NTNU – Trondheim
Norwegian University of
Science and Technology

In-Application Payment from Mobile Apps

A study of In-App Payment

Jøran Christiansen Haines
Alexander Hanssen

Master of Science in Informatics
Submission date: May 2013
Supervisor: Torbjørn Skramstad, IDI
Co-supervisor: Lillian Røstad, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

1 Preface

This thesis was written by Alexander Hanssen and Jøran Haines. For the last two years we have been studying Informatics - Systems engineering and human-machine interaction, after previously studying at the University of Nordland together. The master thesis was written as a part of a ongoing collaboration project between IDI at NTNU and Telenor, and it marks the end of our 5 year long studies.

We would like to thank our supervisors, Erik Berg from Telenor and Lillian Røstad.

Finally we would like to thank our families and friends for the support and help throughout our studies and writing our thesis.

Contents

List of Tables

List of Figures

2 Abstract

The main purpose of this thesis is to evaluate and compare existing service providers of in-app payment solutions and their APIs, and then conclude what would be the best choice for hobby developers when deciding on what service provider to use for their application.

We gained hands-on experience by developing a simple Quiz game app that featured the implementation of in-app purchase. We took a look at the major service providers; Apple, Google, PayPal and Windows to get relevant information and evaluate them based on a comparison model that we created. The comparison model consists of a business and a technical aspect, and within of those aspects we chose the most relevant properties that are necessary to give a fair comparison. We conducted a survey among hobby developers - both students and IT-employees - in order to get insight on which properties within in-app payment they considered as the most important ones. With these results we were able to find some indications on which of the service providers are best suited for the hobby developer.

The survey indicated that easy distribution, good documentation and guides along with an easy development process was the most important aspects among our participants. Surprisingly, the importance of potential profit was rather lower than we expected. Based on what we learned from the survey and the differences we observed during the evaluation of the different service providers we concluded that the best suited in-app payment provider for the hobby developer is Google's solution.

3 Sammendrag

Formålet med denne oppgaven er å evaluere og sammenligne eksisterende tjenestetilbydere av in-app betalingsløsninger og deres APIer, og så konkludere med hva som er det beste valget for en hobby-utvikler når han skal velge en riktig tjenestetilbyder for applikasjonen hans.

Ved å selv utvikle en enkel Quiz app som inneholdte en implementasjon av in-app betaling, fikk vi praktisk erfaring rundt emnet. Vi holdte oss til de største tjenestetilbyderne; Apple, Google, PayPal og Windows. Vi innhentet relevant informasjon rundt disse og evaluerte dem basert på en sammenligningsmodell som vi selv definerte. Sammenlingningsmodellen består av en forretningsdel og en teknisk del, og innen disse to valgte vi flere egenskaper som var mest relevant for å gi et godt grunnlag til sammenligningen. Vi utførte en spørreundersøkelse blant hobby-utviklere, både studenter og IT-ansatte, for å få tilbakemelding på hva slags egenskaper innen in-app betaling og utvikling de annså for å være de viktigste. Med resultatene fra undersøkelsen var vi i stand til å få noen pekepinner på hva slags tjenestetilbydere som var best egnet for hobby-utvikleren.

Spørreundersøkelsen indikerte at en enkel distribusjon, god dokumentasjon og guider, sammen med en enkel utviklings prosess var noen av de mest viktigste egenskapene i følge våre deltagere. Vi fant ut at ønsket om en størst mulig profitt som følge av salget av applikasjon var rangert lavere enn hva vi hadde forventet. Basert på hva vi lærte fra spørreundersøkelsen og de forskjellene vi observerte under evalueringen av de forskjellige tjenestetilbyderne, konkluderte vi med at den best egnet in-app betalingsløsningen for en hobby-utvikler er Google's løsning.

4 Introduction

4.1 Background

Direct communication has grown in importance to all mankind over time. The need for a timely and precise transfer of information has been well exemplified in the conveyance of business transactions. In a historical perspective, the means by which we make these connections has undergone dramatic changes. Most importantly, the implementation of modern technology has allowed us to communicate on a 24-hour, global scale with the possibility to do this while being intransit. From the initial use of postal or telegraph services to the utilization of the telephone in the early twentieth century, we are now offered a more powerful channel for this exchange through the introduction of today's mobile counterpart, the cell phone. In 2004 there existed an estimated 1.752 billion cell phone subscriptions worldwide[3]. Additionally, the world had nearly as many cell phone subscriptions as there were humans on the planet according to a U.N. Telecom agency report from 2012[4]. Earlier versions of the cell phone are commonly referred to as a feature phone, a device with no touch-screen or QWERTY keypad, which operates without an advanced operating system. Its primary functions include voice and text messaging in addition to simple web browsing and e-mail. Today's modern cell phone is labelled a smartphone and it offers a wider variety of advance features. It has a mobile operating system and greater computational capabilities than the feature phone.

Smartphones correspond to personal computers and their ability to run applications opens a vast market for developers. Smart-phone sales are steadily increasing, with 210 million units sold during the first quarter of 2013 – up 42.9% from the first quarter of 2012 [5]. According to a report by Nielsen published in early 2013, more than half of the mobile device users in Australia, China, South Korea, USA, and the UK are smart-phone users[6]. As smart-phones are becoming less expensive, the market is growing in Asia-Pacific and other developing countries as well[7].

The previously mentioned market potential offered to developers lies within the growing popularity of mobile applications, commonly called an **"app"**. A mobile app is an operable program and its purpose may focus on interests such as gaming and general utilities relating to literature, music, food, travel, health, business, finance, weather and news. The potential appears to be unlimited.

Developers may create apps and channel sales and distribution through providers such as Apple, Google, and Microsoft Windows at a fee. To get a sense of the popularity today, according to Apple's press release from early

2013, it was announced the the App Store has achieved over 500 million active accounts and the number of apps exceeding 775,000 for its iPhone, iPad, and iPad touch. Customers had downloaded over 40 billion apps and developers had been paid over seven billion dollars by Apple[8].

Sales of mobile apps is a steadily growing phenomenon and there exists a potential market for both large companies and hobby developers to tap into. We define a **hobby developer** as a person with varying experience in mobile development who work in his or hers own spare time as an independent developer. They may have only recently begun with mobile development or have several years of experience and are often driven by personal motives such as profit, enjoyment, or individual interests.

Should you wish to distribute an app, there are several choices that you may make in regards to a business model for your app. When we discuss business models for apps, we refer to how you wish to generate income from your offer. More specifically, when we mention **"business model"**, we are speaking of the business models the Apple, Google, PayPal, and Windows support. These business models include payment for each application, advertisement, in-application payment, and free application. Our focus in this paper will be in-application payment.

Paying for additional content instead of buying the application itself is an interesting business model. This business model is often referred to as a **freemium model** [9, p.89-98], a business model where the product itself is free but extra feature or functionality requires an additional fee. The way this is introduced in mobile development is by in-application purchases, where extra virtual features are offered within the application which the customers may pay for to enhance their experience with the application.

The trend of DLC(Downloadable Content) in computer gaming where the creators may charge money for additional content has also made it's way into the mobile platform. Instead of making money on your application by charging a one time fee to use the application, some make it free and charge for additional content, while some make it free and use advertisements as their way of generating revenue. By switching over to the freemium paradigm the developers wouldn't have to bother with having to develop a free-to-use "lite" version and a "pro" version of the application. Instead, the application will be free to download, and the user himself can decide what he wants purchase in form of additional features or new content after he has tried the application.

This business model has had a huge increase in popularity as users don't have to pay to try the application, and the developers only need to develop a single version of the game. By making the app free to download more users might download it, and hopefully those who purchase additional content

make up for those who don't[10]. Also, the developers can focus on developing new features and content for the application to secure a continuous income without having to develop a completely new application.

The content that is offered is often something that adds to the experience of the game, such as extra levels or new equipment for your character. Some games employ their own digital currency such as virtual gold to purchase extra content, and sell these packs of gold for a related amount of real money. This is where the in-application payment **service providers** step in. They offer a simple way for the developers to implement a payment model which they feel fits best for their own application.

Apple, Google, PayPal and Windows are some of the service providers a developer can use for delivering extra content at a price, but which fits best for the hobby developer? They all offer an in-app payment service with an API and some tools for the developer to use and we will refer to them as service providers in this thesis.

The people that develop apps on a hobby basis might not have the same knowledge when it comes to economic aspects, unlike large companies. It might be difficult for them to choose choose the ideal payment method that suits the application, but we hope to give some insight and advice which can help the hobby developer.

4.2 Motivation

As students and as tech-geeks, apps is something that has fascinated us for quite some years. Since developing our own simple application during our bachelor thesis, and just being interested in apps in general, we knew that we would like to do something similar for our master. We wish to explore this topic because it is fresh and increasing in popularity[11]. Another point is that we found little relevant reference material on the topic, so there does not seem to be much research published on the topic of in-app payment. In-application payment is currently the new hot business model when it comes to the application market. We wish to find out the pros and cons of using this method compared to paying for the application once. That is why we wish to seek out what the developer that makes apps in his or hers spare time value when it comes to this subject.

This thesis is written for Telenor who also wishes to know what would be the best choice when it comes to the selection of an in-application payment solution.

4.3 Research questions

The main question for this paper is to determine which in-application solution would be the best choice for a hobby/private developer. First we want identify what providers and solutions are out there. Then when looking at the different solutions we will evaluate them based on a comparison model. The questions we want to answer here are what technology they support. What kind of business models do they support? Here we will look at types of payment, business constraints etc. Then we will look at the development process. Are there any requirements to use them? Do you have register an account or pay for it? How are they to test? Do they have any developer support or community? Security is also an important aspect when dealing with money transactions. Therefore we will also look at what security features are supported and how they are to configure. To evaluate the solutions we will need to identify the requirements and expectations of the developers. When we've identified these we can create a set of criteria to evaluate a solution. These criteria will be used to create a model to evaluate a payment solution. Then we can look at how the solutions compare to each other. Then, when we have compared the solutions, we want to answer what would be the best choice based on our case.

To summarize we have the following questions we want to answer:

1. What would be the best choice of in-app payment solution for a private or hobby developer?
2. What providers/solutions of in-application payment APIs are out there?
   (a) What business models do they support?
3. How's the development process?
   (a) What is required to use the API?
   (b) What technology is supported?
   (c) How is the API to implement(ease of use)?
   (d) How is it to test?
   (e) Is there some kind of developer support or community?
4. What security features are supported and how are they to configure?
5. How does the different solutions compare to each other based on the needs of the developers?
6. What expectations do the developers have to the solutions

(a) What are the most the important features and requirements for the developer?

4.4 Method

The first task is to review the in-app payment solutions that are already available today. The resources we are using to obtain completed research, data and general knowledge are Google Scholar[1], IEEE Xplore[2], and NTNU's University library[3]. Then we will evaluate them one by one before moving to doing a comparison of the in-app payment service providers and their solutions. In order to do this we need to create a comparison model. We will identify the most important properties within the business and technical aspects of an in-app payment service to create our comparison model. Our focus will be how one in-app payment solution differs from the others, from the perspective of the developer. We wish to find out what hobby developers value when choosing a method for billing the customers for additional content, so we will be conducting a survey in form of a questionnaire to identify the most important properties and expectations for the hobby developer. Based on the results of this survey and the results from our comparison with the comparison model we will be able to conclude which of the chosen in-app payment services would be the choice for a hobby developer.

We also want to gain some hands-on experience and insight for ourselves, so we will develop a relatively simple application to test out the various ways we can implement in-app payment by using the various payment services.

[1] http://scholar.google.com
[2] http://ieeexplore.ieee.org/Xplore/home.jsp
[3] http://www.ntnu.no/ub

5 Current in-app payment service providers

Today developers have many different options and solutions to choose from when deciding on which and how to implement in-app purchases. In this thesis we will not be evaluating and comparing them all. Instead we will focus on the more well-known ones which are Apple, Google, PayPal and Windows Phone. In this section we will give you a short introduction to the service providers we have chosen and a short list of what else is out there.

5.1 Apple

Apple Inc. was founded in 1976 by Steve Jobs, Steve Wozniak and Ronald Wayne[12]. Their initial focus early was on computers, such as Apple I and II, Macintosh and later on the iMac, Mac Pro and MacBook Pro. Even though the Macintosh sold well when it was introduced in 1984, they went through a difficult period with low sales and was struggling to get ahead of its competitor, Microsoft[13]. Apple really began to shine when they started focusing on mobile devices such as the portable media player - the iPod - along with its own portal to purchase music for this device, iTunes Store[14]. Later they achieved even greater success with the introduction of several other mobile devices such as the iPhone, iPod Touch and iPad, all of which have done considerably well in the market.

One of the reasons why Apple has become so profitable is that they manufacture their own products and the operative system that their devices use are exclusively theirs. Their products are fashionable because of their unique and distinguishable design, and the devices maintain a favorable ease of use. These are a few of the factors that have granted them a huge following across the globe. The simplicity of their products has been a key to their success. The iPhone was introduced in 2007 and the App Store followed one year later which allowed customers to download applications for their device[15][16]. With the ability to download and purchase applications from this store came the possibility for developers to offer additional in-app content for a premium, known as in-application purchase.

5.2 Google

Android is an operating system for mobile devices and it's currently the most popular mobile operating system. Android was originally developed by Android Inc., with the help of the Open Handset Alliance and Google. In 2005 Android Inc. was bought by Google which continued the development of the OS[17]. Android was first revealed in 2007 and in 2008 the first mobile devices running Android hit the stores. After the release the popularity of Android quickly grew and now in 2013 Android is by far the largest and most popular OS. Recent numbers show that the number of mobile devices with Android sold world wide in 2012 was over 144 000 000 units. This means that an impressive 69.7% of device sold were running Android[18].

