a Bass diffusion adoption process.

6. CONCLUSIONS AND FUTURE WORK

The growth of the number of packages depending on a packages can be modelled as a a Bass diffusion process. Overall the Bass diffusion model gave very a good fit for most OSS projects. Using only four parameters, it was able to describe the growth curves from the empirical data. Full statistical rigour would require a more involved analysis using the methods from, for example, Carlos Escanciano (2006) [5], but given the amount of and quality of evidence found we can conclude that most OSS project do follow the Bass diffusion model.

As can be seen in table 1, the Bass parameters p and q are difficult to interpret and compare. A high p does not automatically mean an innovator driven growth: if the q value is also high then the result is simply a lot of growth. For the same reason it is also difficult to compare the p and q between packages. Mahajan et al. (1995) [9] suggests using $\frac{q}{p}$ and $q + p$, this represents the total adoption rate and an imitator/innovator ratio.

Analysing the package dependency graph and its changes over time can provide new insights. Our exploratory study provides some evidence for insights such as how multimedia libraries are being adopted through an innovator driven process with low-level architectural changes happening slowly and through imitation. Further studies could test these hypotheses.

The package dependency graph contains empirical data to test extensions of the Bass diffusion model - extended with discarders. The Bass model and the present analysis is formulated in terms of absolute number of users, but in most applications only sales figures are available. The amount of sales is the first derivative of the Bass model, hence the model is usually applied in its derivative form [9]. As a consequence the model only considers adopters, but does not consider *discarders*. In the `xulrunner` example, we see the package being discarded from 2011 onwards, providing insights in the discarding mechanism. The next step would be to collect more examples of packages being discard, look at their patterns of demise and develop a model of discarding to supplement the Bass model of adoption. One model could for example be the inverse of a Bass curve, this makes sense when the market share of the original package is taken over by a new package. The unique feature of dependency graph analysis to give absolute user numbers facilitates this. Such an analysis can also help in predicting the behaviour of certain projects - hence forewarning the particular project stakeholders.

The scale and complexity of the dependency graphs and open source innovation requires some care. Three notable issues became apparent in this study: First, In the open source community there is a lot of forking. It is not always clear whether a forked project constitutes the continuation of the original project or a separate new project. A more thorough study on the nature of forking could provide the insights to resolve this. Second, due to the public nature of open source development many immature or abandoned projects are visible in the larger datasets. This is good from a scientific perspective: it allows one to research projects from their early beginning and look at projects that failed to grow or became obsolete. But it clouds the 'big picture' with many projects that do not significantly contribute to the overall innovation. In large datasets one would have to devise a relevance metric to select the relevant metrics. Such metrics could be the number of developers, the number users or the number of dependees. Third, the sheer scale of the available OSS databases provide challenges for analysis. Specialist tooling is required to transform the raw data into more manageable formats.

7. REFERENCES

[1] F. M. Bass. A new product growth for model consumer durables. *Management Science*, 15(5):215–227, January 1969.

[2] R. Bloemen, C. Amrit, S. Kuhlmann, and G. Ordóñez–Matamoros. Gentoo package dependencies over time. *MSR'14*.

[3] S. Chen. An economic model of open source software adoption. *The Journal of Portfolio Management*, 2006.

[4] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, Mar. 2008.

[5] J. C. Escanciano. Goodness-of-fit tests for linear and nonlinear time series models. *Journal of the American Statistical Association*, 101(474):531–541, 2006.

[6] S. Haefliger, G. von Krogh, and S. Spaeth. Code Reuse in Open Source Software. *Management Science*, 54(1):180–193, 2008.

[7] Hawaii International Conference on System Sciences. *An Exploratory Study into Open Source Platform Adoption*, 2004.

[8] V. K. Narayanan. *Managing Technology and Innovation for Competitive Advantage.* Prentice Hall, Englewood Cliffs, New Jersey, 2001.

[9] F. M. B. Vijay Mahajan, Eitan Muller. Diffusion of new products: Emperical generalizations and manegerial uses. *Marketing Science*, 14:G79–G88, 1995.

[10] X. Zheng, D. Zeng, H. Li, and F. Wang. Analyzing open-source software systems as complex networks. *Physica A: Statistical Mechanics and its Applications*, 387(24):6190–6200, 2008.