Preface

This volume contains the papers presented at DWSE 2011: 1st Dutch Workshop on Software Ecosystems held on June 29, 2011 in Utrecht.

There were 11 submissions. Each submission was reviewed by at least 4, program committee members. The committee decided to accept 11 papers.

The creation of these proceedings was made possible by EasyChair, my, what a system.

So, my overall findings: we embarked with 22 students to simulate a scientific workshop. It was an absolutely wonderful experience, not only because I’ve met some of the most motivated students in this class, but also because the works produced are in some cases publishable and will add to the body of knowledge about software ecosystems. Well done!

July 6, 2011
Slinger Jansen
Utrecht
# Table of Contents

DIFFERENT REVENUE MODELS FOR APP PUBLISHERS FOR MOBILE PLATFORM ECOSYSTEMS ......................... 1  
Lucian Cancescu and Erik Van Ramshorst

The Eclipse Foundation Ecosystem and its internal members interactions 16  
Stamen Filchev and Ivelina Koseva

Multi-homing in the Apple Ecosystem: Why and how developers target multiple Apple App Stores .......................... 31  
Andrei Idu and Tommy Van De Zande

Implications of Openness in a Software Ecosystem: A Health Perspective 44  
Frederik Mijnhardt and Niels Van Den Berg

What do open source business models of software companies tell us about their ecosystem? .......................... 59  
Katharina Pietzka and Metodi Amov

Requirements Management in Software Ecosystems .................. 70  
Michiel Pors and Peter Van Stijn

A Small Sample Survey on Software Component Supplier Selection for Dutch Software Ecosystems .......................... 85  
Joey Van Angeren and Vincent Blijleven

Improving Activity in Communities of Practice through Software Release Management .................................. 100  
Kevin Van Ingen and Job Van Ommen

Assessing Developers Interdependency and Project Governance in the Open-source CyanogenMod Community .......................... 111  
Martijn Van Zetten and Wilbert Seele

Influences on developer participation in the Debian software ecosystem .......................... 126  
Eric Ververs and Rick Van Bommel

What Information Assets Should You Share with Your Software Ecosystem Participants? .......................... 138  
Wouter Westendorp and Christoph Wagner
Program Committee

Metodi Amov
Vincent Blijleven
Lucian Cancescu
Stamen Filchev  Utrecht University
Mick Hillio
Andrei Idu
Slinger Jansen  Utrecht University
Jaap Kabbedijk  Utrecht University
Ivelina Koseva  Utrecht University
Frederik Mijnhardt  Universiteit Utrecht
Katharina Pietzka  University Utrecht
Wilbert Seele
Joey Van Angeren  Utrecht University
Rick Van Bommel  Universiteit Utrecht
Tommy Van De Zande
Niels Van Den Berg  Utrecht University
Kevin Van Ingen
Job Van Ommen
Erik Van Ramshorst
Peter Van Stijn
Martijn Van Zetten
Eric Ververs  Utrecht University
Chris Wagner
Wouter Westendorp  Utrecht University
Different revenue models for app publishers for mobile platform ecosystems

Lucian Cancescu, Erick van Ramshorst

1 Utrecht University, Buys Ballot Laboratorium Princetonplein 5, Buys Ballot Laboratorium, 3584CC, Utrecht, Netherlands
{l.cancescu, earamsho}@students.uu.nl

Abstract. Business models describe the logic behind the way in which companies plan to earn money. The evolution of mobile platforms (such as Android) and the different business models they use, changed the way software producers develop new software for these platforms and offer new types of services. By attracting many developers to create applications and rich content for the platforms they are developing for, mobile platforms became entire ecosystems. Stand-alone strategies are not sufficient when companies’ success relies on the collective health of all organizations that influence the creation and delivery of the products. A very important part of these mobile platforms is the development community. We are interested how those app publishers generate income by using mobile platforms. In this paper we identified different revenue models being used by app publishers in mobile platforms. Furthermore, an expert review has been conducted with a Dutch app publisher, which led to the discovery of new identified revenue models. Finally, the selection of the most popular cross-platform publishers is graphed in a network diagram in order to show the preferences of app publishers towards a certain mobile platform.

Keywords: Software Ecosystems, Business Models, Mobile Platforms, Innovation.

1 Introduction

Compared to standard mobile phones, smartphones are powerful devices offering traditional wireless functionality as well native software applications, often called apps. They are able to run internet-based services, like e-mail, geo-location, and social networking while providing a good user experience [1]. Smartphones run mobile operating systems that are the platforms for all applications users run. The fast evolution of smartphones and mobile operating systems platforms in the last couple of years has opened the way for new business models for software companies [2]. [3] defines a business model as “A business model is a conceptual tool containing a set of
objects, concepts and their relationships with the objective to express the business logic of a specific firm. Therefore we must consider which concepts and relationships allow a simplified description and representation of what value is provided to customers, how this is done and with which financial consequences.” In other words, a business model represents an organization, its processes and activities, and describes how this organization creates, delivers, and captures value.

In this paper, mobile operating system platforms are considered as software ecosystems (SECO). [4], [5] define software ecosystems as: “A set of actors functioning as a unit and interacting with a shared market for software and services together with the relationships among them.” For a mobile platform ecosystem, the development community is vital for its success [1]. Apps (applications) have become more and more popular, and the developers are accountable for them. An app store is a sort of e-market available easy accessible to all customers that use a smartphone, and each mobile platform has its own means of delivering apps.

Therefore, a large part of the community of the mobile platform ecosystems consists of developers that publish apps. There are thousands of app publishers for mobile platform ecosystems like iPhone and Android. According to the report published by Distimo in 20112 most apps published for the four biggest mobile platforms are not for free. This suggests that a lot of app publishers are trying to generate income in these relatively young ecosystems. So, how do these app publishers make money through a mobile platform ecosystem? How does their business model look like?

What revenue models can be identified at app publishers in the mobile platform ecosystems?

A revenue model is one of the nine elements of a business model presented by [3]. It shows how a company is generating revenue streams. This led to the two sub questions: What are the main differences between the mobile platform ecosystems from an application publisher perspective? and Do app publishers use different revenue models per mobile platform ecosystem? Therefore, a comparison between different apps and the differences for those apps per mobile platform will be made to identify different type of revenue models. To scope this research, the four biggest mobile platforms have been selected namely: Android, iPhone, Symbian, and Blackberry. The main reason for this selection is due to the fact that they are the four biggest mobile platforms, and due to the scope of this research. [6] reports that Symbian OS is still the dominant in terms of world-wide market share with around 32% market share, followed by iOS, around 22%, Android around 17% and BlackBerry around 10%. Therefore, their combined total market share is 81%.

Furthermore, to answer our main research question a comparison of four mobile platforms should deliver sufficient data. Finally, the last goal of this research is to visualize the preferences of app publishers towards a certain mobile ecosystem, therefore a top 10 cross-platform publishers will be graphed in a network diagram.

Please note that in this research “mobile platform” and “mobile ecosystem” are both used with the same meaning in this research.

This paper is structured as follows. The first chapter introduces the topic and the main research question. The second chapter gives a summary of related literature

---

2 The Distimo report can be found on Distimo website: http://report.distimo.com
about existing topics that cover the four mobile platforms and a short description of existing business models used by app publishers. In the third chapter the research method is described. Finally, in chapter the findings of the research are discussed.

2 Related literature

In this chapter we describe three different aspects that need to be taken into account when choosing to develop for a mobile platform, namely: the different perspectives for mobile software producers, the way the revenue streams are modeled, and the difference in the mobile platforms and the change in the value chain. These three aspects are important to our study because they influence publishers’ choice of choosing a platform over another one and ultimately the business model that they apply.

2.1 Different perspectives for mobile software producers

According to [7] there are three aspects mobile software producers need to take into account when choosing to develop for a mobile platform, namely: the development process, distribution aspects, and types of devices.

From the development perspective, [8], [9] and [10] describe the overall process of mobile software development, and explain that every mobile software producer goes through the same steps in order to create an application and reach the customers with it. They make the clear distinction between developers and customers noting that “an increase or decrease on one side of the market induces a similar effect on the other side”. [11], [12], and [13] classify mobile platforms as open and close. This classification is based mainly on the software development kit (SDK) that every developer needs to use a SDK (Software Development Kit) in order to write apps for the mobile platform. The authors describe the closed model as “cathedral method”. Companies like Apple, RIM they have their systems closed. However, Nokia and Google opted for open platforms with the Symbian OS and Android, both open-source projects.

From the distribution perspective, [6], mentions the importance of a third-party between a developers and end-users. He classifies portals as centralized and decentralized. Mobile platforms such as Apple, Google and Blackberry offer a centralized type of portal for developers to publish their applications and users to download them. However the centralized can be seen as fully centralized or decentralized. That means in the case of Apple, there is only one App Store that customers access to purchase and download their apps and developers submit their applications to. While in the case of Google and Blackberry, customers are offered the possibility to download applications published on developers own websites. The other type of portals classification is the decentralized one. Nokia uses a decentralized portal. That means developers can publish their applications on every portal and users can choose where from to download their applications.

From the types of devices perspective, [14] argues that developers can face challenges when choosing to develop their software for diverse devices. The reason
for this is that devices have different hardware features and if developers choose to write the applications for a large variety of devices in some cases different builds of the same application need to be made.

### 2.2 How revenue models are modeled

As told in the introduction in this research our focus is mainly on the “Revenue Flows” from the Business Model Canvass by Osterwalder. The business model canvas is broadly discussed in their book [3]. In this research we are mainly focusing on the revenue streams. The revenue streams show what the customers are willing to pay for the services or products the business offers and how they are going to perform the transactions. However, some other aspects of the business model are mentioned in the results chapter. For example, different distribution channels can be found; the interview proved that customer relationships are vital for the success of an app.

[15] describe the elements, or attributes of which a revenue model consists. A revenue model consists of: Compensator, Effect, Causality, Rating and Charging model, Pricing, and Timing. The compensator provides the compensation (or payment), which is the customer or a third party. The effect tells what the effect of the compensation is, an effect might have no compensation (open source software), it a good or service may have a compensation provided by another good or service, for example Google delivers the search results as compensation. Finally, a good or service might be compensated financially. A compensation or revenue stream can be linked to one or more goods and services, which is called causality. For example if a customer has a subscription on a service, indirectly costs for the support service might be included. The Rating and Charging model concerns the measurement of the usage of software product. In the software industry this usually refers to the amount of licenses, and how this is charged. For example an all you can eat license allows a customer to use all software components of a software product. Pricings concerns different strategies and possibilities to price your software product. There is a distinction between fixed pricings, and differential pricing. Popular fixed pricing models are: pay per use, subscription model, and list price (one-time fee). Finally, timing tells at what time the compensation will happen and what additional conditions may apply.

### 3 Research Method

This research consists of three stages. The first part incorporates a literature study which has already been presented in the previous chapter. The second part consists of an empirical research. First, apps have to be selected in order to identify different revenue models and identify possible differences for those revenue models between the mobile ecosystems. Therefore, a selection based on usage, but also on the cross-platform availability will be used as main criteria. Our inclusion criteria are as follows: an app should be in the top 10 paid apps or top 10 free apps list of a platform, and available for two or more platforms. In order to improve the quality of the selection of the apps, monthly reports of Distimo are used. Distimo publishes
statistical information about apps and their app stores for different mobile ecosystems, for example, popularity ranking or average price paid per app. Furthermore, they provide a tool for developers which enable them to track the success of their app and compare it with its competitors. In this research usage is equal to the total downloads per app, with the assumption that the more downloads an app has, the more successful it is. By selecting apps that are available across different platforms, it is possible to research any differences that app publishers make in selecting their revenue model for a certain platform. In total, thirteen different apps have been selected.

In order to visualize the preferences of app publishers towards a certain mobile ecosystem, a top 10 cross-platform publishers will be graphed in a network diagram. By selecting cross-platform publishers it is possible to show how preferences of app publishers for selecting platform change over time. This will generate an overview of the four mobile ecosystems and the positioning of a certain app publisher between those four ecosystems. For example, if an app publisher publishes over 80% for Android, it will be shown more closely to Android in the network diagram. In order to realize this, data from Distimo will be used. They summed the top 10 cross-platform app publishers in their January 2011 report. In order to have the most adequate data, for each app publisher all published apps per app store have been counted. This has been done due to the fact that the data from Distimo was almost half a year old. Since the top 10 cross platform app are also responsible for a selection of the most popular apps for each of the four platforms, therefore, those 10 publishers are contributing significantly to all of these platforms. Finally, the third stage consists of performing an expert interview with a Dutch app publisher, McWood Solutions.

4 Results and Data Analysis

This chapter presents our findings and the data analysis. It is structured as follows. Firstly we compare different apps and their revenue models, based on free and paid apps and games. Then we present the findings from the interview with McWood Solutions and end up analyzing the top-ten cross-platform app publishers and show the graph with the results.

4.1 Comparison of different apps and their revenue models

The apps selected for our comparison are shown in Table 1, in the Appendix. From left to the right, the name of the app is stated, and the different pricing(s) of the app for Android, iOS, Symbian, and Blackberry OS. The pricings are shown because it can directly show some main differences. However, the pricing in the table, to be more specific is a ‘List Price’, which is a one-time buy price of an app. Note that a pricing model is one of the eight elements of a revenue model according to [15]. Most other elements described by [15] are constant for all apps, or do not generate a revenue stream directly. However, we believe that the pricing is the most important constitute of a revenue model, therefore revenue model is sometimes mentioned although in essence it might be a pricing element according to [15].
Our definition of pricing is: the price a customer pays at a one-time buy. Although for example a licensing model is viewed by [15] as a pricing ‘element’, we view them as different revenue models. The use of an ads-based revenue stream is different from a one-time buy revenue stream. Thirteen apps were selected in total, which can be divided in 3 different groups. The first group consists of apps which companies publish for free because their revenue model is depended on the usage of their service. Second, there is a group of apps which can be both paid and free, but companies want to stimulate the usage of their app in order to tempt a customer for a subscription or to tempt a customer to perform one-time buy. Finally, some of the most popular games were selected for the comparison.

4.2 Free Apps

Free apps are further categorized into two groups. The first category are apps which are free from ads, and do not intend to generate any kind revenue, directly nor indirectly (yet). These publishers, for example, want to contribute to the community. The first group is not interesting for this research, because they have no intention to generate any revenue. The second group consists of apps like Facebook and Twitter. Both are available for free, with no ads (yet) and both want to stimulate more usage of their service(s). For both Facebook and Twitter the more activity and more users will mean more ‘value’ for their business models.

Table 1. Revenue models of different apps across different platforms

<table>
<thead>
<tr>
<th>App</th>
<th>Platform</th>
<th>Android</th>
<th>iOS</th>
<th>Symbian</th>
<th>BlackBerry OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whatsapp</td>
<td></td>
<td>Free (1st year, then $1.99 p/y)</td>
<td>€0.79</td>
<td>Free (1st year, then $1.99 p/y)</td>
<td>Free (1st year, then $1.99 p/y)</td>
</tr>
<tr>
<td>Skype</td>
<td></td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>Facebook</td>
<td></td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>Twitter</td>
<td></td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>NYTimes</td>
<td></td>
<td>Free</td>
<td>N/A</td>
<td>Free</td>
<td>Free</td>
</tr>
<tr>
<td>RemembertTheMilk</td>
<td>Free / One month $2.99 $24.99 p/y</td>
<td>Free / One month$2.99 One year$24.99</td>
<td>N/A</td>
<td>Free (pro $25 p/y)</td>
<td></td>
</tr>
<tr>
<td>App Name</td>
<td>Free, Paid</td>
<td>Free, Paid</td>
<td>Free, Paid</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Angry Birds</td>
<td>Free, paid $0.99</td>
<td>Free $0.79</td>
<td>Free, Paid €2.00, Paid Map Packs €2.00</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Asphalt 5</td>
<td>Free, €3.46</td>
<td>Free €5.49</td>
<td>Only Paid €3.00</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Fruit Ninja</td>
<td>€0.87, €2.19 (HD version)</td>
<td>Free €0.79</td>
<td>Free</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Doodle Jump</td>
<td>€0.69</td>
<td>€0.79</td>
<td>Free</td>
<td>$0.99</td>
<td></td>
</tr>
<tr>
<td>Tom Talking Cat</td>
<td>Free, Paid €0.69, Avatars €0.69</td>
<td>Free, Paid €4.99</td>
<td>Free / Promo is free.</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Shazam</td>
<td>Free Untill 1 Jan 2012 promo, Paid €3.37</td>
<td>Free, Paid €4.99</td>
<td>Free / Promo is free.</td>
<td>Free / €5.99</td>
<td></td>
</tr>
<tr>
<td>Spotify</td>
<td>App Free, pay for (web)services</td>
<td>App Free, pay for (web)services</td>
<td>App Free, pay for (web)services</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Apps that aim to generate (additional) revenue directly

In this section apps will be discussed that are mostly available for free, but publishers want to attract customers for their service, so customers can subscribe for their licensing model, or try to tempt customers to perform an one-time buy. App publishers often launch a ‘free’ version to attract customers offering the app with a limited amount of functionality, or full functionality but a limited amount of usage. These are different ways to tempt a customer to buy the ‘full’ version. Skype, Whatsapp, Shazam, Remember The Milk, Spotify, and NYTimes are all examples. Skype hopes that it will attract more users, and that users will intensify their activity with Skype, and ultimately that users will buy (more) Skype credit. Skype follows their strategy which they also use for their desktop versions, which seems a logical
step. However, often apps available for free are For example, this could be a subscription for an online newspaper, music service, or task manager.

Whatsapp has the same pricing for Android, Symbian, and Blackberry OS. The first year users benefit of a free license, afterwards having to pay $1.99 per year. For iOS users pay once, but they cannot try a free version. A one-time buy versus a license mode, so, a different pricing model for the same app. Shazam offers a ‘free’ and ‘full’ edition, and for two platforms, Android and Symbian they offer a special promotion. The ‘full’ version can be downloaded for free and used until January the 1st of 2012. In ‘free’ version of Shazam you can full functionality, but with a limited amount of usage. Finally, Shazam pursues a strategy with different pricing per platform. Remember the Milk and Spotify are both apps which can be downloaded for free, however if users want to use their ‘full’ functionality they will have to get a paid license. These license models are platform independent, and are thus the same for all platforms.

4.4 Games

Five popular games were selected based on our criteria of popularity and cross-platform availability, namely: Angry Birds, Asphalt 5, Fruit Ninja, Doodle Jump, and Tom Talking Cat. Table 1 shows that publishers use different pricings for their games in the different mobile platforms. For games it is a common method as well to publish a free edition with limited functionality, or just a limited part of a game, and a ‘full’ edition with full functionality and full game content. Publishers use different names but most often they use either the word ‘free’ or ‘demo’ to address it concerns the limited edition, and use the name of the app by leaving ‘free’ or ‘demo’, out to address it concerns the ‘full’ edition. The word ‘HD’ behind the name of an app usually refers to an edition made specifically for most often a faster device, for example apps which are optimized for devices powered by a NVIDIA Tegra chip.

Angry Birds was initially launched for iOS, for which a free version was released with only a limited number of levels, and the paid version included the full game. Rovio, which is the publisher of Angry Birds, chose a different strategy for Android. It decided to launch the game for free, but with ads in it. For this moment it is assumed that the usage of ads is part of the pricings element, however we do not feel confident that it fits the description given by [15]. For Symbian, Rovio decided to try both pricing models. However, for some Symbian versions the game is free and for some not. Furthermore, for Symbian they tried another strategy by publishing the game in 4 parts (map packs) for € 2.00 each. In addition, Rovio started to sell games by using a different distribution channel than the market place from the corresponding mobile platform. Angry Birds for Android is now also available for Android without ads, for $0.99. Rovio is aware of the importance of customer relationship, and for example, made their app is easy to share by using social media. The success of their game made a strong brand image for Angry Birds, and they decided to sell ‘fan’ items like dolls and T-shirts. By using different distribution channels outside the mobile platform ecosystems, Rovio tries to generate additional revenue. In this example of Angry Birds by Rovio we can distinguish differences in pricing, different form of revenue model (free but with ads vs. paid), distribution channel, by using Amazon as
‘appstore’, and finally selling ‘fan’ items through Amazon. Probably Rovio made a lucrative deal by selecting Amazon as one of their partners, however the details of this partnership are not publicly available.

Our second game in our selection is Asphalt 5, one of the most downloaded race games on mobile platforms. We have discussed that (game) publishers often offer a ‘free’ and ‘full’ (paid) version. This holds for both their Android and iOS releases. However, in their Symbian release they only offer a paid version, but for a lower price than the paid versions for the other mobile platforms. This could be seen as a choosing another strategy although using the same revenue model to generate revenue (one-time buy). Our third game in our selection is Fruit Ninja, where there is another example of paid differences between pricing. The version released on Symbian is offered for free and does not include any adds, this might suggest that they would be a partnership deal between Nokia and HalfBrick Studio’s, the makers of Fruit Ninja, although there could no be found any evidence for this assumption. Furthermore, on Android they also published a ‘HD’ version offered for a higher price, optimized for devices which are powered by a NVIDIA Tegra Chip. The fourth game used in this comparison is Doodle Jump. The publishers, called GameHouse uses different pricing models, except for Symbian on which the game is available for free. Again this game offered for Symbian based devices includes no ads, so Nokia might be trying to attract more publishers and popularity of its operating system.

Finally, the last game analyzed is Tom Talking Cat which offers a free and a paid version, but also offers the ability to do an in-app purchase. An in-app purchase is a form of directing users to a website, or mobile platform marketplace / store to buy something. A user can buy additional avatars for the game for €0.69 per avatar.

To summarize, app publishers use different strategies and revenue models to generate revenue. App publishers release their apps for free to stimulate their usage, and therefore stimulate their own business model like Facebook. Other app publishers let users experience with a ‘free’ but limited edition and hope to tempt a customer to make a one-time buy decision. App publishers like Whatsapp Inc. try to attract customers for their licensing model. Furthermore, they differentiate their revenue model and pricing over the different app stores. Finally, app publishers also generate extra revenue by using other distribution channels and by selling ‘fan’ items. According to Popp and Meyer (2010) the revenue models identified in this research are part of a fixed pricing strategy.

4.4 Interview with Martin C. de Jong van McWood Solutions

In this research an expert interview with Martin de Jong, co-founder of McWood Solutions has been conducted. This section contains the main results. McWood Solutions was founded in 2009 and is a young start-up company which develops apps for iPhone, iPad, and iPod touch.

McWood currently has a diverse portfolio of 19 iPhone Apps and 2 iPad apps. McWood uses different forms of revenue models; some apps are available for free and include iAds. iAds is the advertisement platform provided by Apple. The revenue is generated by users that click on the ad, if it leads to a purchase additional revenue is
generated. The paid apps offered by McWood are available for the price of 0.79 cents or 1.59 cents.

McWood experienced that the reviews and comments people give in the iStore (where the app can be downloaded) are essential for the success of an app. Before Apple launched iAds, McWood experienced with their own advertisements within their apps. The results were dramatic, people reacted negative and started to comment and rate accordingly. This combination of negative feedback and ratings in the iStore led to an almost complete stop of downloads. Therefore, customer feedback is important for an app to be successful.

McWood uses a standard framework in all apps made by them, in which you can for example share the app with a friend, share it on Facebook, follow the Twitter account of McWood or sign up for a their newsletter. In addition, if a user shares it with a friend, McWood catches that traffic, and made it possible to create a database with e-mail addresses from present and possible new customers. Furthermore, their framework provides a section where all apps created by McWood are shown and directly links a user to the iStore. Therefore, McWood tries to get most out of each customer. Targeting customers with free apps is also part of their strategy, if a customer uses a free app it might be tempted to buy something from McWood.

The most popular app of McWood is SchoolTalk, currently at around 25,000 downloads. According to Martin the revenue on iAds is really low. The app generated only 20 euro-cent last month and Martin is convinced that if a company uses an ad based revenue model, it can hardly survive. Of all McWood’s apps, the one which generates most revenue is painters, an app which gives all kind of information on different paintings and their painters. As user you have the option to directly order a painting printed on a real scale. These large canvas paintings are between 100 and 200 Euro’s, and are offered by another company. McWood has a partnership with that company, and for each painting sold they receive 10%. This is another creative example how app publishers can generate (additional) revenue.

4.6 Top ten cross-platform app publishers

The most popular cross platform app publishers are shown in Table 2 in the Appendix. Distimo made this ranking by summing the total amount of downloads per app, and summed them per publisher. In this way they made a ranking of the most popular cross-platform publishers based on absolute total downloads. This paper uses the information to visualize the different relations between the mobile platform ecosystems. These network visualizations can be used to not only to show the most popular publishers for mobile platform ecosystems, but it can also used to see changes over time quickly. Interestingly, some app publishers managed to get a very high spot with just a few apps which appeared to be very successful. Note that there is one publisher, Chillingo Ltd. which is only present in the iOS ecosystem. This is due the fact that derived our data from Distimo, and they count iPad as another platform. However Chillingo Ltd., while only publishing for iPhone and iPad is at spot 6 of most popular cross-platform publishers.
Table 2. Top 10 most popular cross platform app publishers (Distimo report January 2011)

<table>
<thead>
<tr>
<th>Publisher name</th>
<th>Android (number of apps)</th>
<th>iOS</th>
<th>Symbian</th>
<th>Blackberry OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rovio Mobile Ltd.</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Electronic Arts</td>
<td>13</td>
<td>85</td>
<td>119</td>
<td>163</td>
</tr>
<tr>
<td>Microsoft Corporation</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Gameloft</td>
<td>11</td>
<td>162</td>
<td>115</td>
<td>154</td>
</tr>
<tr>
<td>Mmmoo   m</td>
<td>3</td>
<td>18</td>
<td>285</td>
<td>184</td>
</tr>
<tr>
<td>Chillingo Ltd.</td>
<td>0</td>
<td>200</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Google</td>
<td>41</td>
<td>6</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Spb Software inc.</td>
<td>24</td>
<td>13</td>
<td>44</td>
<td>1</td>
</tr>
<tr>
<td>Mobistar Media Ltd</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Glu</td>
<td>29</td>
<td>53</td>
<td>24</td>
<td>62</td>
</tr>
</tbody>
</table>
4.7 Data visualization

The graph in Figure 1 in the Appendix shows how the publishers are situated in the mobile platform software ecosystem in relation to the four mobile platforms. Publishers are positioned closer to the platform they built most apps for. Moreover, the size of the circles that represent the mobile platforms is related to the total number of applications submitted by the publishers we selected in Table 2 in the Appendix. The results show that Symbian is the leader in terms of mobile applications of the top ten cross-platform publishers. The second mobile platform with the highest number of mobile applications is Blackberry with almost 600 applications. Very close to Blackberry is iPhone with almost 550 applications. Because the Android platform is newer the most of the software producers that we selected are entering that ecosystem but already have a large number of applications developed on the other platforms. Chillingo is a perfect example of a huge contribution to the iOS ecosystem, but it is not present on all others platforms.

Each circle in the graph (except those that represent the mobile platforms) has a different color, representing one of the mobile platforms. These circles are connected with the four mobile platforms. The thickness of the connection is related to the number of apps the publisher has developed and published on the platform it is connected to. As it can be seen, the company that develops most of the applications is Mmmoooo with most of their apps for Symbian and Blackberry. It is followed by Gameloft which has an almost equal number of applications developed for iOS,
Blackberry and Symbian. The third largest software producer is Electronic Arts with most of the apps for Blackberry and Symbian, followed by iOS. As it is the case for Chillingo, they only develop apps for the iOS platform. Therefore, they have the strongest connection with a mobile platform. Another interesting aspect can be noted for companies that have a small number of apps such as Google or Microsoft. In the case of Google, their “Google” iPhone app has built-in all the services that Google offers. Moreover, similar access to apps is offered for Android users but here there is an app for every feature such as GMail, Calendar, Documents, and so on. In the case of companies like Google and Microsoft, the circles are very small in the graph. The reason for that can be seen in Table 2 in the Appendix in the number of apps offered. These companies developed a very small number of apps, usually around 10 apps in total for the four mobile platforms.

5 Conclusions

The main goal of this paper was to identify different revenue models that app publishers use, and to see whether they use different strategies over different mobile platforms. These platforms evolved substantially in the past few years together with the rise of smartphones and therefore led to new business models for software companies. In this research thirteen apps were selected based on popularity and cross-platform availability, on the four biggest mobile platforms, namely: Android, iOS, Symbian and Blackberry OS.

App publishers use different strategies and revenue models to generate revenue. Some app publishers release their app for free to stimulate usage of the service they provide. By doing so they stimulated their own business model. Other app publishers let users experience with a ‘free’ but limited edition and hope to tempt a customer to make a one-time buy decisions. App publishers differentiate their pricing for the same app per app store. This research found that some app publishers try to pursue a licensing model, like WhatsApp or NYTtimes. Games showed can have the possibility of in-app purchase functionality, like in Tom Talking Cat. Finally, McWood solutions showed that as app publisher it is also possible to generate (additional) revenue by making partnership deals, this was also a form of an in-app purchase. For app publishers which develop games some revenue models were identified which could only be found at games. First, publishers that offer games can try to generate extra revenue by selling ‘fan’ items. Second, some app publishers even sell their own app in other distribution channels then the expected apps store from a mobile platform. According to McWood Solutions it is hard to make money as app publisher, and app publishers need to creative with their business model and have a successful app in order to generate a decent revenue stream.

In this paper the top ten cross-platform publishers have been graphed. Although Symbian lost a lot of market share in the past few years, yet it offers more apps than the other platforms. This graph can be used to see changes the preferences that app publishers have when choosing a mobile platform.

For future research we suggest a more extensive comparison of apps which may lead to more revenue models being used by app publishers. In addition, it would be
interesting to see which revenue model is the most successful, and is this influenced by a mobile platform?

6 Discussion

There are some limitations of this empirical research. First of all, probably not all existing revenue models were found. Only thirteen apps were compared in total, which is rather low compared to all apps available. However, the inclusion criteria: popularity and cross-platform availability, by using data from Distimo, no more apps could be identified that passed the criteria. Furthermore, our assumption that companies pursue different pricing strategies might because due to the fact that in each market the economics principle of supply and demand exists. Therefore, a different price per app store might have nothing to do with a different strategy, but with a supply and demand factor. The network graph presented in this research does not take into account for how long a certain platform exists. For example, Android is relatively young, and app publishers only recently started to develop apps for it. As a consequence, app publishers probably have a lower number of total apps for Android, even if their attention may have switched in the mean time. Therefore, it would be more interesting to make this graph and show the current and planned apps in development.

References

The Eclipse Foundation Ecosystem and its Internal Members’ Interactions

Stamen Filchev¹ and Ivelina Koseva¹

¹Utrecht University, Princetonplein Street 5, Buys Ballot Laboratorium, 3508TB
Utrecht, The Netherlands
{s.filchev, i.koseva}@students.uu.nl

Abstract. In the light of the increasing interest and importance of Software Ecosystems in current communication and interactions between companies, this paper aims at giving a view of the connections between Eclipse Foundation and its internal members, as well as between the Eclipse Foundation’s partners themselves. The paper addresses four main types of members identified: strategic, enterprise, solutions, and associates. Due to the fact that all different types of members have different roles in the Eclipse ecosystem, as well as different benefits, the resulting mash of members’ relationships gives an interesting new perspective of the Eclipse ecosystem. The analysis is based on a descriptive study of the websites of Eclipse Foundation and its partners. As a result, an expressive visualization of the Eclipse Ecosystem is provided. This visualization shows the huge variety of members in this open-source ecosystem, which is one of the important aspects of a healthy ecosystem.

Keywords: Eclipse Foundation, ecosystem analysis, member interaction

1 Introduction

The software organizations of today and tomorrow aim at building beneficial worldwide networks of partners, members, or alliances. These inter-company relationships assure lucrative access to different markets and resources [1]. Still, just establishing these interconnections is far from sufficient. In order to have a healthy inter-company system, organizations need to understand these relations and their respective benefits. This is particularly hard, when all the data about alliances is collected and stored in such a way that it is difficult to process.

For example, if a company has hundreds of partners and has stored descriptive data regarding each partnership in a non-organized text manner, it is a really time-consuming task to understand what the benefit of a particular relation is. It is a completely different story when the data regarding the partnerships is systematic and is describing not only the relation between the organization and its partner, but also the partner’s network of partners. In this case it is easier to see how an organization’s partners bridge the organization to other beneficial markets and/or resources.