Google's app store for Android is called Google Play. This is the official marketplace where developers can sell their applications. If you wish to sell your applications through Google Play you also have to use their APIs and services for handling sales. In March 2011 Google released their API for in-app purchases. Google Play will be used to handle in-app purchases on Android; however we will discuss this later in the report. Today most of the top grossing applications on Google Play are using in-app purchases to generate revenue[4].

In December 2012 Google released their latest in-app billing API, version 3, to the developers. The new version is said to "significantly simplify" implementing in-app purchases in your applications[19]. Through cooperation with mobile operators, Android is also the only operating system which currently support billing over ones mobile subscription. In Norway this was launched by Telenor March 21 2013 as the only mobile operator currently having this payment option [5].

5.3 PayPal

PayPal is one of the largest providers of e-commerce services used on the internet. Its main business is based on handling payments and transactions on the internet, and is widely used in online shops, handling donations, auction sites, etc. PayPal makes money by taking a little fee from every transaction, where the amount of fee is based on the transaction amount, currencies used, national/international transaction and such. In 2012 PayPal had $5.6 billion in revenue and their services are available in most countries[20].

[4] `https://play.google.com/store/apps/collection/topgrossing?hl=en`

[5] `http://www.telenor.com/news-and-media/press-releases/2013/google-and-telenor-norway-make-it-easier-to-buy-apps-and-games-on-google-play/`

PayPal also offer in-app payment APIs to handle transaction in application. They are currently offering several different APIs which can be used depending on what the developers needs. These are available to both Android and iOS. However using these APIs imposes certain restrictions, which we will discuss later.

5.4 Windows

Microsoft was founded in 1975 by Bill Gates and Paul Allen and has created a vast series of operating systems and software products. Windows was first introduced as a graphical extension to MS-DOS which was an early text-input based OS. Several new versions were released from the 80s to present day with the newest being Windows 8. The massive library of software that they have developed and sold over the years has made Microsoft an extremely profitable company and it is the largest software creator when it comes to total sales, profit, assets and market value[21].

Microsoft started their mobile journey in the 1990s by developing operating systems for handheld devices, then called Pocket PCs. In the early year of 2000 the first Pocket PC was released as a pen-based touch screen and the operating system was based on Windows CE 3.0, a trimmed-down and compact OS designed for embedded systems[22]. Several improvements were made shortly thereafter and in 2003 they introduced Window Mobile which was targeted for both pocket PC's and smartphones[23]. This OS was superseded by Windows Phone in 2010[24] and this is the branch of OS that are used on today's Windows-based smartphones. There are two OS's that coexist today, Windows Phone 7 and Windows Phone 8. Windows Phone 7.8 is the newest update and the intention was for older phones to have a longer life span since hardware limitations could not support Windows Phone 8. Windows Phone 8 was made to support high-end hardware devices and is similar to the desktop version of Windows 8, which has the user interface known as Metro - a design which relies on flat colored live tiles and horizontal scrolling to keep interaction clean and simple.

Purchases and downloads on these devices go through Windows official app store, called Windows Phone Store[25].

5.5 Others

There are several other providers that provide their own application store with the ability to add in-app purchases to the apps that are available there. Due to limitations in time and scope in this thesis, we chose not to include the following providers as they do not have the same market size as the biggest providers or may have some other restrictions when it comes to in-app purchase.

The following are divided into two parts, the providers that have their own native OS which they offer content through own application stores and the third-party providers.

Samsung

Samsung[6] is a large South Korean company and is the world's largest information technology company[26]. The newest hand-held devices they offer uses their own version of Android and they have several API's that the developers can use in for the technology limited to Samsung devices. You can make use of their own SDK if you wish to add In-App Purchase, limited to Samsung devices. Samsung Apps is the name of their virtual store where you can download apps to your hand-held device or your Samsung SmartTV.

BlackBerry

The BlackBerry is a brand that sells smartphones, tablets and other services [7]. BlackBerry World is the official application distribution service with approximately 100,000+ apps available today [8].

Amazon

Amazon Appstore[9] was launched by Amazon in March 22. 2011[27]. They currently offer an app store and in-app purchase API for the Android platform. In order to use their app store the developer is required to pay a fee.

[6] http://www.samsung.com
[7] `http://us.blackberry.com/`
[8] `http://appworld.blackberry.com/webstore/?`
[9] http://www.amazon.com

GetJar

GetJar[10] is a third-party service provider. They offer their own payment API and app store. GetJar works quite differently than a lot of the other options in that it uses virtual currency which can also be used across other GetJar applications. We expect to see more of this type of cross-application currencies.

VISA

VISA[11] is also a third-party service provider, but they only have solutions for handling payments by credit cards. In order to use the services VISA offer the developer has to have a registered business and will also have to fill out an application, which means most hobby developer can't use it.

[10] http://www.getjar.com
[11] http://www.visa.com

6 The business aspect

The mobile application market is fairly new and popular platforms such as Apple's App store and Google's Play(earlier called Android Marked) made apps incredibly popular. The App store was released in July 2008, Google Play was released in October 2008 and the revenue that the apps has generated keeps increasing for each year. The business model that the apps have been following have been through some changes when it comes to making the biggest possible revenue.

Some of the different types of business models that the applications use are:

1. **A free application without advertisement.** This is often used for apps that serve educational purposes, such as some online dictionaries, wikipedia or other various apps paid for by either governments or organizations. There can also be apps that are made by companies who already profit by offering their business specialization and want to make their service available on the mobile platform.

2. **Free applications with advertisement.** These applications were very successful and still are to some degree since you can obtain an application for free with only the small annoyance of advertisements taking up a small portion of your screen. This is very popular to use with applications such as games since lots of downloads could mean a great revenue. Some apps are offered as both a free version and a one time fee version.

3. **Free applications with additional content.** This model is also called "Freemium", as in when a product is offered for free but a premium is charged for additional features or digital goods. This model is indeed the topic of our paper and it has grown a lot in popularity in the recent years.

4. **Applications with a one time purchase fee.** A pay-per-download model where you purchase the application and download it with all its features.

5. **Subscription-based applications.** A business model where you pay a monthly fee to gain access to some media such as music, video or news.

When developing an application, several decisions must be made when selecting the business model that best suits the application. Some choose to

make ”lite” versions of the apps with limited features freely available and a full-blooded version that can be purchased for a one time fee, or at a monthly subscription rate. If you are developing a game that allows for customization of a character for example, allowing for in-app purchases and while providing the game for free might boost the total generated income of the app. If you are developing an online game where you play simple card games with your friends, perhaps a model where you offer a free version with ads and a paid version without is the best fit.

The current money maker dominating the app market is the Freemium model. A publication made in 2011 by Distimo shows that half of the top 200 grossing applications on Apple's App Store were generating their revenue using the “Freemium” and in-app purchase model[28]. On Google Play 65% of the top apps were using this model. According to a new study done by app analytics firm App Annie in 2012, these numbers have grown even stronger with freemium apps generating 69% of the app revenue of iOS apps and 75% of all Android app revenues[29]. App Annie tracks over 700 000 apps worldwide by gathering data from various sources and also offer analytic tools for your own apps.

6.1 What type of application fits which type of business model?

When considering what kind of application you want to develop you must carefully select a most suitable business model if you want to maximize your profit. If your goal is not to get maximum profit, there are other business models if you just want to reach a big crowd. If your application is a tool that could improve the usability of your device or adds features which it didn't already have, you might want to consider a one time purchase fee. Examples are new widgets, media players, office tools, utility apps that could be used to assist in normal day tasks.

Many developers release different versions of their apps, one being free and with advertisement and another ”premium” version without ads. This is often a good idea since the application can be tested out for free before purchasing the premium version, and if that doesn't happen - some cash will be generated from advertising. Testing out one-time purchase fee apps is an easier task on Android as compared to iOS. Here, you can pay for an app and still receive a full refund if you delete the application within 15 minutes, whereas you have to go through a more tedious process to get your refund from applications from the App Store. In fact, in Apple's Terms And Conditions it states that every sale is final, although there exist some special

exceptions[30].

When it comes to developing games, the most popular and greatest revenue-generating business model is the in-app purchase model. It suits games very well since it's easy for games to have additional levels, extra gear or valuables that may be purchased. Once again, by offering the game free, it can reach a wider audience who include the preferred kind of customers, ones that continually invest more money on your product. According to a research done by ABI Research a disproportionately small amount of users is spending a disproportionately large amount of money per user and is driving the in-app purchase model[10]. So the key to success is to make a game that is enjoyable and popular, with the possibility of in-app purchase items for the people who are willing to spend money to enhance their experience. Easier said than done, no doubt. But be wary that there are also customers who dislike the idea of not getting the full experience and the fact that they have to pay for several small things instead of getting the full package at a set price, can be off-putting. This is often the case in games where you compete against others and you face challengers who have spent lots of real money, if the content that they bought gives them clear advantages against those who have not paid, it becomes a pay to win scenario which is frowned upon in gaming communities.

It all boils down to what kind of content or service your application is offering when choosing the correct model. It's no easy task to choose the correct one, choosing several different types to meet the needs of different types of customers is no bad idea. In ABI Research's forecast they expect in-app purchase to be the top grossing revenue model when it comes to making application sales in the following years to come[10].

6.2 A deeper look into Freemium as a business model

More and more apps are turning to the Freemium model in hopes of increased revenue. But when choosing this type of business model, one should really think where this will take your product. Even though we can see that in-app purchase revenue from free apps accounted for 71 % of the revenue made on the iOS platform in feburary 2013 in the United States [11], one must not blindly choose this model without thinking it through and hoping it will land you a great deal of cash. One of the greatest advantages that it brings to the table is since it's free - more people will try it and because of this you might gather a larger user base for your product. But this might also be the problem, you get users and not direct customers - you hope that people will fall in love with your product and will be willing to pay for it later [31]. When in-app purchases first got released in iOS 3.0. Users didn't fully

understand the business model. Developers used this model to let users try the app before purchasing anything. However, they reacted differently than expected. Many had downloaded the app expecting it to be free, just to realize they had to pay to get all the levels[32]. This resulted in bad reviews and angry customers. The power of free is a strong one, and an experiment called the "Hershey's experiment" done by behavioural economists at MIT [33] is believed to be one of the starting points that led to the snowball effect of the freemium model. The experiment is done by giving the choice of two chocolates, one Hershey's chocolate which is considered to be cheaper and of lower quality and one Ferrero Rocher chocolate which is an expensive and considered of higher quality. The prices were manipulated as such between the subjects:

Hershey's	Ferrero Rocher
2 cents	27 cents
1 cent	26 cents
Zero	25 cents

The first two conditions showed a roughly even split, but when presented with the option of a free product - 90% chose the Hershey's. This shows that when presented the option of a free product, people will choose it and this is what became the foundation of the freemium school of thought - free is free marketing. You build your customer base with the free app, then you monetize that base by adding extra content or presenting a paid version. This might be easier said than done, and if you look at the expected value per customer - you have to look at the expected value of free. Using the "Hershey's experiment" and doing the math like suggested by the blog Iterative Path [34], we get two cases. 1.When the price was 1 cent for Hershey's and 26 cents for Rocher, the choice was even, that is 50%. So the expected value of the customer is (0.5 * 1 + 0.5 * 26) = 13.5 cents 2.When the price was 0 cent for Hershey's and 25 cents for Rocher, the choice was 90% Hershey's and 10% Rocher. So the expected value is (0.9 * 0 + 0.1 * 25) = 2.5 cents

The expected value per customer is 13.5 cents if you choose to charge 1 cent for the product and the expected value per customer is 2.5 cents if it's free. This means if you choose to offer the free version, you lost 11 cents and have to make it up in a different way. A low priced version alongside a premium version might be an better option, but since there are limits on how low you can price your app on the app stores, this is a challenge and you really have to think through before making your decision.

It's important to remember that even though we see how much in-app purchases generate in revenue, those numbers are from the hugely successful apps and the model itself is no guaranteed key to success.

7 Application development

Our key goal when deciding what kind of application we wanted to develop was to make it as simple as possible in order to get some experience with the different providers and their implementation of in-app purchases. We knew we wanted to make some kind of game since in-app payments are fairly popular in those, but since neither of us had any previous experience in doing graphics we ended up with a text-base game - the Quiz Challenge.

It's a single player game where you answer questions by clicking on one of four alternatives and get a score by the end of the game. It's far from complete, it only contains the core mechanics such as starting a game, choosing a category, answering the questions and a high score board. The important bit was adding a store to the app which allowed for purchase of additional packs of questions. We thought about adding other purchases like new game modes, but decided to keep it simple since just adding an extra category with extra questions would be least time consuming.

We made the app for two platforms, Android and iOS. Microsoft Phone was also considered, but was dropped due to time restrictions. We made two versions on Android, one which uses Google's Payment API and one which uses PayPal's API.

7.1 Android

Since we were going to make two versions of the Android app we decided to first make a ”skeleton” app with the core functionality which is used in both apps. With this done we could start implementing each of the stores and in-app purchases. The skeleton included the starting screen, settings, most of the navigation and the game itself. We did not put that much effort into graphics and design, and won't be expecting an award for our design choices any time soon.

Our starting screen is very simple activity that includes navigation buttons to everything you need in the app, as you can see in figure 1.

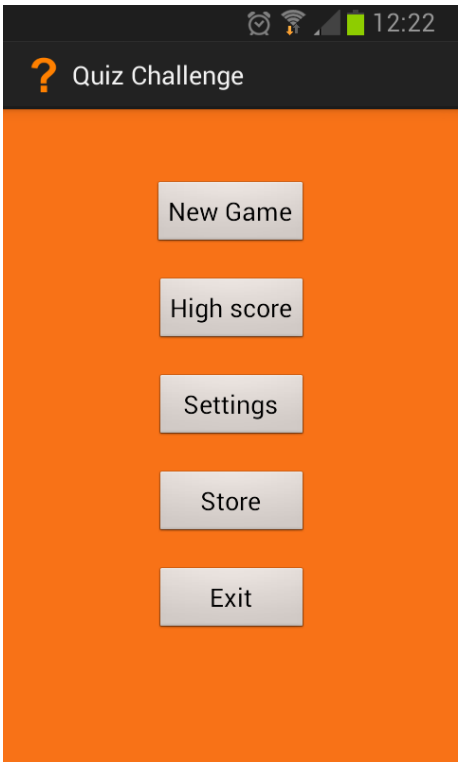

Figure 1: Main menu of our application on the Android platform.

When a user pushes the New Game button he will be taken to the pre-game screen to create a new game. Here he can choose which category he wants to play. The default categories available are TV, movies and sport. You can see this activity on figure 2.

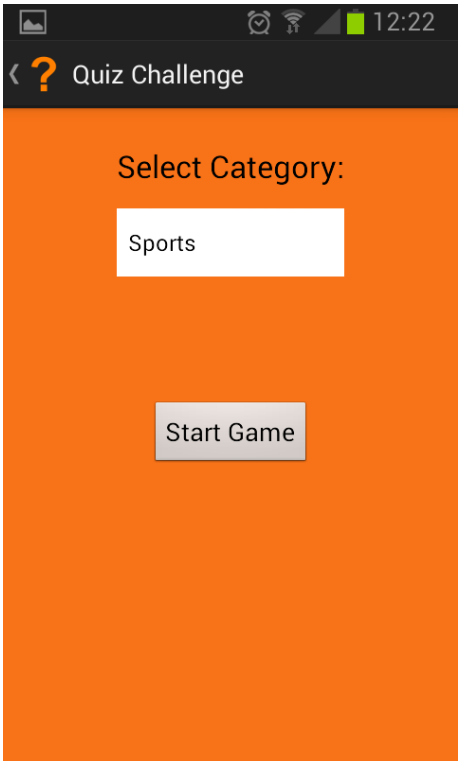

Figure 2: The ability to choose between different categories in a drop-down menu

After the user has chosen his category and starts the game he will be taken to the game activity you can see on figure 3. The game itself is also pretty simple. The user gets a question with 4 alternatives. The game will go on until he answers a question incorrectly.

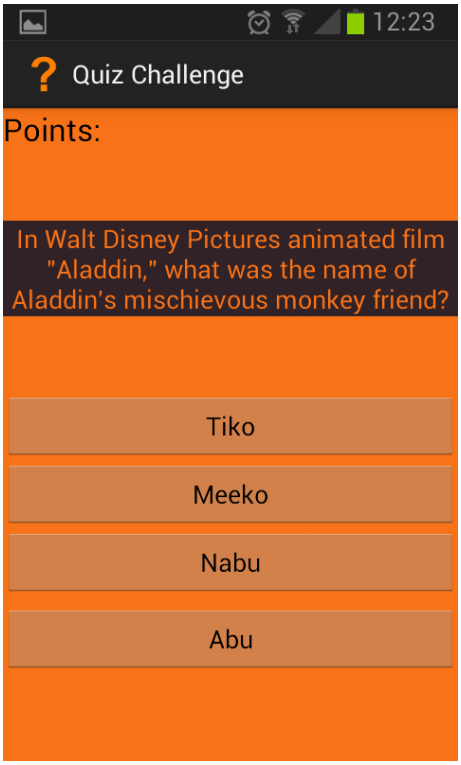

Figure 3: The question and 4 alternatives in the game.

When the game ends the user will be taken to the score screen. Here he can view his score and add it to the high score list if he wants to. He can also start a new game. You can see this screen on figure 4.When we were done with this part of the app we could begin implementing the in-app purchases for the Google Play and PayPal apps.

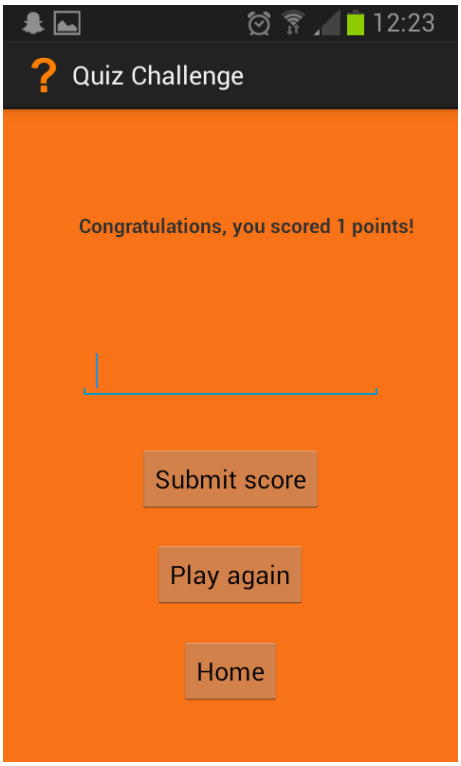

Figure 4: The score screen after an ended round.

7.2 PayPal

When a user pushes the store-button he will be taken to in-app store. As you can see on figure 5 with the PayPal API you have to use one of the three button types following in the API. In the store the user can select what item he wants to purchase. We only created two new categories for our app. After selecting a product he will have to log in on his PayPal-account. You can see this on figure 5. After he logs in the purchase will be completed. The app will then receive a notification telling if the payment went through. If it did, the app will update and the new category will be available to play the users next game.

Here you can see how easily you add the "Pay with PayPal"-button.

```
1  CheckoutButton purchaseHistoryButton = pp.getCheckoutButton(
       this, PayPal.BUTTON_152x33, CheckoutButton.TEXT_PAY);
2  LinearLayout.LayoutParams params = new LinearLayout.
       LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.
       WRAP_CONTENT);
3  params.topMargin = 10;
4  purchaseHistoryButton.setLayoutParams(params);
5
6  purchaseHistoryButton.setOnClickListener(new OnClickListener
       () {
7          public void onClick(View v) {
8                  purchaseType = "history";
9                  requestPurchase();
10         }
11 });
```

```
1  private void requestPurchase() {
2                  PayPalPayment newPayment = new PayPalPayment
                       ();
3          newPayment.setSubtotal(new BigDecimal(8.00));
4          newPayment.setCurrencyType("NOK");
5          newPayment.setRecipient("joran._1350996801_biz@gmail.
               com");
6          newPayment.setMerchantName("H2 Devs");
7          Intent paypalIntent = PayPal.getInstance().checkout(
               newPayment, context, new ResultDelegate());
8          startActivityForResult(paypalIntent, 1);
9          }
```

When the button is added, you will also have to add an OnClickListener. In our listener we will run the method requestPurchase when the purchase button is clicked. In this method we create a PayPalPayment-object. We add all the needed information to in the object to complete the purchase. This contains most of the information about the item that's about to get

purchased, like price, currency and recipient. We then create an Intent which we send to a new activity. In this activity the payment-object gets sent to PayPal while the new activity waits for the response. Once a response is received the app will read the response and act depending on the response. This response could for instance be OK, that the payment was completed, or FAILED.

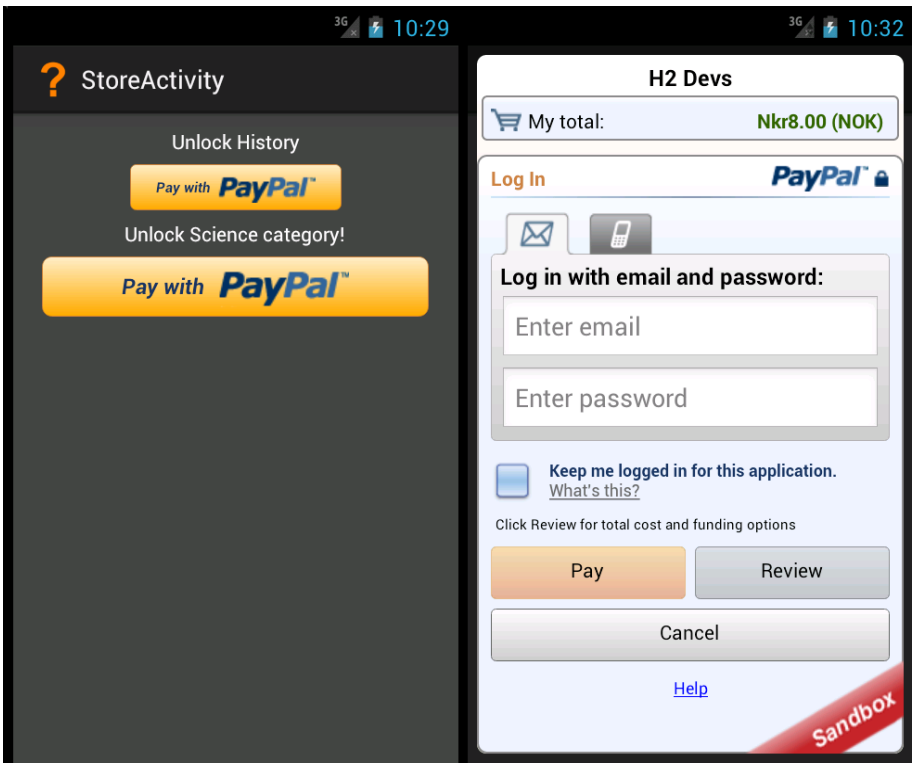

Figure 5: Our store with PayPal purchase buttons and the purchase confirmation.

7.3 Google

For the application created with Google's API you get a lot more integrated functionality you can use in your store. One example of this is that the user doesn't have to log in as it is with PayPal. So when a user on our Google app wants to unlock a new category he enters the store, which you can see in figure 6, he chooses the category he wants. He will then be taken to Google Play. Since the user is already logged in with his Google-account he will be taken directly to the screen where he confirms the purchase. If he doesn't already own this product and payment goes through the app will update the database to unlock the new questions.

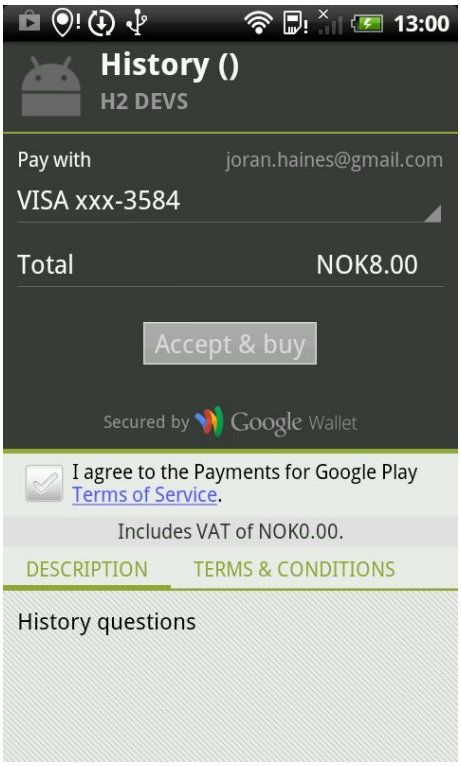

Figure 6: The in-app product presented in the Google Play Store.

There are a lot of classes that you need to add with the payment API version 2, but the most important parts are to add permission to your manifest file so that the application can make use of the billing system, and to compute the public key which you get from the Android publisher site. The products are identified with a string that you choose at the publisher site; in our case we choose the identifier as "history_003" for our history question product. The product is added in our Catalog of products and it contains

the ”sku” or stock-keeping unit which is the unique string of the product, the name ID and the type of product; managed, unmanaged or subscription.

```
1   private static final CatalogEntry[] CATALOG = new
        CatalogEntry[] {
2       new CatalogEntry("history_003", 3, Managed.MANAGED),
3       new CatalogEntry("science_004", 4, Managed.MANAGED),
4   };
```

When the user clicks the ”Purchase history questions”-button this method runs:

```
1   public void unlockHistory(View v){
2       mSku = "history_003";
3       if (mManagedType != Managed.SUBSCRIPTION &&
4               !mBillingService.requestPurchase(mSku, Consts
                    .ITEM_TYPE_INAPP, mPayloadContents)) {
5           showDialog(DIALOG_BILLING_NOT_SUPPORTED_ID);
6       }
7   }
```

This sends a request to Google for a purchase of that product and depending on what the user does, if he chooses to purchase or cancel, this then runs when the transaction is complete.

```
1   public void onRequestPurchaseResponse(RequestPurchase
        request,
2               ResponseCode responseCode) {
3           if (Consts.DEBUG) {
4               Log.d(TAG, request.mProductId + ": " +
                    responseCode);
5           }
6           if (responseCode == ResponseCode.RESULT_OK) {
7               dataSource.unlockCategory(3);
8               if (Consts.DEBUG) {
9                   Log.i(TAG, "purchase was successfully
                        sent to server");
10              }
11          } else if (responseCode == ResponseCode.
                RESULT_USER_CANCELED) {
12              if (Consts.DEBUG) {
13                  Log.i(TAG, "user canceled purchase");
14              }
15          } else {
16              if (Consts.DEBUG) {
17                  Log.i(TAG, "purchase failed");
18              }
19          }
20      }
```

If the purchase went through okay, the dataSource.unlockCategory(3) method runs which unlocks the history questions for the user.

7.4 iOS

The iOS application was made after we finished the Google and PayPal implementations for the Android. We wanted to keep it as similar as possible when it came to look and design, but we had add the bar at the top which allows for navigating back and forward through the application. The flow of the application is identical to the Android version, with a main menu that allows you to start a new game, check out the high score, change settings (not implemented), go to the store to purchase additional questions and exit the application.

It is not recommended to bundle the data that is unlockable by purchases inside the original application. It is recommended to use a server that the application contacts after a purchase is done to retrieve the extra content. We didn't do this since the application was never intended for public release and of course to save ourselves from the extra effort.

When the user enters the app's store and selects a question pack to purchase, a pop-up box appears with the choices of purchasing the selected item or to cancel the purchase as you can see in figure 7. If the user isn't logged into his/hers Apple account, they are asked to do so before proceeding.