A software ecosystem is defined as a set of actors functioning as a unit and interacting with a shared market for software and services, together with the
relationships among them [4]. That is why such thorough and descriptive data could be considered as a documented representation of the software organization’s ecosystem. Furthermore, these data could be enforced with graphical visualization, which would be easier to grasp by the human eye. This is an important matter, which has been researched in the last years. For example, Lungu, Lanza, Girba, and Robbes present a Small Project Observatory, a tool which supports analysis of software ecosystems through interactive visualization and exploration [5].

During the last decade the interest in software ecosystems has increased. This is seen as an inevitable continuation of the global market network. For example, [1] performed an ecosystem analysis in the software industry on a global level, which included 509 companies within a timespan of twelve years. However, this analysis is done on a macro level and that fact makes the particular organization’s position within the industry hard to grasp on a micro level. That is one of the reasons why this paper focuses not on a global ecosystem, but on the ecosystem of a particular organization, the Eclipse Foundation. Another reason for the authors’ choice is that regardless the importance and popularity of the Eclipse Foundation and its ecosystem, not much particular analysis exists [6]. Last, but not least, the fact that the Eclipse Foundation’s ecosystem is an open-source one even spices it up even more.

The paper is organized in seven sections. The first one contains a brief introduction in the Software Ecosystem’s field and an overview of the research. The second section presents the research strategy and method used in the paper. The third section thoroughly describes the different membership types within the Eclipse Foundation ecosystem. The fourth section focuses on the different network metrics used to assess the ecosystem. The fifth section presents the results of the research, while the sixth is concentrated on the analysis of the research results. The seventh suggests some discussion of the research and proposes some ideas for future research. Last, the eighth section concludes the findings of the research.

2 Research Method

The proposed research method is descriptive from a qualitative point of view. That is why it requires a substantial amount of data, regarding the Eclipse Foundation and its members. This paper aims at analyzing thoroughly the Eclipse Foundation ecosystem. In order to succeed, the data collection process starts with an examination of the Eclipse Foundation website (www.eclipse.org), which was aimed at retrieving the full list of members as well as all relevant information to them. Secondly, all the relationships were extracted from members’ websites along with the mutual collaboration within Eclipse Foundation’s projects. Thirdly, these interconnections were aligned within a matrix, which was used as input for the graph visualization tool Gephi (www.gephi.org). Then the Eclipse Foundation Ecosystem graph was generated. The visualization is an important part of the study of an ecosystem, because it clearly presents the significance of the relationships between the members of the ecosystem themselves (excluding Eclipse Foundation). Afterwards, a thorough analysis of the complete ecosystem and specific sub-branches was conducted, using a set of proven network metrics [2],[3].
Because the examined companies have set a number of different types of relationships, that do not match each other, it is not possible to categorize them. Due to this just the inter-firm connections will be examined, but not the weight of their relationship edges. In accordance to the Eclipse Foundation ecosystem specificity, this paper sets the following research question: *How does the internal members’ network within the Eclipse Foundation open source Ecosystem look like?*, together with the subsequent sub-questions: *Does the network embrace a number of larger sets connected to each other?*; *What is the amount of members’ interconnections within the ecosystem?*

### 2.1 Research scope

The paper analyzes the Eclipse software ecosystem that consists of 167 companies in total, including Eclipse. Because there were missing relationship descriptions for some of the companies on the Eclipse Foundation website, they were not included in the survey.

In the process of identification of the existing connections between the Eclipse Foundation’s members it has been found that not all of them have listed their partner/member/alliance links with other firms. That is the reason why the secondary source of organizations’ interconnections is the mutual collaboration within Eclipse’s projects.

### 3 Eclipse Members

In order to ease the understanding of the paper’s hypothesis and the presented results, in this section we present brief descriptions of the different membership types within the Eclipse Foundation. Currently there are five different types of membership, namely: Associate Members, Solutions Members, Enterprise Members, Strategic Members, and Committer Members. Each one of them will be separately explained in the text below [5].

Briefly, *Associate members* are organizations that support and participate in the Eclipse Ecosystem. The associate membership has no voting rights, but is also free for non-profit organizations, standards bodies, universities, research institutes, media and publishing, government and other organization types as defined by the Eclipse Foundation board of directors. For all the rest there is an annual fee of 5000 USD. Associate members can participate in the scheduled annual and quarterly update meetings, submit requirements and participate in all project reviews.

*Solution members* are organizations that offer products and services with Eclipse or based on it. They consider Eclipse Foundation as an important part of their development strategy and want to contribute to the ecosystem. Solution membership provides some benefits, related to Eclipse Foundation’s strategy success. There are also a number of special affiliate programs, discounts, and services. The annual membership fee varies between 1500 and 20 000 USD, depending on the revenue of the organization.
Enterprise members are large organizations that use Eclipse technology as a platform for their projects and want to participate in the development of the ecosystem. Enterprise membership allows an access to intellectual property data, tools and different information sources. There is a fixed annual membership fee of 125,000 USD.

Strategic members are organizations that invest different resources in order to develop the Eclipse technology, because they view Eclipse as a strategic platform. The strategic membership ensures a representative in the Eclipse Foundation Board of Directors and respectively an option for direct influence on the direction of Eclipse. Strategic members are divided into two types: Strategic Developers and Strategic Consumers. Strategic developers are the one that make the contribution, while Strategic consumers are mainly users of Eclipse technology. Both of them pay annual dues set by the Eclipse Foundation, which varies between 25,000 and 500,000 USD, depending on their annual revenue.

Committer members are individuals that contribute and commit code to different Eclipse projects. They can be independent or a part of a large organization.

Having in mind the different types of membership and corresponding benefits, a correlated dependence is expected. Because of the importance of the Strategic members it is considered that they will play the role of hubs in the ecosystem, which are “the firms with a disproportionately high number of links” [1]. Based on the participation that associate members have in the Eclipse ecosystem it can be hypothesized that their connections will not be so strongly expressed. On the other hand, associate members together with the solution members may use Eclipse Foundation as a source for finding more relations that will probably support their performance on the market, as well as use it as a communication channel and access to external resources.

Committer members, who are individuals, are not subject of this research, as its purpose is to present the relationships between the member companies, and the way they support each other.

4 Network Metrics

In order to measure the Eclipse Foundation ecosystem and to provide an answer to the set research questions, four metrics are used: degree of links (weighted degree), average path length, clustering coefficient, and network density [1]. These metrics are chosen because of their significance and application in other domains concerning different types of ecosystems [2], [3]. For each of the metrics a definition and an example is provided in the text below.

4.1 Degree of Links (Weighted Degree)

The degree of links is a metric that presents the number of links that a firm has at a given time [1]. Figure 1 presents a simple graph with seven nodes and eight edges between them. If we look at node a - its degree of links is 4, which is the number of
existing links from $a$ to the existing nodes in the graph. Nodes $g$ and $f$ have degree of links $1, d – 3$ and so on. This metric can be applied to only one member or to the whole ecosystem, like in that case the average degree of links can be found.

4.2 Average Path Length

Path length defines the number of steps that a company should take in order to reach another company in the ecosystem [1]. For example, in Figure 1, the path length form node $d$ to node $g$ is $3(d \rightarrow a, a \rightarrow c, c \rightarrow g)$. The average path length is the sum of all path lengths, divided to the number of the companies. This metric will be computed for the whole software ecosystem excluding the Eclipse Foundation. The reason for this is that there is no point to calculate it with Eclipse, because all of the companies are connected to it and the maximum number of steps is 2.

4.3 Clustering Coefficient

Clustering coefficient is “the degree to which a firm’s partners are also partners with each other” [1]. In figure 1, node $d$ has 3 partners – $a, b$, and $e$. From the possible three connections between $a, b$, and $e$, there are two existing. This means that the clustering coefficient of $d$ is $2/3$ or 0.67. This metric can be calculated both for a single company and for the whole ecosystem. The closer the clustering coefficient is to 1, the better is the linkage between the partners of a member.

4.4 Network Density

Network density is the ratio of the existing links and the number of all the possible connections within the ecosystem [1]. Figure 1 illustrates a graph with 7 nodes, which have 8 connections between them. The number of possible links in this case is 21, so the network density is 0.33 ($7/21$). This metric can be applied just to the entire software ecosystem or to the part of it, examined separately. The closer the ratio is to 1, the higher is the density within the ecosystem.
5 Results

This section presents the results of the research over the Eclipse Foundation ecosystem. It is aimed only at providing understandable set of results, for the analysis of the result, please refer to the next section.

In order for the results to be more understandable, they are presented in five different views. The most general view addresses the complete Eclipse Foundation ecosystem. Then, the scope is slightly changed, just by excluding the Eclipse Foundation itself – this action ensures an unbiased view of the overall connectedness between Eclipse Foundation members. Last, the scope is calibrated to different types of memberships. The members from the Strategic and Enterprise membership types are merged, because there are only two companies with Enterprise membership and this type of membership is a lot more similar to the Strategic one. The resulting three different views are targeted at understanding the interconnections between the Strategic, Solution and Associate members.

In the following paragraphs each perspective is accompanied with a visual graph of the related members as well as a table addressing each network metric, described in the previous chapter. On top of that, table listings of top 10 members with regard to each degree of links and clustering coefficient for each view was built and analyzed. The coloring of the nodes (members) depends on the number of connections the particular company has. The size of the points is dynamic and depends on the degree of links as well. The gradient used is varying from black – for very few connections, to sharp red, which means lots of connections. Furthermore, the coloring is based on the maximal number of connections per company within each view, meaning that some companies might be sharp red in some views, but rather dark red in others.

5.1 View 1: The complete Eclipse Foundation Ecosystem

Figure 2 depicts the complete Eclipse Foundation software ecosystem and the relationships between the companies taking part in it. In order to be visualized in a more clear way the dot representing each company increase its size with the increase of the numbers of relationships it has. The ecosystem consists of 169 members with 616 relationships examined between them. This represents 4.2% of all theoretically possible relationships between them.

As expected, Eclipse is the one with the most relationships, being a central hub within the ecosystem. The clustering coefficient shows inversely proportional relationship to the number of links a company has and just minor fluctuations can be observed in that trend. The high clustering coefficient and the low amount of links support reciprocal relationship between number of links and clustering coefficient, examined in the most densely interconnected organizations. Another interesting fact is that the Strategic companies can be clearly seen as leaders, as four of them follow Eclipse Foundation in the top 10 list of links and the seventh company (Cisco, 25 links) within the list is already with just half of the links of the company right after Eclipse Foundation, which is IBM with 52 links. Still, it is important to mention that a number of Solution members are also within the top 10. The clustering coefficient
states that a lot of the companies have 1 as a result, but this is due to their few number of links, which happened to be mutual partners.

![Figure 2 The complete Eclipse Foundation Ecosystem](image)

Table 1 presents the monitored metrics of the ecosystem. The average path length of the ecosystem is 1.964 which is expected, given the fact that Eclipse is related to all of the examined parties. Each of the partners has about 7.3 links on average, which result in slightly below 0.45 clustering coefficient and 4.2% network density.

Table 1 Overall Network Metrics for the complete Eclipse Foundation Ecosystem

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average path length:</td>
<td>1.964</td>
</tr>
<tr>
<td>Network Density:</td>
<td>0.042</td>
</tr>
<tr>
<td>Average Links:</td>
<td>7.29</td>
</tr>
<tr>
<td>Average Clustering Coefficient:</td>
<td>0.446</td>
</tr>
</tbody>
</table>
5.2 View 2: The Eclipse Foundation Ecosystem, excluding Eclipse Foundation

Figure 3 displays the whole network of Eclipse partners and the direct relationships between them. The isolation of Eclipse is required, because it allows an unbiased study on the relationships within the ecosystem without the central hub. Few differences are witnessed in the top ten most interconnected companies in the ecosystem. This fact was expected because of the fact that only one partner, which is common for all parties is not examined in that case. Furthermore a slight decrease in the clustering coefficient is witnessed. The least connected companies also remain the same.

Figure 3 The interconnectedness between all the members in the Eclipse Foundation Ecosystem, excluding Eclipse Foundation

Table 2 Overall Network Metrics within the members' part of the ecosystem

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average path length:</td>
<td>2.647</td>
</tr>
<tr>
<td>Network Density:</td>
<td>0.031</td>
</tr>
<tr>
<td>Average Links:</td>
<td>5.35</td>
</tr>
<tr>
<td>Average Clustering Coefficient:</td>
<td>0.177</td>
</tr>
</tbody>
</table>
Table 2 presents the metrics examined for the ecosystem without the presence of its centralized hub. This results in increase in the average path length up to 2.65. A noticeable decrease is also examined in the network density and the average number of links. With regard to the top 10 companies on each network metric, the observations are similar to the previous view - the Strategic members can be seen as leaders. The only difference is that each company advances one place up, as Eclipse Foundation is excluded.

5.3 View 3: The Interconnectedness between Strategic and Solution Members

The interconnections between the Strategic and Solution members are shown in figure 4 below. As could be seen from the graph, it turned out that the Strategic members are not necessarily the companies with the highest number of connections. Still, the most of them have more relations than the Solution members.

In order to clearly comprehend the data represented in the graph, we applied four network metrics – degree of links, average path length, clustering coefficient and network density (see section 4). Two of these networks are applied to the complete sub-network, while the other two are applied to the complete network as well as each member within. The results for the complete sub-network are presented in table 3 below.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average path length:</td>
<td>2.439</td>
</tr>
<tr>
<td>Network Density:</td>
<td>0.093</td>
</tr>
<tr>
<td>Average Links:</td>
<td>6.69</td>
</tr>
<tr>
<td>Average Clustering Coefficient:</td>
<td>0.285</td>
</tr>
</tbody>
</table>

Furthermore, the listings with top 10 members with respect to degree of links and clustering coefficient show that there is a rather good balance of the Strategic and Solution members, each group having five representatives. Even though the top four are just Strategic members, this listing still proves that there is a balance between these two membership types and strengthens the proposition that these two membership types are the leading force within the ecosystem.
5.4 View 4: The interconnectedness between Solution and Associate Members

The interconnections between the Enterprise and Associate members are shown in figure 5 below.

As could be seen from the graph, there are very few Associate members with significant amount of relations. This is expected, because these members are usually smaller companies with underdeveloped partner network. Another reason is that some of these companies, even if big, have their main focus not directly connected to the software industry, making the relationships with lots of members within the ecosystem redundant. Still, the Associate members are the members that diversify the Eclipse Foundation Ecosystem in terms of different industries.
In order to clearly comprehend the data represented in the graph, we applied four network metrics – degree of links, average path length, clustering coefficient and network density (see section 4). Two of these networks are applied to the complete sub-network, while the other two are applied to the complete network as well as each member within. The results for the complete sub-network are presented in table 4 below. What is easily noticeable is that the network density is 8.5% and the clustering coefficient is rather low – below 28%. The average degree of links is also lower, just above 4.5. Furthermore, the companies with most connections are Oracle and IBM with 22 links to other members in this network fragment. The connectivity rapidly decreases and the company on 10th place in the top 10 members with highest degree of links has only 7 connections.

Table 4 Overall Network Metrics Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average path length:</td>
<td>2.574</td>
</tr>
<tr>
<td>Network Density:</td>
<td>0.085</td>
</tr>
<tr>
<td>Average Links:</td>
<td>4.582</td>
</tr>
<tr>
<td>Average Clustering Coefficient:</td>
<td>0.275</td>
</tr>
</tbody>
</table>
5.5 View 5: The interconnectedness between Solution and Associate Members

Figure 6 on next page illustrates the connections between Solution and Associate members, the big red nodes presenting the Solution members and the small gray nodes illustrating the Associate members. This figure is the one that has the lowest number of interconnections between the companies. The reason for this is that most of the firms are small to medium size, with very few exceptions. These companies are more interested in contacting bigger organizations, instead of building connections between each other.

Table 5 below presents the general results for this view within the ecosystem. What is easily noticeable is that the average path length now is over 3 and the clustering coefficient is rather low – just above 30%. Compared to the other graphs, this is the one which is less connected. The highest number of links here is 17. The clustering coefficient is also low with the exception of the companies that take the first two places from the companies with highest clustering coefficient. They have clustering coefficient 1, due to their number of links, just 2.

Table 5 Overall Network Metrics Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average path length:</td>
<td>3.021</td>
</tr>
<tr>
<td>Network Density:</td>
<td>0.048</td>
</tr>
<tr>
<td>Average Links:</td>
<td>4.6</td>
</tr>
<tr>
<td>Average Clustering Coefficient:</td>
<td>0.309</td>
</tr>
</tbody>
</table>

Figure 6 The interconnectedness between the Solution and Associate members within the Eclipse Foundation Ecosystem
6 Analysis of the Results

The presented results in section 5 managed to give the answers to the research questions posed in this paper. From all the visual representation and the evaluation of the ecosystem through the network metrics, it can be concluded that the Eclipse Foundation ecosystem is a healthy and diverse ecosystem.

From all the five perspectives which were used during the analysis, there were no critical bottlenecks observed. As expected, the Strategic members proved to be the backbone of the network, but none of these members is of such importance, that the overall connectedness in the network is threatened. This means that the network is not just a union of larger sets, related through few links, which was one of the main points of interest of this paper. The next paragraphs present the interpretation of the results of the network metrics.

The range of the results of the *average path length* metric varies between 1.964 and 3.021. Knowing that the lowest result is when Eclipse Foundation itself is included in the analysis, this is expected. The presence of Eclipse limits the maximum path length to 2, because if two companies have nothing in common they can reach each other through Eclipse. The more interesting fact is that when Eclipse Foundation is excluded, so there is no maximal limit, the results do not drastically differ: the values are between 2.439 and 3.021. This is an argument in favour of the ecosystem’s health and overall connectedness — most members can reach each other in no more than 3 “hops”. In comparison, the average path length observed by [1] in their study is in the interval of 5.1 to 7.5. Of course, in their study the scope is a lot bigger, so this is one of the reasons of the larger value for the average path length.

The range of the results of the *network density* metric is between 3.1% and 9.3%. What is of importance is that the highest percentage is representing the network density between strategic and solution members. This is very good, because these two membership categories are the “driving force” of the ecosystem. It is even an authors’ suggestion that Eclipse Foundation should stimulate a further increase up to 13 to 15 percent. On the other side is the lowest result, which is based on the complete Ecosystem, excluding the links through Eclipse Foundation itself. Even though 3.1% is not low for a complete ecosystem, the authors believe that Eclipse Foundation should put some effort in increasing the network density in this view to at least 5%. In comparison, the results in the study of Iyer, Lee, and Venkatraman vary from 4.9% in 1990 to 1.3% in 2001, which again is reasoned by the number of companies (509) that are included in their study.

In the five different views, the results for the *average degree of links* metric vary from 4.11 to 7.29. The highest result is when Eclipse Foundation is included in the view, which is logical, given that it has links with all other companies. But even without Eclipse, the maximal result achieved is 6.69, which is reasonably high. As expected, this result is synchronous to the result of the network density, meaning that is achieved by the view over the Strategic and Solution partners. This metric, in a single-company view, is useful for the identification of hubs within the ecosystem as well as other different ecosystem-specific roles.

The *clustering coefficients* vary between 0.177 and 0.446. The maximal result differs significantly from the others, because it represents the whole ecosystem including its center (Eclipse Foundation). Still, when the data is “unbiased” by
excluding the center, the maximal value is 0.309, which is a rather good result. This points out that the ecosystem is diverse and the density is not really high, which confirms our expectations. However, the limitations of the data sources (no proprietary sources) used might have biased the results.

From all the above it can be concluded that the Eclipse Foundation ecosystem is a healthy one – it is diversified, it does not have any crucial bottlenecks and there are the typical roles of leaders and followers within.

7 Conclusion

Software companies are creating partnerships in order to get some benefits like new markets and access to external resources. By doing this they create networks that can be determined as a software ecosystems. This paper studies the Eclipse Foundation software ecosystem and presents the existing connections between the different members within the partnership types. The data for the relations is extracted from the Eclipse Foundation’s and its partners’ corporate websites and stored in a matrix. Based on it, different visualizations are presented. Together with the explanations of the Eclipse Foundation’s different membership types and the defined metrics, a good base for a discussion and analysis is proposed. Each metric contributed to the assessment of the ecosystem. The average path length confirms that the overall connectedness is good, with most companies reaching all others in three “hops” at most. The network density suggests that the network is dense enough and that the densest sub-network is formed by the driving force of the ecosystem – strategic and solution partners. The average degree of links is an evidence of good communication channels within the ecosystem. Last, the clustering coefficient supports the authors’ hypothesis that the ecosystem is diverse.

The metrics’ results suggest that Eclipse Foundation ecosystem is an example of healthy software ecosystem – it is diverse, with no detectable crucial bottlenecks, and easily perceivable roles of leaders and followers within.

The main limitation of this research is the limitations on data sources. Due to the fact that not every company lists all its relations to other organizations publically, our data could be improved by the use of proprietary data sources covering financial, marketing, consulting, and research and development relations. Because of time and space constraints, only the relation between different Eclipse Foundation’s membership groups is presented. There could be presented some of the interesting relations between companies based on other role classifications.

There are a few important steps, which can be taken in future research. The authors think that the most important is the correlation of inter-company links and time. If the view on the network is mapped to the time – dimension, it is believed that a far more accurate model is to be built and even some predictions on the evolvement of the ecosystem could be made. In order to provide better comparison, more metrics suitable for the current ecosystem, can be identified. What is more, a better observation of the roles that the different partners play in the ecosystem can be conducted.
References

Multi-homing in the Apple Ecosystem: Why and how developers target multiple Apple App Stores

Andrei Idu and Tommy van de Zande
Department of Computing and Information Sciences, Utrecht University
{a.idu,t.jacobus}@cs.uu.nl

Abstract. The Apple ecosystem is divided into three sub-ecosystems: iPhone, iPad and Mac. This paper looks at companies that target more than one of these sub-ecosystems, also called multi-homers, analyzing their behavior across them. The set of companies analyzed is composed of the publishers of the top 300 free and paid applications within the three ecosystems’ App Stores in May 2011. Findings showed that there is a strong relationship between the number of platforms a company targets and the type of applications it publishes and the number of these applications. The most encountered trend was of first targeting the iPhone platform and then the iPad. Multi-homing from a strategic perspective is motivated mainly by a wider customer base and the portability the Apple sub-ecosystems offer. This research offers important insights into the strategies and tactics adopted by multi-homers in the Apple ecosystem.

Keywords: Software Ecosystems, App Store, Multi-homing

1 Introduction

Software ecosystems (SECOs) have created new opportunities for software companies. Acting as SECO participants, these companies are able to develop software based on an open platform provided by a SECO orchestrator [4]. However, for these companies it is of strategic importance to understand the SECO in which they exist [12]. They are dependent on an orchestrator for a vital infrastructure such as operating systems, libraries, component stores, and platforms [9]. A practice that poses an even more complicated strategic challenge is the fact that an increasing number of companies are active in more than one SECO. Data from as early as 2004 [2] showed this practice of “multi-homing” software companies developing software products for multiple platforms.

Apple currently provides three different software platforms, for which it provides an App Store. These App Stores are governed in a very similar, tightly controlled and organized way, but they serve different platforms: the iPhone App Store for their mobile phones, the iPad App Store for their tablets and the Mac App Store for their personal computers, which are considered sub-ecosystems of the overall ‘Apple ecosystem’.
The iPhone App Store launched in July 2008. As of July 2011, there are more than 377,000 apps available in the iPhone App Store\(^1\). The iPad App Store launched together with the introduction of the first iPad, in April 2010. As of July 2011, there are 107,000 apps available for the iPad\(^1\). The latest App Store Apple launched is the Mac App Store, which launched in January 2011. As of July 2011, more than 5600 apps were available\(^1\).

Distimo B.V., a Dutch company specializing in App Store market intelligence, provides monthly reports where several interesting statistics about several major App Stores are presented. In their February 2011 report, they elaborate on the three different Apple App stores \(^5\). The report shows that while some publishers create their apps for just one of these platforms, others target two or even all three of these App Stores. This paper looks at the reasons behind these strategic choices of crossing the boundaries at a software ecosystem level \(^9\). This brings us to the following research question:

**What are the trends and characteristics for multi-homing App Store publishers and their apps?**

This research question is divided in the following sub-questions:

- Is there a relation between the type of application and the number of platforms it targets?
- Is there a relation between the size of the publisher and the number of platforms it targets?
- Is there a relation between the first platform a publisher targeted and the second platform it targeted?
- Why do publishers make these choices?

The practical contribution of this research is to provide some fundamental information that supports publishers in developing a portfolio road-map, choosing what ecosystems to target, and when. As a scientific contribution to the field of software ecosystems, the results of this paper will provide insights in the behavior of multi-homers. Trends in multi-homers business decisions on an ecosystem level are identified as well as a characterization of multi-homers in order to identify different multi-homer strategies.

This section introduced the software ecosystem domain and the concept of multi-homing publishers. The Apple ecosystem was given as an example to illustrate the behavior of such publishers. The remainder of the paper is structured as follows. Section 2 describes a literature study, which provides a theoretical background for the analysis of the characteristics and behavior of multi-homing publishers. In Section 3 the conducted research method is described. Section 4 presents the data that was gathered about the publishers that are active in more than one of the three Apple App Stores. In Section 5 an analysis of the data is presented together with the findings of the research. In Section 6 we discuss some limitations and threats to validity. Finally, conclusions and proposed further research are presented in Section 7.

\(^1\) Source: Appshopper.com. Last accessed on July 4th, 2011
2 Literature Study

In the field of Software ecosystems research different definitions are used [10, 1]. This research paper uses the definition proposed by Jansen, Brinkkemper & Finkelstein [8]. They define a Software ecosystem as “a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts”. Based on this definition, the iPhone, iPad and Mac ecosystems are considered as sub-ecosystems of the larger ‘Apple ecosystem’. Jansen at al. also talk about three different scope levels that SECOs can have. These are the organizational scope level, the SECO scope level and the SECOs scope level.

A mix of the organizational level and the SECOs level has been discussed in literature by identifying participants of SECOs that are active in two or more SECOs at the same time. This behavior has been characterized in different manners based on the perspective of the analysis. Rochet & Tirole [11] call it multi-homing and state that it is derived from the consumer’s desire to take advantage of different functionalities in an environment of non-interconnected platforms. The perspective in Rochet & Tirole’s research is market oriented [11], so their conclusion is that multi-homing appears when there are gains to be reaped from trade on incompatible or not interconnected platforms. The authors also explain that a multi-homing supplier will benefit from a greater potential market share that can be tapped into.

Boudreau [2] examines the “network effects” that an increasing number of buyers and suppliers around a platform has. His study looks at mobile software developers on specific platforms. Even though there is no mention of the term this is indeed a software ecosystem study. The interesting information in this paper is the fact that there were already multi-homing companies active between 2004 and 2007. Multi-homing software developing firms accounted for 1% of the data used by the research.

Burkard et al. (2011) analyze multi-homing in SaaS market places [3]. They base their research on Rochet & Tirole and mention the fact that SaaS providers sometimes employ multi-homing behavior by selling the same service on different platforms or marketplaces. This kind of behavior can be traced to the paper of Hagel, Brown & Davison [6] in which they call this type of participants hedges. A hedger develops the same product for multiple platforms. The main benefit of being a hedger is that the business risk is spread in more markets, while the main drawback is that there are higher costs incurred when duplicating the effort for supporting multiple platforms.

Jansen et al. [8] mention a role that has similar attributes to multi-homers. The role is that of a “bridge” between different SECOs. This kind of a SECO participant can transfer resources between ecosystems, however its main driver is providing connect-ability and not necessarily providing the same product for different ecosystems.
To summarize, existing literature provides different definitions for the types of behavior or roles a participant that is active in more than one ecosystem can be classified as. These are: multi-homer, hedger or bridge. As a base for this research the term multi-homer is the most relevant. This is because the companies that are examined further provide either the same product or different related products to different sub-ecosystems in the Apple ecosystems. The iPhone, iPad and Mac environments are considered sub-ecosystems because, based on the SECO definition, they provide different platforms and different markets for participants. However, they are sub-parts of the Apple ecosystem, because they share similar characteristics by having the same keystone player [7], having the same governance model with similar policies, using the same programming language, etcetera.

3 Research Method

This section describes the research method. First a literature study in the field of software ecosystems research was done. This created a theoretical background to base the analysis of the research results on. Emphasis was put on the strategic perspective for participants in software ecosystems and on the characteristics of the software ecosystem level [9]. Also, we studied and tried to unify the different terms used for companies which participate in multiple software ecosystems.

To answer the research question and sub-questions, a small software tool was created to extract app and publisher data out of the different Apple App Stores. This tool was used to download several RSS-feeds provided by Apple, which included extensive data about the 300 most downloaded paid applications, as well as the 300 most downloaded free applications of every US App Store. We based this research on data from the US App Stores because it currently has the most apps. The tool extracted relevant data, such as app title, publisher name, price, date first added and category, for every app from the RSS-feeds. This data was stored in separate XML-files for every store. Data from these XML-files was then combined and converted into CSV-files, so it could be loaded into SPSS, to perform statistical calculations on it in order to analyze the results. In the CSV-files, our tool counted the total number of apps per App Store for every publisher. Also, it stored the number of apps per category (23 in total, such as Social, News, Games etc.) per App Store, and the dates that the first app has been launched per App Store. All data was extracted on May 19th, 2011.

To evaluate the results, several publishers have been contacted to answer a small questionnaire to discover the reasons for their decisions. In this questionnaire, we asked publishers in which stores they were currently active, in which store they started out, what were the strategic reasons for making these choices and if there were positive or negative effects after this behavior.
4 Results

The extraction of the App Store RSS-feeds, as described in the research method, resulted in a data-set of 1,060 different publishers, which had at least one top 300 app in one of the three App Stores. The total number of apps published by these publishers was 1,800. Most publishers (70.8%) have only one app in one of the three App Stores. The biggest publisher (Electronic Arts) has 44 different apps in the three App Stores. The average number of applications per publisher (mean) is 1,698, with a standard deviation of 2,203.

A large part of publishers in the App Stores (28%) consists of game publishers. It is important to also look at the results without taking game publishers into account, since these game publishers could distort the results for other types of publishers. When all game publishers are filtered out of the data-set, the remaining number of publishers is 761. Most of them still have only one app in total (74.8%). The biggest non-game publisher (Apple) has 23 apps over all three App Stores. The average number of applications per publisher is 1,474 with a standard deviation of 1,415. These descriptive statistics are summarized in table 1.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1060</td>
<td>1</td>
<td>44</td>
<td>1,698</td>
<td>2,20382</td>
</tr>
</tbody>
</table>

4.1 Store and Category characteristics

In Apple’s App stores, each application is placed into one of 23 categories. For example, there are separate categories for apps that relate to business, education, navigation and music. As noted before, the Games category is the dominant category in each store. Games account for 284 apps of the total 600 iPhone apps, while on both the Mac and iPad App Store, games account for 197 of the 600 apps of those platforms.

When looking at the popularity of categories on the different platforms excluding games, we see that there is a clear difference in category Popularity over the different platforms. The category Productivity is, with 87 applications, the most popular category on the Mac. On the iPad, Productivity is the third most popular application category and on the iPhone it is the 8th most popular type. The most popular type of applications on the iPhone and iPad are in the Entertainment category, where on the Mac App Store this application category is ranked 9th. Figure 1 gives on overview of the fifteen most popular categories per platform combined, and their respective number of application per store.
4.2 Multi-Homers

In total, 182 publishers publish an application in at least two stores (for example one app in the Mac App Store and one in the iPad App Store). 93 of these 182 publishers (51%) have an even amount of apps in both the iPhone App Store and iPad App store. 22 publishers have one or more apps in all three App Stores. In figure 2, a pie chart shows the division of multi-homers in the Apple ecosystem. As this chart shows, most multi-homers publish only in the iPhone and iPad App stores (74%), 12% publishes at least one app in all three stores, while 10% targets the iPad and Mac stores and 3% combines the iPhone and Mac App Stores.

5 Analysis

In the previous section, some interesting observations from the App Store and its publishers were extracted from the dataset. In this section, we will answer the actual research questions, by analyzing the results and calculating the correlation between several characteristics. The final sub-question, if there are specific reasons behind the choices publishers make, will be answered by analyzing survey responses.
5.1 Is there a relation between the type of application and the number of platforms it targets?

As we have seen in the previous section, there are differences in the popularity of certain app categories on different platforms. Games are the most popular category on every platform, but Productivity apps are second most popular on Mac, while these types of apps are far less popular on iPhone and iPad.

Looking back at Figure 1, some categories have a large number of apps on two platforms, but far less on another platform. For example, the entertainment category, has 66 apps on the iPhone and 61 on the iPad, but only 17 on the Mac. Some categories, such as books, are even non existent on the Mac, while they are present on both the iPhone and iPad stores. On the other hand, Productivity has 87 apps on the Mac, 50 on the iPad but only 15 on the iPhone. As mentioned before, the most popular category on all platforms is the Games category.

It is clearly observable that there is a relation between the type of platform and the popularity of some specific categories, however, other categories are about evenly distributed among the three platforms.
5.2 Is there a relation between the size of the publisher and the number of platforms it targets?

A logical assumption is that the size of the publisher correlates with the number of apps on different platforms. To compute the correlation between the number of stores a publisher targets and its size, we counted the number of apps in all stores for every publisher in the data-set. We defined size of a publisher as the total number of apps it provides over all three stores. Table 2 presents the result of this computation.

Table 2. SPSS output of the correlation between publisher size and number of stores it targets

<table>
<thead>
<tr>
<th>Correlations</th>
<th>Number of Stores</th>
<th>Total number of Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Stores</td>
<td>Pearson Correlation</td>
<td>Sig. (2-tailed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total number of Apps</td>
<td>Pearson Correlation</td>
<td>Sig. (2-tailed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**. Correlation is significant at the 0.01 level (2-tailed).

With a correlation coefficient of 0.524, there is a significant correlation between the size of a publisher and the number of platforms it targets. However, it is not a perfect correlation. There are still large publishers who do not target all three stores. One example is Electronic Arts. It is the biggest publisher in number of total apps, but it only targets the iPhone and iPad App stores.

5.3 Is there a relation between the first platform a publisher targeted and the second platform it targeted?

For most publishers, the iPhone platform was the first platform that they targeted. This is not because of any strategic reasons, the iPhone App Store was simply the first App Store available. It seems a logical choice to also create an iPad app when the platform launched, since it requires relatively low effort (Apps are built using the same language, same SDK, same APIs etc.). However, looking at the significant differences in the number of available apps among the stores, it seems that most publishers still only launched an iPhone app.

When looking at the relation between a first platform and a second platform, it is important to first look if there is a correlation between any of the platforms
at all. We have calculated Pearson’s correlation coefficient for all possible combinations, with all publishers who have an app in at least two different stores. Those combinations are the correlation between number of iPhone and iPad apps per publisher, iPhone and Mac apps per publisher and iPad and Mac apps per publishers.

Table 3. SPSS output of the correlation between stores

<table>
<thead>
<tr>
<th></th>
<th>iPad Apps</th>
<th>iPhone Apps</th>
<th>Mac Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPad Apps</td>
<td>Pearson Correlation</td>
<td>1</td>
<td>0.763**</td>
</tr>
<tr>
<td></td>
<td>Sig. (2-tailed)</td>
<td>0.000</td>
<td>0.047</td>
</tr>
<tr>
<td>N</td>
<td>182</td>
<td>182</td>
<td>182</td>
</tr>
<tr>
<td>iPhone Apps</td>
<td>Pearson Correlation</td>
<td>0.763**</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Sig. (2-tailed)</td>
<td>0.000</td>
<td>0.975</td>
</tr>
<tr>
<td>N</td>
<td>182</td>
<td>182</td>
<td>182</td>
</tr>
<tr>
<td>Mac Apps</td>
<td>Pearson Correlation</td>
<td>0.148*</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>Sig. (2-tailed)</td>
<td>0.047</td>
<td>0.975</td>
</tr>
<tr>
<td>N</td>
<td>182</td>
<td>182</td>
<td>182</td>
</tr>
</tbody>
</table>

** Correlation is significant at the 0.01 level (2-tailed).
* Correlation is significant at the 0.05 level (2-tailed).

Table 3 presents the SPSS output for the correlation coefficients between the number of apps in the different stores. The correlation coefficient between the number of iPhone apps and the number of iPad apps is 0.763, so there is a significant correlation between the number of iPhone apps and the number of iPad apps that a publisher has. The correlation coefficient between the number of iPad and Mac apps is low, but still significant: 0.148. There is some correlation, but it is not very strong. The correlation between the number of iPhone apps and the number of Mac apps per publisher is really low: 0.002. There is no significant correlation between the number of iPhone and Mac apps.

When looking at the first and second platform publishers target, it also shows the strong correlation between the iPhone and iPad apps. 72 out of the 135 publishers that have both an iPhone and iPad app, launched their iPhone app first and then later launched an iPad app, while only 15 publishers did it the other way around: they launched an iPad app before they launched an iPhone app. Interesting to note is that 48 of the 135 publishers launched both their iPhone and iPad app on the same day. These apps are so-called universal apps; apps that run on both platforms. They are present on both stores, but are technically the same app and therefore share the same launch date.
Of the 19 publishers active on both the iPad and Mac, 16 have launched their iPad app before they launched their Mac app. The remaining three have launched a Mac app before they launched an iPad app. The multi-homers who have an app on only the iPhone and Mac App Store, all six have first created their iPhone app, before they launched a Mac App.

In total, 22 publishers have apps in all three stores. Of those 22 publishers, 10 publishers have first developed an iPhone app, then an iPad app and finally a Mac app. Two publishers have first launched their iPad app, then their iPhone app and later their Mac app. Eight publishers launched their iPhone and iPad apps at the same day (universal app) and later launched a Mac app. The remaining two publishers first created an iPhone app, then a Mac app and later an iPad app.

Summarizing, the iPhone platform is often the platform of choice for a new entrant in the Apple ecosystem. If the developer desires to target another App Store next to the iPhone, in most cases it will be the iPad app. If a publisher does not create an iPhone app, it most often first creates an iPad app and later a Mac app. The other way around, from a Mac app to an iPhone or iPad app, is rare.

5.4 Why do publishers make these choices?

Out of all 182 multi-homers found in the data set, 30 were contacted to answer a short questionnaire about the reasons behind their multi-homing behavior. The reason for contacting only some of the publishers is the fact that contact data was not available in the data set extracted from the App Stores. Thus, gathering contact data had to be done manually, which was a labour intensive task. The publishers were chosen based on the number of apps they had in the stores. Even though the response rate was low, 13%, time constraints did not permit for contacting more publishers. Despite this, the responses to the questionnaire provide some interesting insights into the reasons behind the behavior that was observed from the App Store data and confirm some of the statistical findings.

The respondents were all active in the iPhone and iPad ecosystems, with only one publisher being active in the Mac ecosystem as well. All respondents to the questionnaire listed a wider potential customer base as the main reason for targeting more than one platform. One respondent active in both the iPhone and iPad ecosystems stated that it did not target only the iPhone because “the iPad represents a significant number of [additional] users, as well as revenue”. Targeting a new platform does not only bring new customers into reach it also includes potential sales to the same customers that have more than one Apple product. Companies believe most Apple users own more than one Apple product.

Another strategic factor for targeting more than one platform is the fact that using the same programming language for all platforms makes it very easy to port an application from one platform to another. Developer resources are mainly used for one platform with less than a quarter of these resources being used for a different platform.
Also mentionable is the fact that companies see that some categories of apps are more popular on a specific platform. For example the iPad has been observed to be used more in enterprises and so Productivity applications are more popular on this platform than on the other portable platform, the iPhone. This confirms the statistical analysis per category that we presented earlier.

From the perspective of related benefits and negative effects of multi-homing there is a general opinion that customers of the Apple ecosystem view the three sub-ecosystems without strict boundaries. On the positive side, targeting a new platform will increase sales for an existing product on a different platform. This is enhanced by efficient cross-platform promotion of the products. On the negative side, however, users see a product that is available on more platforms as the same piece of software and expect it to behave in the same manner. A respondent gave the example of a “Tips Guide” for the iPhone app: “the content of which covers only the iPhone, but by being an universal app, some customers thought it would also cover the iPad”. Another issue is that negative reviews on one platform will have effects on the other platforms as well, for example, one respondent stated that there is a “risk that some bugs or design decisions in one version will result in poor user reviews that will be shown to the users of the other platform”. There is also a tendency from users to compare the portable platforms to the Mac platform, in this case Mac applications are seen as over priced even though they do require more effort and resources than their simpler and smaller portable versions.

Summarizing, multi-homing from a strategic perspective is motivated mainly by a wider customer base and the portability the Apple sub-ecosystems offer. This behaviour is observed to bring higher sales, however there is also the risk that a product inherits the same (potentially bad) reputation on all platforms it targets.

6 Discussion

Even though the data set used for this research was large and detailed enough to perform relevant statistical computations, there are some limitations posed by the collection method used and sample data that was gathered.

The first limitation is that the population consists only of the top 300 popular free and top 300 paid applications in each of the App Stores at a certain moment in time. Publishers could be in the top 300 only on one of the three platforms even though they do publish in more than just one App Store. These occurrences were omitted by the method used to collect data, thus making the data have a certain degree of incompleteness. Another limitation incurred because of the collection method is the fact that the selection of publishers was not random. The selection was based on popularity. The rationale behind this choice to consider only the top 300 most popular applications, was that this was the only App Store data that was freely and publicly available. For further research in the Apple ecosystem it would be interesting to select publishers randomly in order to have a more representative data set. Lastly, the low response rate of the
questionnaire, even though it provided important insights from the publishers’ perspective, does prevent us to conclude that these are the only reasons why publishers target multiple ecosystems.

Other limitations come from the fact that the concept of the App Store is relatively new. The Mac App Store launched less than a year ago, however there have been applications for Mac before this. The market within the Apple ecosystem is continuously developing and evolving, so data from this research can quickly become obsolete.

7 Conclusions & Further Research

This paper analyzed the trends and characteristics of software companies within the Apple ecosystem and its three sub-ecosystems: iPhone, iPad and Mac. The focus of the analysis was on multi-homers within these ecosystems. A multi-homer is a software publisher that offers products on more than one platform. The dataset that the research used was composed of the top 300 most popular free and 300 most popular paid applications in each of the three App Stores.

The results show that there is a relation between the category or type of application and the number of platforms that it targets. Apps categorized as Games are the most popular on all three platforms, while apps categorized as Books are only popular on iPhone and iPad and productivity apps are popular on Mac. Statistical correlation has shown that the number of apps a publisher has, correlates with the number of platforms it targets, however the large game-publisher Electronic Arts is an exception. It only targets iPhone and iPad while it is the publisher with the highest number of applications over all.

Focusing on the order which publishers use to target the ecosystems, the iPhone platform is usually the first platform a publisher targets. This does not only have to do with the fact that the iPhone was the first platform to have an App Store, because we used the current top 300 most popular paid and free apps. A better explanation is, probably, the market size of iPhone apps compared to iPad or Mac apps. Publishers that target more than one ecosystem usually proceed by launching an iPad app after the introduction of their iPhone app.

Also observable was the fact that if publishers first target the iPad ecosystem the next to be targeted is usually the Mac ecosystem. When asked directly, multi-homing publishers stated that targeting more potential customers and the high portability the three ecosystems offer were the main reasons for targeting more than one platform.

The findings of the paper enable better decision making for publishers in the Apple SECO. These findings show what the current general behavior of multi-homers in the ecosystem is and also identify strategic benefits and risks that need to be taken into account when deciding on adopting a multi-homing approach. This information can be used by publishers to enhance their strategic analysis of the environment.

As stated in the discussion section, further research in this area is necessary. A first direction would be to perform the same research on data-sets that
are selected randomly, in order to improve generalizability of the results. Also, responses from more publishers should be gathered in order to better understand the different motivations behind their strategies. Another interesting aspect would be to look at the data from the App Stores in more than one moment in time in order to get a more detailed and accurate trend analysis. An approach to multi-homer behavior within other platforms could also be a cause for further research. For instance, multi-homers within the iPhone, Android and Windows Mobile ecosystems or multi-homers in the Mac and Windows ecosystems could be analyzed.

References

Implications of Openness in a Software Ecosystem: A Health Perspective

Frederik Mijnhardt, Niels van den Berg

Utrecht University, The Netherlands
{f.mijnhardt, n.vandenberg2}@students.uu.nl

Abstract. For a software ecosystem to prosper, it is essential to be in a healthy state. An important factor for an ecosystem’s health is the robustness of its network. The most influential partner in any ecosystem is the keystone organization. Keystones have a variety of options to open up their business towards partners in their ecosystem. In this paper we research the relationship between openness and ecosystem network health, to work towards an openness-health model. By performing a literature study, a survey with an expert group and two case studies, we determine which strategic, tactical or operational decisions have the biggest impact on ecosystem health. Results show for an organization to increase their network health through openness, only a small number of openness options have to be addressed to get a high impact. Furthermore, organizations can compensate the lack of openness by investing heavily in other options, leading to a combined positive health for the ecosystem.

Keywords: software ecosystem, ecosystem health, openness, mobile platform, keystone

1 Introduction

In modern business environments, companies are highly dependent on others for delivering an end product. These networks of organizations working together are called business ecosystems [1]. Business ecosystems are characterized by being more than just the classic value chain. The organizations act as partners, working together to create a high value product. To deal with barriers between the ecosystem partners it is important to align strategic decisions and focus on communication channels.

An area where business ecosystem have evolved into software ecosystem, is the software industry. Software organizations in particular depend on service providers, software suppliers, value added resellers and even customers [2]. For the purpose of this research, we therefore use the definition of Software Ecosystems (SECO), introduced by Jansen, Finkelstein and Brinkkemper [3]: “A software ecosystem is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are
frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts.”

The most important player in a SECO is the keystone organization. The keystone is the driver for keeping the ecosystem healthy; it can increase ecosystem productivity by enabling partners and is characterized by having high influential powers within the ecosystem. An example, explaining the role of a keystone, can be found in the mobile app industry. This is a dynamic industry in which a large amount of stakeholders cooperate and compete. Examples of keystones in their respectively own ecosystems are Apple iOS and Google Android. While the ecosystem consists of numerous smaller and larger organizations, the keystones steer the direction of the entire ecosystem, define what new technologies will be implemented, promote new business opportunities and keep the ecosystem robust for the future.

For a software ecosystem to survive it is essential to be healthy. The factor of health as a performance indicator for business ecosystems was introduced by Iansiti and Levien [4] and was further extended by Den Hartigh, Tol and Visscher [5]. Ecosystem health is for a large part influenced through the behavior and strategies of the keystone. Looking from the keystone perspective it is therefore particularly interesting to find business areas in which improvement could directly lead to better health of the ecosystem.

One such area, that is especially of value to Software Ecosystems is the openness of the keystone organization. Openness was first introduced by Jansen, Brinkkemper, Luinenberg and Souer [6]. They propose a model for determining openness choices, giving companies the ability to assess the level of openness for five of their organizational departments.

The goal of this paper is to determine to which extent the openness strategy of a keystone player influences the health of an ecosystem. Our research question is as follows: “Can a keystone’s openness strategy, influence the health of the ecosystem in which it operates? And, to which degree is openness positively related to ecosystem health?”

This paper continues by discussing the research method in section 2, continued by a literature study in section 3, to identify known relationships between specific openness options and health factors. In section 4 we work towards an openness-health model, to completely map the influence of all openness options on ecosystem health. Section 5 presents the analysis from the case study research. The last sections are dedicated to discussion, drawing conclusions and suggestions for future research.

2 Research method

The paper’s research approach is threefold. By first performing a small literature study we determine whether a relationship between openness and health could exist. Subsequently, by arranging an expert group meeting of master students an openness-health model is proposed, which aims to map the influence of all openness options from the openness model to the factors that constitute ecosystem health. Finally, this model is evaluated by performing two case studies on similar SECOs.
As mentioned above, a openness-health model has been developed in cooperation with an expert group of fourteen master students at Utrecht University. After dividing the openness options between the students, they were asked to rate the implied impact of openness options on three network health measurements. This rating is based on a four point scale, respectively meaning: no implied impact, low implied impact, high implied impact or a fourth option to leave the question open if the student found him/herself not to have the relevant knowledge to answer the question. The questionnaires were then discussed with the expert group to gain further insights. To further justify results, where possible related literature was used to confirm results given by the expert group.

To evaluate our proposed model and determine whether a link could be drawn between ecosystem openness and its health, we performed two case studies on very similar ecosystems: Apple iOS and Google’s Android OS. These two ecosystems were selected based on the amount of organizations in the ecosystem, similarities between the keystones and the keystones strategies and position in the mobile industry ecosystem. For the case study analysis we derived techniques proposed by Yan-Ru Li in his analysis on the Cisco ecosystem [7]. The position of the ecosystems, subject of this case study is depicted in figure 1.

![Fig. 1. Chosen case studies position in the Mobile OS Ecosystem.](image)

For each ecosystem we described the characteristics of the keystone player and the ecosystem environment. In succession, we adhered the openness [6] and categorized them in matrix form. Information retrieval was done by investigating Apple’s iOS and Google’s Android OS developer guidelines and company policies. Lastly we used the health measurements [5] to rate the health of both ecosystems.
3 Literature study

The basis of this research lies in two papers that explain the concepts of openness and ecosystem health. In this section we determine whether a relationship between the model and health measurements can be found. The concepts of openness options and network health measurements are explained as follows:

There are 59 options organizations have to open up their business [6]. The factors are divided into five horizontal themes: Governance, Research and Development (R&D), Software Product Management, Marketing and Sales and Consulting and Support services. The factors within these themes are also grouped vertically on three organizational levels: Strategic, Tactical and Operational. Through this matrix, external parties can easily assess an organization's openness and determine whether the governance strategy of an organization is very open, as opposed to the operational factors of the R&D department.

The notion of Ecosystem health is best described by Den Hartigh, Tol and Visscher [5] who use the health factors [4] to introduce practical measurements for determining an ecosystem's network health. Measuring network health is done by assessing the number of partners, the visibility in the market and the covariance of partner variety within the market, as displayed in figure 2. These measurements reflect the overall robustness and strength of the network.

![Fig. 2. Ecosystem health and its underlying measurements, as described by Den Hartigh, Tol and Visscher (2006)](image)

This research will only adhere to network health measurements from the health model, because partner health measurements are focused on individual partners and their financial health. For a large ecosystem this cannot be calculated within the scope of this research. Furthermore, since focus lies on the perspective of the keystone, these contextual variables play a lesser role.

As a result of research performed into business- and software ecosystems a number of papers have been published on: ecosystem health, increasing of the number of
partnerships and organizational openness. However, no research has been done on finding a relationship between openness and ecosystem health. By taking a closer look at the openness options and network health measurements we can determine a relationship is possible:

For this study we first looked into the network health measurement: Number of Partnerships (NoP). As higher connectedness leads to a higher health [8], we searched for examples where as a result of an openness option a higher number of partners have emerged. An example of this happening was mentioned by Iansiti and Levien (2004), while discussed the case of E-Bay. Where the openness option, the creation of new business models for partners, has led to a great increase of new people being able to auction materials using their platform, resulting in over 70 million partners.

The second important factor is the popularity of the company, relatively measured to other companies, leading to a higher health [8]. In this research referred to as Visibility in the Market (VitM). An example is mentioned by Bushee and Miller [9], discussing an investor (or partner) program giving a network of similar smaller investment firms the ability to grow their VitM. Due to their higher visibility they created a number of new business opportunities with large investors, they before could not reach.

The last network health measurement is the Covariance of Partner (CoP). A relative high covariance of partners leading to higher health [5]. This can be underlined by an example that showed how a number of IT partner operating under different roles added more value to the end product, and network, than another competing ecosystem not having the same CoP [10].

We will not claim a relationship existing between openness and health based on the above mentioned examples, though we see a certain tendency towards a possible relation between the two. The results of this literature study has led us to continue our research, to further work towards an openness-health model.

4 Towards an openness-health model

In light of our research question, we propose a new model to indicate the impact of openness on an ecosystem’s health. By combining implied impact from openness options for a keystone and health measurements for an ecosystem in a model, it becomes more easy for an organization to identify the health or unhealthiness of the ecosystem it resides in and the steps it could take to make their ecosystem healthier. The factors which we combine are described in the following subsections.
The openness-health model is described in Table 1. For each section, strategic, tactical and operational openness options have been identified and categorized in the five themes. For all options the implied impact on the ecosystems network health. Three network health measurements have been used: NoP, VitM and CoP. The impact shown in the model are based on 159 openness options, gathered per theme and summarised in the model. The implied impact per openness option has been rated by the expert group. The summary shows high implied impact when more than half the openness option for that section are rated high. In the following subsections the openness model will be discussed. For the governance theme the openness options for the operational level show the highest impact. By making the ecosystem explicit they keystone can give a high positive impact on network health. By making the ecosystem explicit the keystone can impact all three network health measurements. In addition by creating user groups and partnership models have a high impact on health. Lower impact is expected on the tactical level. In total 20% of the openness options for Governance were rated high, while 78% was rated low. 2% does not have impact

Research and development, having the most openness option, results in high implied impact on the operational level and for the number of partnership also on the governance level. For the operational level providing, facilitating and helping others in their development process, by for example publishing api’s, promoting co-development and creating reuse enabling architectures has shown to have a high effect on especially the NoP and the VitM. On the strategic level the sharing of source code and technological roadmaps has a high effect on health. In total over 41% was rated high, though very evenly spread out over the 54 openness options for R&D. 57% has a low impact, and 2% has no impact.

Software product Management score specifically high on the strategic option, though with 2 openness options, 4 out of 6 health measurements were rated high. Where sharing of platform strategy and sharing of product life cycle plans for products are rated high. In total 29% of openness options were rated high, with 71% rating low.

Expert group results show the openness options for marketing and sales have a very high impact on the VitM. out of nine Openness options, six have been rated high. Most important openness option is the development of multiple distribution channels, leading in high impact for all three network health measurements. Two other high impact openness options are the development of innovative business models and creation of in- and external components markets. In total 44% of the openness options have a high impact on health and 56% having a low impact.

Consulting and Support Services show to have least impact on network health. With only 11% of openness options having a high impact. The most important openness option is on the operation level by using collaborative workspaces for customers leading to a high number of partnerships and high covariance of partnerships. In total 85% of openness options have a low impact and 4% have no impact.
Based on the expert group the most important themes for affecting network health are the Research and Development and Marketing and Sales themes. Least important is Consulting and Support Services, with Governance and Software Product Management being equally important. Another interesting aspect is the correlation between openness options affecting multiple network health measurements. In 50% of the cases when an openness option affects one of three health measurements, it also affects the second, or even third health measurement. Meaning that 16 openness option have a high implied impact on health for either two or three health measurements. Thus improving on only a small number of openness options could lead to a relative high impact on network health

5 Case studies

In the following subsections the two subjects of the case study research, Apple’s iOS and Google’s Android Platform have been briefly introduced. In subsection 4.3, the SECOs are compared with the openness-health model that has been proposed in section four. Furthermore, an elaborate description of circumstantial factors is given for each organization to qualitatively position findings in their context. To conclude these case studies, the results and their implications are discussed in section 5.

5.1 Case study: Google Android

Android is Google’s mobile operating system. Android was first released in October 2008 and is now used as an operating system for multiple types of mobile devices from different companies. Android is an open platform under the Apache 2.0 license that makes use of a Linux kernel. Android Market is Google’s mobile application store on Android. Through this platform, developers can distribute software to Android users. Figures from Distimo reports show that Android Market had 294,738 available apps on May 1st 2011 [11]. Android Market is currently only available for developers in 29 countries. Although Google has stated to be opening up their platform to more and more countries in the near future, a relatively large amount of potential developers can not join this ecosystem at the moment.

5.2 Case study: Apple iOS

iOS is Apple’s mobile operating system. iOS was released in June 2007 when the first version of the iPhone came to market. A proprietary variant of the Unix based OS X runs on multiple types of Apple’s mobile devices such as the iPhone, the iPad and the iPod Touch. Figures from Distimo reports show that the App Store had 381,062 available apps on May 1st 2011 [11]. To develop software for iOS, one must register with Apple as a developer at an annual fee of $99,- or as a software company, for $299,- a year. This last membership model includes code-level feedback from official Apple developers.
5.3 Case study comparison

Tables 1 to 5 show comparison tables of all 32 openness options that were valued through the expert group meeting as being of high influence on any three of the network health measurements. Options that were rated to have a low or no impact are left out, because they are considered to be unimportant for this case study.

A platform can either comply to an openness option, which will result in yes or not comply, which results in no. The most left column of each table indicates the organizational level wherein the option lies, strategic (s), tactical (t) or operational (o). When no satisfactory answer could be found, the cell was marked with unknown. Some ratings are marked with an asterisk, to indicate a nuance is in place. The remaining part of this section is used to explain how each of these values has been determined and what nuances apply.

5.3.1 Governance

Table 2. Compliance to governance openness options

<table>
<thead>
<tr>
<th>Open up governance</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create partnership model</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Coordinate contributions to other ecosystems</td>
<td>no</td>
<td>unknown</td>
</tr>
<tr>
<td>Make ecosystem explicit</td>
<td>unknown</td>
<td>unknown</td>
</tr>
<tr>
<td>Create a partner directory</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create user groups</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

iOS is strictly governed by Apple. No outsiders have a direct say in future developments. With Android this is somewhat different. Google actively monitors the Android community and derives future changes from there. Also Android partners, like Samsung and HTC have a say in how the mobile operating system should behave.

Both Apple and Android provide partnership models to developers in their ecosystem. Especially the Apple iOS, where developers can sell their applications, is a key point in the business model behind iOS.

Both platforms do not show to be actively contributing to other ecosystems than their own. Of course developers in both ecosystems are active in multiple ecosystems, but they should not be considered part of the platform itself. However, since Android is an completely open platform, that is not exclusively bound to its own hardware, we assume that Google might be active in ecosystems of device developers like HTC and Samsung. Yet this is not explicitly stated, therefore the cell is left blank.

It is unknown to what extent the iOS and Android platform are made explicit. Especially because both platforms iterate fast and migrate to different device types, Google and Apple might be reticent in defining strict borders to what their platform exactly encompasses.

Both Apple and Android provide a partner directory on their websites. They also facilitate the creation of user groups.
5.3.2 Research and Development

Table 3. Compliance to R&D openness options

<table>
<thead>
<tr>
<th>Research and Development</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share technology and research roadmap</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Stimulate open standards</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Share source code</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Share innovations</td>
<td>unknown</td>
<td>yes</td>
</tr>
<tr>
<td>Outsource tasks</td>
<td>no</td>
<td>unknown</td>
</tr>
<tr>
<td>Certify third-party components</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create and publish (content) APIs and SDKs</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create reuse enabling architecture</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Do co-development</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Provide developer training</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Propagate software operation knowledge</td>
<td>no*</td>
<td>yes</td>
</tr>
</tbody>
</table>

Apple is known for enthusiastically sharing their technology and research roadmap in their keynote presentations and specially organized developer conferences. Google does this as well, but on a lower profile.

Both organizations stimulate open standards. Android, however, is founded and thriving on its open nature, iOS is not. Nevertheless, Apple also sees the value of open standards and supports many of them in extension to their own platform. One of the most important differences between Apple and Android, is that the latter has a completely open source code. The source code of iOS is proprietary. The business models of both mobile application stores rely heavily on the fact that the platform owners leave opportunities for developers. Sharing innovations therefore is something both Apple and Android do. The option of outsourcing tasks was hard to determine for both platforms. No evidence was found that Apple outsources certain tasks of their core platform to third parties. In the case of Android it is even harder to determine, since the latter is a completely open platform, that is used by many other companies that could be considered part of the ecosystem, but are not actually performing tasks in the name of Android.

Certification of components is done by Apple. Apple also exclusively lets certified retailers sell their products. Finally, Apple provides certifications for professionals using their software. Android also certifies device manufacturers in using their platform.

Supporting and providing good APIs greatly improves the openness of a platform. Both Google and Apple provide numerous APIs, that allow developers to use the OS, and the devices it runs on to their full extent, in a standardized fashion.

Apple does not openly claim to have co-developed parts of the iOS platform with third parties. Android however does. To wrap up the openness options within the research and development department, both Apple and Google give developer training. The asterisk for Apple is in place, because Apple does not provide much...
information about this to outsiders. But only the fact that their conferences like Apple’s worldwide developers conference (WWDC) contain all sorts of workshops for developers, developer training can be considered supported. The last openness option in the R&D department that was found of high importance is the propagation of software operation knowledge (SOK). Google has recently added a variety of SOK-features to the Android platform in order for developers to be informed when their software malfunctions. Apple does not have this yet, but announced to be supporting SOK in iOS 5.

### 5.3.3 Software Product Management

#### Table 4. Compliance to SPM openness options

<table>
<thead>
<tr>
<th>Software Product Management</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share product lifecycle plans for products</td>
<td>yes</td>
<td>unknown</td>
</tr>
<tr>
<td>Share platform strategy</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Outsource requirements engineering to partners</td>
<td>no</td>
<td>unknown</td>
</tr>
<tr>
<td>Share and adjust product (line) roadmap(s)</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Open up requirements management process</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

In sharing product lifecycle plans, Apple is more open. This is probably due to the fact that Apple is not reliant on third party hardware, and therefore can determine themselves whether a OS version is still supported or not. Because Google cannot always do this with Android, the cell is left blank. There is insight in the platform strategy of both iOS and Android.

Requirements engineering for iOS is done exclusively by Apple. However, it could be said that obtaining interesting functionalities from partners, for the platform is a way of requirements engineering. It is to what extent Android does this, since device developers can adapt the OS to their own liking if needed. Yet, because this form of making adaptations is device-specific, and not merged with the original Android version, they are not considered part of the platform.

Apple and Google also both share the roadmaps for their product lines. Apple does this mainly through publicly available keynotes. Google on the other hand, keeps more quiet, until it gives access to early beta versions. The exact working of the requirements management process are hard to grasp, since neither platform has a dedicated tool for this. However, based on information available on the websites of both platforms we assessed Android has a more open requirements management process, since they have an active community of practice around the core development of the platform. It is reasonable to assume Apple also listens to what developers want, but the one-way nature around the development aspect of the core platform makes this impossible to verify.
5.3.4 Marketing and Sales

Table 5. Compliance to Marketing and Sales openness options

<table>
<thead>
<tr>
<th>Option</th>
<th>IOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share market vision</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Develop innovative business models</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create sales partner program</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Share market information</td>
<td>yes</td>
<td>unknown</td>
</tr>
<tr>
<td>Develop distribution channels</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Certify partners</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Create in- and external components markets</td>
<td>yes</td>
<td>unknown</td>
</tr>
<tr>
<td>Involve partners in marketing and sales</td>
<td>yes*</td>
<td>yes</td>
</tr>
</tbody>
</table>

On the marketing and sales side of openness options, Google and Apple share their market vision. They also develop innovative business models, and share opportunities with partners.

Apple has a sales partner program, while Google does not. Also Apple likes to share statistics concerning market-, customer- and supplier information with their developers and shareholders. Android does this as well, but provides less information to the general public. As mentioned, the role of the distribution channels within the OS platforms is critical for the platform’s success. Apple and Google both know this and give their app stores a prominent place in the OS and the ecosystem. Both organizations certify partners, as mentioned in the SPM-part earlier. Apple has created an extensive internal- and external components market with which it supplies a lot of sales and new development opportunities. The same is harder to say for Android since it is not so much reliant on their own devices. Both parties involve partners in marketing and sales. Apple however is much stricter in this and only allows few selected retailers to sell their products.

5.3.5 Consulting and Support services

Table 6. Compliance to Consulting and Support Services openness options

<table>
<thead>
<tr>
<th>Option</th>
<th>IOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outsource imp. projects to partners</td>
<td>unknown</td>
<td>yes</td>
</tr>
<tr>
<td>Use collaborative workspaces for customer</td>
<td>unknown</td>
<td>unknown</td>
</tr>
</tbody>
</table>

There are only two relevant consulting and support options with effects on ecosystem health. The first is the outsourcing of implementation projects to partners. Google does this with Android, for example by letting device manufacturers deploy their own Android versions. Apple does not, with iOS. We could not find satisfying
answers to whether either of the platforms provides collaborative work-spaces for customers.

5.4 iOS and Android ecosystems health

In section three the network health measurements have been shown as a part of ecosystem health. In this subsection we determine and compare these for both iOS and Android. Figure 3 shows the results. Based on the results, no significant unhealthy network measures have arisen. The visibility in the market shows somewhat similar results, by taking into account insignificant results both iOS and Android are very popular and well known within the market. Although Apple iOS has a higher number of partnerships, this is compensated by the higher covariance of partners. Based on these results both networks are in similar health.