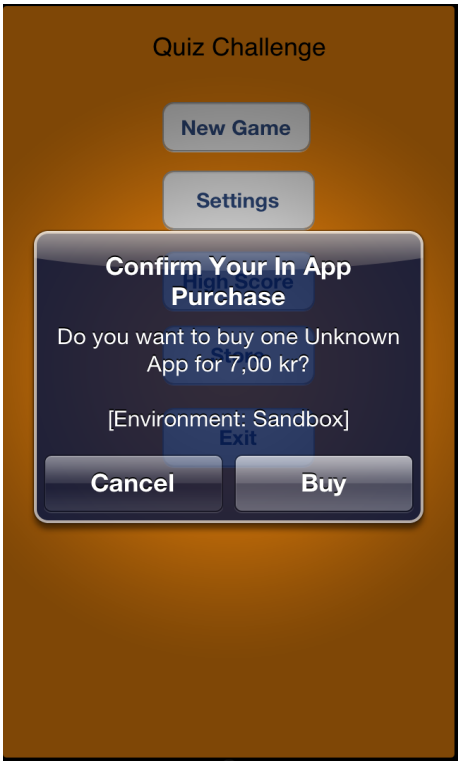

Figure 7: The pop-up box that is displayed after selecting a question pack to purchase.

If the purchase is started the method purchaseProUpgrade is called. The payment is added to the payment queue.

```
1  - (void)purchaseProUpgrade
2  {
3      SKPayment *payment = [SKPayment
           paymentWithProductIdentifier:
           kInAppPurchaseProUpgradeProductId];
4      [[SKPaymentQueue defaultQueue] addPayment:payment];
5  }
```

After the queue has been updated it checks if the payment was completed, failed or restored. So the necessary code for adding the purchased content is done inside the case ”SKPaymentTransactionStatePurchased”. The code below shows this, but the code that unlocks the purchased content is not shown.

```
1  (void)paymentQueue:(SKPaymentQueue *)queue
       updatedTransactions:(NSArray *)transactions
2      {
```

```
3         for (SKPaymentTransaction *transaction in
              transactions)
4         {
5             switch (transaction.transactionState)
6             {
7                 case SKPaymentTransactionStatePurchased:
8                     [self completeTransaction:transaction];
9                 //unlock purchased content.
10                    break;
11                case SKPaymentTransactionStateFailed:
12                    [self failedTransaction:transaction];
13                    break;
14                case SKPaymentTransactionStateRestored:
15                    [self restoreTransaction:transaction];
16                    break;
17                default:
18                    break;
19            }
20        }
21    }
```

This is the method that is called when a transaction was successful.

```
1     - (void)completeTransaction:(SKPaymentTransaction *)
          transaction
2     {
3         [self recordTransaction:transaction];
4         [self provideContent:transaction.payment.
              productIdentifier];
5         [self finishTransaction:transaction wasSuccessful:YES
              ];
6     }
```

We didn't have to add that much code ourselves, besides adding the functionality to unlock the content.

After our hands-on experience with three out of four of the service providers in-app payment solutions, we could move on to create a comparison model which we could later use to evaluate the service providers with.

8 Comparison model

In this section we will define and explain what aspects and points we will be evaluating and comparing. This will be the model we use to compare the different providers and solutions of in-application payment APIs.

Since we are going to study and compare from both a business and technological aspect, we decided to divide our comparison model into two parts, a business model and a technological model. In each model we have identified several categories that we want to compare. Each category may also have several topics we want to compare. Some of the topics might be easy to compare, but others might require much more effort in order to make them comparable. Conducting user surveys and interviews with developers will help us get some data on what they prioritize and what topics are the most important ones.

This could help hobby developers decide which In-Application payment solution works best for them, based on what they have selected as important, less important and not important.

8.1 Business

8.1.1 Fees

Here we will compare what fees and how much the developers have to pay to sell their products. So for a hobby developer the less fees, the better. Fees we will be looking at could be:

1. Registration fee
2. Yearly/monthly fee
3. Fee per purchase

8.1.2 Profit

The profit is how much the developer is left with after the service provider has taken its share of the pie. How much is the developer earning per sale? This is one of the most important aspects when rating the service providers. The more profit that is left for the developer after each sale, the better it is for him or her. To compare the total profit from the application will use the following case:

We assume that each free application will sell an average of 100 in-app products each month, which is a total of 1200 purchases for a whole year,

which we see as a realistic amount for a small scale developer. We will then subtract whatever fees the developers have to pay that year.

We will also take a look at what happens if an application sells 75 000 products.

8.1.3 Business models

What kind of business models does the provider support. This can for instance be some of the following:

1. Pay-per-app
2. In-app advertisement
3. In-app purchase
   (a) One-time purchase
   (b) One-time-one-use-purchase
   (c) Subscriptions

Here we will mainly look at which of these features they offer to the developers. We will also take a closer look into each one to see if there are any difference for the different providers.

8.1.4 Distribution

Here we will look at how the developers can distribute their application. What distribution channels can the developers use? Are there any differences or limitations? This could for instance be Google Play, Apple's App Store or third party application stores.

8.1.5 Market size

We will take a look at how many apps that have been downloaded since start, how many users pay and how many apps are available today. Evaluate the pros and cons of an already large market. The better the market is, the easier and more potential there is to earn a profit.

8.2 Technical

8.2.1 Registration

Look at and compare the process of creating and configuring an account to support in-application payment.

8.2.2 Development

Here we will evaluate and compare the actual development and implementation of the in-application payment API into the application. This could include the difficulty, knowledge and experience required, how much code has to be written and time used.

8.2.3 Testing

It's important for developers that they can test their in-app store to find flaws and weaknesses. Most providers have test servers that the developers can use to test their applications without fear of letting users test it before it's ready for the "real" market. So here we will evaluate the test environment that is provided, ease of setup and use, and take a look at how quickly you can test new code.

8.2.4 Payment

Compare what types of payment are offered and how they can be configured. An example of this is dividing the profit from a transaction to several receivers.

8.2.5 Ease of use

How easy is it for the user to pay for an application, how easy is it to set up an account and add a credit card. This will directly affect how much profit you earn. The easier it is for customer, the more likely he is to make a purchase in you application.

8.2.6 Documentation

Evaluate how well documented the steps of implementing is done, how easy it is to find what you are looking for and overall quality of the documentation. For inexperienced this is very important to make the process of implementing the API's easiest possible.

8.2.7 Supported technology

Compare what technologies the different implementations support such as operating systems and programming languages among others.

8.2.8 Support and community

Evaluate how easy it is for the developer to get the help needed from the providers own support crew or from the community. Check how easily navigated the help pages are and maybe find info on response time from the provider. The communities can be compared in form of size and active users.

8.2.9 Security

Things we will look at are what threats you are vulnerable to, how much malware is out there, how easy it is to configure the security and what security features the platforms offer.

In the next section we will evaluate Apple, Google, Paypal and Windows using this comparison model.

9 Evaluation of the service providers

In this section we will go through Apple, Google, PayPal and Windows and use our comparison model to evaluate each of the service provicers.

9.1 Evaluation of Apple

9.1.1 Business

Fees

When you wish you start developing and distributing iOS applications you need to enroll in the developer program which has a yearly fee of $99. The price is the same for individuals and companies unless a company wants to create proprietary applications - then the yearly fee is at $299.

Apples cut on each transaction and purchase is 30%.

Profit

Apple uses a "price tier list", so the prices you can use are set by Apple. They will also take 30% of every transaction, which gives you a 70% profit. This amount is the same for buying the application and in-app purchases. We will use the first tier in the list as a basis for the comparison. The first tier is of course the lowest amount, which is $0.99. So for each sale your profit would be $0.693. Since Apple require developers to purchase a developer license for $99 each year, you would need to sell 143 items to make a profit each year. Though if you get any sales in Norway, you would get a bit more profit from each sale.

So if you sell 100 items each month on average for $0.99 you would get the following:

The first month: 100 * 0.693 - $99 = $-29.7 The following months we get 100 * 0.693 = $69.3

Month	Profit($)
1	-29.7
6	316.8
12	732.6

Table 1: Apple profit after 1200 in-app sales.

So after a year, with 1200 sales you would get a profit of $732.6. If we increase the number of sales to 75000 we get:

75 000 * 0.693 - $99 = $51876.

If these were sold in Norway with the increased price you would be left with substantially higher amount.

Business models

Apple supports pay-per-app, in-app advertising, subscriptions and in-app purchase. The purchase types that are offered within in-app purchase are

1. **Consumables:** Items that can used once in an application, like a potion or in-game gold.
2. **Non-Consumables:** Items that a bought one time and then persist. These items are available on all registered devices for the user.
3. **Auto-renewable Subscriptions:** This subscription type allows a user access to a item or service for a set duration. This subscription auto renews at the end of the duration.
4. **Free Subscriptions:** Subscription type that delivers content free of charge for the duration.
5. **Non-Renewing Subscriptions:** Subscription type where the user has access to content for a limited duration.

Distribution

The main method of distributing your applications is through Apple's App Store. Installing apps that are available on this store onto your iOS devices requires authentication and can only be done via the App Store and/or iTunes.

Applications can also be distributed by companies hosting their own in-house software for iOS devices via Intranet or restricted web pages available only for employees. This requires the developers to enroll in the iOS Developer Enterprise Program.

For testing purposes Ad Hoc distribution is used to install the application to a total of 100 iOS devices, and these devices must be registered manually by their ID. This feature is supported by both the iOS Developer Program and the iOS Developer Enterprise Program.

The last distribution model, although not legal in all countries, is via third-party sites. This requires you to unlock/jailbreak your device which

removes the limitations and permits root access to the iOS operating system, which in turn allows you to install applications outside the App Store[35].

Market size

According to Apple's press release on January 28. 2013 [36] App Store now offers over 800 000 applications to it's users, and has had a total of 40 billion downloads since 2008. Since then Apple has paid out over 7 billion dollars to the developers. During the most recent earnings call from Apple(January 23, 2013), Tim Cook said that "well over half a billion iOs devices" have been sold in total. So there is a great potential in the iOs market.

However, Distimo estimates in the 2012 yearly review report, that the growth in daily revenue from App Store only increased by 21% the last four months of 2012[37]. If we compare this number to the growth in January 2012, which was 51%, the growth of App Store has slowed down. Although this number is based on data from 20 countries. Even though App Store hasn't had the biggest increase in revenue, they still have the highest revenue. On an average day in November 2012 the daily revenue from App Store was an astonishing 15 million dollars. An interesting point in the report is that of the revenue in November, 10% of it came from the top 7 applications.

9.1.2 Technical

Registration

There are several different programs you can choose between when you want to start developing iOS applications. The most common is the iOS Developer program which is offered both to individuals and companies. To register as a company you need a D-U-N-S number which is an unique number given to a single business entity. When you register as an Apple developer you get free access to developer tools and resources needed to create iOS and Mac applications. You also get access to sample code and videos from their world-wide developers conference which can be of great help for both new and experienced developers. You register by either signing in with your pre-existing Apple ID or you create a new Apple ID. The Apple ID is your main account when dealing with everything that you deal with Apple such as purchasing and downloading apps from the App Store, using iTunes to buy songs and iTunes Connect to distribute your content. Having only one account that is integrated into all Apple services keeps it simple for the developer. All in all it's a very simple and painless experience.

Development

When we started developing our application for iOS our experience with Mac, XCode and Objective-C was from very limited to non-existent. Neither of us

had used XCode or Objective-C prior to this. Due to this and our lack of time, we decided to cut down the application even more, and focused primarily on implementing the in-app billing along with some basic functionality.

Before we started writing any code we had to find a good source of information and knowledge and headed over to the iOS Developer Library to read up on the SDK and in-app billing API. As we entered the site we were overwhelmed by information. Since we had never developed using Objective-C we had to depend on Apple's guides and information sites, which we found to be very confusing and hard to find what we were looking for. So we abandoned the site and found other sources for information. We'll talk more about this later.

After we had found the information we needed, writing the code wasn't that difficult, but it's quite different from Java and the Android SDK. Basically we had the following steps to implement in-app purchases to our application:

1. Create a unique App ID
2. Generate and install a new provisioning profile
3. Update the bundle ID and code signing profile in Xcode
4. Submitting application metadata in iTunes Connect
5. Add the products for in-app purchase
6. Write code for fetching the product from your store
7. Wait for products to become available

The implementation and the amount of code needed was very manageable, even for rookie developers. One thing holding us back was the bureaucracy of Apple. A lot of time was spent filling out forms and waiting for responses and updates. All in all we spent about a week developing the application for iOS.

Testing

It's important for developers that they can test their in-app store to find flaws and weaknesses. Most providers have test servers that the developers can use to test their applications without fear of letting users test it before it's ready for the "real" market. So here we will evaluate the test environment that is provided, ease of setup and use, and take a look at how quickly you can test new code.

We ran into some problems when testing on a device at first because of an older Mac OS preventing us from using the newest iOS SDK. Using this old version created a conflict with the mobile device which only supported the newest iOS code. This was solved by getting up to date with the newest OS, even though it cost us money in order to upgrade.

Testing is possible without external devices by using the iOS Simulator which is included in Xcode 4. The simulator is quite good and it even has support to test out in-app purchases in the newer versions. When you haven't implemented in-app purchase in earlier projects, this certainly speeds up the process of getting it done.

Testing on devices is a more tedious process than just plugging your device into your computer and running it. You need to add the device IDs of the devices that are going to be used for testing into the iOS Provisioning Portal. You need a distribution certificate in your keychain and an ad-hoc provisioning profile which contains the app ID and the device IDs. Then you can create the package with the application for the tester. The testers need that file and the Ad-hoc provisioning profile file in order to test. Quite a lot of work the first time around.

Payment

In order to use the App Store you must use your Apple ID to create a iTunes Store account. When creating this account you need to enter your billing information and select a credit card to make purchases from. Application and in-app purchases will then be made with this stored card information.

Ease of use

Since it's required to have an Apple-ID account to download applications on App Store, a user will already have a credit/debit card registered to his account. So when trying to make a purchase in an application, all a user has to do is just press purchase and confirm the purchase when a dialog box appears. This process is so simple that unknowing kids can purchase items without knowing what they're actually doing. Which has been a bother to some people. Other than confirming the purchase, the in-app store design is mostly up to the developers.

Documentation

The documentation provided for in-app purchase is very thorough and detailed. The API programming guide that is presented on Apple's developer site is divided into several sections to give introduction, explain what is

needed and the steps involved to finalize the implementation. The step-by-step process is perhaps the number one thing that developers are on the outlook for when they are in need of a quick solution. An developer unfamiliar with iOS development and iTunes connect might not find much help in the in-app purchase documentation as some additional knowledge is required in order to create an unique App ID, a provisioning profile and a few other steps that need to be done prior to the in-app purchase implementation.

So this led us to search the web for a more first-time and "newbie" friendly guide. We found a blog containing a tutorial [38] which gave us a good starting point and a good overview of what needed to be done.

The documentation that Apple provide is of very high quality, but it can be quite overwhelming at times.

Supported technology

Apple is very strict with what technology can be used to make application for their hardware. This inflexibility turned out to be quite a problem for us. First of, you have to use a Mac and you should use Xcode. Ok, so you have to get a hold of a mac. Then to test we couldn't use our available iPhone since it was old and did not have the right version of iOS. Because of this we had to get the newest iOS SDK, which required we buy Mountain Lion. We experienced that if you don't have to newest products from Apple, you're going to have a bad time. Other than that, as mentioned earlier, you code in Objective-C and it only works on Apple products. There are ways to create iOS applications outside using the native tools that are available, one way is to use Xamarin, a cross-platform development tool which allows you to create iOS, Android and Windows apps by writing the code in C#.

Support and community

Apple's support site for developers is mainly centred around FAQs sorted in topics as the different types of developer programs, the App Store and other technical issues. If you click one of these topics it covers what needs to be done in order to register for one of these programs, what tools are available and the necessary resources to get started. These pages link to the development center for iOS, Mac and Safari which are heavily documented and contains all things needed for a developer to start up from scratch including sample code, tutorial videos and documents. The developer center site also links to the official developer forums where developers can ask and answer questions. The official forums are quite active, with over 117 000 threads, not including the archives. There's also a lot of activity on StackOverflow

for iOS, with a total of 130 000 questions posted, of which only 33 000 is unanswered. Members of the developer program have to option to write two TSI support tickets per year to get further help if you can't find any answers on the forum.

Security

Apple's strict process of reviewing applications before being made available to the App Store is one reason why there are less headlines regarding iOS when it comes to malicious apps. A case study of security on different mobile platforms in 2011 concluded that *"a non proficient attacker would not choose to use the iOS as a privacy and security attack vector, since it is the platform having the most defensive mechanism in place (i.e. application testing, controlled application installation vectors and remote application removal) and being difficult in application development"* [39]. In quarter 1 to 3 in 2012, only 1,1% of all new mobile threats was found on iOS [2].

Apple is a large and successful company, it has many dedicated users who place a great deal of trust in that their products are secure. Apple has a great responsibility and takes security very seriously, a few of the steps that they have taken to secure their reputation and trust of their users are some of the following: App code signing which requires all apps that are to run on the iOS system to have its executable code signed by an Apple-issued certificate. This means all third-party apps must be validated and signed before being allowed to run on the device. Developers must register with Apple and be verified before being able to develop and distribute applications and in addition the application is reviewed to ensure that the application that is uploaded is operating as the developer has stated.

All third-party apps are sandboxed in the sense that they can't access files from other applications or make any changes to the device itself. This is done to keep applications separated and prevent them from getting information stored by other apps[40].

Over the years Apple has had one problem, and a very simple one at that. Purchasing apps and in-app items has been so easy that even small children can complete a purchase. Through the years there has been some cases where these kids have spent hundreds of dollars with their parents knowing. One case went so far that the parents sued Apple[41].

9.2 Evaluation of Google

9.2.1 Business

Fees

To register as a publisher you need to pay a one time fee at $25. This means that once you have paid this fee, there will be no other monthly or yearly fee that needs to be paid.

As for fees per purchase, Google will take 30% of every type of transaction/purchase.

Profit

As mentioned in the last section, Google will take 30% from every transaction. This includes purchasing the application itself, in-app purchases etc. Most transactions will be for around $0.99. This means Google will take 30.6 cents for each purchase, which leaves the developers with $0.693. So in order to earn a profit for an application you need to sell at least 37 products in you app. As 37 sales * $0.693 = $25.64.

Now let's look at if your application gets 100 purchases each month for $0.99.

The first month we get (100 sales * $0.693) - $25 = $44.3. The following months we get 100 * $0.693 = $69.3.

Now if your application gets a bit more popular and you get 75 000 purchases the first year, we get the following profit the first year:

75 000 * $0.693 - 25 = $51950

Month	Profit($)
1	44.3
6	390.8
12	806.6

Table 2: Google profit after 1200 in-app sales

Business models

Google supports pay-per-app, subscriptions, in-app advertising and in-app purchase. The in-app purchase product types that Google offers depends on which API version you choose, 2 or 3.

They recommend to use version 2 if you want to sell subscriptions in your application, which are items that are sold with a recurring bill interval until the user cancels it. Version 2 also included Managed per user account and unmanaged product types. "Managed" are items that can only be purchased once per user account and Google Play stores that transaction information which allows for restoring later on. Unmanaged are items that do not have their transaction information stored on Google Play, which leaves the developer of the app to keep track of these items.

Version 3 is recommended if you want to sell in-app products only, not subscriptions. This version only has one product type; managed in-app products. This product type has the ability to offer non-consumable and consumable items, similar to version 2's managed and unmanaged, but this time you can keep track of the user's ownership of in-app products.

Distribution

Android strives to be an open platform, and therefore offers a lot of flexibility and choices to developers. One of these choices is how to distribute your app. Android applications can be distributed through marketplaces, web sites or even e-mail.

The main distribution channel for Android applications is Google's own app store, Google Play. For a developer to share his application on Google Play he has to accept the terms of agreement. This is the Developer Distribution Agreement and the Developer Program Policies[42]. In the Developer Program Policies Google states the following:

"App purchases: *Developers charging for applications and downloads from Google Play must do so by using Google Play's payment system.*
In-app purchases: *Developers offering additional content, services or functionality within an application downloaded from Google Play must use Google Play's payment system as the method of payment, except:*

1. *where payment is primarily for physical goods or services (e.g. buying movie tickets; e.g. buying a publication where the price also includes a hard copy subscription); or*

2. *where payment is for digital content or goods that may be consumed*

outside of the application itself (e.g. buying songs that can be played on other music players)"

This section specifies that when using Google Play for distributing your app, you cannot use a third party library to handle in-application payments and transactions. The only exceptions being if you are selling digital content which is used outside of the application itself, or if you are selling physical goods or services. You of course also have the usual no sexually explicit material, bullying or violence, illegal activities, infringement of intellectual property etc.

There are also many third party app stores available to developers who want to use other or more channels to distribute their app. An example of this is Amazon App Store which is growing rapidly.

Market size

It's hard to get exact numbers on which of the competitors are in the lead. When it comes to shipping out new devices, Android completely crushes iOS devices with its recently 75% over iOS's 14,9%, third quarter 2012 [43]. There are several large manufacturers shipping units with Android such as Samsung, Sony and HTC and only Apple shipping iOS units. Google bought Android in 2005 and sought out to make it the future of mobile computing. Instead of the manufacturers making their own operative system for their own devices, they built their own variations on top of the already free Android system provided by Google. This is what makes Android so popular and widely used, and it is still on the rise. Devices with Android are also generally cheaper than iOS devices which gives them an advantage.

Our main focus however, is on applications and its market. How many applications are available, how many applications are downloaded and how much revenue is generated from these applications are some the the key questions. Another aspect is to look at the demographic of the users and which part of it uses and spends money on apps more frequently.

As of November 2012, Google said that about 700,000 applications were available for download [44]. In September 2012, Google announced that 25 billion apps had been downloaded. These numbers are easily observed compared to other figures such as sales, revenue and how many of these are generated by in-app purchases. According to a publication made by Distimo in 2012[37], a typical day for Google Play generated $3.5M USD in revenues when looking at the 20 largest countries using the store. Looking at the top 10 publishers in Google Play 2012, only 3 of these publishers had an average price paid per app, whereas iOS's top 10 publishers had only 2 that didn't have an average price paid per app.

9.2.2 Technical

Registration

In order to upload applications one has to register for a Google publisher account and if the application is to support payment of some sort, a Google Wallet(also known as Google Checkout)[12] account is needed also. To register as a publisher you need to pay a one time fee at $25. The nice thing is that you use your gmail account to register for all these services, so if you already have set up your Google Wallet which you use to pay for apps, you pay the publisher account fee with that. Then you register as a merchant in order to be able to charge for your in-application products, which is free. All you have to do then is sign in with your Gmail account, select your location and agree to the terms and then you are good to go. The only requirement being that you need a active credit/debit card.

When this is done you're ready to start implementing in-app payment in your application and configure your in-app products.

To sum up; you need a publisher account, a Google Wallet account and a merchant account which all use your previous Google account information to register with.

Development

Before implementing the in-application billing we made our own simple application which we will not go into too much detail right here. To keep it short, the idea was a simple quiz application where there are some questions in a certain category and the user has the opportunity to purchase additional categories with new questions, which is where the in-app billing comes into play. We had some basic knowledge of the Android SDK and when we considered our application to be functioning well enough to implement the important feature, we followed a guide provided by Google on how to do so[45]. There were 7 steps to be followed in order for the application to support in-app billing:

1. Download the in-app billing sample application. This step encouraged to upload the sample application and try the in-app billing functionality before adding to our own project, so we did.

2. Add the IMarketBillingService.aidl file to your project. An .aidl is an Android Interface Definition Language file which generates a Java

[12]For more information: http://www.google.com/wallet/

interface file to your projects gen folder. You can then use this interface to make billing request to Google.

3. Update your AndroidManifest.xml file. We needed to define a new permission for the application in order to use the billing functionality called "com.android.vending.BILLING". We also needed to add a BroadcastReciver with some intent filters and a local Service to bind with the IMarketBillingService. This was all accomplished by using their sample code. Since the sample application also contained all the classes needed for steps 4-6, we choose to skip to part 7 after copying the needed classes into our own application. We did however look at the classes to understand what was going on, even though one should just appreciate that "it just works!".

4. Create a Service and bind it to the MarketBillingService so your application can send billing requests and receive billing responses from Google Play.

5. Create a BroadcastReceiver to handle broadcast intents from Google Play. Create a security processing component to verify the integrity of the transaction messages that are sent by Google Play.

6. Modify your application code to support in-app billing. We modified one activity class in our code called StoreActivity since this class had two simple buttons which were to be used as redirections to the Play Store. So in simple terms we had to add functionality so that when the user presses the button to purchase the category History, the app would then send all the necessary requests to Google so that the Play Store displays the item that the user wished to purchase. This will be told in greater detail in the next section, Testing.

The guide goes through all of these steps in much more detail with example code and explaining how the different parts and classes interact. This, at first, seemed quite easy, but as we got started it didn't go quite that smoothly.

We used the example code combined with the guide as a starting point, but we found it a bit confusing. To make a good implementation we needed a lot functionality, which needed a good amount of coding. Even with the guide we didn't know where to start. There were a lot of classes and methods that were needed, but after carefully going through the example code step-by-step we figured out most of it. All in all it took some time to implement, and wasn't as easy as "just implementing a few methods". It requires a great deal of knowledge and configuration.

One thing that should be noted is that when we were developing, we were using the In-app Billing API version 2 which now is superseded by version 3. While we haven't tried implementing the in-app billing using the newest API, what we've seen from the documentation on how to implement the API - it seems easier and less confusing.

Testing

There was a lot of back and forth for us when doing the testing. We ran into a problem which can be a bother for developers that are either developing alone or do not have access to an extra phone to test on. Emulators can be used to test almost everything except interaction with the Play store. So in order to test your in-app billing functions, you need an account which is listed as your test account and you of course need to use a phone that has Android. We didn't think we'd run into problems here since we had a phone with android, but one thing stopped us from "finalizing a sale" when we tried to purchase an item from our own sample application. We had set up our publisher account and merchant account with the same gmail account we were using on our testing device. And it turns out that you can't purchase your own applications and/or in-application products, not even for testing. This can be solved either by

1. Resetting your phone to set up a new primary account, which you make as one of your test accounts. This will require you to add a gmail account that is not the same as your merchant account. This can be bothersome since you will then need to reset these settings again if you wish to use your phone with your original gmail account.

2. Buy/borrow someone else's android phone, and add that gmail account as a test account.

3. Replacing your in-application product ID with a fake product ID that android offers to test your application with, called android.test.purchased. When you make this request, Google Play responds as though you have purchased an item and you can then unlock the content that you wanted to be your product. There are also other reserved test methods called "android.test.canceled", "android.test.refunded" and "android.test.item_unavailable".

This could be an issue for the spare time developer if he/she doesn't have access to an extra phone to test the purchase of own items. While waiting

for an old android phone to be sent to our house, we attempted to test the application on one of our room mates phone.

The tedious thing when testing like this is you first have to upload and sign your current version of the application and then upload that version manually to the phone via an USB-cable. When we came across an error in our code we had to update the manifest so that it wouldn't conflict with the current version that was already uploaded and export the project as a .apk all over again. When uploading the new version to the publisher account, we then had to wait a few hours since it took some time for the in-app products to be available again. This really puts a halt to your developing process since you might not be sure what is causing the error when you are running the application on your cell phone.

Payment

Compare what types of payment are offered and how they can be configured. An example of this is dividing the profit from a transaction to several receivers.