![Figure 3. iOS and Android network health](image)

6 Case study analysis

By comparing the mobile platforms of Apple and Google in the previous section, we have gained insight in how these two differ in their openness strategy. Even though both platforms share similar approaches to most openness options, some interesting findings remain. This section elaborates on these findings and how they affect the health of the ecosystem at hand.

Starting by looking at the two ecosystems on department level, we can see that Apple is particularly open in their marketing department. Google, in its turn is more
open in the core of the platform, which is mostly covered in the research and development department. Therefore it can be said that Android is in essence a more open platform, for by definition, its core source code is publicly available. Apple somewhat negates this difference by showing its value and providing extensive support and sharing opportunities with developers in their ecosystem. The implications of these different approaches on ecosystem health are therefore hard to capture, since Apple makes up for the lack in core-platform openness with contextual openness. Nevertheless, based on these results we can say that Android increases the health of its ecosystem in a more sustainable way, and makes fewer concessions that could potentially scare developers. This point is strengthened through Android being more accessible for new developers, since no development fee is required.

7 Discussion

The openness-health matrix as shown in section 3 can be considered a way to qualitatively estimate the effect of a keystone’s choices in openness on the health of its ecosystem. The most important reason for the fact that this is not a quantitative measurement, is that each openness option is only weighted on a three-point scale, due to the preliminary nature of this research. By either having no, a low or a high effect on the health of the ecosystem, this simplistic scale gives just enough granularity to sort what can be considered important and unimportant factors from one another, but it does not give a basis to underlie statements about these effects with rich data that would for example be suited to make quantitative comparisons. In addition, external factors could also influence network health, clouding case study results.

Before working towards a better model, the criteria for selecting members for the expert group should be increased, ascertaining quality of results. In addition more room should be given to discuss the openness options, as a quantitative study alone will not represent the entire context.

Furthermore, this research was performed in context of case studies on ecosystems of commercial mobile phone platforms. One could argue that the weight we assigned to each openness option could differ when applying the same method to other types of ecosystems, for example non-commercial ecosystems tend to have a slim, or no marketing department, so any openness options that coincide with this department are less relevant.

There is also some critique on the network health measurements, as proposed by Den Hartigh et al. [5]. For example the visibility in the market is measured by the number of search results shown when a Google search query is performed, on the name of the platform. We find this an highly unreliable way of data gathering, since first of all, the exact name used in the search query is ambiguous to the search engine, and can incorporate many unrelated results. Next to that the number of pages found varies regularly, not only because more sources are created, but also because Google adjusts their algorithm.
A final remark on the weighting in this matrix is that some factors can either stimulate or stagnate openness, completely depending on the way it is executed by the organization at hand.

8 Conclusion

In this paper we present the results of a literature study done on openness and network health measurements. We propose an openness-health model, based on an expert group meeting and literature. The proposed model has been evaluated by performing two case study researches on similar software ecosystems. The two case studies have been compared on their openness options and their ecosystems network health.

Based on the results of the expert group meeting we see the most important openness themes residing in the Research and Development and the Marketing and Sales departments. Where over 40% of the openness options have a high impact on network health. A real correlation between Strategic, Operational or Tactical openness options and network health cannot be found. For each department a different category of openness options seems to be important.

Another conclusion is the correlation found between openness options affecting multiple network health measurements. For over half the cases the openness option affects multiple network health measurements. To improve an ecosystems network health, a keystone can now work on a number of ‘quick wins’ to greatly increase the NoP, VitM and CoP. Three main ‘quick wins’ are: Certifying third-party components, developing distribution channels and making their ecosystem explicit.

The case study comparison leads to the conclusion that complying to different openness options has different results for the health of the ecosystem. However we can not exactly say how these relate, in the case of Apple any negative influences from the closed attitude around the source and development of the source seems to be negated by a very open marketing and sales department and supplying good APIs and developer support.

Based on this research we can determine that the keystone organization can, by adjusting certain openness strategies, influence the network health of a SECO. Also, the expert group results show the relation between openness options and ecosystem network health have, as long as they are executed well, a positive impact on network health.

For this model to become more mature, future research could be devoted to analyze individual implications of openness options more profoundly. This way, a quantitative way of measuring the performance on each option could lead to insightful data that can be used to for example show trends in an organizations’ ecosystem health performance in terms of openness.
9 Acknowledgements

We thank the MBI students from Utrecht University who participated in developing the openness-health model.

References

Investigating the relationship between open source business models of software companies and their ecosystem.

Metodi Amov and Katharina Pietzka

Utrecht University, {M.Amov, K.Pietzka} @students.uu.nl

Abstract. This paper looks at open source software companies’ business models and their surrounding ecosystems. After digging into the literature on this topic a framework for evaluation of software business models was selected as a base for the paper. This framework was applied to a case study of two enterprises, their ecosystems and products. The business models of the companies are thoroughly described and their relationships with the ecosystems discussed. The presence of these relationships is presented and discussed. Implications of decisions from the business model over the ecosystems are also elaborated on.

Keywords: Business model, open source, software business models, software ecosystem, open source software ecosystem

1 Introduction

In recent years the importance of having an “innovative” business model has been under discussion. The continuing process of business model innovation provides companies with a parallel way to outperform competition [1]. The innovation in business models have brought billions of dollars of revenue and reshaped entire industries [2]. Organizations in the software industry can no longer be satisfied by the standard product software business model with a straightforward revenue system or a distribution channel. With the fast developing ICT environment business models need to be adapted faster to the changing situation. After the dot com bubble in the beginning of the second millennium, software companies became more pragmatic and safety oriented, as investors became cautious about investment in the industry of software development. Chesrough [3] even says that a better business model often will beat a better idea or technology.

The introduction of Open Source software products made the profit generating task for some software providers even harder. The free software idea did not immediately become mainstream [4], but in recent years it has emerged as a major cultural and economic phenomenon. For the purpose of this paper we use Open Source
Initiative’s\(^1\) definition of open source software: software which source code is freely available to the public, possibly under a license (GPL, GNU etc.). Today thousands of such projects exist with many supporting developers, and users. Even though the software might be under some license, the fact that the source code is freely available to all and vast amount of contributors allowed for the introduction of different (from commercial development) software development practices which involve great number of revisions and many viewpoints over the structure and contents of each project.

All these developments accounted for high quality software and the widespread usage of such products. Having this in mind more companies are considering on “opening” their products or starting open source projects. There are many different open source business models (OSBM) associated with different license approaches to regulate usage \[5\]. Because different licenses exist a business model for an open source product gains more opportunities for restructure and innovation. The different types of such business models have different impact over the community, the platform they are using and the ecosystem as a whole. If a business model is not well thought, it could make a good product non-desirable for the public, therefore limiting the expansion of the ecosystem it resides in. On the other hand the flexibility of OSBM can generate a better performance within the ecosystem and higher involvement from developers.

The topic of OSBM is interesting for research for both the open source community and for commercial software communities, as some pattern, if such existing, could be picked up by commercial developers to insure a healthy ecosystem. Furthermore open source projects on average are more prone to sharing information and thus being easily accessible by researchers. Another point considered while determining this research’s agenda is the lucrative business opportunities that OSBM provide for companies. The more managers understand the links between business models and the reactions of a particular business model from the ecosystem, the better response they could have when designing new models, products and satisfying product users.

The aim of this research is to analyze the business models of products within an open-source ecosystem. This analysis aims at determining any factors within the ecosystem that influence the way business models are structured or internal and external factors of the ecosystem that drive companies towards choosing the models they have. Factors of influence of the business models towards the ecosystem will also be considered. We argue that common characteristics of the business models of open source software ecosystems exist. This is why our research question is:

What are the relationships between an open source ecosystem and the business models of products within it and how do they influence each other?

The paper is structured as follows; chapter 2 will give an overview about the related literature concerning open source business models and software ecosystems. In chapter 3, the research method is being described that is used for conducting the research. It will cover the approach to identify and gather the relevant data, and the way to analyze it according to the research goal. Building up on the previous, chapter 4 consists of the case study. In the following part, the results of the case study

\(^1\) http://opensource.org/docs/osd
comparison. Chapter 6 consists of the discussion; limitations of the research are presented, as well as the relevance and suggestions for further research.

2 Literature Analysis

Business models are often used, but sometimes not clearly defined. Richard Rosenbloom and Chesbrough developed a six point definition of a business model [6]. They look at a business model as consisting of six key elements - a value proposition, target market, value chain, revenue mechanism, value network or ecosystem and competitive strategy. Furthermore Chesbrough [3] describes different types of business models in which companies can belong. The six types are - companies that have an unidentified business model, have some differentiation in their business models, develop a segmented business model, have an externally aware business model, and integrate their innovation process with the business models and companies whose business model is as adaptive platform. This classification is good foundation for determining a company’s current business model state.

When analyzing business models, different approaches could be taken. Different techniques have been applied for evaluation of software business models [7][8][9]. As a basis for this research’s investigation the framework of Rajala et al. [9] has been used. In their work [9] investigated business models for software companies and developed a framework for the analysis of software business models. This model was chosen because it looks specifically on business models of software companies. Furthermore it provides a holistic and coherent view of the value proposition of software companies. Four core issues that have to be considered when analyzing business models in this research. These factors are “Product Strategy”, “Distribution Strategy”, “Revenue Logic” and “Services and Implementation”. Those categorizations have the purpose to help identify key characteristics of the businesses and compare them. The first factor “Product Strategy” deals with the description of product development focus and approach of a software company. The next category “Distribution Strategy” is dedicated to the marketing and sales operations in order to sell the product. “Revenue Logic” is for analyzing the way the company generates profit from the product. The way how the software is delivered to the customer is described in the last factor “Services and Implementation”. All those categories are interrelated and they are all part of the environment surrounding a company, or its ecosystem.

In order to asses a relationship between business models of companies and their surrounding software ecosystems a clear definition of an ecosystem is needed. Jansen et al. [10] define software ecosystems as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. This definition is used for the purpose of this research as it is also aligned with Rajala et al.’s [9] framework, which also can be used for analysis of any business as a part of a network. As argued by Jansen et al. [12] the openness of an ecosystem can stimulate the ecosystem and a company’s business goals. Thus the business model is an essential part of an ecosystem’s existence.
Jansen et al. [11] develop the Open Software Enterprise Model (OSEM) used to open up a software producing organization. The model uses five openness options aligned with three layers of openness. The model can be used to assess a certain company’s degree of openness. However, the usage of this model should provide a better understanding of the business model and the software ecosystem of a particular company. For instance the openness of one product could be a threat to the business model of another product within a similar ecosystem [12].

3 Method Description

This section includes the approach the authors took in order to conduct the research. It includes all activities needed to select the appropriate data, gather it and analyze it.

The research is qualitative and observational in its essence as most of the data was collected from the official web sites of the two case studies and their respective products. Official articles of the two companies were analyzed as well as other documents. Appendix A contains sources of information used in the case studies.

The two companies were chosen in accordance to the available official information that could have been obtained. Furthermore the companies needed to satisfy the criteria of being open source, have a structured (well defined) business model and have available information online, from which it can be determined the type and characteristics of the business model. Our choice of companies that have more information available is due to reasons of openness. As discussed by Jansen et al. extended enterprises can “open” their models by sharing innovation, know-how and information [11]. We considered this as important for our findings. In addition the products chosen needed to act in the same or a similar ecosystem so the contribution could be discussed.

The research considers only two companies, due to the fact that we are aiming to understand the implications of the business model towards the ecosystem in regards to the opinions of the contributors’ as well as the business outcomes. This is why the research looks at available official information (documents, websites, and financial reports) and user feedback (forums, discussions).

The assessment criteria of identifying the different business models are taken from Rajala et al.’s framework for analyzing software business models. The four dimensions (categories) of analysis are product strategy, revenue logic, distribution strategy and services and implementation. This research evaluates how components in each of the four categories shape relationships in the ecosystem of open source software. Furthermore the notion of openness was considered, when gathering information.

The scientific contribution of this paper is aimed at giving a better understanding of the relation of the impact of the ecosystem and the business model. Furthermore it can help practitioners in determining the outcomes of the business model strategies and decisions discussed.
4 Case Studies

This section includes a description of case studies on open source software companies’ business models. We will describe two software ecosystems: Mozilla and Red Hat Enterprise Linux. Even though both ecosystems operate in the open source sector, they have different business models.

4.1 Mozilla

Mozilla encompasses different organizations developing software. The Mozilla Foundation is a non-profit organization with the purpose of maintaining and developing the Mozilla software. According to the American IRC, Mozilla Foundation is tax exempt. The Mozilla Corporation is a commercial organization targeted at developing, marketing, supporting and sponsoring of the Mozilla software. It is a sub organization of the Mozilla Foundation. Mozilla messaging is a commercial daughter company of the Mozilla Foundation with the purpose of developing software for Internet communication. Mozilla is embedded in ecosystems of open source software usage. The software is used by private persons and companies.

The most common applications of Mozilla are the web browser Firefox, the e-mail client Thunderbird and a calendar application that can work as a stand-alone product or as an extension of Thunderbird. Furthermore, there is the bug tracking tool Bugzilla, Camino, which is a Mac OS optimized browser and the IRC client ChatZilla. Mozilla also supports other various applications.

The Mozilla Application Framework is a platform independent framework for the development of applications, which should be able to run on different operating systems. The purpose of the framework is to address the framework from different environments with different programming languages. It mainly consists of the Gecko Rendering Engine, the XUL-user interface-toolkit, the Necko network library and XPCOM, a cross-platform component collection.

The Mozilla software is open source. The whole source code is available for the users. Furthermore, it is also free, i.e. the users do not have to pay for Mozilla applications.

Mozilla generates revenue mainly from a sponsoring contract with Google. Mozilla provides the Google search engine by default in its browser and uses a customized Google website as a start page. Furthermore, Mozilla accepts donations.

4.2 Red Hat Enterprise Linux

Red Hat Enterprise Linux (RHEL) is a Linux distribution provided by the company Red Hat. It is suited to be used in an enterprise environment. It is published under the GPL license. It was developed with the purpose to offer a Linux version especially suited for professional customers; therefore it offers support and training. The

---

2 http://www.mozilla.org/about/organizations.html
ecosystem Red Hat operates in is highly professional with a high demand for reliability.

There are two main variants of RHEL: the server version and the desktop version. The server version consists of the Entry Server and the Advanced Server distribution. The desktop variant contains the Red Hat Desktop and Red Hat Enterprise Linux Workstation.

The RHEL is an enterprise operating system. This means, it considers the needs of larger companies in terms of long term stability and maintenance. Therefore, it is suited to be used on systems which are supposed to run very stable. Furthermore, it complies with the security standards of the Common Criteria Standard to guarantee a high level of security. Additionally, RHEL offers insurance against law actions concerning intellectual property.

The revenue for RHEL comes from subscriptions for the operating system. Red Hat offers trainings and schoolings as well as support for the system, which have to be paid [13].

5 Results

This section includes the results from the conducted research. The companies are being compared along the four dimensions of Rajal et al.. Afterwards, the business model of the company is put in a context with the ecosystem to establish relationships.

Mozilla’s product strategy is to reach a broad audience of private persons as well as enterprises who use their Internet applications like the browser or the e-mail client. The widespread recognition is what brings most the value of the company and with it, the incentive for third parties to invest in Mozilla. RHEL has the professional enterprise environment which depends on stable, reliable software solutions as target audience. They offer the system with maintenance and training contracts.

Mozilla gains money from partnership contracts like the one with Google. Third parties are interested in a cooperation with Mozilla because it is widely used and therefore a broad advertising platform. Additionally, they also generate money from donors, but there is no chance that Mozilla can generate all the needed revenue by subscriptions of the users for obtaining the software because the users will not be willing to pay for a browser or an e-mail client. RHEL’s revenue logic is more based on the classical subscription model; the customer pays a regular fee and gets the operating system and also the promise of a stable system for a long period plus the maintenance and trainings for the employees working with the system. Furthermore, big hardware and application vendors promise interoperability with the RHEL systems.

The distribution strategy of Mozilla involves online offerings with the option for a download from the company’s web site. It is not possible to obtain them from a local reseller or via CD or any other device. RHEL can be obtained via the website or via authorized resellers.

Concerning services and implementation, Mozilla leaves it to the customer. The user has to download the product and install it himself. There is a FAQ website and a
forum, where users can get help, but only to a certain extent and without the guarantee that the problems are solved. RHEL offers an implementation service and additionally a 24/7 online and phone support. So RHEL makes sure that the systems are working properly.

The following table shows a summary of the comparison of both ecosystems.

**Table 1.** Comparison of Mozilla and Red Hat Enterprise Linux.

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Mozilla</th>
<th>Red Hat Enterprise Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product strategy</td>
<td>reaching a widespread audience which obtain the product for free</td>
<td>providing a stable, reliable solution for a professional environment</td>
</tr>
<tr>
<td>Revenue logic</td>
<td>partnership contracts, donorships</td>
<td>subscriptions, maintenance, training, schooling</td>
</tr>
<tr>
<td>Distribution strategy</td>
<td>online downloads, no shops etc</td>
<td>online resellers</td>
</tr>
<tr>
<td>Services and implementation</td>
<td>download, self-installation, no further services, except FAQ</td>
<td>offering implementation services, 24/7 online and phone support</td>
</tr>
<tr>
<td></td>
<td>maybe forum support</td>
<td></td>
</tr>
</tbody>
</table>

To find out what business model is suitable for the product, the provider always has to take into account the ecosystem around the product; what are the needs of the ecosystem, how much money is there to obtain from the users, how much are they willing to pay? If the direct client is not willing to pay, what other ways of generating money are there?

Mozilla is embedded in the open source community. Since in general, web browsers are available for free, it does not seem to be a good strategy for Mozilla to sell their Firefox, since the users might switch to other providers. But Mozilla benefits from the community in another way. Since it is open for everyone, there are a lot of additional plugins, which makes the applications suitable for almost every user. This leads to a good product, a good reputation and of course to a widespread usage of the product. With this, parts of the revenue can be gathered from donations for the product. Users can donate for using the products of Mozilla. But this only makes a single-digit percentage of the revenue. Most of it is achieved by a cooperation contract with Google. Google has the right to integrate its search engine on the start page of the browser and in the search toolbar. Google gets a huge direct market share by having this arrangement and Mozilla becomes a value adding company.

Red Hat Enterprise Linux is integrated in and targeted for a business environment. A business has different needs. To guarantee reliable services, it needs to have a system that is reliable and stable. It has to be able to interact with hardware and other applications for a longer period without the need of updating and migrating too often. A company cannot take the risk of having operating systems that have too long
downtimes or problems with interacting with other standard application. Additionally, there are legal regulations for some industries about security. Other application vendors only open their solutions for operating systems if they have a long-term support guarantee. So, if the vendor promises a stable and reliable operating system, the using company will be willing to pay for it, if they will get assured maintenance and support.

6 Discussion

When we compare the two products, we are looking at two really different ecosystems. Even though both products are open source, they serve the needs of different types of customers (users); they have different revenue models, provide different incentives for collaboration and provide services in different levels. The outcome is that even though both products are open source, their business models have shown a great difference in the expansion and collaboration tactics of the ecosystem as well as its survival.

Both companies appear to be key-stone players in their ecosystems. They set the rules within the ecosystem, control their vision and make strategic decisions that give different outcomes.

In comparison of the product strategy, both companies share the collaboration aspect of allowing users to change their products, as they are open source. However Firefox’s development follows a hierarchy of development and collaboration that allows members of the community to work on the project. Both Mozilla and Red Hat use internal employees for development and decisions on the products, but Red Hat provides a proprietary model where development occurs within the company. In Mozilla’s case the focus is on use of volunteers for product changes. Red Hat however prohibits that in the proprietary version, by using own personnel. However the open source model gives the community a vote on important decisions. This decision of opening the collaboration process has brought Mozilla’s product strategy to a well-known product with a community that has a sense of belonging and autonomy. Red Hat members sense the presence of decision censorship, which could drive some supporters away. However Red Hat insures closeness and a feeling of belonging for its users through training seminars, awards and certifications.

The revenue logic part of the business model is where the two companies differ a lot. This is partly due to the different purpose of the products, however the decisions on such revenue logic affects the ecosystem. In Mozilla’s case the revenue logic is the part where it expands the ecosystem, by working with partners (mainly Google and ad-on developers). This is also where Firefox expects to see ecosystem actors’ appreciation with the possibility of donations. However due to the decision not to support corporate needs for the product, Firefox is limiting the donation revenue stream and relies more on private donations. Red Hat on the other hand uses the revenue logic to tighten up the connections in the ecosystem. By providing different packages and support solutions they assure quality for the price given. It makes a distinction between users that collaborate with money and the ones that collaborate with usage and pays off with quality to the one subsidizing it.
Both distribution strategies rely on their official websites for product distribution. However Firefox has limited the choice only to the official website. There are additional ways of distribution, such as peer-to-peer transfer, but these are non-official. Red Hat on the other hand, uses its revenue logic and distribution strategy more efficient in terms of adding new partners to the ecosystem. It uses resellers for the product.

Mozilla treats everyone the same in terms of services. Problems with the product and within the ecosystem are solved by its participants. There is no difference between a corporate user and a home user. This is a good strategy for keeping an overall feeling of fairness in the ecosystem; however as their revenue strategy suggests the appreciation is to be shown by the amount of happiness with the product. Unfortunately corporations have different needs and levels of support than home users and thus situations such as one developer from the core team (with sufficient knowledge) is trying to contribute to more than thousand computers, which does not satisfy neither side. Red Hat on the other hand shows the most of appreciation when it comes to services and implementation. The community solves problems for users that are not paying for the services, but with some disadvantages of waiting. However if the user pays for services, quality is assured.

Summarizing, it is crucial for a company to figure out in which environment they want to operate. If the company is targeting at businesses, it has to ensure a certain amount of stability, even if it is offering free open source products. It can generate income from maintenance and support contracts. If a company is providing rather small applications that are targeted at private users, there is less chance of generating money from the users. So, the offering company should maybe target at finding cooperation with another company that might benefit from being integrated in the product. Every decision made within the business model has a consequence on the life of the ecosystem.

6 Conclusion

This paper gave an overview of the importance of adapting the business model to the software ecosystem, a company is operating in. The company which wants to compete in a market has to be clear about the possibilities of how to generate money. It can either get the money directly from the users or find a third party that might be interested in investing in the company or product. Making thought through decisions in this context is crucial for a company’s success.

One limitation of this work is that more research on this topic should be done in order to support the results presented in this paper. Two case studies can already give an insight into the topic but are not able to cover the whole bandwidth of companies operating in software ecosystems. An analysis of the activity within the ecosystem should be done with a focus on user contribution. Also, interesting results could be generated if the relationship between the two ecosystems is studied.

Further research is recommended in the following fields; more companies should be investigated to validate the findings in this paper and maybe find more important factors influencing the relation of business models and ecosystems for software
companies. From more elaborated research, a framework could be derived in which companies could position themselves. This could serve as a decision support for companies willing to develop a business model in that market or rethink their existing one. Furthermore, it might be interesting to investigate the story of some successful and less successful companies to figure out the reasons for succeeding and possibly reveal some dependencies that are not known yet.

References


Appendix: Data sources

http://www.mozilla.org/about/  retrieved 01.06.2011
http://www.redhat.com/rhel/  retrieved 03.06.2011
http://www.mozilla.org/projects/  retrieved 02.06.2011
http://www.mozilla.com/about/ retrieved 01.06.2011
http://blog.mozilla.com/addons/2008/06/29/state-of-the-add-ons-ecosystem/ retrieved 03.06.2011
http://www.internetnews.com/dev-news/article.php/3590756 retrieved 05.06.2011
http://www.mozilla.org/about/ retrieved 01.06.2011
http://www.mozilla.org/projects/technologies.html retrieved 05.06.2011
https://www.redhat.com/wapps/store/catalog.html retrieved 03.06.2011
https://www.redhat.com/partners/ retrieved 03.06.2011
http://forums.mozillazine.org/index.php
http://www.mozilla.org/about/forums/
Requirements Management in Software Ecosystems

Peter van Stijn and Michiel Pors
Utrecht University,
Department of Information and Computing Sciences,
Princetonplein 5, 3584 CC Utrecht, Netherlands

Abstract. This work aims to study how on of the key process areas in software product management, requirements management, is performed in software ecosystems. To achieve this goal, a literature study is conducted to identify how the work of a software product manager is altered when working in a software ecosystem. This research will help product managers to identify problems and anticipate on structural changes in a software company.

Keywords: Software Ecosystems, Software Product Management, Requirements Management

1 Introduction

Software ecosystems originated when customers demanded more functionality than one software product company could deliver. The software vendors started negotiating with third party developers to cope with increasing customers' needs. Some ecosystems are around for some decades already, beginning in the 1980s with the expansion of operating systems’ capabilities with applications from other software companies. Today, all software vendors maintain relationships with other software companies. This network of companies can be gathered around a single platform, resulting in a software ecosystem. An illustrative example of a software ecosystem in present time is the mobile software ecosystem of iOS, comprising of a rich number of external developers making applications for iOS devices.

Although a software company can follow different approaches when transitioning from a more traditional software product line to a software ecosystem (SECO), the change basically affects the whole company. The transition has impact on the business strategy, architecture, development processes and, research and development organization [2]. However, the most dramatic change is in software product management [2]. Software product management is concerned with governing a software product over its whole life cycle. It is involved with four key process areas. These areas are portfolio management, product roadmapping, release planning and requirements management [4]. They are based on the deliverables and corresponding business functions of a software producing company.
Both for companies already operating as part of a software ecosystem, as companies who are transitioning from a software product line to a software ecosystem, it is of great importance to understand the software product management in these ecosystems. However, no direct research has been performed to address this combination of the two research areas and little best practices and case studies exist. This research aims to fill this gap in literature, by collecting and structuring the performed research on the border of both areas. Relevant implications related to the software product manager are identified and best practices to cope with SPM in SECOs are proposed. For scoping purposes, only the requirements management processes, one aspect of software product management, are addressed. The other business areas are left for future research. With this, the research question is defined as follows:

How is requirements management performed in software ecosystems?

In the next section, the relevant existing knowledge both on software ecosystems and software product management is provided. In section 3 the research method is explained and in section 4 the results of the literature study will be provided, elaborating on each of the process areas. Section 5 describes the conclusions of the research, of which the limitations are addressed in the final section.

2 Background

This section aims to provide a background on the concepts used in the research question, firstly summarizing research on software product management in general, then explaining the different types of ecosystems and players in an ecosystem.

2.1 Software Product Management

As stated in the previous section, software product management (SPM) is concerned with governing a software product over its whole life cycle [4]. This life cycle involves portfolio management, product roadmapping, release planning and requirements management. These elements originate from the structure that a portfolio consists of products, a product consists of releases and a release consists of requirements [6]. All four aspects from this hierarchical structure are incorporated in the software product management competence model, shown in Fig. 1 and described below.

This competence model is introduced by Bekkers et al. [6] and is an enhanced version of the original software product management reference framework introduced by Van der Weerd et al. [4] towards a more simplistic and intuitive model. As can be seen, more elements are involved in the competence model than only the four business functions. In the left side of the model, the three external (groups of) stakeholders are visualized, interacting on the software product management through all stages. On the right, the seven internal stakeholders are positioned. The stakeholders should be self-explanatory by the
names, but short descriptions of each can be found in Van der Weerd et al. [4]. Furthermore, the four business functions are divided into multiple focus areas. These divisions are described below, and are heavily based on the descriptions provided by Van der Weerd and Brinkkemper [8].

Requirements management, the continuing process of dealing with the content and administrative data of individual requirements, is divided into the gathering, identification and organization of requirements. A requirement can be defined as a *statement on an action that the product is requested to do, or a quality that the product is requested to have* [7]. Here the first focus area, requirements gathering, concerns from what sources and in which way these requirements can be obtained. When a requirement is gathered, it is still a market requirement, i.e. a customer wish related to current or future markets, defined using the terminology and context of the customer. These have to be analyzed and rewritten into product requirements, i.e. a requirement to be covered by future product releases described in the company’s own terminology and context. This process is captured in the requirements identification focus area, where one market requirement can result in one or more product requirements, and multiple market requirements could all resemble the same product requirement. During requirements organization, the product requirements are organized based on shared aspects.

Release planning, the process dealing with the set of requirements of each release in order to plan, manage and launch the release, is split on the different stages this planning process goes through. First, the requirements are prior-
itized, from which a release definition is written. This definition is validated and is checked against the ever changing requirements requests (scope change management). After this, the whole release definition is either revised, or the development is started. After the development (not shown in this management model), the build is validated and the launch is prepared.

Product planning concerns the creation of a product roadmap, and is split into three focus areas. The first focus area, roadmap intelligence, involves gathering all preliminary information needed for the roadmap. The second focus area, product roadmapping, involves the actual creation of the product roadmap. The final focus area, core asset roadmapping, concerns the creation of the roadmap for the components that are shared by multiple products, the core assets.

Portfolio management involves the decision making across the entire product portfolio, and the related information gathering. This is split into market analysis, product lifecycle management and partnering & contracting.

Obviously, the focus of software product management is not on the development of the actual software product, but merely all operational and strategic aspects performed before and after this development. Current research addresses this focus (as introduced by Van der Weerd [4]), where product management is considered a separate business function within the organizational structure of the company. This separate function is needed in order to avoid conflicts of interests between e.g. the sales and development department [8]. All software product management focus areas are performed and controlled internally, even though the development can be outsourced.

One of the results of this paper will be to what extent this approach is still applicable in large software ecosystems. This is essential to understand for companies which aim to expand their current business model from internally controlled product management (and development) towards an ecosystem approach (as described e.g. in Bosch [2]).

2.2 Software Ecosystems

The definition of a software ecosystem (SECO) provided by Jansen [1] is as follows: a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. This definition takes a slightly different approach than the definition of Bosch [2], which states that a software ecosystem consists of a set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions. Where the definition of Jansen is more from the point of view where actors are creating software, the definition of Bosch is more from the point of view where the software is connecting the actors. No clear choice for either of these definitions has been made in scientific literature, although the definition of Jansen [1] is used fairly more often.

Software ecosystems (as defined by Jansen [14]), can be viewed upon from three different scope levels. The most detailed level is the organizational one, in which actors and their relationships are studied. The SECO scope level models
the software supply networks and their different relationships, taking a higher-level view. At the highest level, the SECOs scope level, the SECOs themselves are the objects of study [16].

In a healthy SECO, two roles can be identified [14]. The first is the keystone player, providing the platform or technology. The second role is the niche player. This party creates business value by using the platform or technology. A third role, the dominator, has not proven to be successful in the long run [14]. In this dominator role, an organization eliminates other players in the ecosystem, and therefore needs to fulfill the customers’ needs by itself. Typically, the ecosystem gets destroyed in these cases or needs external regulation to survive.

Besides being able to view SECOs from a different scope level, there exist different software ecosystems, each with their own characteristics and (maybe) own differences in requirements management processes compared to traditional product lines. For example, an ecosystem can be a group of users creating stand-alone applications for a certain (operating) system (e.g. mobile apps), or it can be a group of users all working together on one large framework (e.g. a large open source project such as ruby).

One approach to categories the different types of ecosystems is provided by Jansen, Brinkkemper and Finkelstein, who defined four so called restriction boundaries [14]. These boundaries are the following: Market, Technology, Platform and Firm. Actors in a SECO centered around a market function as a unit in that they supply customers with similar products and the relations among them are competitive [14]. Examples are the Internet browser market and operating system market. Participants in a technology-oriented ecosystem are of any kind, though the keystone player is typically found to be the owner of the technology’s intellectual property [14]. The Ruby programming language is an example. According to the platform boundary, the SECO is focused around a specific platform or product. The functionality of a platform typically is extended with components by using an application protocol interface. The keystone player provides the platform. Examples are the Eclipse platform and Ruby on Rails framework. Finally, a SECO can be defined around a firm, in which the firm plays the role of the keystone supplier in several platform ecosystems [14]. The Apple SECO and Facebook SECO are examples.