With Google's in-app billing you can't customize the payments as advanced as with other payment providers(PayPal). When a user purchases an item in your app the amount is withdrawn from the users Google Wallet account. So buying from Google Play, of course, requires that you have a Google Wallet account, which requires a credit/debit card.

The Google Wallet account is set up when the user tries to purchase something from Google Play. Your credit/debit card is stored in the Wallet, which means it stores the account information on their secure servers until you wish to remove it. The goal of storing the information is to make it more convenient for the user to make additional purchases from the Play store without having to manually enter your payment information each time. The charges to your card will appear on your credit/debit card and a complete order will give you a receipt via email.

Some mobile network operators offer an alternative to this, by purchasing apps via carrier billing. So instead of paying with your card, the bill will be posted to your mobile billing account after a 15 minute return period is expired. This is something that's still in the works as it is only currently available in some countries and for some mobile operators. As of early 2013, customers of Telenor in Norway and Sweden are able to pay for apps by using carrier billing via Google Play.

Ease of use

It is quite easy for the user to pay for an application or an in-app purchase. If they already have a Google wallet linked to their Google account on their device, the purchasing procedure is pretty straightforward. You need to be logged in on your Google account in order to download applications on the Play Store and if the application has a download price, you press a purchase button which contains the price of that application. When clicking that button you are asked to agree to the payment Term of Service, then to finalize the purchase you have to click "Accept and buy" which starts the download if your credit/debit card has sufficient funds.

In-app purchases are implemented by the developer, but the payment is done using the same procedure as normal application purchases. When an in-app purchase is implemented into an application, the user clicks on the item that he/she wishes to purchase and is then redirected to the Play Store where that item is listed and paid for.

Documentation

On the Android developer site, Google offers a very well documented guide on how to implement in-app billing in your app as well as reference information on the API. With the changes in API version 3.0, the guide is now written in 5 steps with a detailed description and code examples. In the reference guide you can look up technical information on methods and messages used in the API. The API also comes with a working example application which you can use to get some more insight and examples on how your in-app billing could be implemented.

During our development we found the guide a bit confusing at first and needed to read through it a couple of times before we finally fully understood how to implement the API.

Supported technology

In-app billing for Android is of course implemented in Java. The APIs currently support Android 2.2 and above, which would support over 90% of active devices running Android.

Android applications are coded in Java, which means applications can be developed on almost any platform like Mac and Linux. You can also use .Net and Visual Studios using the right tools. So implementing in-app billing with Google's API is highly flexible.

Support and community

For support related to coding/implementation Google provides links to several community forums on their Android developer web page, in form of Google Groups and IRC channels. IRC, Internet Relay Chat, is a protocol used for chatting with channels for various discussion topics that allows for group communication or one-to-one communication. The community is already quite large due to the amount of people developing applications and is steadily rising. They also have a help center where you can get help with other issues related to Google Play.

Stackoverflow[46] is a popular site used by developers to ask and answer questions regarding programming problems, algorithms and coding techniques. Questions can be tagged with relevant subjects and the tag "android" has been tagged in almost 300 000 questions. Of these 90 000 are unanswered. So chances are if you have an issue with your current implementation of in-app purchase, a Google search might lead you to that site where your question already may have been answered.

Google does not offer any other kind code-level support then the forums, however if your problem is related to Google Play or your account you could write a support ticket for help.

Security

Android being the largest actor in the mobile market [18] and with it's focus on openness makes it the most attractive target for hackers. With no control or code review of applications on Google Play and the increasing use of independent third-party app stores, the distribution of malicious applications is easy on Android. Because of this, Android is the most targeted operating system for mobile devices [2]. In Mcafee's threat report for Q3 2012 [1] goes as far as saying "In fact, we see very few mobile threats that are not directed at Android phones". F-secure reported over 51000 unique samples detected during this quarter, although only a few were actually found to be on Google Play. So the amount of malware for Android is extensive and new exploits are detected almost daily.

A study in 2011 that focused on Google's In-App Billing API, found that over 58% of the top applications on Google Play at the time were vulnerable to an automated attack which would make all the items in an in-app store free of charge[47]. This attack is attack is called the FreeMarket attack. The way it works is that a fake app store is created. Then the application to be attacked is rewritten to use this fake app store instead of the real one, resulting in free purchases. They could then release the app to give other

user access.

When developing your application and implementing the in-app billing, Google provides guides and information on how and why you should make your more application more secure [45]. This makes it easier for more inexperienced developers to make a secure application and also gets more focus on security. Basically the security in Android breaks down to the manifest.xml file and application signature. In the manifest developers explicitly define each permission the application needs. All the permissions an application has defined can be viewed by the customers prior to purchasing and downloading it. So it's very much up to the users to be careful when getting new applications for their device. So if a developer follows Google's best practices and guide, configuring the security correctly shouldn't be too big of a challenge.

To prevent the worst malware to spreading Google has what is referred to as a "kill switch". This is a feature which Google can use to remotely remove malware from devices. On Feb. 2. 2012 Google revealed their next step against the malware on Android. They called this "Bouncer" [48]. Bouncer is a security service that automatically scans newly uploaded applications on the Play Store in search of malicious code. This new feature was said to remove 40% of malware in the store, but it didn't take long before methods to circumvent Bouncer appeared.

9.3 Evaulation of PayPal

9.3.1 Business

Fees

PayPal is free to use for private developers, so there's no registration fee and yearly fee, but they do have a small fee on sales. PayPal offers offer two options for payment plans with different fees. They have the standard plan and the micro payment plan. Other factors which affect the transaction fee are national/international transactions, currency types and your revenue.

As you can see on table 3 and table 4 the transaction fee is based on how much revenue you got. The more you sell, less of a fee you have to pay. To get this benefit you have to apply for Merchant rate, and it is always based on your sale volume the previous month.

Monthly revenue	Fee per transaction($)
0,00 NOK - 20 000,00 NOK	3,4% + 2,80 NOK
20 000,01 NOK - 80 000,00 NOK	2,9% + 2,80 NOK
80 000,01 NOK - 400 000,00 NOK	2,7% + 2,80 NOK
400 000,01 NOK - 800 000,00 NOK	2,4% + 2,80 NOK
>800 000,00 NOK	1,9% + 2,80 NOK

Table 3: Norwegian fees for PayPal

A retailer can apply for a micro transaction plan, which you would use for in-app payment. This plan is customized for transactions below $10.With this plan you get a transaction fee which is 5% + $0.05. International transactions includes a 1% additional cross-border fee, and an additional 2,5% fee for currency conversions. To use this plan you either need to have a Premier account or a business account, which you should have anyway to be able to support payments from credit and debit cards.

Monthly revenue	Fee per transaction($)
0,00 NOK - 20 000,00 NOK	4,9% + 2,80 NOK
20 000,01 NOK - 80 000,00 NOK	4,4% + 2,80 NOK
80 000,01 NOK - 400 000,00 NOK	4,2% + 2,80 NOK
400 000,01 NOK - 800 000,00 NOK	3,9% + 2,80 NOK
>800 000,00 NOK	3,4% + 2,80 NOK

Table 4: International fees for PayPal

Profit

Here we will assume that most developers will get the micro transaction plan, since this will be the most ideal choice with such low transactions. So then the fee you have to pay will be 5% + $0.05 + 1% for cross border fee. Since there is no registration or yearly fees, you'll have a profit from your first sale. You can also set the price to be whatever you want, but we will assume that you sell for about the same amount as the others, $.99 or 6 NOK. Profit per in the same nation will be ($.99 - $0.05) - 5% = $0.8905 and $0.8806 for cross border sales with currency conversion.

For national sales you will get the following each month: 100 * $0.8905 = $89.05. Assuming that half the sales will be international we get: ($89.05 / 2) + (50 * $0.8806) = $88.555

Month	Revenue(National)	Revenue(with 50% int. sales)($)
1	$89.05	$88.555
6	$534.30	$531.33
12	$1068.60	$1062.66

Table 5: PayPal profit after 1200 in-app sales

If we increase the sales to 75 000 we get:
75 000 * $0.8905 = $66787.50
and with 50% international:
75 000 * $0.8806 = $66045.00

Business models

The available business model depends on what account type and API the developer decides to use. However, you cannot implement a "Pay-per-app"-model with PayPal, as this has to be done through the distribution channel the developer chooses. You also can't directly earn money through ads with PayPal. PayPal is just handling transactions; the rest is up to the developer.

PayPal offers 4 different payment APIs, but the only ones with In-app payments, and therefore the most relevant, are the Mobile Payment Libraries (MPL) and Mobile Express Checkout Library (MECL).

	MPL	MECL
In-app payment	+	+
Runs without back-end API integration	+	
Quick integration	+	+
Credit card checkout (no PayPal account needed)		+
Supports auth/settle payments		+
Supports Recurring Payments	+	+

Table 6: What kind of functionality the MPL and MECL support

As we can see from table 6 both APIs support in-app purchases and subscriptions. The main difference is that with MPL a customer can't make any purchases with a credit card without having a PayPal account. But other than that both APIs support one-time purchases. This means they also support one-time-one-use; it's just up to the developers code how to handle it. You won't get a free database to store your sales.

Distribution

PayPal can be implemented on both Android and iOS, but they both have some requirements for using their native app stores. In their Terms of Agreement it states that you are not allowed to sell your application on either Google Play or Apple's App Store unless you are using their own APIs to handle sales. So when you choose to use PayPal your application is not eligible to be distributed on them. You therefore need to use third-party app stores and distribution channels. This greatly limits your marked size. PayPal does not at this date have a way to distribute applications themselves, so the developer will have to use third-party sites like Amazon[49], Handster[50] or GetJar[51].

Market size

Even though you can't use Play or App Store, there are many independent third-party app stores out there, but the market size for applications that use PayPal will still be highly reduced. Let's look at one of the biggest third-party app stores, Handster. They currently have over 50 000 apps for Android and over 20 000 apps for the iPhone. The interest for these app stores is increasing, but they're still far behind. One positive thing with a smaller app store is that you will make it easier for customers to find your app in a store with thousands.

9.3.2 Technical

Registration

When you first arrive at the PayPal developer site you get a good overview of the different solutions they have available. Based on this overview you can easily decide what solution you better use. They even have some example use cases describing what the solutions support and should be used for. We ended up with PayPal Mobile Libraries as this API best suited our needs for our application.

After you have decided on what API to use you have to start registering an account on PayPal. These accounts are free to register for both personal and companies/non-profit organizations. After you've made an account with PayPal you need to register with x.commerce. This is quite easy as all you have to do is to log in with your PayPal account to import your account to x.commerce. You will also have to create an account with PayPal Sandbox. The Sandbox is a testing environment where developers can test their calls to PayPal, which of course is very important.

Development

Now the next step is to find the right SDK/API and start developing. PayPal has "How To"s and "Getting started" guides for everything, but we experienced some confusion to which one was correct for us. After some time and some searching we found our starting point and proceeded to download the MPL API.

The SDK comes with sample code and some good documentation. The sample code is a simple application which shows how the API can be implemented into an app, although not all of it is needed. In the documentation you get a step-by-step guide to how implement the API. It gives you a good description of what features it has, method calls, response codes etc.

Implementing the Mobile Payment Library in Android was a quite easy and quick task. There are a lot of resources that help you implement it. But all you really need is already included in the SDK.

While developing we found a few things that weren't perfect. First off, you have to use PayPal's official "Pay with PayPal"-button. You cannot change it, and it only comes in three sizes. Also, the user interface for this activity has to be hard coded for you to be able to add the purchase button, which is quite a hassle and not very good practice.

After you press the purchase button you will be take to the payment activity. This activity you cannot change at all. Although this is annoying, it gives the users the same interface as every other PayPal payment window.

If you really want to use PayPal as your in-app payment solution, you better have some kind of back end server to register purchases. PayPal does not keep track of what a user buys. This will give you a problem if a user for example has downloaded your app, bought an item in your store, uninstalled the app and then at a later point reinstalled it. Without a back end you won't be able restore his purchased items. Other elements that we noted were that the size and loading time of the MPL library is horrific, which makes the application bigger and the purchase process slower when compared to the application that uses Google's API.

All in all, after we found the documentation and sample code the implementation was quite easy and was done within a few hours. You basically just have to write a few methods with a few lines of code.

Testing

For testing, PayPal offers a sandboxed testing environment where developers can test their apps and the payment API without having to worry about security, money and other potential problems. It's a safe testing environment.

Before you can use PayPal Sandbox you have to create test accounts for both seller and buyer. You can use pre configured test accounts, or you could create them yourself. For testing, we just went for the easy choice of using the ones that were pre configured.

To use the sandbox you have to make a few configurations in the application. You have to sign the application with a fixed appID. This ID-code says that the application is a test app. The second thing you to do is setting the app to use the sandbox server.

The code below shows how the PayPal API is initiated. The second parameter in the method call says that the app is a test version and that it should use the Sandbox server.

PayPal ppObj = PayPal.initWithAppID(this.getBaseContext(), "APP-80W284485P519543T", PayPal.ENV_SANDBOX);

In the sandbox developers can set the money balance on the test accounts which is used to simulate real transactions. The seller account will also be updated when a transaction goes through. This way we can easily monitor transactions.

Payment

The PayPal API offers several different payment methods, which gives developers ways to implement quite advanced payments. Developers can choose

from what PayPal calls simple payment, parallel payment and chained payment.

- Simple payment is the most basic type of transactions. It only supports payment to a single recipient.
- Parallel payment gives developers the opportunity to divide the payment between several recipient. Using this method changes the user interface a bit so that the customer sees who the money goes to.
- Chained payment is also way of handling dividing the amount between several recipients, but here the whole amount is paid to a single receiver, called the primary receiver, who then passes it to several other receivers.

Using PayPal requires that a user has a PayPal account with funds or a credit/debit card. And as mentioned earlier, PayPal offers both subscription type payments and one-time payments.

Ease of use

When a customer presses the Pay with PayPal-button he will be taken to an activity where he has to sign in with a PayPal account. This process is very much like all other payment solutions from PayPal. This will make it easier to use for customers. There is also an option to save username and password.