Bosch has provided a different approach to categorize software ecosystems with his Software Ecosystems Taxonomy (SET) [2]. This SET is two-dimensional and maps the platform for which software is created (desktop, web or mobile) against the abstraction level at which the software ecosystem exists. These levels are the operating system level, the application level and the end-user programming level. Comparing to the boundaries provided by Jansen [14], these three levels all fit within the platform boundary. Software ecosystems falling in the operating system category are domain independent. The operating system provides development tools to third-party developers so they can build applications, which offer value to customers. Examples are Windows, Google App Engine and iOS. In contrast to operating systems, application centric software ecosystems are domain-specific. The software company of the application pro-
vides an application protocol interface so third party developers can extend the functionality of the program. Examples of these ecosystems are Microsoft Office and Facebook. The last category is end-user programming software ecosystems. These are communities of end-users creating and sharing their own application built with a software platform. These platforms in this case provide a configuration and composition environment with which this can be (intuitively) realized.

3 Research Method

This research can be labeled as exploratory research. It will structure the research area where software product management and software ecosystems meet. In addition, this is a secondary research, i.e. a summary of existing research. The literature study will be conducted in order to create a thorough understanding on software ecosystems, software product management and the boundaries of these concepts.

The literature research consists of exploring existing research performed on each of the key process areas and how these areas relate to software ecosystems. Typical questions that need answers are what implications arise when a company transitions to an ecosystem, if the areas are still relevant when transitioning to an ecosystem and on what areas more research must be done.

When searching for relevant literature the search engine Google Scholar was used. The queries used, were all combinations possible taking zero or one of each of the two sets of key phrases:

1. software ecosystem
2. software product line
3. open source project
4. open source software
5. stakeholder collaboration

1. software product management
2. requirements
3. requirements management
4. requirements engineering
5. requirements gathering
6. requirements identification
7. requirements organization
8. requirements prioritization

All of the combinations resulted in at least five pages of results (>50 results). Only the first five pages have been searched. For each of these search results, a certain amount of papers was pre-selected, based on the title, looking for a referral to a community, ecosystem or platform and a main subject of requirements management. This resulted in a set of almost 100 papers. Then, abstracts and conclusions have been read to further thin the selection, resulting in around thirty papers. Relevant parts of relevant papers have been copy-pasted and stored
in a text document (using Google Docs) containing our literature findings. Once
a relevant part has found a place in our final paper, it was removed from the
database to keep an overview.

4 Results & Analysis

This section describes for the requirements management business area of the
software product management competence model, introduced in Van der Weerd
et al. [4] and shown in Fig. 1, our literature findings on the interface of this model
with software ecosystems. It will describe what research has been performed
on each of these business areas, the relevance of the area within a software
ecosystem, best practices, and what aspect of requirements management is very
important within ecosystems, but only to a lesser extent in traditional software
development.

The results described in this section relate to SECOs focused around a spe-
cific platform and SECOs based on a specific technology, i.e. the platform and
technology restriction boundaries [14]. Within the platform boundary, the oper-
atting system, application and end-user programming level [2] are all described.
The other boundaries, i.e. market and firm, are out of scope because too much
different types of ecosystems and software producing organizations fit in these
boundaries.

The section is structured as follows: first, the essential role within the re-
quirements management processes of the platform provider are described. Then,
the implications for the requirements management processes of working in a
software ecosystem are described. This is performed separately for each of the
focus areas (requirements gathering, identification and organization) and some
words are dedicated to the process of communicating the requirements and the
importance of this focus area within software ecosystems.

4.1 The role of the Platform Provider

Bosch and Bosch-Sijtsema define SECOs as a software platform, together with
a set of internal and external developers and a community of domain experts
in service to a community of users that compose relevant solution elements to
satisfy their needs [3]. The internal developers build products on top of the
software platform and the external developers extend these products with new
functionality. According to the authors, the keystone firm should take a commu-
nity perspective, including all the actors participating in the ecosystem, i.e. the
external developers, domain experts and customers. It requires the organization
to collaborate and coordinate in a community-centric way [3].

The authors further state that this keystone firm, providing the platform,
need to broaden their coordination scope when transitioned to a software ecosys-
tem. By broadening this scope, especially the software product management is
affected [2]. When coordinating the development and future plans of the software
platform, the external developers and development teams need to be involved in
this process as well, instead of only involving the team within the company. This concerns all the phases in the life cycle. According to the authors, “during the road mapping and planning process, external developers often have strong views on the priorities and sequencing of platform functionality. During development, the architecture, specifically the interfaces between the platform and the products, should be developed in collaboration with the external developer community, not just published. Finally, during the validation of the new release of the platform, the external developer community needs to be involved in order to minimize the unintended breaks in functionality when rolling out the next platform release.” [3]. A real world example of this last phase is the seeding of beta-versions of a new major release of iOS from Apple to developers. The developers then can modify their existing applications so they function in the new release.

Bosch describes the implications that arise when a company is transitioning to a software ecosystem. He categorizes implications as a result of this change in three main areas, i.e. coordination mechanisms, engineering agility and product composition [2]. The first area, coordination mechanisms, implies that the processes used, such as requirements management and roadmapping initiatives, are not designed for a software ecosystem approach. The other two areas are not relevant for requirements management. According to Bosch a solution in the coordination mechanism area, is to move to a decentralized approach. He states that in the case of software ecosystems, it’s necessary to decentralize activities like requirements management, architecture evolution, integration, quality assurance and SCM and to move to a compositional approach to software development. One of the changes this would result in is the replacement of centralized requirements management and roadmapping with bottom-up, team driven roadmaps and requirement specifications [2]. Bosch describes this situation already exists between the platform company and its external developers in most companies, but some companies apply the same approach internally as well. This means that each component team announces its own roadmap and the requirements that it will release at the end of the next iteration cycle.

In [14] the concept of the Extended Software Enterprise (ESE) is introduced. It is based on the role of the Extended Enterprise played by an organization in a business ecosystem in the business ecosystem model of Moore [18]. An ESE is defined as a software enterprise that has abolished selected barriers surrounding its intellectual property to create value by sharing it with the surrounding SECO [14]. Every participant in the SECO can be an ESE, niche players as well. However, generally keystone players are identified as an ESE, because they play a much more important role in the SECO than the niche players. In that paper the authors indicate what an ESE can do to share knowledge regarding product management. They mention sharing road maps, coordinate release times and opening up the requirements engineering process, so that customers can vote on important features [14].

Van den Berk et al. use the term platform strategy as opposed to product strategy [1]. They state that the theory of product strategy can also be applied to the software ecosystem domain and that platform planning in a software
ecosystem is similar to software product planning for an independent software vendor. Kittlaus and Clough state that the main elements of software product planning are roadmap definition, release planning and requirements management and specification [9]. These tasks will therefore need to be performed in platform planning too. The difference for a software product manager in an ecosystem compared to an independent software vendor is that these processes need to be carried out while dealing with the niche players in the software ecosystem.

Van den Berk et al. also present a software ecosystem strategy assessment model [1]. This model describes the key characteristic of an ecosystem and it can help the central hub in strategic decision making. One example that is assessed in the model is the platform planning characteristic. It consists of the four processes of software development: portfolio management, roadmap definition, requirements management and release planning. According to the authors, these processes are executed in a different manner in software ecosystems. However, they state that it is unclear how exactly and that good SECO platform planning methods do not yet exist.

4.2 Requirements Management

Open sources software projects are a subcategory of software ecosystems, usually within the platform and technology boundaries. The fundamental need to discover and understand the process of requirements development in such projects is described in [10]. According to Riehle [17] there are two types of open source software: community and commercial open source software. The former is developed by a broad community of volunteers, and determine which contributions are accepted and where the software is headed. Decisions about the software are made by individual developers, an example is the Apache Web server ecosystem. Commercial open source software is defined as software owned and developed by a for-profit entity [17]. The company maintains the copyright and makes decisions about roadmap issues and release planning. An example is the MySQL ecosystem.

In [11], Scacchi has provided a description of the differences of requirements engineering in open source software systems (a type of software ecosystem) compared to traditional requirements engineering (RE). Although the traditional requirements engineering described there differs slightly from the one we have chosen to follow in this research, a lot of overlap exists. One of Scacchi’s findings, is that the requirements management function in open source projects does not have the status of an assigned and recognized task, but is mainly performed informally as a “socio-technical process that entails the development of constructive social relationships, informally negotiated social agreements, and a commitment to participate through sustained contribution of software discourse and shared representations” [11]. The requirement engineering activities are not structured by processes but rather emerge routinely as a by-product when the stakeholders discuss functional features, quality aspects and constraints of the software. The findings of Scacchi relate to the community open source software ecosystems.
The role of requirements management in software ecosystems is described in more detailed in the following three sub-sections, each covering one focus area of the software product management competence model (requirements gathering, identification and organization). A fourth sub-section is devoted to elements of requirements engineering not covered by the competence model, but an element that has been described by many authors (e.g. in [11] as an important aspect within ecosystems: requirements communication.

**Requirements Gathering** Scacchi observed that in open source software system requirements often simply appear to exist, without really the possibility to trace these to a point of origination [11] as what would be the case in traditional requirements engineering, where requirements are elicited from stakeholders, customers or end-users. The study further could not find evidence for a requirements elicitation effort arising in such open software development projects. The author also mentions how the gathering of requirements for these kind of projects would have to be broadened.

The same author follows in a study two years later with the comparable observation that requirements are established as threaded messages or discussion, not gathered in the traditional manner but posted on web sites by certain users for open review, after which it is elaborated on and refined before being used by other users to be captured as requirements [12].

According to Noll, requirements elicitation in open source software development differs from conventional software development projects [19]. Developers assert the requirements, in contrast to customers' needs that are elicited. Henderson describes how typical software requirements elicitation techniques work in developing software for the open-source community [20]. The seven techniques described are Introspection, Questionnaire Interviews, Open-ended interviews, focus and application development groups, discussion, protocol analysis, and discourse analysis and are listed by Goguen and Line [21]. Henderson states that for typical open-source projects requirements elicitation is not needed, because in general the developer is the customer. But in an open-source community not all members are programmers, the community consists of users with no programming skills as well.

For these types of communities, Henderson describes how requirements elicitation techniques can be used. The first technique treated is introspection. Henderson says introspection is needed, no matter what type of project. Other techniques need to be used as well however, because a discrepancy can exists in the meaning of the customer's requirements and the understanding of the developers.

About questionnaire interviews and open-ended interviews, Henderson says that the two techniques fit well in the open-source community, because they both can be used on the Internet and are easily implemented. However, they still have their well-known drawbacks. According to Henderson, the discussion technique is widely used in the open source community. However it's more used as a general communication technique, rather than specifically eliciting requirements. The author says about the focus and application development groups technique that
it is better used as a bulletin board, because of time problems: parties might be geographically dispersed. Finally, in discourse analysis taking turns is important. However, the Internet is an environment in which a person doesn’t necessarily needs to wait for someone else’s respond. Therefore, discourse analysis is not really suitable to be used over the Internet.

Feller and Fitzgerald state that the customers are a “critical feature” of open source projects [22]. Users should be seen as the source of new requirements. According to Scacchi, requirements elicitation in open source software development is not processed using traditional methods and that the requirements emerge from open forums. However, he states that this process is in fact effective [11, 12].

Requirements Identification In traditional requirements management, the identification of requirements involves mainly the deduction of product requirements from market requirements [6]. Here, market requirements are feature requests as provided by customers. Obviously, it can occur that multiple feature requests exist for one actual feature, or that one feature request incorporates multiple features. Product requirements are in this sense the lowest level features as they will finally be implemented in the system, and are translated towards the company’s standards.

According to Scacchi, little (if any) of such analysis of software requirements specifications exist [11]. He observes that the participants in these communities, however, are able to understand the functional requirements, quality aspects and constraints to an extent that is sufficient for the successful implementation of these. Henderson states that, in order to gain this clear understanding, usually multiple forms of requirements gathering have to be performed [20].

Most developers create solutions for what would be market requirements in the traditional requirements engineering and have little priority on formally translating the market requirements into product requirements first. This results in the observation that players (programmers) in a software ecosystem do not see this analysis of requirements as an assigned or recognized task [12]. However, the author did observe wide-spread practices such as “reading and sense-making of online content, interlinked webs of discourse that effectively traces, condenses and hardens into retrospective software requirements” [12]. This implies that although the identification process is not formally performed, product requirements are informally created as a combination of multiple similar market requirements. The requirements identification focus area is thus, in software ecosystems, merely the analyzing and cross-validation of feature wishes, identifying the real requirements.

Requirements Organization The focus area of requirements organization is by Bekkers described as the process which organizes the requirements throughout their entire life cycle based on shared aspects, which also describes the different relations between the product requirements [6]. Little on this focus area related to software ecosystems can be found back in literature. This is probably due
to the observations by Scacchi, described in the previous sub-section, as lack of requirements identification makes that little extra information on the requirements exists except how they were gathered as market requirements, and thus there would be no additional organizational possibilities for these requirements [12] - there is no link between market and product requirements.

Requirements communication There is generally a collaboration of many stakeholders needed for bringing new products and systems in an ecosystem to a success [13]. Fricker introduces requirements communication networks to visualize and facilitate these communications [13]. Scacchi also emphasizes on the importance of communicating requirements [11]. Fricker provides information on how his requirements communication networks can support process development for improving the communications of requirements in ecosystems [13]. When the requirements are communicated well within the software ecosystem, these become community knowledge - replacing the formal requirements organization focus area.

5 Conclusion

Just as requirements management plays a vital role on the more traditional software development process, it does for companies operating in a software ecosystem. Many similarities between the more traditional approach on gathering, identification and organization of requirements and the one used in software ecosystems have been identified and some interesting differences have been discovered. Our research question was formulated as follows: "How is requirements management performed in software ecosystems?". Figure 2 pictures the answer to this question in a way consistent with the software product management competence model introduced by Bekkers et al. [6]. Below, a summary on how this answer has been developed is provided.

Fig. 2. Requirements Management within Software Ecosystems

In a software ecosystem, there is no formal entity that handles the elicitation of requirements. For requirements gathering this implies that, besides the fact that the requirements are gathered differently, they are not gathered by one single person but merely by a community. This holds especially for open source software development. Techniques mainly used for requirements gathering in software ecosystems are introspection, interviews, and development and
discussion groups. However, it is often left unclear which of these community inputs actually are the requirements.

The requirements identification part, rewriting the market requirements into product requirements, is not performed formally. Instead, the requirements identification is merely a validation process of the inputs provided by the community, to determine which actually are the requirements.

Organization of requirements is hardly performed at all, as no formal links between market and product requirements exist. Instead, this organization is merely a process of communicating the requirements to all stakeholders, creating community knowledge. Thus, for companies acting as a platform provider in a software ecosystem, collaboration with external stakeholders is crucial. Even more in ecosystems than in traditional product management, a company needs to collaborate with their customers and external partners. The firm can open up the requirements engineering process, by enabling customers’ and external developers’ involvement. This involvement means coordinating the evolution of the software platform with the external parties.

6 Discussion and Future Work

The results of this literature study are solely based on existing literature; the validation of the resulting competence model fragment is left as future research. Although triangulation has been used to create the resulting artifact (Figure 2), by combining as many literature sources as possible, this has not been performed with other sources than that, such as case studies. This also implies no real empirical research has been performed (which was anticipated as this research was designed solely to be a literature review), and validation threats such as internal validity, construct validity and empirical reliability [15] are not applicable.

However, there still remains a threat to external validity - the generalizability of the results. Many of the results are based on sources describing requirements management within open source software development communities. These communities, as described, only form one category of software ecosystems - where even this categorization is not validated to be complete. This means it cannot be stated with full certainty whether the found requirements management process refers to all actual software ecosystems. In order to have this certainty, boundaries of software ecosystems must be able to be determined and a complete categorization must exist, after which an empirical research should be conducted (besides this literature review). Up till that point, the results can only be generalized to requirements management in all open source communities. One point in favor is that these communities form a large part of all software ecosystems.

Future research would also have to be performed to determine how the other three software product management business areas (release planning, product planning and portfolio management) are performed within software ecosystems. Furthermore, in order to use the results of this literature study and create a complete competence model for SPM in SECOs, an empirical validation has to be performed on the resulting competence model fragment provided in figure
2, as explained in the earlier paragraph. This could be done by interviewing domain experts: e.g., product managers of companies who recently shifted their product development towards the ecosystem, or active members of open source communities. Furthermore, performing multiple case studies would add a great amount of knowledge to this still quiet immature field. One must keep in mind, that when extending this research with an empirical research, more validation threats arise. Jansen and Brinkkemper [15] described, besides internal validity, construct validity and empirical reliability as such threats. Especially construct validity has to be carefully approached, taking into account the different classifications of software ecosystems and how the results refer to which of these categories of ecosystems.

References

A Small Sample Survey on Software Component Supplier Selection for Dutch Software Ecosystems

Joey van Angeren and Vincent Blijleven

Department of Information and Computer Sciences, Utrecht University
Princetonplein 5, 3508 TB Utrecht, the Netherlands
{j.vanangeren,vblijle}@cs.uu.nl

Abstract. A software product is created out of multiple components, not all of those components are developed in-house, as a consequence software vendors depend on suppliers. Suppliers deliver software, hardware, services or intellectual property. In this paper we address the relationships with these suppliers and some of the selection criteria for these suppliers in the form of an exploratory research. The results presented in this paper can be used by software vendors when assessing their own software supply networks, but also form a basis for future research. We analyze results from a small sample survey conducted at Dutch product software companies. To relate supplier selection and relationships to the importance of a certain component for the overall product, we propose a matrix, also used for the analysis, that distinguishes products into core and context components and further distinguishes them into critical and non-critical components. Results show that software vendors opt for differing supplier selections, from a minimal dependency strategy on suppliers to depending on large software ecosystem orchestrators. Also, the importance of a certain product component has an influence on the average grade of intimacy with a supplier and the criteria that are at play in the supplier selection process.

Key words: software ecosystem, supplier relationship, supplier selection, software supply network, product deployment context

1 Introduction

The Dutch product software industry is flourishing and is playing an important role in the Dutch economy. Various examples of successful products are computer games, navigation systems, administrative software and enterprise resource planning products. For the purpose of this paper, we define product software as: “a packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market” [1].

As noted from the definition, software products are a configuration of numerous components. Often, a software vendor will not develop all of these components in-house, rather there will be a number of other organizations that supply
them with hardware and software components, services and intellectual prop-
erty vital for the products they offer. Because of this, software vendors become
dependent on service providers and other software vendors in order to leverage
their products to the customer. We refer to the network of actors that is the
result of this phenomenon as a software ecosystem. In this paper we define a
software ecosystem as; “a set of actors functioning as a unit and interacting
with a shared market for software and services, together with the relationships
among them” [2].

Software ecosystems can be studied from different scope levels. These differ-
et scopes involve different levels of abstraction and view software ecosystems
and their interactions from different perspectives [3]. In the most detailed scope
level, one specific software ecosystem is studied including the actors that are
part of this ecosystem and the relationships amongst these actors. On the total
opposite, there is a scope level that addresses the organizational perspective and
the relationships between different organizations and ecosystems. In this paper,
we will use the second scope level, in the form of Software Supply Networks,
or in short SSNs. Within this scope level the network of actors of hardware,
software and service organizations is studied that cooperate to satisfy market
demands [4]. This SSN can be used to give an insight into first-tier buyer-supplier
relationships for one specific software vendor of interest, including the resulting
flows and dependencies. Also, creating a Product Deployment Context (PDC)
for this software vendor that describes the structure of software products in its
running environment in a stack view, showing how the product is composed out
of different components [5] can provide further insights. Combining the insight
into how important a certain component is for the portfolio of a certain product,
including the grade of dependency on a certain supplier can be used to identify
weaknesses. This can be useful to gain insight into the factors that are at play
when selecting a certain supplier. Furthermore, this is valuable information to
be aware of for a software vendor, to make the right decisions on a both strategic
business as well as a software architectural level.

In this paper we address supplier relationships from a portfolio perspective.
We will propose a matrix to classify components and services and their perceived
grade of importance for the final delivered product. This matrix will be utilized
to perform a pattern analysis on the SSNs, PDCs and tables that describe the
grade of intimacy of relationships with suppliers of Dutch product software com-
panies. These data have been gathered through a small sample survey. We will
examine different supplier relationships, strategies and dependencies, based on
the importance of the delivered component or service for the product leveraged
by the Dutch software vendor. Furthermore, we will address the choices for sup-
plier strategies from a software ecosystems perspective by addressing reasons to
join an ecosystem as a customer to obtain software components or services, or
even as a partner. Choosing for a certain supplier might result in a high level of
dependence or lock-ins, having great implications for, for example, the business
model of a company. Also, changing strategies or decisions about components
or services by one of the suppliers can cause big implications and problems for
the software vendor. In the contrary, intimate relationships with suppliers can result in advantages, such as, shortened support and maintenance lines or secondary benefits. As a result, a software product company gets confronted with trade-offs. Because of the potential impact such dependencies and trade-offs can have, more research needs to be carried out within this domain.

Because of the exploratory nature of this research, we chose for a structure in most ways similar to the one used in [6]. Results will therefore be presented early on. The remainder of this paper continues with a description of the research approach in section two, in which we will elaborate on the research methods we employed as well as the survey participants selection process. In section three, we will present a matrix to classify components of a software product or its direct running environment. In the fourth section, we present the results of the survey concerning organizational size, business models and software ecosystems. An analysis and interpretation of findings from the survey results, will be presented in section five, including several benchmarks with similar studies that have been conducted in the past. In section six, we discuss encountered validity threats and make statements about generalization possibilities of the presented results. In the last section we draw the most important conclusions and provide suggestions for future research.

2 Research Method

In this paper we use empirical data gathered from the Dutch product software industry, gathered between September and November 2010. The used data collection technique is best described as a small sample survey. The total dataset consists of twenty-seven contributions.

2.1 Data Collection

The data collecting process took place during the Product Software course at Utrecht University, which is part of the bachelor in Information Science curriculum. Twenty-seven couples of bachelor students selected a small to medium-sized Dutch product software company, where they wanted to conduct their research in order to get familiar with the Dutch product software industry. The main prerequisite for a software vendor to qualify for participation in this survey was that their number of employees had to be at least ten. Furthermore, they had to be registered at the Dutch Chamber of Commerce. During two or three meetings, the teams of students gathered information about three key themes; organizational structure, business models and software ecosystems. Each assignment addressed one key theme, and each assignment was graded separately. Because of the nature of the data collecting process, the survey consisted mainly of open questions. The open nature of this survey also provides for a broader insight into drivers and motivations software vendors have for strategic decisions. All couples conducting the research used a similar approach when conducting the surveys.
The first part of the data collecting process was edged on gathering information about the participating software vendors. Information was gathered about the products they offer and the way in which the company is organized. This structure has been captured in an organizational structure, including details on the number of employees. The middle part of this survey was edged on business and revenue models. Data was captured by filling in Osterwalders’ business model canvas [7]. In this business model canvas, many components of the business model are addressed, for example, key partners, activities and revenue models. Furthermore, accompanied by a representative of the company, they filled in a SWOT Matrix [8] to identify the main strengths, weaknesses, opportunities and threats for the respective company.

Software ecosystems were addressed in the last and most comprehensive part of the survey. For each of the companies, one product of interest, that the software vendor develops and delivers has been selected. For this product, the Software Supply Network and Product Deployment Context has been described and captured, according to the modelling techniques described in [5] and [9]. An example of this, from the dataset, is depicted in figure 1. The PDC describes a client-server product delivered by one of the product software companies that took part in this survey. The labels in the SSN, depicted at the right side of the figure match with those included in the PDC. The SSN also includes other suppliers of components, services and content. This way, the SSN gives an overview of first-tier buyer-supplier relationships and the exchange of products, services, data and money flows between these actors. Furthermore, in this part of the survey the perceived level of intimacy for each of the actors within the SSN has been collected in a table.
2.2 Selection Criteria

Because of the nature of the data collection process and to enhance the quality and integrity of the dataset, we formulated a number of inclusion criteria for each of the contributions. The contributions that were to be included into the dataset we used had to apply to the following criteria: (1) All three assignments have to be handed in and need to be accessible; (2) The average grade for the entire contribution has to be at least 7.5 on a scale of 1-10 where 1 is the lowest possible grade and 10 is the highest possible grade; (3) The grade for each assignment has to be at least 7 or higher; (4) Each assignment has to be entirely executed and complete; (5) Each company can be included only once into the dataset, and in case of a duplicate the one with the highest average grade will be included.

After applying the inclusion criteria to the initial dataset, in total seventeen out of twenty-seven contributions were included into the final dataset, 63% of the initial dataset. This dataset will be subject to the analyses that form the basis of the findings presented in this paper.

2.3 Data Analysis

The main research question of this paper is as follows: “How does the perceived level of importance of a component, that is part of a software product, influence supplier selection”? To be able to answer this research question and to benchmark with previous findings, we perform both qualitative and quantitative analyses on the empirical data gathered from the Dutch product software industry. Furthermore, notions from existing literature and about the Product Deployment Context will lead to the creation of a matrix that classifies product components.

A brief quantitative analysis is performed and employed to contextualize the dataset. We therefore elaborate on various organizational characteristics, such as, organizational size and business or delivery models. Using these findings as a prerequisite, a qualitative analysis is performed to provide an answer to the research question. The SSN and PDC, together with the table that describes the perceived level of intimacy with each of the actors within these networks, will be subject to a pattern analysis. Through this analysis, we identify multiple supplier strategies and their resulting trade-offs. The matrix that will be presented in section 3 is then utilized for further analysis. The analysis is employed to identify factors that influence supplier selection and to determine to what extent these factors are perceived as important when selecting certain types of components.

3 A Matrix for the Classification of Components within a Product Deployment Context

Product software is constructed out of multiple components. Most relevant components are either hardware or software components. Apart from that, additional
services and the inclusion of added value by the software vendor will result in a final software product. As already elaborated on by Jansen, Brinkkemper & Finkelstein [10], some of these components obtained from suppliers are more easily replaced than others. An interface grid, for example, can be replaced quite easily within most software products for a product with similar functionality from another supplier without heavily affecting the end product. In the contrary, totally migrating a software product to be compatible with another operating system will be a challenging and time consuming process. Based on this notion, we developed a matrix that classifies components that are at play within the running environment of a software product, as displayed in Product Deployment Contexts.

![Fig. 2. A Matrix for the Classification of Components within a Product Deployment Context](image)

The matrix in figure 2 distinguishes four types of components from an architectural perspective, that are part of the software product or its direct running environment. First, we distinguish between core and context components. When speaking of core components, we refer to the fundamental building blocks of a software product that are vital to allow the software product to be run and provide no value-added functionalities to the customer. Core software components are regarded as the heart of a software product. Context software components then, are software components that add specific values to the product. For example, certain functionalities that make the product unique to a customer or that add additional functionality to the product. Unlike core software components, context components are not a necessity for the software product to be run.

When examining core and context components, we can make a distinction between critical and non-critical core and context components. Components are
critical if they cannot be easily interchanged with another component and contribute significantly to the added value of the software product and its resulting overall functionality. A simple PDF plug-in, for example, provides an additional functionality for the product and is therefore a context component, but since it is easily interchangeable by a substitute with equal functionality from a different supplier it is considered as non-critical.

The proposed matrix complies with the PDC modelling approach defined by Boucharas, Jansen & Brinkkemper [5]. Within the stack view of a product and its direct running environment, a distinction can be made between optional and required components. Typical examples of optional components, again are plug-ins or components that are easily interchangeable with another component. Furthermore, the place a product has within the stack view depicted in a PDC, typically products that are lower on a stack are core components (e.g. operating systems, frameworks, databases), as well as the description that always accompanies a PDC provide for a last distinction within the four identified categories.

Because the PDC provides insight into product components and the running environment in a stack view that is based on a certain level of abstraction, this matrix is created out of a simplified architectural perspective. Since factors like the degree of coupling and the lines of code necessary to incorporate a certain component into the final product cannot be measured by assessing the PDC, they are not considered as factors influencing the classification of components. Since we are dealing with PDCs within our dataset, this simplification step will not have an influence on the results.

4 Results

To conceptualize the dataset, we divided the participants in three distinct company size categories. Companies having ten up to and including twenty-five employees are categorized as small. In addition, companies with an employee amount of twenty-six up to and including one hundred have been categorized as medium-sized companies. The last category is for companies categorized as large, who have more than one hundred employees. This categorization of participants based on company size differs from the one proposed by the Dutch Chamber of Commerce, because that categorization does not provide enough detail for this research. An overview of the result of this company categorization is displayed in figure 3.

Furthermore, the software delivery model for each company within the dataset is relevant. 55% of the companies sell their software products as traditional (on-premises) software, where the product is both installed and running on computers of the person or organization using the software. In addition, 41% of these companies also offer a software as a service (SaaS) solution as an alternative next to their traditional solution. Additional revenue streams for on-premises software will often come from subscriptions to maintenance and support. Furthermore, three participating companies are dedicated SaaS solution providers.
The remaining two companies solely offer a platform as a service (PaaS) solution. An overview of solution types offered in terms of percentage can be seen in figure 3.

The core of the dataset is formulated around Software Supply Networks, Product Deployment Contexts and a table that describes the perceived level of intimacy with each supplier that is part of an SSN out of the software vendors’ perspective. Furthermore, a textual description accompanies these artefacts to provide insight into the motivation for the decisions made by the product software company. Out of this data, we distinguish four main supplier strategies; product integration with a hardware component supplier, depending on a large software ecosystem orchestrator, inclusion of open source components and a minimal supplier dependence strategy. When a software vendor has to decide to opt for one or more of these strategies they will have to face and cope with effects from possible trade-offs. Choosing to be less dependent, for example, can result in having to dedicate more resources to development or having less benefits. An overview of the strategies and its resulting trade-offs are collected in table 1.

Several companies choose to integrate their products with a hardware component supplier. Companies need to make a trade-off between having a streamlined integration process by working with a hardware supplier with whom they have built up an intimate relationship and thus becoming dependent on this vendor, or having a less streamlined product integration process without being dependent on the hardware supplier, since in a lot of cases it is easily replaceable by another one.

Some companies choose to become fully dependent on large software ecosystem orchestrators. In a lot of cases this is regarded as an opportunity. Various reasons for this choice have been given, such as; “the company has a strong market position”, “the company has a big market share” or “continuation is guaranteed since this company will still exist in ten years from now”. One of the main reasons for this strategy can be to benefit from niche creation within the ecosystem. In case of heavily depending on a large software ecosystem as a supplier, it becomes common to join its partnership model. This model will bring additional benefits for both parties, can provide for additional education of employees of the participant and can shorten direct support and maintenance lines to the supplier. Another trade-off needs to be made here, whether to rely
### Table 1. Supplier selection strategies and their resulting trade-offs

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Trade-off</th>
</tr>
</thead>
</table>
| Product integration with a hardware component supplier | Y (+) Streamlined integration process by working with an “intimate hardware supplier”  
N (-) Become dependent on a supplier |
| Depending on large software ecosystem orchestrator | Y (+) Benefit from participating in partnership model  
(+) Benefit from niche creation  
(+) Direct contact & support lines with the supplier  
(-) Become dependent on a large software ecosystem orchestrator  
N (-) No benefits from niche creation  
(-) Less partnership model possibilities  
(-) Indirect contact & support lines with the supplier |
| Inclusion of open source components | Y (+) Ability to steer an open source software project into a favourable direction  
(+) Less license fees  
(+) More support and maintenance responsibilities  
N (-) Avoid liability issues  
(-) Less support and maintenance responsibilities  
(-) Few strategic influence on the development of components |
| Minimal dependency on suppliers | Y (+) Develop components in-house to decrease direct supplier dependencies  
(-) More resources required  
N (-) No additional resources required  
(-) Remain dependent on suppliers |

Software vendors indicate to be reviewing their current supplier relationships because of the advent of open source software components as alternatives for proprietary components. With the benefits from open source come new supplier selection challenges, like carefully examining licenses of open source software to avoid liability issues. Being able to leverage continuous support and maintenance to customers is an additional drawback of open source, since more of these responsibilities will end up with the software vendor rather than its suppliers, in this case open source communities.
The larger companies within the dataset strive for less supplier dependencies. Vital components for the leveraged product are developed in-house, if resources are available. A trade-off is made between the advantage of not needing any additional resources because of being dependent on a supplier, or developing components in-house and thus decreasing direct supplier dependency. This in the contrary to smaller companies that rely on basic vital components from suppliers. They create some additional value on top of this, in an attempt to serve a niche. Apart from the mentioned suppliers, some software vendors are dealing with service providers and intellectual property providers. An example of this, is that most SaaS vendors indicate to maintain intimate relationships with their hosting providers. They cannot permit themselves to have downtime or security breeches, so short support and maintenance lines are needed.

5 Analysis

In the results we distinguished four types of supplier strategies and their trade-offs, but did not yet discuss the factors that are at play when choosing for strategies for certain components. The component classification matrix serves as a basis for the analysis of the survey results. The goal of this analysis is to capture the influence of the type of a software component, as defined in the matrix, on supplier selection criteria and supplier relationships.

With regard to the perceived level of intimacy, a relationship with a component supplier can be classified as either intimate, familiar or unfamiliar. In this sense a familiar relationship can be, for example, a partnership with a supplier but also just having regular and direct contact with the supplier. In the contrary, an unfamiliar relationship with a supplier is applicable for normal buyer-supplier relationships in which the software vendor just obtains a product or service. As noted in section 4, software vendors indicate to select suppliers based on their capability to supply their components without disruptions, their financial condition and their company size or reputation. Furthermore, they strive for participation in partnership models of large software ecosystems. They are keen on intimate relationships with these organizations, especially when it concerns a critical core component or a critical context component supplier. Examples of this are, for example, relationships with a platform host that forms the basis or fundament of the software vendor’s product.

From a software ecosystems’ perspective, some of the indicated motives for supplier selection show parallels with ecosystem health indicators. Ecosystem health was first defined by Iansiti & Levien [11, 12] as an overall performance indicator of an ecosystem. This is further defined by three determinants; robustness, productivity and niche creation. For supplier selection, especially robustness is relevant, since it indicates the capability of an ecosystem to face and survive disruptions. Software vendors indicate that for them, continuity of a software ecosystem is vital when selecting a supplier for the most important components of a product. Software vendors therefore opt for stability, by selecting
software ecosystems to join, as a customer or partner, based on these simple indicators. Related to this, are some of the indicators to measure ecosystem health, as defined in [13]. The visibility of a software ecosystem within the market is indicated as another important reason to choose for a certain supplier, especially for large ecosystems that provide core components. Furthermore, niche creation is an important reason to join a software ecosystem. This is especially the case when selecting a platform that forms the basis for the product.

Including open source components as alternatives for proprietary source becomes more interesting for software vendors. Examining open source licenses therefore, becomes more prominent in the supplier selection process. This examination is performed regardless of the type of component, since inclusion of any component with a certain open source license, can have serious consequences for the rights of use or redistribution for the total software product. Ruffin & Ebert [14] discuss several major legal aspects and three major risks with regard to the inclusion of open source software and how to mitigate them during product development. The first risk concerns directly integrating open source software directly into the source code, and then reproducing or selling the product without permission of the licensor. As a consequence, the licensor might claim damages or force the product vendor to terminate production, delivery and sale. Secondl, open source software is often a compilation of code from many sources. Because of these many sources, it is not an easy task to identify which parts, if at all, relate to protected intellectual property rights. Infringement of third parties patents or other intellectual-property rights might be the consequence of being unaware of including open source software that is protected by intellectual property rights. Third, when an open source tool is used to, for instance, generate output that contains tool-created comments and a specific structure, the resulting work might be considered a derivative work. In this case, the owner of the tool and thus the copyright holder will be given certain rights to the resulting product. In all three discussed risks, an examination by an expert (e.g. judicial expert) often becomes emergent.

An interesting notion is that several survey participants primarily belonging to the large company size category, mentioned that for certain components they prefer open source over closed source. The primary reason they gave was that using open source components leads to less supplier dependence which makes it an attractive alternative. They did not, however, mention anything about possible copyright violations due to the inclusion of open source software protected by intellectual property rights, making them vulnerable for the associated major risks as described in the beginning of this section. It is questionable whether the survey participants are aware of the major risks concerning the inclusion of open source software within their own commercial products. Important to note is that an increasing awareness of licenses is urgent for all categories of software components. However, this is most prominent for context component, since most core components are often well-known.

Support and maintenance flows play a decisive role when selecting both critical core and context component suppliers. Especially when a component can
be classified as critical and is thus of importance to the software vendor, a well organized support and maintenance flow is essential. Therefore, continuity of maintenance and support and direct lines with supplier are identified as important triggers for supplier selections. These maintenance and support interactions make supply management in the (product) software industry different from this practice in other industries [10].

<table>
<thead>
<tr>
<th>Category</th>
<th>Factor</th>
<th>Critical component</th>
<th>Non critical component</th>
<th>Critical context component</th>
<th>Non critical context component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier related factors</td>
<td>Perceived level of intimacy</td>
<td>Intimate</td>
<td>Familiar</td>
<td>Intimate</td>
<td>Unfamiliar</td>
</tr>
<tr>
<td></td>
<td>Continuity</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Visibility within the market</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>Niche creation</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Product related factors</td>
<td>Product &amp; license type</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Support &amp; maintenance</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 2. Classification of factors influencing supplier relationships and selection per product component type

Table 2 is the result of classifying what is described in this section. By taking the selection criteria as described into account, we created a classification table to identify which factors are specifically important when selecting a supplier for a certain type of component and what the average grade of intimacy is for this supplier relationship. It is important to note, that selection criteria related to functionalities or perceived added values of a certain component and costs are not included in this table, since they will be applicable for all categories and are therefore trivial.

6 Discussion

In this paper, we addressed supplier selection out of a software ecosystems and portfolio perspective. We also addressed component classification, supplier strategies and the way in which the perceived level of importance of a component influences factors that are at play when selecting suppliers and strategies. The data used for this analysis, have been gathered through a small sample survey carried out by bachelor students, during the Product Software course that is part of the bachelor in Information Science curriculum. Because of the nature
of the data collection process, we chose for a data selection process to enhance the validity of the final dataset. Nevertheless, the generalization of the results is limited because of the uncommon way in which the data has been gathered. Furthermore, we can not state that the Software Supply Networks include all the suppliers or that the Product Deployment Contexts contains all relevant components. Some software vendors may not be aware of some small (open source) components that have, consciously or unconsciously, been incorporated into the leveraged product.

The next step to take to generalize the findings presented in this paper, is carrying out a large sample survey with product software companies and perform an analysis on the dataset, similar to the one employed in this paper. Also, case studies using unique or representative cases [15] can contribute to provide a higher level of generalization of the findings presented in this paper. The same also applies for the proposed matrix. Further validation, from a PDC or system architectural perspective is needed.

7 Conclusion

In this paper we presented the results of a small sample survey performed on the Dutch product software industry. In total, seventeen out of twenty-seven small to medium-sized Dutch product software companies were selected to be part of the dataset after applying selection criteria to enhance the quality and integrity of the dataset. Based on this dataset, we addressed supplier relationships and the supplier selection process from a portfolio and software ecosystems perspective. Software products consist of multiple components, and software vendors develop only part of these components in-house. Therefore, they depend on suppliers of software, hardware and services. Because not all components are as critical for a system to run or to offer its added-value, we created a matrix that classifies components of a software product including the components that are part of its direct running environment. Within this matrix we defined four types of components. First of all, a distinction between core and context components was made. Core components are the essential building blocks and can be seen as the heart of the product and its running environment. Context components then, provide additional functionalities to the product that gives the product its added-value to a customer. Furthermore, in a second level of decomposition, on the other axis of the matrix, we distinguished between critical and non-critical components. In this classification, a critical component is a component that is not easily interchangeable with another component offering similar functionalities or that adds a significant amount of added value to the product, whereas a non-critical component does not.

Out of the small sample survey, different supplier strategies were noted. Some software vendors choose to become fully dependent on a large software ecosystem orchestrator. While it heavily increases supplier dependence, this is regarded as an opportunity, because of a perceived guarantee of continuation of support and a strong market position. In addition, the often offered partnership models by
large software ecosystem orchestrators is another driver in becoming dependent on them. On the total contrary, several software vendors opt for a minimal dependency structure. Results show companies belonging to the larger company size often choose to be more independent than small and medium companies, primarily because they consider dependence on suppliers to be a threat. Software vendors also indicated to be reviewing their current supplier relationships, because of the advent of open source components as alternatives for proprietary components. Having to leverage continuous support and maintenance to customers, however, is experienced as a drawback of open source, as more responsibilities of support will end up with the software vendor rather than being provided by an open source organization that supplies the component. In addition, open source software licenses need to be carefully reviewed to avoid risks associated with the inclusion of open source software within the software vendors’ final product.

Software vendors mentioned several selection criteria when looking at potential component suppliers. Relationships with suppliers have a perceived level of intimacy, being either intimate, familiar or unfamiliar. We found that critical core component suppliers have a higher perceived level of intimacy than non-critical component suppliers. In addition, suppliers are also selected based on their software ecosystem health, which is regarded as an overall performance indicator of an ecosystem. As open source components are becoming interesting alternatives for software vendors, for example to reduce their supplier dependence, an examination of open source licenses becomes more prominent in the supplier selection process due to major risks associated with the inclusion of open source software. Also, support and maintenance flows play a decisive role when selecting both core and context component suppliers. This is especially the case when a component can be classified as critical and is thus of importance to the software vendor, since a well organized support and maintenance flow is then essential. With the selection criteria as noted as a basis, a table has been created to provide an overview of which selection criteria are specifically important to each type of component.

The identified strategies and the classification table of supplier relationships including the factors that are at play for supplier selection related to component type, has to be extended. Further studies need to be addressed on supplier relationships and the supplier selection process in order to gain a broader insight into this topic. Furthermore, more research needs to be employed in order to further validate and evaluate the presented classification matrix. This matrix can prove to be valuable not only from a Product Deployment Context perspective, but also from a system architectural perspective when discarding the simplification step employed in this paper to make it applicable from a PDC perspective. Furthermore, studies need to addressed on software ecosystem selection to gain a broader insight into mechanisms that are at play when making a decision on what ecosystem to join.
References

Improving Activity in Communities of Practice through Software Release Management

Kevin van Ingen¹, Job van Ommen²

¹ Utrecht University, Faculty of Science, Postbus 80125, 3508 TC Utrecht, The Netherlands
² {k.l.vaningen, j.vanommen}@students.uu.nl

Abstract. Keystone players need to nurture its inhabitants to keep the ecosystem active. Many modern ecosystems have communities of practice where knowledge is transferred by collaborative problem solving, sharing of ideas, software components or configurations. In recent years a widely spread medium for communities of practice is the use of online discussion boards. This research proposes a method to analyze the relationship between software releases and activity in a community. This paper explains how software ecosystem keystone companies can use software product release management to cultivate communities of practice using an illustrative case study. In this case study a comparison is made between two communities in the Android ecosystem. The results are validated with Google Trend analytics.

Keywords: software ecosystems, communities of practice, knowledge management

1 Introduction

Software vendors have become networked and integrated with suppliers, resellers, and even customers. This phenomenon has been the enabler for the research field of software ecosystems. A software ecosystem is “a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them” [1]. Within these ecosystems we can recognize different roles such as Keystone or Niche. The Keystone role within a software ecosystem belongs to the entity that behaves in such a way that its effects propagate through the entire system. In each case, the keystones occupy richly connected hubs that provide the foundation for creating many niches, regulate connections among ecosystem members, and work to increase diversity and productivity [2]. The Niche role within a software ecosystem belong to the entity that acts to develop or enhance specialized capabilities that differentiate it from other firms in the network, leveraging resources from the network while occupying only a narrow part of the network itself [2]. Among the ecosystem actors are developers who use the platform of the keystone player to develop their own products. Because of their contribution to the ecosystem developers can be considered niche players. Actors in an ecosystem interact with each other by sharing information about software solutions and by doing so become part of a community of practice [3]. These communities are not necessarily limited to
developers. Ecosystem enthusiasts, customers, or even hardware vendors can partake in these communities of practice.

The goal of this study is to analyze the dynamics of an online community in relationship with software releases from a software ecosystem perspective. By examining the way niche players are affected by actions of the keystone this research contributes to the body of knowledge on software ecosystems. The results of this research will provide keystone players in a software ecosystem with new tools to influence the community. Providing insight on the effects that different types of platform releases have on online communities of practice is a contribution to the body on knowledge of software product management. The main research question addressed in this survey is:

“How do platform releases impact activity in an online community of practice?”

This research question is answered by these two sub questions:

1. Is there an effect on communities of practice during a product release?
2. Do the effects on a developers community correlate with the effects on a community with a broader audience?

The remainder of the paper is structured as follows. Section 2 describes the research method of this case study. Section 3 describes the theoretical background for software ecosystems, online communities of practice, and the relationship between communities and software ecosystems. Section 4 presents a description of the results of the case study and an analysis of the data. Section 5 provides the answers to the research question and sub questions. Section 6 discusses this research short comings and future research directions.

2 Research method

This research identifies the relationship between software releases as an event and activity in an online community of practice. A case study with statistical analysis will provide insight in recurring patterns.

The research field of software ecosystems is a relatively young field of research. This research needs to be repeated and validated in other industry domains. The research method used for this research is very explicit but can also be generalized for use in other domains.

The online community of practice subject to this research needs to offer information about the members and their activities in a significant time range so that we have several releases and release-intervals to measure. This research follows the following steps:

1. Conduct preliminary literature research to discover hypothesis, measurement, and case selection criteria;
2. Construct a list of event in the Android community that potentially influences community activity.
3. Select one or multiple case study subjects and conduct a preliminary assessment on the community dataset;
4. Choose largest possible case and construct an scraping tool to obtain community data;
5. Data gathering using the custom built scraper and use Extract, transform and load (ETL) tooling to transform the cleaned dataset into a reporting tool;
6. Construct a timeline of community activity in the community;
7. Construct a timeline of Google trend analysis for the Android platform;
8. Transpose the list of Android related events on the timelines;
10. Explain the effect of Android related events on the community in the results.

The case study is validated through the use of multiple data sources for the analysis.

2.1 Case study criteria

The Android system is an open source software stack for mobile devices that includes an operating system with eight major releases and dozens of small releases in the past four years. This information can be obtained through the Android Developers website [4].

Data about software releases and community members and activity are analyzed in this research en need to be obtained. Criteria for a community are:

- The community has to be active from 2008 until now and have at least 10,000 post per year (about 30 per day) to identify trends;
- The community has to display meta-data about threads (date, replies, views);
- The community has to display meta-data about members (post started, join date);
- The community has to have post and member data publicly available on the internet in some sort of structured form in order to be suitable to build a scraper for;
- The community preferably has an URL structure which enables an external party to do some preliminary data filtering on inactive users.

2.2 Data mining and analysis

The information from the selected community is obtained through a custom-built scraping tool. This tool is similar to a search engines bots or spiders that crawl the web for new links. When given a base URL it filters out relevant information and URLs on a webpage using text based pattern matching in the source code of the webpage. New URLs on its turn are checked for relevant information and recursively,
new URLs. Relevant information about posts is filtered out and stored in a temporal dataset. For analysis purposes ETL is used to structure information in a generalized format needed for this study. The data is saved in a database and is structured as described in Table 1.

<table>
<thead>
<tr>
<th>Field</th>
<th>MySQL type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadid</td>
<td>INT</td>
<td>Thread identifier</td>
</tr>
<tr>
<td>postid</td>
<td>INT</td>
<td>Post identifier</td>
</tr>
<tr>
<td>member</td>
<td>VarChar(255)</td>
<td>User who posted a contribution to a thread</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
<td>Date stamp of the thread</td>
</tr>
</tbody>
</table>

Table 1. Data structure of mined Androidcommunity.com messages.

The data obtained from the communities will be analyzed and standardized into a comparable format. The standardization is done using an index scale for activity where for every time-interval there will be an index number relative to the maximum amount of activity which will be indexed at 100%. The indexing of activity is a necessary step to ensure the comparability of data from different sources.

For the data on platform releases we use the roadmap and version history provided by the platform owner [4]. Because in mobile platforms a software release for the end user is always preceded by a corresponding software development kit (SDK) release we use the release dates of the latter. The availability of the end user version differs with every carrier and model so the SDK release is more suitable for providing a single point in time which may be correlated with a change in the activity of communities of practice. Especially because the communities of practice contain developers who depend on the SDK for their role as niche players, the authors consider the release date of SDK versions the starting point of the actual platform release.

3 Theoretical background

In this chapter we use related literature to position this research. We divide this chapter in two sections. First, we describe research in the field of software ecosystems. Secondly, we describe research in the field of online communities of practice.

3.1 Theoretical background ecosystem view

The role of a keystone player within an ecosystem is to promote the overall health of the ecosystem [2]. By definition the platform supplier is a keystone firm [5]. Because in this review we are looking at the effect on platform releases on the community of practice we hope to contribute to the understanding of how a keystone player can revitalize the community and thus the ecosystem. For this research it is not relevant who plays the role of keystone as long as it can be identified, so we will not get into the depths of exploring who exactly owns the Android platform and is thus the
keystone player. It is sufficient to recognize that a platform is released by a keystone player and that the rest of the ecosystem may utilize this new release.

One of the key success factors for companies employing a keystone strategy like Microsoft and eBay did was appealing to a community of people [2]. The value of a community is also evident in the field of product innovation and development and has successfully been leveraged by companies, such as for example Ducati and Eli Lilly [6]. A community perspective in the software ecosystem includes external developers, domain experts and users. The role of this community is vital to the health of the software ecosystem and thus for the keystone player whose health depends on that of the ecosystem [2]. In the next section we will elaborate on (online) communities and their dynamics.

3.2 Theoretical background community view

Actors within the community may have different roles. Not every person in the community will be a “niche player”, some will have more passive roles such as end-users who ‘just’ like to be up-to-date. Similar to Open Source Software (OSS) Communities the roles of community participants are not distinct: all users are potential developers and thus potential niche players [7]. Any modification, improvement, and extension made to an OSS system also influence the OSS community. The dynamics of the OSS and the community associated with it can affect the level of participation a member has within the community. Because of this evolution of the communities topic is important for keeping the community alive [7].

One of the success factors for a platform like Android is to constantly extend the set of features to remain attractive for the developer community. It is also a pitfall because functionality created by the developer community can be incorporated so that it becomes a commodity, thus alienating the developers of that particular functionality [5]. Especially on the internet the importance of a community of 3rd party application developers has become central to the strategy of many companies.

4 Results

In this chapter we present the results from the case study of the Android community and the effects of Android releases on the activity in the community. This chapter starts with the descriptive results, after this the research results are presented and explained.

At the time of this research the open source Android system seems to have multiple very active communities. Different Android communities where selected and assessed based on research criteria. Androidcommunity.com was found capable of supplying proper information for this research. Androidcommunity.com originated as a news and discussion platform from the creators from Slashgear.com. Androidcommunity.com targets people that have interest in Android mobile hardware
and software for tablets and phones. Its users are end users that have interest in news from manufacturers about new phones and operating system updates but there are also developers for the Android apps.

Google Groups is a second community and was considered as a good complementary source of information. It is a discussion platform primarily focused on Android developers. It is integrated in the Android development documentation because of its linkage to Google. It is important to keep in mind that the user population of this community is more homogeneous because it is targeted at developers.

Different communities will be affected by new product releases. A way to access information from the communities is by web search engine. Google being the leading search engine at this point in time offers statistics on search queries performed. The meta-data of search trends are used to validate findings in the communities.

To illustrate the actual process of data mining, conversion, and analysis the process is depicted in Error! Reference source not found.. The different data-sources are mentioned and depicted as a server. The different high-level processes: data mining, ETL, and analysis are depicted sequentially. The data sources concerning the different steps are connected to the corresponding related processes inside the high-level processes.

**Fig. 1.** Data mining and analysis process
This study covers the results of two Android communities. Both communities offer similar meta-data suitable for this research. The communities are both active since the beginning of Android development in 2008. Descriptive results of the sizes of both communities are depicted in Table 2.

### Table 2. Descriptive statistics of the Google Groups community and Androidcommunity.com

<table>
<thead>
<tr>
<th>Metric</th>
<th>Google Groups</th>
<th>Androidcommunity.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of members</td>
<td>49,245</td>
<td>22,369</td>
</tr>
<tr>
<td>Amount of posts</td>
<td>156,241</td>
<td>313,874</td>
</tr>
<tr>
<td>Average post/month</td>
<td>3.633</td>
<td>7.300</td>
</tr>
</tbody>
</table>

A list of platform releases was gathered using Android platform release notes. The platform release notes also classify the impact in of the release as a major or minor release. An overview of platform releases is depicted in Table 3. The active versions column provides a current overview of the different Android versions that have accessed the Android Market in a 14 day period [8].

### Table 3. Android platform releases

<table>
<thead>
<tr>
<th>Date</th>
<th>Platform version</th>
<th>Classification</th>
<th>Active versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-09</td>
<td>Android 1.0 SDK</td>
<td>Major</td>
<td>Not available</td>
</tr>
<tr>
<td>2009-02</td>
<td>Android 1.1 SDK</td>
<td>Major</td>
<td>Not available</td>
</tr>
<tr>
<td>2009-05</td>
<td>Android 1.5 SDK</td>
<td>Major</td>
<td>1.9%</td>
</tr>
<tr>
<td>2009-10</td>
<td>Android 1.6 SDK</td>
<td>Minor</td>
<td>2.5%</td>
</tr>
<tr>
<td>2009-11</td>
<td>Android 2.0 SDK</td>
<td>Major</td>
<td>Not available</td>
</tr>
<tr>
<td>2010-01</td>
<td>Android 2.1 SDK</td>
<td>Minor</td>
<td>21.2%</td>
</tr>
<tr>
<td>2010-06</td>
<td>Android 2.2 SDK</td>
<td>Major</td>
<td>64.6%</td>
</tr>
<tr>
<td>2010-12</td>
<td>Android 2.3 SDK</td>
<td>Major</td>
<td>9.2%</td>
</tr>
<tr>
<td>2011-02</td>
<td>Android 3.0 SDK</td>
<td>Major</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

In this research the focus was primarily on activity expressed by the amount of posts in the community. For interpretation purposes the activity of both the communities was indexed on a percentage scale. The index numbers of both the communities are combined into a new index called the average community index. To ensure validity a correlation between the two datasets is constructed. The total dataset has a low correlation of 0.38. Close examination of the dataset revealed a significant decrease in the activity in Androidcommunity.com. A correlation of the two datasets for the years 2008, 2009, and 2010 results in a correlation of 0.76. The strong correlation is interesting considering the differences between the audiences of both communities. The correlation indicates a relatively similar response on Android SDK releases in the developer community and a community with a much broader interest.

**Fig. 2** depicts the indexed amount of activity for each month and the Android platform releases over the years 2008, 2009, 2010, and 2011.
Visual analysis of the graph indicates that a relationship between releases and activity in the community is present. For most releases there is an increase in activity in the community. The details of the release can affect different communities in different ways. Some releases primarily application developers, others enable new hardware opportunities like tablets and multi-core processing, and some affect both. In order to illustrate this we also show the releases of a few notable phones. As the graphs show the releases of phones coincide with the release of important updates. The graph in Fig. 2, primarily shows peaks that occur in both datasets. The classification of the impact of the release seems to have no significant effect on the activity.

A significant result in this graph is the difference of length of peaks. Peaks range from one or two months to 4-6 months. An observation is that high peaks (high amplitude) typically have a short duration.

The findings obtained by both communities are compared to Google Trend analysis. Google Trends provides data about popular search queries. In this research some keywords are selected and a weekly search popularity ranging from August 2008 until May 2011 was combined in an average index. The keywords are selected in a way that they represent popular topics in both the Google Group community and androidcommunity.com. To ensure representativeness of the queries a correlation matrix was constructed. The correlation matrix described in Table 4 show highly correlated search trends.
Table 4. Correlation matrix of Google Trend analysis

<table>
<thead>
<tr>
<th></th>
<th>Android</th>
<th>Android SDK</th>
<th>Android Phone</th>
<th>Android Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Android SDK</td>
<td>0,97</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Android Phone</td>
<td>0,95</td>
<td>0,96</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>Android Release</td>
<td>0,84</td>
<td>0,88</td>
<td>0,90</td>
<td>1</td>
</tr>
</tbody>
</table>

The popularity of different search queries is plotted in a graph combined with platform releases depicted in Fig. 3. Visual analysis of the graph indicates that a relationship between releases and Google searches is present. For most releases there is an increase in searches in the Google search engine.

Fig. 3. Google trend indexes combined with Android platform releases

Both of the Android communities and the Google trend analysis indicate the existence of a relationship between activity in a community and platform releases. There is no visible relationship between the classification of the impact of the release on the response of the community. It is important to mention that peaks in activity or search trends cannot be fully understood by new releases of the Android platform. There seem to be two distinctive patterns in the activity peaks. There are short living
bursts of activity with high peaks that end within one month and peaks with a smaller amplitude that have a longer duration of some months.

5 Conclusion

This paper addresses the research question: “How do platform releases impact activity in an online community of practice?” The results of this research show peaks coinciding or following platform releases in both the communities and in the Google trend analysis. It is therefore safe to say that platform releases cause higher activity in online communities of practice and increased interest for the entire ecosystem.

The results show a positive effect on the activity after each platform release. The overall effect on the communities in the case study is not enough to keep them from degrading. Of all the factors affecting the activity in communities of practice the topic evolution part – in this case a platform release - has now been established. This answers the first sub question: “Is there an effect on communities of practice during a product release?”

The activity in a community of practice with a homogenous population of developers corresponds to that of the community with a broader audience. The correlation between these two communities along proves platform releases have an effect on the entire ecosystem and not just on niche players. In addition to the rise in activity in communities of practice, the rise in global interest for the ecosystem was clearly visualized by the Google search trends. This answers the second sub question: “do the effects on a developers community correlate with the effects on a community with a broader audience?”

A bi-product of this research is the creation of an research method which is explicitly mentioned in this paper and generalized to the level that it can be used in other research areas.

The results of this research show that keystone players can utilize platform releases to improve the overall health of the ecosystem. It also shows that an important factor in online community research is the evolutionary aspect of the community’s topic, a factor which has hitherto largely been ignored.

6 Discussion and Future Research

Future research must investigate how this factor can be leveraged by the keystone player. An important weakness of this research is to ensure that a certain event is correlated or even causes an effect in the online community of practice. Because before the actual platform release the community is already aware of this pending event, thus activity may rise in anticipation. Even worse, activity may also rise due to rumors of a pending event. Other events, such as conferences or hardware releases,
may also affect the community’s activity and coincide with platform releases. This problem has also been visualized in the figures but has not been explored in this research. Such ecosystem events need to be researched so that we can establish a full set of measures affecting communities of practice within the ecosystem. In this study we were unable to differentiate between events on a detailed level. By knowing the details of an event it may be possible to perform an improved analysis which could explain the differences in corresponding activity. A study of a classification of the nature of product releases in the context of software ecosystems could provide a better explanation of community response. Such a classification system may also be used to formalize the meaning of notions such as major and minor release. Another weakness is the data we used to determine activity. In this research we use posts per timeframe to determine the community’s activity. Forum statistics show other important factors like “users online” on a given date. Unfortunately we were unable to obtain such factors and thus the measure used for activity may not be as accurate as possible.

References

Assessing Developers’ Interdependency and Project Governance in the Open-source CyanogenMod Community

Martijn van Zetten and Wilbert Seele

Department of Computing and Information Sciences, Utrecht University, Princetonplein 5, 3584CC Utrecht, The Netherlands
{m.d.vanzetten, w.j.seele}@students.uu.nl

Abstract: The CyanogenMod firmware is a well-known firmware within the Android rooting scene. It aims to deliver the latest Android technology as possible and has two major releases each year. In this paper we present a visual representation of the community to identify how developers depend on each other and we provide an overview of the governance structure within the project. The data have been gathered using a data-mining tool. The visualization of the community shows us that most work is being done on generic functionality and that there are a small number of contributors who add value to a large number of repositories. We have identified three types of developers in the community. Furthermore, we see that a basic governance structure is in place, based on the fact that each developer should do what he finds most interesting, to keep the motivation is high and to make sure deadlines are met.

Keywords: Software ecosystem, Android OS, CyanogenMod community, open-source governance, developers

1 Introduction

Android is a software stack that includes an operating system, middleware and key applications [1]. The Android OS for mobile devices (developed by Google) is growing fast in the market of mobile operating systems [2]. However, Google is not the only developer for Android-based operating systems. Short after the introduction of the HTC Dream mobile phone in September 2008, the Android community discovered that it was possible to ‘root’ Android devices. Rooting refers to using exploits to achieve system-level access on a device that was locked by the manufacturer [3]. This discovery allowed the phone’s stock firmware to be modified.

CyanogenMod (CM) is an open-source, Android-based firmware and is well-known within the ‘Android rooting-scene’. We define firmware as a collection of small programs that controls an electronic device. Within the context of this paper (and the Android community) it has come to mean the operating system and all attached components and hardware drivers. CM is developed within the Android eco-
system\textsuperscript{1} by the CM community. In this paper, we will use the definition of a software ecosystem (SECO) as provided by Jansen, Finkelstein and Brinkkemper in \cite{Jansen2014}, who define a software ecosystem as “a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them”. We do not view the CM community as an ecosystem itself, but as a community within the Android ecosystem, due to the small size of the community and the limited number of developers (Fig. 1).

\begin{figure}[h]
    \centering
    \includegraphics[width=0.5\textwidth]{image.png}
    \caption{Positioning of the CM community.}
\end{figure}

CM has a very active community and has major releases about twice a year\textsuperscript{2}. CM aims to deliver the latest Android technology as fast as possible. CM claims to increase performance and reliability over official firmware releases by Google. Besides a very active community of developers, CM has a large user base with about three and a half thousand installs per day in June 2011\textsuperscript{3}.

The CyanogenMod software offers the user a wide variety of features not available in the official software from Google. Examples of these features are support for VPN, support for a wider variety of media codecs, multi-touch, a reboot menu and WiFi/Bluetooth and USB tethering\textsuperscript{4}. Some of the mentioned features are present in the current official Android software, but were supported by CM long before supported by the official Google releases. Furthermore, due to several updates to the software, the developers of CM have managed to extend the battery life of mobile devices.

For each supported mobile device\textsuperscript{5}, the firmware base needs to be adjusted for use on said devices. Because each device uses different hardware and drivers, there is no uniform CM firmware. In order for the firmware to work correctly on all supported devices, the CM software needs to be adjusted to the hardware of each supported

\begin{flushleft}
\textsuperscript{1} Seeing as Android is based on the Linux kernel, we see the Android ecosystem as part of the Linux ecosystem, as can be seen in Fig. 1.
\textsuperscript{2} According to the release scheme at \url{http://www.cyanogenmod.com/}.
\textsuperscript{3} According to the official CM statistics at \url{http://stats.cyanogenmod.com/}, retrieved at 3 June 2011.
\textsuperscript{4} According to the official CM website: \url{http://www.cyanogenmod.com/about/features}
\textsuperscript{5} Cyanogenmod is supported on 27 different (mainstream popular) devices at the time of the 7.0 release. Supported devices can be found at \url{http://www.cyanogenmod.com/devices}.
\end{flushleft}
device. For example: each device uses different hardware for baseband reception, modem use and wireless connectivity, so the software needs to be tuned to work with that hardware.

This paper is organized as follows: in the following sections, the problem statement and research question are described. In section 2 we discuss some related literature. In the following section (section 3), we will describe the research method. In section 4, we describe the data mining process and the acquisition of the data, along with some visual representations. In section 5, we describe the governance of the CM project. In the last sections, we will give our conclusions, discussion and future work.

1.1 Problem Statement

When looking at the CyanogenMod community, we see that some of the developers are developing for specific devices, while some are developing cross-device components (generic functionality). However, it is unclear which developers are working on cross-device components and who are working on device-specific components. Furthermore, it is unclear if developers tend to work on different projects, both cross-device and device-specific. When a new release is launched for CM, all devices are supported. This means that there needs to be some kind of governance on the release scheme for each device.

1.2 Research Question

In this article, we want to research how development of different versions of the CM firmware is connected when looking at its developers. We will do this by answering the following research question: "How do developers in the CM community depend on each other, seeing as there is parallel development for cross-device and device-specific functionality?" Furthermore, a second research question we pose is: "Is there any kind of governance for the simultaneous release of a new version of CyanogenMod, and if so, what does this governance consist of?"

2 Related literature

As Jansen, Finkelstein and Brinkkemper describe in [8], they distinguish three different perspectives on SECOs: the software ecosystem level, the software supply network (SSN) level and the software vendor level. Seeing as our study focuses on the CM community, we can see this as a study on the level of a software vendor. This level is described as: “all products and services supplied by the vendor and the vendor itself”. Although CM cannot be seen as a traditional software vendor (as there is no financial flow), the community of developers can be regarded as the vendor in this case. For reference, the SSN is described in Fig. 2.