Documentation

As we discussed in the development section, the SDK comes with most of the documentation a developer might need. It's a simple step-by-step guide and it's so easy to follow that even developers with very little experience will be able to complete the implementation. You can also look at the code example to see how it is done. All in all the quality of the documentation is good. The only thing that's missing is anything regarding security configuration.

Supported technology

PayPal currently has two libraries available for mobile developers, MPL and MECL and are mentioned on page 65. Both these libraries are available for Android version 1.5 and higher. As for iOS the libraries are available for(iPhone, iPad, iPod) iOS 3.0 and higher.

Support and community

PayPal has an official forum where developers can ask other developers for help. The forum doesn't seem to have very high activity however. Most of the posts on the board have a few responses, if any. Other than the official forums there seems to be some activity on Stackoverflow. Whenever we had any problems, we found most of them on Stackoverflow or on some kind of developer blog. So there is a small active community. Other than the forums, PayPal doesn't offer much code-level support.

Security

PayPal has millions of users all over the world, and is one of the biggest payment providers on the internet. Their payment libraries are used for everything from big web stores to small donations to a private site. PayPal is a widely known brand that most people have heard of or used earlier at some point. This gives PayPal a great deal of trust, which will lead to more people willingly using the technology. There have been some security issues in the past, as shown in the National Vulnerability Database, but none of them are related to the mobile library[52].

As for configuring security in your application, there was very little information to be found, and it was not even mentioned in the guides. This can be an issue. Misconfiguration could easily lead to a potentially serious flaw in an application.

When you configure the PayPal transactions you have to define the amount to be paid, what kind of currency should be used and to whom the money should be transferred. By following the guide and code examples you would end up with the following lines of code:

```
newPayment.setSubtotal(new BigDecimal(8.00));
newPayment.setCurrencyType("NOK");
newPayment.setRecipient("joran._1350996801_biz@gmail.com");
```

All that defines to where the money should be transferred is that last line of code. If you were to change this line, the money would end up somewhere else. Obviously, this wouldn't be very good for security reasons. If a hacker decides to reverse engineer your application, he could quickly locate that line, change it to a merchant account he controls, and release a "spoof" version of the application.

Since there's no app store, there's no approval or control of applications. When using independent app stores, the risk of malware and bad software is higher[53].

9.4 Evaluation of Windows

9.4.1 Business

Fees

To join the Windows Phone Dev Center you need to register an account which requires an annual fee of $99. They also offer a free of charge subscription if you are a DreamSpark student. You can register as a DreamSpark student and get access to free designer and developer tools if your school is one of the ones that are eligible for this program. As NTNU students we were able to register as a DreamSpark student and get free access, which is great for our purpose.

Microsoft has no limit to how many paid-applications you can submit to the marketplace for free, but if you're submitting free applications, there's a limit of 100. For very free application you want to submit past 100 submissions you'll have to pay $19.99 for each application.

Microsoft will also take 30% off of every purchase in your application.

Profit

Without actually testing this ourselves, we found it to be quite confusing trying to find exactly what the minimum price is for sales in Windows Phone Store. When looking at the documentation in the Developer center we found that the minimum price in the price-tier is $1.49 [13]. But when looking at the actual prices in the app store we see most applications selling for $0.99 [54]. Since this seems to be the going price, we decided to use this amount for comparison.

How easy it is to make a profit when developing and release an application on Windows Phone depends on your whether or not you're a student at a DreamSpark school. If you are, you won't have to pay the $99. So here we will have two additional cases when looking at your profit. One case if you're a DreamSpark student and another for non-DreamSpark developers.

So if you're one of the lucky students and don't have to pay the license fee of $99, you'll start making money from your first sale. Microsoft will take 30.6c for each sale in your application, which leaves the developers with $0.693. So if we take a look at the first case where you sell 100 items each month for $.99.

For each month you will get (100 sales * $0.693) = $69.3.

[13] `http://msdn.microsoft.com/en-us/library/windows/apps/jj193599.aspx#pricetiers`

Month	DreamSpark($)	Non-DreamSpark($)
1	$69.3	$-29.7
6	$415.8	$316.8
12	$831.6	$732.6

Table 7: Windows profit after 1200 in-app sales

Now, if you have to pay the $99, you need to sell a few more items in order to start making a profit.

The first month with only 100 sales you will get: 100 * $0.693 - $99 = $-29.7 The following months we get 100 * $0.693 = $69.3

If we increase the number of sales to 75 000 we get:

75 000 * $0.693 = $51975 for DreamSpark 75 000 * $0.693 - $99 = $51876 for non-DreamSpark

In order to receive payment at all from Microsoft you have to have at least $200 in revenue. Anything below this and you won't receive anything [55].

Business models

Here we will mainly look at which of these features they offer to the developers. We will also take a closer look into each one to see if there are any differences for the different providers.

Some of the business models that are supported are:

- **Paid apps -** Regular paid for each download of an application, not all countries are supported for this however.
- **Trial Apps -** An option for the developer to let the application act in "trial mode" which later can be enabled into a full working application. The developer decides what functionality is available in trial mode or if it should be fully working for a limited amount of time. This is a good way to allow users to test the application before buying it [56].
- **Advertising in apps -** You can include advertising in your applications by using the Microsoft Advertising SDK.
- **In app purchase -** Available in two different product types; consumable and durable. In-app purchase is only available in Windows Phone 8 [57].

Distribution

The main channel for distributing Windows Phone 7 apps is the Windows Marketplace. The Marketplace offers apps, games, music, movies, TV shows and podcasts and can be launched from the device itself or via the web. In Windows Phone 8, you use an improved version simply called Store or Windows Phone Store which also recommends apps based on what you like.

Windows also offers a solution for companies that want to distribute apps internally for their own employees [58]. The company can bypass the Windows Phone Store by getting an enterprise certificate from Symantec, creating an application enrollment token and developing a Company Hub app. The employees can then enroll for company app distribution on their phones and install the company apps. This is only available for Windows Phone 8.

There's also a possibility to install non-marketplace and homebrew apps and this is done by unlocking your Windows Phone and by using a procedure called sideloading. Sideloading is when you transfer data between two local devices, such as a computer and a mobile device via USB or memory card. Without going into too much detail, this is a alternative endorsed by Microsoft themselves for homebrew app developers called ChevronWP7 [59] which is a service that costs $9 which lets you "unlock" your phone to install a limit of 10 apps from the web via your computer using developer tools. This type of "unlocking" isn't the same as iOS Jailbreaking, but it gives the same privileges as an developer gets in order to run applications for debugging etc. This service was discontinued in August 2012.

Market size

Microsoft's Windows Phone Store had an explosive growth during 2012, and during this year they doubled the amount of applications in their app store from 75000 to 150000 applications in total, according to Microsoft's developer blog. They also said that the average amount of application downloaded per device running Windows Phone is up to 54 [60]. Many of the applications are only available for Windows Phone 7 and 8.

Windows Phone also almost doubled their market share in the last quarter of 2012 from 1.8% to 3.0% [18]. This indicates it's a growing market for Windows Phone.

9.4.2 Technical

Since we didn't develop our application for Windows Phone we decided to remove the Development process section.

Documentation

When trying to find documentation for implementing in-app purchases for Windows there was a lot of back and forth browsing. After spending some time and effort we found several pages on Microsoft's Dev center[14] that would help us, and we felt we had the information needed to complete the development process. The problem was that we couldn't easily find one page with a good overview of everything related to in-app purchase. Even though we found documentation on the implementation, there was very little information on the business side.

Besides the challenge of finding the information needed, the documentation itself was decent. It came with step-by-step guide with a lot of code samples and a good API reference guide.

Supported technology

Compare what technologies the different implementations support such as OS's and programming languages among others.

To develop apps for Windows Phone 8 you need the Windows Phone SDK 8.0 which includes Microsoft Visual Studio Express and an Emulator for testing. This SDK requires 64-bit Windows 8 Pro, which means that you can't develop Windows Phone 8 apps on older Windows versions such as Windows 7. The SDK 8.0 supports development for Windows Phone 8 and 7.5 devices.

The supported programming languages are C#, VB.NET, C++. C++ was not supported until Windows Phone 8 [61].

Support and community

Microsoft has their own support site at the Windows Phone Dev Center [62]. On this page they give the developers an overview of the official and some other unofficial communities. This includes the Developer forums and links to other communities like Stack Overflow and a subforum at Reddit [63]. They also provide links to How To's where developers could find a solution to his problems.

[14]Windows Dev Center: https://dev.windowsphone.com/en-us

At the developer forums, the sub forum for Windows Phone development currently has over 11000 threads where well over 5000 have been answered, with a total of 73000 messages. Over at the Stack Overflow they have over 15000 threads tagged with windows-phone. Since the release of Windows Phone in late 2010 the community has quickly grown into a large community, and with the increase in market share and popularity, it will most likely continue to grow.

If you don't get the help you need on the forums you could also write a support ticket, but there will some waiting for a response.

Security

Most people have used a Microsoft or Windows product in their life, and are familiar with its areas of use. Even though Windows Phone is fairly new and has a low market share, the brand Windows is well known. Since its release, Windows Phone has statistically been one of the most secure mobile OS's. Very few threats have been identified according to F-Secure's most recent Mobile Threat Report [2]. Only 0.6% of all threats to mobile devices found during Q1-Q3 2012 were related to Windows Phone. This doesn't necessarily mean that Windows Phone has better security. As we mentioned earlier, Windows' market share is quite low compared to other operating systems and therefore doesn't make it an equally attractive target.

Microsoft has applied several security measures to protect the device and its data from threats. "Trusted Boot" ensures that only authorized code can run before they are allowed to load the operating system [64]. This means all apps must be signed properly to execute. Windows Phone 8 keeps applications separated by keeping no communication channels between apps other than through the cloud. Each app is also granted just the capabilities it needs to perform its use cases, and these are available to the user to check upon installation. Apps are checked and go through a certification process before being made available to the Windows Phone Store.

In the next section we will compare each of the service providers based on what we have learned and evaluated in this section.

10 Comparison

In section 9 we individually covered each of the chosen service providers and evaluated them based on our comparison model. In this section we will see how they compare to each other. We will follow the comparison model and go through it comparing them section by section. So we will start with the business part of the model and then move on to the technical.

10.1 Business

10.1.1 Fees

To compare the fees we looked at registration fee, yearly/monthly fees and fees per purchase. We found that it was mostly the registration and yearly fees that were different. The fee per purchase was the same for all providers except for PayPal. The results are summed up in table 8.

When looking at registration fee, the only one that required this was Google with a fee of $25. This is just a one-time fee and is the only fee required to develop and distribute apps for Android. Windows and Apple on the other hand both have a yearly fee of $99. Although Windows is free if the developer is a student. As for the fee per purchase it is 30% for Apple, Google and Windows. PayPal has a more complicated fee plan which you can see in table 3 and table 4 on page 63, so we won't go in-depth on that here. Basically with the right payment plan you will pay 10-15% of the amount on a $1 transaction.

PayPal is clearly the winner if the developer wants to spend as little money as possible to be able to develop and release applications. A one-time fee versus a yearly fee could make a difference for a person who wants to start

	Apple	Google	PayPal	Windows ($)
Registration Fee	None	$25	None	None/Free
Yearly Fee	$99	None	None	$99/Free
Fee per purchase	30% of each transaction	30% of each transaction	See Table x	30% of each transaction

Table 8: The different fees for the providers

doing mobile applications for the first time. It costs no money to implement PayPal's in-app purchase system, but the downside is that you won't get your application distributed at the App Store or Play Store.

It is important to keep in mind that those providers that charge more and more frequently might be doing so to provide better services and the application reviewing process most likely is a full-time job for lots of employees.

10.1.2 Profit

The profit is the amount of money the developers receive after all the fees have been subtracted. To compare the profit from the different service providers we used two cases. The first case was how much profit the developer would get when selling on average 100 items for $1 each month for 12 months. The second case was how the profit would compare if the application was a bit more successful and sold 75000 items in a year. We also looked at how many sales an application needed in order to make a profit.

If we start to look at how much the developer will get for each sale, we see that they will mostly be the same since they take 30% of each sale except for PayPal. For Google, Apple and Windows Phone the developer will get $0.693 from a $.99 sale. From PayPal the developer will receive $0.8905. We also want to mention that with the price tier list some of the developers will get a higher profit in some countries. An example is that in App Store if a app costs$1 in the US, with the fixed prices it costs 7 NOK in Norway. This is at the time of writing an increase of almost 20% based on the exchange rate.

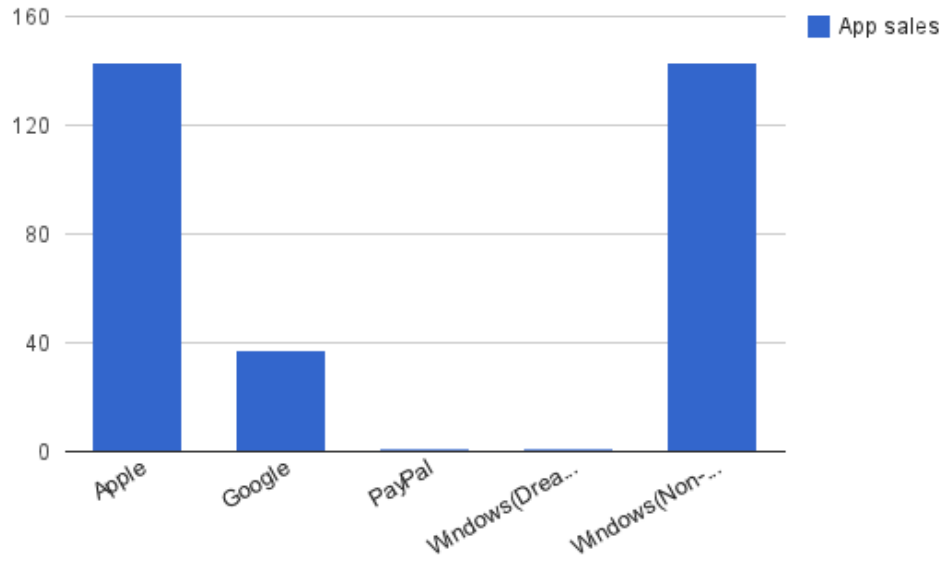

Figure 8: A graph showing how many sales are needed to make a profit

With the different fees and profit per sale calculated we can now compare how many sales you need in order to make up for the registration and yearly fees. As we can see in figure 8 on page 77, PayPal and Windows(DreamSpark) has no registration/yearly fee, and therefore you will have a profit from the first sale. With Google's one-time fee of $25 you will need to sell 37 items. Apple and Windows on the other hand, with its $99 yearly fee, the developer has to sell at least 143 items each year to make a profit.