In their paper [7], Kabbedijk and Jansen identify elements within a SECO, characteristics of such a SECO, the descriptives of a SECO and the roles within a SECO. To
aid their study, they have provided a visual representation of the Ruby programming
language SECO. Furthermore, they have provided an “interaction map” of the de-
velopers in the SECO. In our study, we want to perform a similar research, but in a dif-
ferent ecosystem.

In our study concerning open-source governance, we have found several approac-
hes. Shah [5] describes that there are two broad sets of governance mechanisms for
open source projects: Decision-making rights (code control and domination of mail-
ing list interaction) and property rights (ownership of source code, restrictions on
user, modification and distribution of source code, proprietary modifications) that
influence developer participation. He concludes that usually there is a hybrid gover-
nance strategy.

In their paper [6], Lerner and Tirole describe that a basic institutional innovation of
an open source project must be the crafting of a governance structure, which enables
investment without crowding out donations. They describe a basic governance struc-
ture for open source projects. They describe the most basic governance structure as
two levels:

- **Technical level**: This level contains a definition of the work process and the
division of labor. This level defines the main schedule of the project. It also
incorporates a role that is similar to a project manager, to guide the project
and guard deadlines.

- **Level of analyzing the incentives**: an analysis of the incentives is needed to
make sure that the actors perform according to the assigned tasks. This is a
more abstract level and is more present in open source projects than in cor-
porate projects, since open source projects are run by volunteers.

Goeminne and Mens [9] state that traditionally, software studies rely on the soft-
ware source code to analyse and predict how software evolves. However, the authors
are of opinion that researchers need to focus on more elements to get a full picture of
software evolution. For instance, we also need to look at the human aspect (the de-
velopers) and the interaction between developers. This will be one of the foundations of
our study, as we will focus on the developers and the dependencies between them.
3 Research Method

We will obtain our research data by using the CM repositories\(^6\) (located at the CyanogenMod repositories at Github\(^7\)) and extracting the data needed using the Greppel tool, developed by students at Utrecht University. By using this data-mining tool, we can extract metadata about development (commits, developers, activity) from the CM repositories with which we can analyze the development of the software, and identify trends in its development.

Our research will focus on the developers, their relations and their activities in the CM project. For the purpose of this paper, the CM project will be viewed as a community within the Android ecosystem. The project activity database produced by the Greppel tool will allow us to make a visual representation of the developers within the CM project. The visual representation is an aid to understand developer’s dependencies.

For the visual analysis of the data gathered with the Greppel tool, we will use a Social Network Analysis tool called Gephi. Gephi is open source software for graph and network analysis [4]. The tool allows us to display a large network in real-time and speed up the exploration. The use of a Social Network Analysis tool is ideally suited to our developer-centric research. The quantity of data gathered through our Greppel-driven data mining will allow us to provide a graphical visualization of the CM community, and will help us to see how developers depend on each other for the development of a CM release.

We will also perform a short expert interview with one of the head developers of CM. We have a series of questions relating to the use of repositories, development and developer growth for helping us in answering the primary research question. This interview will serve as validation for our findings, seeing as a head developer can be seen as an expert concerning CM.

3.1 Greppel data mining tool

The tool we use for the data mining (and part of the visualization process) is called Greppel. It is a tool that is developed by students at the Utrecht University, one of whom is co-author of this paper (Wilbert Seele).

Greppel is a tool designed to support software ecosystem research from a developer- and software-based perspective. It achieves this by tracking software-project activity through version control systems. Supported systems include Git and Bazaar, but as the system is plug-in based this can be extended. It differs from the version control systems own view on projects by compiling the project data into its own unified projects database. This enables users to map entire dependency maps and developer associations and activities. Greppel also allows the user to map these pictures over a time-frame as well. When we have extracted the data from Github, we will further

---

\(^6\) A (software) repository is a storage location from which software packages may be retrieved.

\(^7\) Github Social Coding - https://github.com/
analyze the repositories using Gitstats\(^8\) for statistics such as lines of code (LOC) per developer, number of commits and time active.

### 3.2 Open Source Governance

The answering of the second research question relies on both an interview with one of the head developers of CM and existing models described in scientific literature. We will identify some models used in open-source governance, this will be elaborated upon in section 2. Furthermore, a section of the expert interview will concern the governance of the CM project. Based on the expert interview with one of the head developers of CM, we will compare the existing governance structure to the ones proposed in the literature. The interview will consist of questions concerning the basic structure of the project, governance of the projects and the guarding of deadlines.

### 4 Results

The research data are obtained from the data and metadata of the CM project repositories at Github. At this time (June 2011), the CM project has 202 public repositories and 53 members (developers). A graphical representation of the growth of the project in number of developers (since the startup in 2008) is shown in Fig. 3. This graph shows us how the community (and thus the number of dependencies) grows over time. The green line depicts the release of the 5.0 version of CM. The red line and the purple line depict the release of the 6.0 and the 7.0 release, respectively. According to one of the head developers of CM, before the 5.0 release, nearly all the work was done by the lead developer alone. Before the development of the 5.0 release, the core team (four head developers) came together. After that, the number of developers grew steadily. The release of new CM firmware seems to be consistent: when Google releases a new version of Android, a new version of CM is released roughly three months later.

![Fig. 3: Growth of the CM project in no. of users.](image)

\(^8\) Gitstats – GIT History Statistics Generator - http://gitstats.sourceforge.net/
The number of developers stabilized after the 7.0 release, which is due to the fact that the project has reached an ideal number of developers for the project, with new developers mostly working on adding new devices.

The repositories are divided in cross-device Android repositories, device-specific repositories, frameworks, hardware, packages and apps and kernel. For the sake of this paper, we will look in more detail into some development repositories both cross-device and device-specific (such as the Android kernel in section 3.1) and we will provide an overview of the division of developers over the entire CM project. We will look into the amount of value developers add to CM and the amount of “work” they put into it. This amount is measured by the number of commits and lines of code.

4.1 Visualization of the community

The CM project consists of over 200 repositories (although there are repositories without contributors). Due to limited time and resources, we have randomly selected twenty-six repositories for the visualization of the community. However, we have selected a few repositories from each category (Android, device, external, packages (apps), hardware and system). Although this is not a complete visualization, it gives a good picture of what we want to describe in this paper. The community is depicted in Fig. 4.

What we see in Fig. 3 is that the development repositories are in the middle, and the developers are circled around it. The grey lines indicate that a developer is contributing to the connected repository. A disadvantage of the visualization tool is that the strength of a relationship cannot be derived from the figure. We can, however, conclude that the number of connections of a developer indicates how important a developer is for the project. The fact that there are gaps between the developers on the left and the developers on the right is because of better readability and has no further meaning.

4.2 Cross-device repositories

When looking at the development of the kernel, we can see that it relies heavily on the general development of the Linux kernel. This can be seen in the top twenty contributors of the kernel. Linus Torvalds, chief architect of the Linux kernel, is the top contributor of the Android kernel development with 4.08% of all commits (9208). This is an indication that the CM community is not only reliant on the Android ecosystem, but also that it is greatly influenced by the bigger ecosystem, the Linux ecosystem. This has already been visualized in Fig. 1. This picture does not contain other communities and ecosystems connected to the depicted communities.
As Kabbedijk and Jansen describe in [7], the top \( n \) developers (in the case mentioned in the paper \( n=5 \)) contribute a significantly bigger amount than the rest of the developers. When looking at the development of the kernel (not only for CM, but for all Android systems), we see that only eleven of the fifty-three CM developers contribute to the Linux/Android-kernel (Fig. 5). The top twenty contributors (of roughly 7500 in total, 0.27%) are responsible for 45.36% of all commits, so our findings seem to confirm earlier findings (for example [7]). In Fig. 4 we see the kernel repository in the middle, and the CM developers linked to that.

When we look at some cross-device repositories, we see that a big number of developers contribute to the cross-device repositories (Fig. 6). The CM project has 53 developers, some of whom are more active than others. If we look at the cross-device repositories, we see that the top five developers are responsible for more than 57% of
the work. The number one contributor (Steve Kondik, who is the lead developer of the CM project) is responsible for more than 25% of the work alone. Furthermore, we see that there are a number of developers who have added value to multiple repositories. This leads us to the conclusion that the community relies heavily on the top contributors of the cross-device repositories.

![Fig. 6: Developers of a cross-device repository.](image)

### 4.3 Device-specific development

An analysis of the more device-specific repositories shows us that they are developed in the same trend as the cross-device repositories. However, the number of developers working on a device-specific repository is smaller than those working on the cross-device repositories. On cross-device repositories, there are 10+ developers on average. On device-specific repositories, however, there are one to five developers on average. A typical example of developers on a device-specific repository is shown in Fig. 7. The analysis of the device-specific repositories showed that a small number of developers is active in most of the device-specific repositories. The user "Steve Kondik", for instance, is active in over a hundred repositories. In device-specific repositories, however, the developers who add to a lot of different repositories are not the leading contributors in the device-specific repositories. These are usually developers working on a single or a few repositories (and can spend all of their time on those repositories). According to one of the head developers, the most work (>50%) is usually done by a single user, supported by the other developers contributing to that device.

According to the developers of CM, the work on device-specific repositories does not take as much time as we initially thought. Once the groundwork for a device is laid, it is rare that a device stops working. Although the development of device-
specific repositories is certainly important, the bulk of the work is being done on the generic functionality.

Fig. 7: Developers of a device-specific repository.

4.4 Types of users

When looking at the community, we can distinguish 3 different kinds of developers (Fig. 8A-C), distinguished by the number of repositories they work on and, related, how important they are to the community. These categories are distinguished based on the developer’s structure (described in detail in section 5.1).

1. **Cornerstone developer**: A developer who spends his time contributing to a large number of repositories (8+), both cross-device and device-specific. Furthermore, they are responsible for governance. The loss of such a developer would greatly influence the community, since he contributes to a large number of repositories. This group consists of six users (~11%) and was formed at the beginning of the project.

2. **Multi-repository developer**: A developer who contributes to a small number (2-7) of different repositories, possibly both cross-device and device-specific. It is possible; however, that he spends his time only on device-specific repositories or only on cross-device repositories. This is the biggest group and consists of 31 developers (~59%). Most of the multi-repository-developers joined in 2010 (after the cornerstone developers).

3. **Single-repository developer**: A developer who spends his time contributing to only one repository, but is the major contributor to that repository (and often is the sole contributor to that repository). The replacement of a single-repository developer is easier than with the two above, since only
one repository is affected by that developer. This group consists of 16 users (~30%). Although single-repository-developers joined throughout the entire duration of the project, most of them joined in late 2010 and beginning of 2011. We think that this is due to the fact that the number of developers on cross-device-components has reached an optimal number, and only new developers for new devices are attracted.

### 4.5 Dependency between developers

In the introduction of this paper, we stated our research question as follows: *How do developers in the CM community depend on each other, when looking at the parallel development for each supported device?* We see that there are certain roles for developers in the project (Fig. 8). The cornerstone developers are the most influential developers in the project and contribute the most to the project. They are active in both the cross-device and the device-specific repositories and can be seen as the glue that holds the project together. They are not the most active developers in the device-specific repositories; those are often single-repository developers. We see that the cornerstone developers are all part of the top two layers, as shown in Fig. 9.

![Fig. 8: Three different kinds of developers: Cornerstone developer (A), multi-repository developer (B) and single-repository developer (C).](image_url)

However, the parallel development of CM is not as deep as we expected. Most of the work still lies in the development of the generic functionality, so most of the development depends most on the contributions of the cornerstone- and multi-repository developers. The contributions of the device-specific single-repository developers,
however, make sure that the CM firmware will work on all supported devices when a
new version is released.

When we map the types of users in Fig. 8 to the growth of developers in Fig. 1, we
see that first the cornerstone developers joined, and the multi-repository-developers
joined later. The latest additions are mostly single-repository-developers, with the
task of adding new devices.

5 Open Source Governance

As many open source projects, CM relies entirely on the efforts of volunteer software
users and developers, not paid managers and employees. According to Lerner and
Tirole [6], the motivation of volunteer software developers lies in the development of
the individual reputation (which is either comforting in itself or may be used as a
talent signal on secondary markets, e.g., the job market), rather than wages. Open
source projects are exemplars of a fundamentally different organizational model for
innovation and product development - referred to as collective invention, private col-
lective invention and community based invention [5]. The data we have collected in
this section are primarily obtained from the expert interview.

5.1 Governance structure

The CM project team consists of a layered structure (Fig. 9). On top, there are the
“system-wide administrators”. This group consists of the lead developer (Steve Kon-
dik), and three other core team developers. One of these developers was the source of
our information. The core team can do anything in the project, including: deleting
repositories push code up, skipping review and submitting to any repository in the
project. Below this core team, there is a group of several “head developers”. These
developers are able to submit code to any repository in CM as well, but cannot change
the administrative levels of other users. The third subset is device maintainers. These
people have the ability to push to their own personal device and submit. This level is
generally granted to the device maintainers who have been with the project for a long
time and are trusted by the core team to not break devices. The final subset is the
general users. Anyone with an account can alter repositories, but everything will be
reviewed by higher users before the code is pushed.

When looking at CM, there seems to be a basic governance structure similar to the
structure described above. The technical level is clearly present. The division of work
is done based on the interest of the developers. The progress is not really guarded,
because it is not really an issue. According to a developer in the core team: “The won-
derful thing about a group of people that do this on their own time is that they love
doing this. Sometimes we will post a reminder for someone to finish something up as
a deadline approaches, but for the most part it is not a problem.”
When looking at the incentive level, we see that each developer develops the parts that he actually likes to do. Since the development of CM is on a strictly volunteer basis, nobody is expected to *have* to do something. However, if there are features that someone wrote that are broken, that person is expected to fix that feature. In doing so, the developers maintain a high motivation, which contributes to deadlines being met.

The deadlines, however, are easily adjustable, with developers sometimes releasing a little later than the main group of devices. However, we see that deadlines are (almost) always met with a working firmware, but the status of that firmware can differ. On the set deadline, it is possible that a “gold” version (software that is ready for use by the customer) of the firmware is delivered, but it is also possible that a release candidate (RC) is released (a version with potential to be a final product). This seems to be decided based on the number of bugs found before the deadline. If not all bugs can be fixed, the CM development team will release it as an RC instead of a gold version. Several RCs can be released before a gold version is released. The 7.0 release of CM, for instance, has had four release candidates before going gold.

6 Discussion

The goal of this study was to provide an assessment of the interdependency of the developers of the community and to provide a case study in the field of software ecosystems, on the software vendor level. Some implications of this study are that we have provided insight in developer’s dependency in an open-source, volunteer community. We have also strengthened other research (for example, [7]) by confirming their findings.

The visualization of the community provided in section three is not a complete visualization, though it gives a good picture of what we want to describe. We have used twenty-six repositories for the visualization, of a total of over 200. A large number of repositories are maintained by one or two developers and some repositories are not being developed any further by CM developers. Adding all repositories will give a complete picture of the community, but for the sake of this paper, the visualization we have given will suffice.
For the analysis of the governance structure of the CM project, we were only able to get one developer to comment on the governance structure. In further research, we would like to expand the number of developers to interview and expand the interview with more questions, to get a more complete view of the governance structure.

In future research, a good continuation of this research would be to perform a study addressing the relations between the CM community and other communities within the Android ecosystem, or (communities in) other ecosystems. Furthermore, a study investigating the relations between the CM developers and Google would help to develop a more complete understanding of open-source SECOs.

The possibilities of future research in the field of open-source SECOs could lead to a more complete understanding of the field, which in turn could lead to more efficient ways of governing and executing projects.

7 Conclusion

In this paper, we have performed a study concerning the CyanogenMod firmware community in the Android ecosystem. The aim of our research was to see how the developers of the CM project team depend on each other, when looking at the parallel development for each supported device and to see what governance structure was in place for the CM project.

We have provided a visualization of the community. What we can learn from this visualization is that we can distinguish three kinds of users: the cornerstone developers, multi-repository developers and single-repository developers. We have also identified when these developers joined the project, this seems to be a sequential (first cornerstone-developers, then multi-repository-developers and single-repository-developers last). The three types of developers are closely related to our research question how developers depend on each other, when looking at the parallel development of the supported devices. The parallel development is not as deep as we expected. The most development concerns the generic functionality (cross-device repositories), most of which is done by the cornerstone developers. We see that most of the development for the device-specific is done by single-repository developers. Our findings have been validated by one of the head developers of the CM project.

The results show that there is a lot of interdependency between the developers of the CM project. We see that developers often commit to multiple repositories within the project. There are a number of developers which are the cornerstones of the project. They commit to the most repositories, make the most commits and add the most lines of code. The lead developer (Steve Kondik), for instance, is irreplaceable for the CM project, as he develops for over 50% of the repositories and is often responsible for a large percentage of the total number lines of code. We see that these cornerstones are most active in the cross-device repositories of the CM project, whereas the device-specific repositories are developed most actively by smaller contributors (often they are active for a single device-specific repository).

The second part of our research concerned the governance of open-source software projects, and whether the CM community has such a governance structure in place.
The governance structure we found seems to be similar to the structure described by Lerner and Tirole in [6]. The governance structure consists of two levels: the technical level (defining the work process and the division of labor) and the more abstract incentive layer, which is concerned with the incentives of each developer. This means that developers are assigned parts of the project that they actually like to do, to keep the motivation high and make sure that deadlines are met.

Our paper can be used as an indication in the inner workings of volunteer, open-source projects. It depicts how roles are divided among developers, and how the project is dependent on certain users. Furthermore, it adds some knowledge to the young field of ecosystems and aids in a more complete understanding of the field, specifically on the topic of open-source SECOs. Although the results have been validated, we cannot yet generalize our findings.

References

Influences on developer participation in the Debian software ecosystem

Eric Ververs¹ and Rick van Bommel¹,

¹ Utrecht University Buys Ballot Laboratory, Princetonlaan 5, 3584CC Utrecht, Netherlands
{L.J.Ververs, R.A.A.vanBommel}@students.uu.nl

Abstract. Nowadays, more and more open source software developers are starting to create software in decentralized communities. For these software ecosystems and their many end-users it does not always guarantee contributions where needed. This paper maps the influential factors that determine developer participation with a case study on the open source software Debian website. During the data gathering part, event data is obtained that can potentially influence developer participation. Later on, data is mined from the Debian Git repository regarding average commits per developer and predetermined time span. In the data analysis part, the two previous data results are compared in a correlation study to derive relationships. In 11% of the types of events that were studied, there was a reoccurring relation with in- or decreased developer participation. Ecosystem project leaders can stimulate or prevent these events to prevent missing in action developers and welcome new developers.

Keywords: Software ecosystem, developer involvement, developer activity, developer participation, open source software, free software, Linux, Debian.

1 Introduction

Software development can be divided in two types, i.e., centralized and decentralized. Examples of centralized software developers are Red Hat and Microsoft. An example of a decentralized software project is Debian. Decentralized and open source projects are not necessarily less capable of creating large sized distribution as the centralized projects [1]. However, especially in open source and decentralized software projects, it does not always guarantee contributions where needed. Jansen et al. [2] proposed a study on the orchestration of ecosystems. Jansen et al. define in [2] a software ecosystem as: “a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them”. Messerschmitt and Szyperski [3] describe all the aspects of the software ecosystem in detail. Not uncommonly do decentralized Open Source Software (OSS) projects not provide financial rewards to stimulate participation in development [4]. This understudied type of rewarding exposes decentralized OSS projects to a risk that centralized and paid development projects do not entail. If fewer developers participate in developing OSS, the continuation of the OSS project is at risk. In order to regain Missing In Action (MIA) developers, or to attract new developers, it is
important to know what events influence developer participation. By reusing the positively affecting influential events mapped in this paper, decentralized OSS project orchestrators (such as project leaders) can more effectively increase developer involvement in their projects.

Following up on this background, this paper targets to map the relations between influences and developer involvement in terms of the software ecosystem. By doing so, attempting to answer the following research question:

What factors influence developer participation in the Debian software ecosystem?

The research is divided into the following sub questions:

• What is the total number of developers and average number of commits in each week?
• Which events can potentially influence developer participation?
• What reoccurring relations can be found between events and peaks/valleys in average software commits and total developer participation?

In order to identify these relations, a case study is performed on the decentralized OSS Linux distribution Debian website. In May 2011, Debian had 1643 developers, which were spread out over the world [5, 6].

The participation of developers can be researched by looking at what drives OSS developers in the development process, as described in a research performed by Hars and Ou [7]. This research was not conclusive, because it only looked at personal motivations, instead of outside factors.

Roberts et al. [8] went a step further by describing the relation between motivation, participation and performance (by studying the quality of the submitted code) of OSS developers. In practice, this usually means looking at the commits. According to Goeminne and Mens [9] a commit can be defined as: “atomic changes done on the source code”.

Merely investigating the quality of source code, is not sufficient according to a recent qualitative framework design and study on ecosystem evolution [9]. In order to fully understand the workings of an ecosystem the community needs to be studied as well. For example, this can be done by looking at bug trackers, mailing lists and forums.

The scope is this paper is the Debian ecosystem of the last 11 years (2000-2011).

Based on the acquired information, an analysis can take place to see if there are any reoccurring relations between events and developer participation changes.

Data obtained from the Debian software version control system (Git) will be used to find new relations between events and developer participation. The structure of the paper first details the chosen research approach, after that the results from the three sub questions leading towards the last results chapter, where the main research question is answered. Some discussion is opened to elaborate on uncertainties, newly derived questions and applicability of this research.
2 Research approach

In this research, a case study is performed on the OSS project Debian. Debian is an OSS operating system for desktops and servers. Development of Debian is a collaboration between developers (1600+) working from all over the world. The operating system is based on the Linux kernel and is very popular among users. Debian is used as a base for many other Linux distributions, because of the attitude towards stability and security when releasing software. What makes Debian so special compared to other software projects, is the capability to create an OS including 29,000 packages (a lot from separate smaller projects within the Debian community), while using a decentralized development team.

The methods outlined later in this chapter are aimed towards the use of the Debian Git repository and Debian project website to gather data from the year 2000 to 2011. The approach was chosen to forgo personal interpretations of developers and since it would otherwise only be possible for a select part of the total developer community. The Debian Git repository and Debian website, on the other hand, hold a sizable range of historical data of past developer activity and events.

Data mining in this paper targets the following two parts: Debian Git repository and the Debian website. In order to complete the first part, a Linux machine was set up and two Git commands for each developer project were issued in the Bash terminal:

```
  git clone http://git.debian.org/git/[project]
  git log >> [project].txt
```

Respectively resulting into the download of all Debian projects into one local folder and an extract of all Git commit logging data into a plain text file. The plain text files containing the log data on the developer activity were then converted to a Structured Query Language (SQL) database by using a small Hypertext Preprocessor (PHP) script containing various regular expression matching functions (preg_match) specifically applied for this task. The data was arranged in an aggregated weekly view of the number of commits, developers and average commits per developer in order to perform a meaningful analysis. Two perspectives then became visible: the network total and the average degree of participation (figure 1, 2).

The second part of the data mining consists of looking for events that could potentially influence developer participation. For that the Debian website was used as a source. All the events together with their corresponding category are also stored in the same SQL-database as the data from the first part. After all the data was collected and categorized, the database was queried to find a relation between the categorical events and developer participation. To find this relation, the Pearson linear correlation co-efficient was calculated for each category of events in the range of four weeks before and after the event occurred. This statistic measures the strength and direction of a linear relationship between two variables [10].

Eventually in order to evaluate the results, the running Debian leader (2011) checked this paper, and gave feedback on the accuracy and the scope of the research method and results.
3 Results

3.1 Total number of developers and average number of commits in each week

The Debian Git repository holds key historical information on software revisions. For the purpose of this study, the Git database was downloaded and aggregated into an average developer commit and total number of developer perspective. Time intervals were set at one week, and data was collected from 2000 to 2011. In total, 1,000,000 commits were downloaded and used for this paper.

A clearly noticeable change in mid-2009 in figure 1 and 2 define a continuing decrease in total commits by the Debian developers, which is still ongoing in 2011, when the data was extracted. There is a wide variety of events that have occurred before and during the time span 2007-2011. It is hard to locate definite causal factors, since there is only one such event and there are many potential influences.

A more definite answer can, however, be given for multiple identical events of the same category which are dispersed over 10 years. Such studies can, to a lesser degree, be influenced by de- or increasing trends in the worldwide Debian developer community. This is exactly what this study is focussed on.

3.2 Potentially influential events

The Debian project is transparent in its information sharing regarding internal and external events, making it a very suitable matter for study. The study was aimed to recognize and gather events that could potentially influence the level of activity of the Debian ecosystem. Each event was stored together with the corresponding date, category and week number. In total 116 were deemed feasible. These were extracted from the following Debian website sources:

- Mailing lists (lists.debian.org), announcements relevant for developers such as frozen releases. General announcements regarding the Debian project and community spin-offs that take form as new projects. Furthermore, a large effort is
put into matching enthusiastic and capable developers with projects that were abandoned. Such events can influence the activity of developers. They can for example be the root-cause for new developers to sign up or contribute in regaining activity to existing projects;

- **Bug tracking system** (bugs.debian.org), some release-critical bugs might spur the developer community to become more active. Especially when a general release is around the corner. A bug report is published in the mailing list that shows the current number of critical bugs, essentially depicting the degree of preparedness for a new general release. Such events might play a vital contribution to participation;

- **Quality Assurance (QA) group** (qa.debian.org), software quality and continuity are aspects which the QA group monitor and steer. Missing developers, dependency issues and mass bugs filings that could have a great impact on the developers are particularly interesting for this study;

- **Wiki** (wiki.debian.org), release notes, pending issues, upgrades and a wide variety of project information, which might lead to some influential events;

- **Event list** (debian.org/events/), Debian hosted multiple events for community groups, for the developer group a Debian conference is hosted. In addition a work camp is organized to stimulate cooperation between developers that would not do so otherwise. Furthermore, an open day is held to introduce the Debian project to anyone interested. These activities could help in gaining more members in addition to increasing activity and commitment with current developers;

- **Voting area** (debian.org/vote/), general resolutions have marked important events in the past. Some of which could potentially influence the degree of participation.

The sources previously mentioned, led to the formulation of the following potentially influential categories in which the events were placed:

- **Stable release**, a pending stable release can be influential in a way that it stimulates developers to put in extra efforts, or when the release is over to stay inactive for a while because their projects are not likely to reach the public anytime soon;

- **Change of Leadership**, each period a new leader is elected. This could for example influence developer morale;

- **Community spin-off**, the community provides significant input that had effect on the Debian project. With these events working as a spring board for new developments in the project, it is not unlikely that some have worked in favour of developer participation;

- **Incidents**, some notable incident were found such as a fire that burnt a major operations server;

- **Debian Conference**, increased morale and cooperation that might occur after these events is not unlikely to influence the participation of developers;

- **CeBit**, CeBIT is one of the most important computer exhibitions. New people are introduced to the Debian project which might lead to increased participation;

- **Debian Day**, several speakers talk about current developments. Furthermore, a Debian conference which is open to anyone that is interested in free software;
• **End of support for Debian release**, once a release is out-dated, the support ends, which could potentially influence developer participation;

• **Annual Southern California Linux Expo**, this exposition brings together open source companies, Linux, developers and users. Perhaps converting users to enter the developer community;

• **Awards**, Debian has been nominated and chosen numerous times as best project by magazines and websites;

• **General Resolution (GR)**, some examples of GRs are: electing a new leader, changing developer regulations and many more which can be found at the Debian voting area;

• **Major bug**, major bugs can be for example bugs that hinder vital functionality of the system. Such bugs need to be fixed as soon as possible and might cause increased activity shortly after;

• **Dependency issues**, some events were seem that had influence on hundreds of packages that depended on it. For example, when the developer changed its popular package, a lot of developers that had packages that depended on that package had to change theirs;

• **Release frozen**, a frozen release marks the end of any new package contributions. By doing this, the Debian community can focus on the upcoming stable release, rather than introducing new packages;

• **Introduction of developer service**, making the job of the developer easier, allowing them to do a better job in a shorter time, resulting in increased participation.

### 3.3 Reoccurring relations between events and peaks/valleys

Based on the data in chapter 3.1 and 3.2, in this chapter relationships between influences and peaks/valleys in developer participation will be identified using statistical analysis.

There are some influences that are happening at the same time or very short after on another. This will create noise in the research results because it cannot be determined based on what particular influence the developer participation was affected. Therefore, the influences are divided to fifteen categories. If influences are now taking place short after one another, the combination of the influences in the category will compensate for the potential disturbances that take place. This will lead to the reduction of noise in the research results.

An influence can affect developer participation in the future and the past. For example, there could be a deadline for uploading commits before a conference. But it is also possible that developers will start submitting commits after discussing with other developers at that conference. To find all these relationships, every week, from 4 weeks before, to 4 weeks after the event were investigated for changes. 4 weeks were chosen, because it is the average time between individual events.

As mentioned earlier, the developer participation is split up in two perspectives, i.e., the total amount of developers and average commits per developer. When looking at the amount of developers, it can be determined if a certain influence resulted in more or less active developers. But the shortcoming of only using that measurement is
that it only says something about the community as a whole rather than the degree of participation of individual active developers. To overcome this problem, the measurement of average commits per developer will also be determined.

To determine what effect the influences had on the two perspectives, the *Pearson linear correlation co-efficient* was used [10]. This calculation is suitable for this particular situation, because it can be used to show the strength and direction of a linear relationship between multiple sets of variables. In order to determine the correlation of a point in time prior to the influence, the variables (average commits per developers or amount of developers that submitted a commit) of that point in time and the variables of the time of the influence are used (figure 3). To calculate the correlation of future points in time compared to the time the influence took place, a similar approach is used, but with a varying timeline (figure 4). Because of this, the same correlation number means something different for past and future events.

![Fig. 3. Example of a negative correlation (from past to present)](image1)

![Fig. 4. Example of a negative correlation (from present to future)](image2)

The results of applying the correlation to all the categories of influences can be found in table 1. To interpret these results, some sort of scale is required. The scale proposed by D.J. Rumsey, PhD [10] is used in this research. The scale is as follows:

- **Exactly -1**: A perfect downhill (negative) linear relationship;
- **-0.70**: A strong downhill (negative) linear relationship;
- **-0.50**: A moderate downhill (negative) linear relationship;
- **-0.30**: A weak downhill (negative) linear relationship;
- **0**: Linear relationship;
- **+0.30**: A weak uphill (positive) linear relationship;
- **+0.50**: A moderate uphill (positive) relationship;
- **+0.70**: A strong uphill (positive) relationship;
- **Exactly +1**: A perfect uphill (positive) linear relationship.

The result tables show the categories per week where the correlation was either positive (+0.30 or higher) or negative (-0.30 or lower). Because the developer participation is split up in two groups, there are also two result tables. Table 2 shows the correlation of the average commits per developer in a particular week. Table 3 depicts the correlation of developers that submitted commits.
## Table 1. Correlation table

<table>
<thead>
<tr>
<th>Categories</th>
<th>Amount of influences</th>
<th>Linear Correlation Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 weeks before influence</td>
<td>3 weeks before influence</td>
</tr>
<tr>
<td></td>
<td>Average commits per developer</td>
<td>Amount of developers that submitted a commit</td>
</tr>
<tr>
<td>Annual Southern California Linux Expo</td>
<td>-0.286</td>
<td>0.137</td>
</tr>
<tr>
<td>Awards</td>
<td>0.144</td>
<td>0.095</td>
</tr>
<tr>
<td>CeBIT</td>
<td>-0.290</td>
<td>0.044</td>
</tr>
<tr>
<td>Change of leadership</td>
<td>0.083</td>
<td>-0.039</td>
</tr>
<tr>
<td>Community spin-off</td>
<td>-0.181</td>
<td>-0.028</td>
</tr>
<tr>
<td>Debian Conference</td>
<td>0.042</td>
<td>-0.015</td>
</tr>
<tr>
<td>Debian Day</td>
<td>-0.233</td>
<td>-0.136</td>
</tr>
<tr>
<td>Dependency issues</td>
<td>-0.092</td>
<td>0.055</td>
</tr>
<tr>
<td>End of support for Debian release</td>
<td>0.169</td>
<td>0.037</td>
</tr>
<tr>
<td>General Resolution</td>
<td>0.064</td>
<td>0.107</td>
</tr>
<tr>
<td>Incidents</td>
<td>0.353</td>
<td>0.094</td>
</tr>
<tr>
<td>Introduction of developer service</td>
<td>-0.287</td>
<td>-0.484</td>
</tr>
<tr>
<td>Major bugs</td>
<td>0.222</td>
<td>-0.019</td>
</tr>
<tr>
<td>Release frozen</td>
<td>0.320</td>
<td>-0.101</td>
</tr>
<tr>
<td>Stable release</td>
<td>-0.186</td>
<td>0.001</td>
</tr>
</tbody>
</table>

: Correlations with a weak, moderate or strong positive or negative relationship
Based on these tables, the following results can be derived from the correlation of the average commits per developer in a particular week:

- The categories Annual Southern California Linux Expo, awards, change of leadership, community spin-off, Debian conference, end of support for Debian release have no effect on the activity of developers;
- 2-3 weeks before and 4 weeks after CeBIT started there was an increase in developer activity;
- 2 weeks prior to Debian day, there is a decrease in developer activity. After Debian day there is a decrease in developer activity with the lowest point 2 weeks after Debian day. After these 2 weeks the developer activity significantly rises again with the highest point after the event at 4 weeks;
- 1 week prior to dependency issues there is an increase in developer activity, 2 weeks after there is an increase, followed by a decrease in the third week;
- 2-3 weeks before a general resolution there is a decrease in developer activity;
- Incidents have increased developer activity 3 weeks after the incident occurred;
- 2 weeks prior of major bugs, there is a weak increase in developer activity;
- 3-4 weeks before a release is frozen, there is a decrease in developer activity;
- 2 weeks after a stable release, there is a weak increase in developer activity;
- The introduction of developer services leads to a decrease in developer activeness 3 and 1 week before and leads to a decrease in developer activeness in week 1 and 3 after the influence.

### Table 2. Correlation of the average commits per developer in a particular week

<table>
<thead>
<tr>
<th>Time before influence</th>
<th>Category</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 weeks before influence</td>
<td>Incidents</td>
<td>0.353</td>
</tr>
<tr>
<td>3 weeks before influence</td>
<td>Release freeze</td>
<td>0.320</td>
</tr>
<tr>
<td>2 weeks before influence</td>
<td>CeBIT</td>
<td>-0.383</td>
</tr>
<tr>
<td>1 week before influence</td>
<td>General Resolution</td>
<td>0.444</td>
</tr>
<tr>
<td></td>
<td>Introduction of developer service</td>
<td>0.605</td>
</tr>
<tr>
<td></td>
<td>Release freeze</td>
<td>0.310</td>
</tr>
<tr>
<td>1 week after influence</td>
<td>CeBIT</td>
<td>-0.370</td>
</tr>
<tr>
<td></td>
<td>Debian Day</td>
<td>0.414</td>
</tr>
<tr>
<td></td>
<td>General Resolution</td>
<td>0.319</td>
</tr>
<tr>
<td></td>
<td>Incidents</td>
<td>-0.353</td>
</tr>
<tr>
<td></td>
<td>Major bug</td>
<td>-0.362</td>
</tr>
</tbody>
</table>

### Table 3. Correlation of developers that submitted commits

<table>
<thead>
<tr>
<th>Time before influence</th>
<th>Category</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 weeks before influence</td>
<td>Introduction of developer service</td>
<td>-0.405</td>
</tr>
<tr>
<td>1 week before influence</td>
<td>Introduction of developer service</td>
<td>0.369</td>
</tr>
<tr>
<td>2 weeks after influence</td>
<td>Introduction of developer service</td>
<td>-0.301</td>
</tr>
<tr>
<td>3 weeks after influence</td>
<td>Introduction of developer service</td>
<td>0.495</td>
</tr>
</tbody>
</table>
In the joined or hosted events, Debian works on introducing the project to new and unfamiliar developers and users, which could be a probably cause for increased participation. However, during the study of the Debian community it became clear that the new developer approval process tends to take months instead of weeks. It is therefore more likely that groups of developers that were less active, started to become more active, or that already active groups started to become more active.

The conclusions based on the correlation of weekly active developers are as follows:

- The categories Annual Southern California Linux Expo, awards, CeBIT, change of leadership, community spin-off, Debian conference, Debian day, dependency issues, end of support for Debian release, general resolution, incidents, introduction of developer service, major bugs, release frozen and stable release have no effect on the amount of developers that submitted commits;
- Only the introduction of developer services seems to have a reasonable effect on the amount of developers that submit a commit. 4 weeks before the influence, there is an increase in the amount of developers. The first 4 weeks after the event all show a decrease in the amount of developers, with a peak in the first week after the event.

And finally, the conclusions about the entire research in general:

- Influences have more effect on the degree of participation by developers, rather than causing more or less developers to become active;
- In 98.32% of all cases there was no or a weak relation;
- In 10.82% of all cases there was a weak, moderate or strong relation;
- There seems to be no relationship between the degree of participation and the amount of developers that submit a commit.

4 Discussion and Future Work

The use of categories to group influences, resulted in the reduction of noise in the analysis. But even with this reduction, it remains uncertain if there was any noise. The categories with the highest amount of influences have less noise than the categories with few influences. Also, the influences limited to the Debian ecosystem itself were considered, because these influences are more likely to be relevant than external factors. To get even more accurate results, other factors should also be taken into account. Debian was exclusively investigated. The results can be generalized to decentralized OSS ecosystems, however, to further verify the results in this paper an identical study needs to be performed. Other decentralized OSS projects can also use the results of this research, but because the results are not verified with a second case study, the reliability of this research remains uncertain. Furthermore, to find out whether sub groups (e.g., dormant, sporadic or hyperactive) exist within the ecosystem that respond differently to internal and external ecosystem influences,
types of commit frequency and amounts need to be analysed. By doing so, a more
detailed picture can be obtained for supporting software ecosystem orchestration.

Another study can be done on the visitor statistics of the Debian website in relation
to events that happened, such as study is particularly interesting in the Debian project,
because there is a clear website and purpose separation due to the use of subdomains
(e.g., vote.debian.org, qa.debian.org and bugs.debian.org).

This study aimed to map positive and negative influence factors that have been
found in the last 11 years of the Debian project. In the study it became clear that a
general decline in activity was occurring between the years 2009 and 2011 (figure 1
and 2). One could hypothesize that the economic turmoil was the underlying cause for
the decline during 2009-2011. This calls for an additional study that also takes
additional factors into account (e.g., socio-economic factors), because this
observation could not be attributed to one specific event.

5 Conclusions

Developer participation in OSS development communities is very important. The
continuation of the project is at risk without a constant flow of new commits. In order
to maintain developer participation it is important to know what influences affect
developer participation. All the data of the commits for Debian in the year 2000 to
2011 were collected. In total, 1,000,000 commits were collected and analysed. 117
influences were found. These influences were spread out over 15 categories of
influences that could potentially influence the developer participation of the Debian
community. Based on the obtained data, a statistical analysis was performed to find
relationships between the commits and the categories of influences.

In only 10.83% of all measurement points, there was a weak, moderate or strong
relation. It can be concluded that although some influences have a clear change in
developer participation, the effect is not large. The biggest influences proved to be
hosted or joined events (i.e., CeBIT, Debian day), new or frozen releases, incidents,
dependency issues and the introduction of new developer services. Debian and other
OSS projects can use this research to pay more attention to the events that cause a
change in developer participation. By doing this, they can obtain a more streamlined
orchestration of their developer ecosystem.

Acknowledgements

We thank the current Debian leader Stefano Zacchiromi for his comments on our
research results and providing us with new insights into the Debian community. We
also want to thank Slinger Jansen for introducing us to the subject software
ecosystems.
References


What Information Assets Should You Share with Your Software Ecosystem Participants?

Christoph Wagner and Wouter Westendorp

Utrecht University, PO Box 80.089, 3508TB Utrecht, The Netherlands
{c.wagner, w.a.westendorp}@students.uu.nl

Abstract. Gaining software platform leadership is not simply a matter of sharing technology, but requires sharing relevant information with the software ecosystem partners as well. Still, the question remains what kind of information should be shared, or kept secret, to increase the total value of the ecosystem. This paper defines an index of typical information assets brought into existence by software producing organizations that is used to study the information sharing characteristics of three different software ecosystems. Although the actual information sharing strategy strongly depends on the business model of the platform supplier, this index provides the basic foundation to evaluate software ecosystems as a researcher or aspirant platform leader.

Keywords: software ecosystems, software platforms, information sharing, knowledge sharing

1 Introduction

In the modern software development industry, software vendors do not operate as isolated units with a traditional supplier-customer value chain. Instead, software vendors are forced to operate as part of a tightly integrated network: the software ecosystem (SECO). Jansen, Brinkkemper, Luinenberg and Souer define a software ecosystem as: “a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them” [1]. A SECO is often based on technology specific for the SECO: the software platform. Jansen defines a software platform as: "a collection of software artifacts that form a coherent whole on which applications can be built" [1]. The central actor within the software ecosystem is the software ecosystem owner, or platform supplier. A platform supplier is "the organization that maintains and supports the platform and that actively stimulates the use and adoption of it" [2]. To maintain power within the ecosystem, the platform supplier should gain platform leadership. Gawer and Cusumano describe the concept of platform leadership as “the common objective to drive innovation in the ecosystem”. Since the value of a platform depends on the availability of complimentary products, it is in the interest of the platform leader to stimulate and channel innovation on complements [3]. Therefore, the emergence of a platform leader is not only due to the software, in the form of binary executables or plain source code, but also depends on the accessibility of information about the software,
the platform leader, and the software ecosystem for the ecosystem participants. A common example of a SECO is Microsoft Windows, that is based on the Windows operating system, but entails a large network of professional software suppliers, vendors, integrators and even hardware suppliers who work together and thereby add value to the entire Windows software ecosystem. In this example, the Windows operating system is the platform and Microsoft is the platform supplier.

Different platform suppliers have different policies with regard to the disclosure of platform related information. The characteristics of these policies determine the “openness” of the platform provider. Jansen et al. define openness as “the degree to which an organization has opened up its business processes within all its business domains” [1]. For example, the Django Software Foundation that steers the development of the open source Django web-development framework, has chosen to make all documentation, road-maps and bug-reports publicly available. It thereby should gain a high score in the degree of openness. However, within the Apple iOS ecosystem, the official iOS documentation is available for registered developers only and bug information is not accessible to them at all. These aspects will rank Apple lower within the degree of openness.

For software producing organizations (SPOs), making available road-map details is often subject to strategic decisions. For example: Disclosing future plans for a software product might influence the competitors behavior and thereby reduce competitive advantage. On the other hand, it might be worthwhile to share details about future product changes with ecosystem partners and allow them to prepare for a “soft landing” of the upcoming features.

For example, Figure 1 shows part of the road-map document of the Django Software Foundation [4]. It lists the must-have, maybe and rejected feature requests. It also contains the status of the requests, the release schedule and some information about the release process.

**Scaffolding platform leadership aspirations**

For future platform leaders, it might be an interesting question whether or not to disclose API documentation, publish the road-map details or open up the usage data. It is interesting how existing platform owners approach these questions.

Jansen et al. already show that opening the source code is completely dependent of the business model [1]. However, there is still little understanding how SECO owners disclose other relevant information. This boils down to the following research question: What information assets should a platform owner share with its software ecosystem participants?

To answer the research question, a list of typical information assets of platform providers is needed. During our preliminary research, it appeared that such an index does not yet exist. As a result, a complete list of essential and relevant information
assets that may be shared within a SECO is investigated. This leads us to the following sub research question: What information assets does a platform owner produce? To scope this research, information assets are defined as any kind of informative artifact that is the outcome of an strategic, tactical or operational business process of a software producing organization.

**Django 1.1 Roadmap**

Fig. 1. Example Roadmap document of Django 1.1.

This paper is structured as follows: after the introduction, the paper elaborates on the research approach. The related literature chapter follows the research approach.
where the current body of knowledge related to information sharing is explored. The fourth section presents the results of the study. Data is analysed the in the subsequent section. The paper finished with a discussion and conclusion of the research.

2 Research Approach

The first deliverable of the study is an index of typical information assets that can be used in the remainder of this research and subsequent research to investigate the effects of information sharing within software ecosystems.

Defining a list of information assets

The foundation of the information asset index is based on the overview of the Open Software Enterprise (OSE) model by Jansen et al. [1]. This model lists a number of factors at the strategic, tactical and operational level of a software producing organization that influence the degree of openness within the following business domains: Governance, Research and Development, Software Product Management, Marketing and Sales, and Consulting and Support Services. For each factor, the outgoing information asset is extracted. For example, Jansen et al. describe the factor “make ecosystem explicit”. From that, the “Ecosystem Model” information asset is derived. Within the index, we will maintain the business level and business domain associated with the asset to maximize model compatibility.

The case studies

Given the list of information assets, three software ecosystems are observed as part of the case studies: Apple (in general), Google Android, and Django. These ecosystems have been selected based on personal interest of the authors and their familiarity with those. Also, it has been taken into account that the general reader is familiar with those ecosystems in a certain way as well. Apple and Google have both gained platform leadership in recent years. While the revenue growth of Google is flattening after a few years, Apple is considered one of the most successful companies of the last decade [5]. The Django ecosystem is kind of a stranger in the midst, but we will elaborate on that later. For now it is important to know that Django is an open source web-application framework written in the Python programming language.

For each SECO within the case study, the platform owners website is searched for the identified information assets. This search will be conducted by browsing the websites with the default navigation and site-maps. Furthermore the sites’ built-in search engine is used if it is not possible to locate the asset by normal click-through browsing.

Furthermore, to find how the available information is divided over different areas, the number of public web pages available is counted. Google and its special site:
operator is used for that purpose. For example, to count the number of public pages on
the Apple homepage, query Google with ‘copyright site:www.apple.com’. The results
page then shows the number of matching pages found. It is thereby assumed that each
document contains a copyright notice, often found at the bottom of the web-page

3 Related Literature

Current literature shows that during the past century, most companies organized
their innovation processes and collaborations within their own company (e.g. R&D
department, market research etc.). The general idea was to protect the companies’
knowledge, like blueprints or product designs, from being copied or stolen by others.
Suppliers and buyers were not involved in any kind of product development or
improvement [6].

At the end of the past century, companies and academics discovered real value in
corporate partnering and reliance on external collaboration [6]. Moving towards a
view of the own company as part of an ecosystem, rather than a stand-alone
compound within its market defending its position against anybody engaging them.
The ecosystem view is strongly supported by an awareness of collaboration and
sharing of information [7].

Toyota might be the first company that used their ecosystem extensively and
earned a competitive advantage from their tight engagements [9]. Dyer and Nobeoka
observed that the special partner relationships that Toyota had developed, had a great
impact on business with these partners [6]. The researchers conclude that the network
knowledge-sharing activities resulted in a strong advantage of Toyota’s competitors.
As for General Motors, an American car manufacturer, they proved to be much less
efficient than Toyota just because of Toyota’s information sharing strategy.

Further research in this area showed not only that information and knowledge
sharing results in a major competitive advantage [6]. Similar to the software and IT
industry, the bio-technical industry is also a field with rapid technological
developments [9]. Powell et al. showed in their research, that sharing of information
and knowledge also impacts the innovation process. They conclude that significant
amounts of innovations are created in “networks of inter-organizational relationships
that sustain a fluid and evolving community” [9].

Due to the increase in the use of information systems and technology, innovation can
be facilitated in our days. It goes much faster than back in the days, as information is
accessible from anywhere and anytime. Especially companies concerned with the
Internet business-wise, show significant growth because it is easy to access a lot of
partnerships via the Internet and even build software ecosystems that only exist
virtually. Also new business models have emerged, like Software-as-a-Service
(SaaS), which is basically a software ecosystem. Other examples for software
ecosystems that have a high level of virtuality are Android and iOS.
As indicated, an ecosystem requires involvement of different partners, which means a company has to disclose information to specific partners. One would want that, but one would also want to keep specific information secret, as competitors might be interested in that information. Depending on the companies’ role within the ecosystem, a company wants to, or does not want to, disclose information.

Iansiti and Levien (2007) identify two major roles within software ecosystems: the dominator with few collaborations, and the keystone who is highly collaborative. A dominator basically rules his ecosystem. At some time in the future, the dominator will have absorbed the ecosystem he reigned in. Collaborations are limited to just a few. Innovation is being made by the dominator and even if another ecosystem participant proposes a new product or adds it directly to the ecosystem, the dominator typically will attack that participants market position by selling a similar one on its own, or by integrating it into the platform the ecosystem is about. A keystone player also rules his ecosystem, but he is a highly cooperative entity within the ecosystem. Innovations within the ecosystem made by participants are actively supported by the keystone organization and encouraged. The keystone views the ecosystem as a large network that works together on a product and where every contribution is positive and strengthens the ecosystem and the platform itself [7].

4 The Index of Information Assets

The following table shows the identified information assets per business domain and their appearance within the SECO’s of our case studies. We have first observed Google as a Keystone player in the mobile market space. Then we observe Apple as a Dominator that is strengthening its control within the entire ecosystem of hardware devices, the software application development and distribution, and media consumption of end-users. Finally there is also a look at the Django Software Foundation, a relatively small and open software development niche player to further broaden our horizon.

Table 1. List of information assets and their public availability.

<table>
<thead>
<tr>
<th>Business Domain</th>
<th>Level</th>
<th>Information Asset</th>
<th>Google</th>
<th>Apple</th>
<th>Django</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
<td>S</td>
<td>Governance Policy</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td>Process Descriptions</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Product Lifecycle Plans</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Acquisition Policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>Knowledge Management Strategy</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Partner Development Process</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Partner Governance Process</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Ecosystem Model</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Partner Directory</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Customer Directory</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Business Domain</td>
<td>Level</td>
<td>Information Asset</td>
<td>Google</td>
<td>Apple</td>
<td>Django</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------</td>
<td>---------------------------------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>User Feedback</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Reusable License</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R &amp; D</td>
<td>S</td>
<td>Technology Roadmap</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Research Vision</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Development Procedure</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>API Standard</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Source Code</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Non-core Innovations</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Inter-product data</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Source Code Reuse Policy</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Quality Ratings</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Bug Tracking System</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Testing Process</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Functional API’s</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Content API’s</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Test Results</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Candidate Releases</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Developer Training</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Performance Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Crash Reports</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPM</td>
<td>S</td>
<td>Platform Strategy and Vision</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>SPM Process and Lifecycle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Product Roadmap</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>IP Policy</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Requirements Engineering</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Process Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Release Planning</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M &amp; S</td>
<td>S</td>
<td>Market Vision and Expectations</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Business Model</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>Partner Model</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Market Information</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Certifications</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Current and Potential Customer Information of</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Partners</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Incentives</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSS</td>
<td>S</td>
<td>CSS Knowledge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>CSS Strategy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Project Process Knowledge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>Quality Measures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>CSS Project Details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>Consultant Training</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5 The Case Studies

First call, Google

Google Android is referenced to as an open and innovative mobile operating system. Since 2009, it achieved huge market share increases and is at the time the most used mobile operating system. Since Google approaches an open thinking, they were experienced in the area of information sharing even before Android. Google established many directories and sub-domains for the Android project, to empower a large ecosystem with a broad variety of information and possibilities for innovation. Google fosters third-party innovation by providing major information on Android.

![Development Workflow at Google](image-url)

Fig. 2. Development Workflow at Google.

The main hub for every target group is www.android.com. The page provides links to overviews for the media, developers and partners. It also directs to jobs and general
high-level information of Android. Furthermore, there are two major sites tailored towards the developer group. The first one (developer.android.com) is specifically made for app developers and beginners. The second one (source.android.com) is made for core developers and generally more technical oriented developers.

developer.android.com provides exhaustive information about the whole Android universe. The site offers the source code and the general reference. It also contains a developers guide, which is very large and consists of various development topics and best practices for beginners and experts. Also, up-to-date videos and blogs are provided, which exclusively deal with the newest developments in the Android universe. Google also shares a bug-tracking system, different communication channels (IRC, forums) and licensing and security information on this page.

source.android.com is generally used by more advanced developers like mod-developers. The page gives additional insight in fundamental Android technologies like Dalvik and encryption mechanisms. Google does an outstanding job when it comes to sharing the patch and bug process descriptions and life cycles. Developers can view these processes and get an exact insight of what will happen in case a bug or patch is submitted. Google also tries to create a bound to the developers by stating the mission, vision and overall philosophy of their Android project.

### Table 2. Division of Google Android information over sub-domains.

<table>
<thead>
<tr>
<th>Sub-domain</th>
<th>Target Audience</th>
<th>Number of public pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.android.com">www.android.com</a></td>
<td>General Public</td>
<td>3,090</td>
</tr>
<tr>
<td>developer.android.com</td>
<td>App Developers</td>
<td>1,130</td>
</tr>
<tr>
<td>source.android.com</td>
<td>Core developers</td>
<td>10</td>
</tr>
<tr>
<td>code.google.com/android</td>
<td>Core developers</td>
<td>16,900</td>
</tr>
</tbody>
</table>

Generally, large amounts of information can be found on the Android websites. Google maintains two specific pages with a lot of information on their system. They also provide very detailed guidelines and manuals, with regard to operational and even more strategic topics. More strategic and innovation related information is also provided via some blogs. Google puts information in specific centered places and uses only a few subdomains for all the information. This enables the user to easily navigate through all the information he eventually needs without leaving the information resource.

### A glimpse through the eyes of Apple

During the research of the Apple website, insight into the perspective of Apple to its multi-sided market have been found. While navigating the public parts of the Apple website, it has been found that the information is divided over a small number of sub-domains. Apple is notorious for limitations, apparently even if it comes to sub-domains. What is interesting is that each of those domains could almost directly be
mapped to a target group within the ecosystem. The following table lists these sub-domains and the amount of pages on that part.

**Table 3.** Division of Apple information over sub-domains.

<table>
<thead>
<tr>
<th>Sub-domain</th>
<th>Target Audience</th>
<th>Number of public pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.apple.com">www.apple.com</a></td>
<td>Potential Buyers</td>
<td>72,000</td>
</tr>
<tr>
<td>store.apple.com</td>
<td>Actual Buyers</td>
<td>32,000</td>
</tr>
<tr>
<td>discussions.apple.com</td>
<td>Current and Potential Customers</td>
<td>3,130,000</td>
</tr>
<tr>
<td>support.apple.com</td>
<td></td>
<td>96,900</td>
</tr>
<tr>
<td>training.apple.com</td>
<td>Current Customers</td>
<td>636</td>
</tr>
<tr>
<td>consultants.apple.com</td>
<td>Consultants</td>
<td>288</td>
</tr>
<tr>
<td>channelprograms.apple.com</td>
<td>Resell Partners</td>
<td>13</td>
</tr>
<tr>
<td>investor.apple.com</td>
<td>Investors</td>
<td>335</td>
</tr>
<tr>
<td>developer.apple.com</td>
<td>Developers</td>
<td>37,300</td>
</tr>
<tr>
<td>opensourse.apple.com</td>
<td>Open-source Developers</td>
<td>240,000</td>
</tr>
<tr>
<td>advertising.apple.com</td>
<td>Advertisers</td>
<td>5</td>
</tr>
<tr>
<td>itunes.apples.com</td>
<td>Current Customers</td>
<td>14,600,000</td>
</tr>
</tbody>
</table>

First, and most prominent, is the primary part of the website (www.apple.com), that is almost entirely concerned with providing product information, thereby targeted at potential buyers. On this part, almost each product is accompanied with a blue ‘Buy now’ button that takes the consumer to the web store (store.apple.com) that is optimized for buying. For example, it offers the localized pricing information and allows the customer to customize the product and buy accessories. This web-store thereby not only allows for up-sell, but more importantly, it is the direct-sales channel of Apple. This distinction also explains how Apple converts potential buyers to actual buyers on-line.

A little bit hidden behind product support buttons is the ‘discussion’ forum at discussions.apple.com. This part of the website is the place where current customers have the ability to get support from Apple employees or other Apple customers and the contents of this part are largely generated by those customers. While a lot of questions or complaints are addressed directly to Apple, the number of responses from Apple employees seems to be quite low. This part of the website is an interesting candidate for further research, but the impression of this part of the website is that it at least allows Apple customers to directly respond to Apple, and provides them an official place to make their voice heard. Apple has a process in place to take this feedback into account, but being not very supportive into solving individual problem cases, and in a certain way we understand: how would you support all your individual customers on this scale?

Then, hidden between the web-page bottom links, are the investor.apple.com and the consultants.apple.com and training.apple.com websites. While the former speaks for itself, and apparently is an important target group, the latter offer information about courses and certification for professionals. Then, there is also the
opensource.apple.com part, where Apple pays its open-source license obligations, and more importantly, the developer.apple.com part. This developer website contains some public information about developer programs, but requires formal registration and a partnership fee to gain access to the enormous amounts of technical software information. Finally, Apple appears to be only exploring advertising as a new market given the small amount of information over there. What is an important market given the number of available pages, is the iTunes platform that is concerned with the distribution of music and video of external media providers to Apple customers.

Comparing apples with oranges

Our final pick is concerned with the Django Software Foundation. Django is a web-application development framework written in Python and distributed under the open source BSD license. Strong points in favor of other frameworks are, amongst others, the sophisticated Object-Relational-Mapping (ORM) layer that abstracts entity data and behavior from the relational database characteristics and the flexible Model-View-Controller (MVC) implementation that allows for user and search engine friendly URLs with a modular code lay-out. Given the nature of the product, the primary users of Django are web-application developers. The contents of the Django website are therefore quite technical in nature. What makes Django easily accessible for developers is that the foundation embraces the mantra of “docs, or it doesn’t exist”. Contributors are therefore obliged to first document new features before they will be merged into the code-base. The result is a well written and up-to-date documentation part of the website (docs.djangoproject.com) that covers all major and minor public features. Only if the developer needs to touch the lower-level API’s or feels the need to investigate Django’s inner workings, he or she should dive into the source code. Part of the official documentation is the “Contributing to Django” chapter. This chapter explains the whole development policy and release procedure which are thereby formalized.

As with most open source projects, Django users are encouraged to contribute to the Django framework by issuing bug-reports and feature requests or suggest fixes with code patches. The code.djangoproject.com sub-domain is designed for this purpose. Compared with the ‘read-only’ documentation part, this part is designed for both reading and writing. Information that should not be put under the source code documentation may be contributed through the wiki. For general discussions or support questions, official mailing-lists and an IRC channel are in place. Users are encouraged to consult this resources first before formalizing the request with a ticket to mitigate ticket “pollution”. This application of the wiki and mailing-lists allows anybody to share their saying with the Django developer community and constitutes the open character of Django. Finally, the source code is also made accessible for web-browsing over here.

The smaller part of the website is the primary (www.djangoproject.com) part. This part covers the homepage that states Django’s unique selling points, possibly of
interest to managers or developer team-leads. Furthermore it holds the formal weblog where the framework developers share their word with the Django community.

What is not on the official website is a formal link to 3rd party ‘apps’, the Django term for custom modules. It appears that the Django Software Foundation is not yet concerned with providing an official way to browse and search this kind of contributions. The focus is on the platform code-base and its documentation, while the management and maintenance of the ‘external’ part of the platform is left to the ecosystem.

The following table shows the distribution of information over the official Django project domain. It shows that a large part of the official information is product documentation. The ‘code’ sub-domain holds a single page for each source code file and each issued ticket which counts towards the large amount of pages on this sub-domain.

Table 4. Division of Django related information per sub-domains.

<table>
<thead>
<tr>
<th>Sub-domain</th>
<th>Target Audience</th>
<th>Number of public pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.djangoproject.com">www.djangoproject.com</a></td>
<td>General Public</td>
<td>387</td>
</tr>
<tr>
<td>docs.djangoproject.com</td>
<td>Developers</td>
<td>773</td>
</tr>
<tr>
<td>code.djangoproject.com</td>
<td>Contributors</td>
<td>17,600</td>
</tr>
</tbody>
</table>

7 Discussion

The research is mainly based on the paper of Jansen [2], as the paper elicits information assets in a sound way. Additionally we encountered some more specific assets within the objected software ecosystems and added them. However, the list might not be complete for every company in the diverse software sector, as every company owns specific internal assets that nobody from the outside world will ever be able to access or even know of.

It is also known that this research is very limited. Case studies comparing the same business models are encouraged as this would help to reveal even more typical sharing behaviours in software ecosystems. Since the motivation behind the choices for Apple in general, Android as a product and Django as a platform is to show distinctive ecosystems, and how information sharing is handled within different ecosystems, a comparison between the picks is only possible to a certain extent.
6 Conclusion

A decent information sharing strategy is essential to gain and maintain platform leadership. The development and execution of this information sharing strategy largely depends on the business model of the platform supplier and the characteristics of the software ecosystem.

This study has contributed to this field by defining an initial index of information assets produced by software producing organizations. The index is derived from, and compatible with the Open Software Enterprise model of Jansen et al. [1]. Where the OSE model focuses on “openness” factors, this paper has focused on the availability of the resulting artifacts in three different software ecosystems. We believe this index is a valuable tool to further research software ecosystems in future research.

During the research, we have found that most platform suppliers organize their information by target group. By reviewing the platform suppliers’ websites, we were able to get a view on how the platform suppliers organized their ecosystem, even in absence of a formal ecosystem model.

Furthermore, our limited research shows that the niche player shares the most information with its surroundings. Because we did not investigate any other niche players, our conclusion that niche players have to share in order to survive, cannot be drawn as a general assumption. Further research in the area of niche players will have to be done to confirm this hypothesis. Though it is valid as a first assumption in this field. The niche player has to distinguish himself in some way from the big players, this does not necessarily mean that only the product is different, but also the relationships within the companies’ ecosystem might be stronger.

Google also shares large amounts of knowledge via two main portals, but as this is a really huge ecosystem, it might not be possible for Google to care as much about the ecosystem relationships as it is for niche players.

The dominator within our case studies appeared to share the largest amount of information in number of webpages, but in contrast does not share a proportional number of information assets. A reason might be that Apple is not strictly a software vendor. Given that the operating system updates of Apple are sold for a very low price, it shows that this market is not the primary money maker. Instead, Apple makes a lot of money with their popular and high profit margin hardware devices. Then, Apple raises their 30% tax in the very large consumer app and media market. Within this context, the software platform provided by Apple is only the glue that integrates these markets. Our analysis shows that this business model is reflected by their information sharing strategy.
References


2. Jansen S. When the Foundations have already been laid: four case studies of extendible software platform architecture.


## Author Index

<table>
<thead>
<tr>
<th>Author</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amov, Metodi</td>
<td>59</td>
</tr>
<tr>
<td>Blijleven, Vincent</td>
<td>85</td>
</tr>
<tr>
<td>Cancescu, Lucian</td>
<td>1</td>
</tr>
<tr>
<td>Filchev, Stamen</td>
<td>16</td>
</tr>
<tr>
<td>Idu, Andrei</td>
<td>31</td>
</tr>
<tr>
<td>Koseva, Ivelina</td>
<td>16</td>
</tr>
<tr>
<td>Mijnhardt, Frederik</td>
<td>44</td>
</tr>
<tr>
<td>Pietzka, Katharina</td>
<td>59</td>
</tr>
<tr>
<td>Pors, Michiel</td>
<td>70</td>
</tr>
<tr>
<td>Seele, Wilbert</td>
<td>111</td>
</tr>
<tr>
<td>Van Angeren, Joey</td>
<td>85</td>
</tr>
<tr>
<td>Van Bommel, Rick</td>
<td>126</td>
</tr>
<tr>
<td>Van De Zande, Tommy</td>
<td>31</td>
</tr>
<tr>
<td>Van Den Berg, Niels</td>
<td>44</td>
</tr>
<tr>
<td>Van Ingen, Kevin</td>
<td>100</td>
</tr>
<tr>
<td>Van Ommen, Job</td>
<td>100</td>
</tr>
<tr>
<td>Van Ramshorst, Erik</td>
<td>1</td>
</tr>
<tr>
<td>Van Stijn, Peter</td>
<td>70</td>
</tr>
<tr>
<td>Van Zetten, Martijn</td>
<td>111</td>
</tr>
<tr>
<td>Ververs, Eric</td>
<td>126</td>
</tr>
</tbody>
</table>
Wagner, Christoph 138
Westendorp, Wouter 138