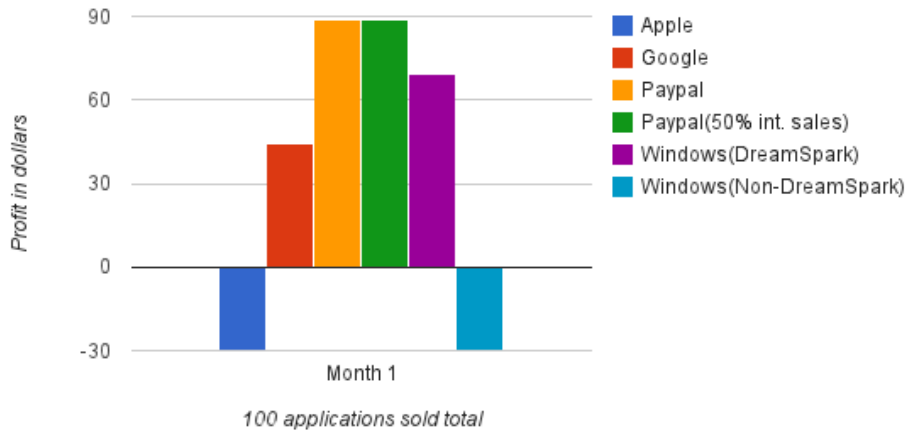

Figure 9: Profit after 1 month of sales

Now let's look at how the services compare to each other when selling on average 100 items each month. At the first month after selling 100 apps, only Apple and Windows are not making profit. This is because of their high yearly fee, whereas Google's low one-time fee and PayPal's no registration fee takes them to the lead.

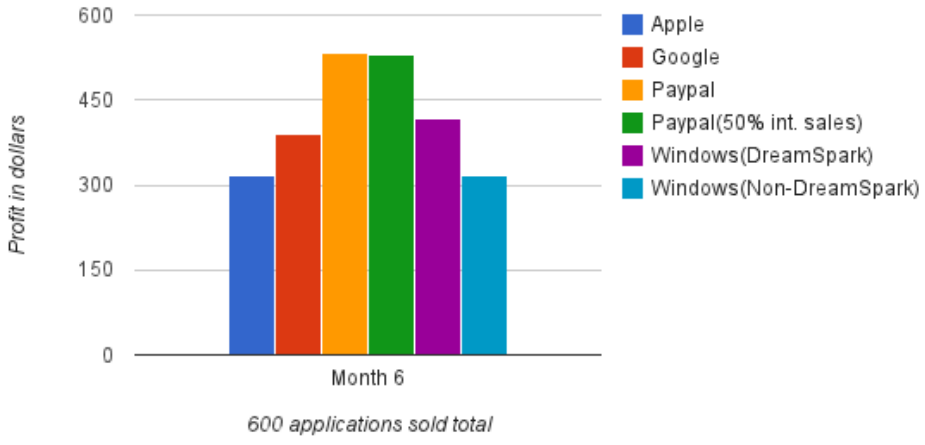

Figure 10: Profit after 6 months of sales

After 6 months of selling 600 applications on each platform, Apple and

Windows are getting more profitable while PayPal is still making the most profit.

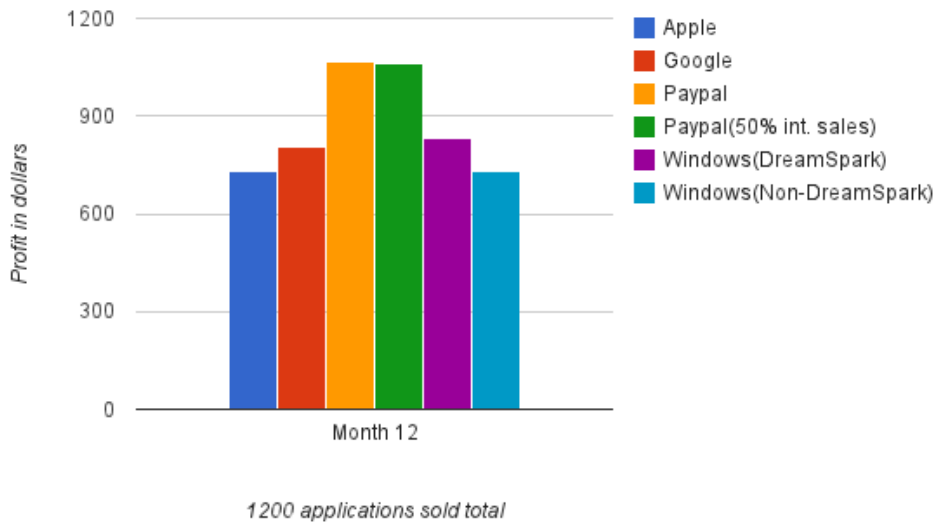

Figure 11: Profit after 12 months of sales

After 12 months PayPal has the highest profit ending up with $1083.00, with Windows (DreamSpark) and Google following behind with $831.6 and $806.6. And with the worst profit we have Apple and Windows at $732.6. So when only looking at profit compared with an equal amount of sales we see PayPal giving the developers the best profit.

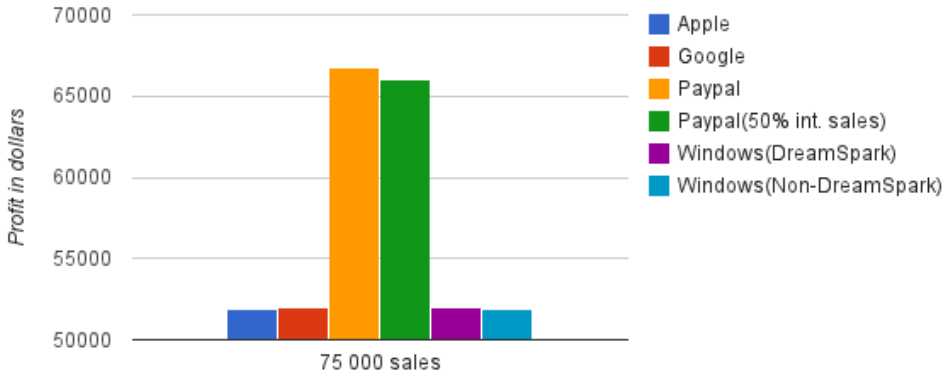

Figure 12: Profit after 75 000 applications sold

If we increase the number of sales to 75000 we see how much difference the fees will affect your profit if your application is successful. Not surprisingly PayPal comes out on top with its low fees resulting in a profit of $66787.50. Then we have Windows (DreamSpark) with $51975, Google at $51950 and finally Apple and Windows with $51876. The calculations are based on $1 sales, but as mentioned earlier, with the fixed prices on a price-tier list you will make more money in certain countries.

10.1.3 Business models

First, let's take a look at what business models the service providers support.

	Apple	Google	PayPal	Windows
Pay per App	x	x		x
In-App purchase	x	x	x	x
One time purchase	x	x	x	x
One time one use purchase	x	x	x	x
Subscription	x	x	x	
Trial application				x
In-App advertising	x	x		x

Table 9: Business models supported by the different service providers

All the service providers have their own distribution channels except PayPal; they just handle the transactions and leave the rest up to the developer. In app purchase is supported by all providers, even though Windows didn't support it until Windows Mobile Phone 8. The different types of in app purchases are available for all providers except Windows which don't support subscription based payments.

Windows is the only platform that offers trial applications as an implementation choice when developing. This could be done manually by creating "stripped down" versions of your paid application on iOS and Android.

The main difference between the providers when implementing in-app purchase is that the service is more integrated on iOS, Android and Windows compared to PayPal. If we picture a scenario where a user purchases a sword in a game, then installs this game on a new device - he would of course like his purchased sword in this game as well. Since applications are bound to the users account along with all its purchases on iOS, Android and Windows Phone 8 the only step necessary to retrieve it is to simply download it again. If you want this done with your PayPal implementation, you will have to create a service that recognizes the user that you have registered in your own database to confirm that he/she has purchased this item previously. So even though PayPal offer some of the same type of business models as the others, there might be more work involved if you want to get on par with the other providers quality of service.

When it comes to In-App advertisement, Apple, Google and Windows have their own SDKs to help and provide services like Apple iAd, Google AdMob and Microsoft Advertising.

10.1.4 Distribution

As we mentioned in the previous section, the only in-app purchase provider that doesn't have an official or a distribution channel of their own, is PayPal. Apple, Google and Windows on the other hand have their own channel of distribution. Since they are developing and manufacturing both mobile device and the operating systems, they develop and customize their own services, which PayPal or any third party service provider could never do. Using the official app stores will of course make it easier for most users to access the store and purchase an app. They will also feel a lot safer when they know it's controlled by big companies such as Google and Apple. For developers using PayPal, they have to depend on other unofficial third-party distribution channels. This of course also has its advantages and disadvantages.

First of we have the size of the app stores. Apple and Android can boast with number of apps up in the 700000-800000 with billions of downloads,

whereas the third-party app stores have from a few hundred up to 250000. The larger ones being Amazon App store[49], Getjar[51] and Handster[50]. Second, using unofficial app stores has its security issues. They often have no app reviews or control of apps, and a lot malware is coming from these app stores.

10.1.5 Market size

In this section it is easier to look at the numbers and compare them in order to see who has the most applications, downloads and users, but we also have to take into consideration what pros and cons there is of an already large market. Getting the correct numbers that are up to date isn't easy as those numbers aren't announced that often by the providers themselves.

We will focus on the amount of applications that are available on each the markets first; Apple's App Store, Google's Play Store and Windows Store are the biggest ones since they are the main distribution channel for each of the providers. Applications that use PayPal's payment system are not allowed on these markets due to the providers own payment systems, and PayPal does not have its own marketplace. Gathering numbers on how many apps that are using PayPal's in-app purchase is impossible since they can be scattered around several third-party site marketplaces.

	Available apps	Apps downloads	User activations
Apple App Store	850 000(May 2013[65])	50B(May 2013)[65]	500M(Jan.2013 [8])
Google Play Store	700 000(Oct 2012[44])	48B(May 2013[66])	900M(May 2013 [66])
Windows Phone Store	145 000(Feb 2013[67])	1.7B [15]	31M [16]

Table 10: Available apps, amount of downloads and users for each provider

One thing to take into consideration is that the App store was released 2 months prior to the App Store(previously Android Market) and the Windows Phone Store was released 2 years after this. We can see from table 10 on page 83 that Apple currently has the most applications available. But since Google's number is so outdated it is reasonable to assume that they aren't that far apart in total available apps. Windows is still at an early stage with its phone market so it "only" has 145 000 applications available.

Having so many applications available could also have its downsides. It could be hard for a new application to break through and gain audience if there already are similar applications out there or just too many applications in general. Publishing an application on a newer market with less available apps could make it more popular than if it were to be published in an already over saturated market.

[15]This number is from where an average user have downloaded 54 apps [60] multiplied by total users.

[16]Total users is added manually by checking multiple Gartner reports [18], [5], [68],[69],[70],[71]

10.2 Technical

10.2.1 Registration

The registration processes aren't that different between the service providers. For all of them you first have to register a "basic" user account, and then find out what type of account is needed for a developer. For Apple this is the iOS Developer Program. For Google you will need a Google Publisher and Wallet account. With PayPal you need a x.commerce and Sandbox account for testing. There are only some minor differences.

PayPal is the only one where a credit is not required. All you have to do to use PayPal is to create the account. If you want to pay with PayPal you could transfer some money to that account. When you're going to use this account as a merchant you could apply to get merchant discount on your transfer. For using PayPal you should also apply for the micro transaction plan. This process was also quite easy, but it did take a couple of months to get approved for the micro transactions.

With Apple you need to have an Apple-ID. With this account you can access and join the iOS Developer Program. This costs $99 each year, but it's required for in-app payments and to publish your app. As for Google, you need to have a Google account. When you have this account you can easily register for the other services Google offers, this includes the Wallet account. After this you have to register for an Google publisher account. This is needed if you're going to use in-app payment or publish your application and this requires the one-time fee of $25.

We would say that based on our experience, the easiest registration process was Apple. Apple was quick and easy to find what you needed. Then we have Google and PayPal. With Google there was some confusion due to name changes, but all in all it wasn't that bad. PayPal was also easy, but there was a long wait to get the approval for micro transactions.

10.2.2 Development

The development was done on two platforms, Android and iOS, with PayPal implemented on Android. These platforms are very different in both implementation and policies. Under the development sections in each service provider we went through in some detail how the implementation was done. We decided we would compare the processes by evaluating them in four categories; level of difficulty, knowledge/experience required, lines of code and time used. When we developed the Android application, we first made a "skeleton" app so we could use this as a basis for our PayPal implementation as well.

All the development that was done of course requires some basic knowledge of programming, but with good guides and helpful forums one does not need to be an expert to get the in-application payment implementation working. If you have no previous experience with any programming languages, the learning curve will be pretty steep and perhaps a bit harsh if you want to go straight ahead and develop an mobile application. Programmers who have basic knowledge about coding in general will have a smoother transition if they have coded in advance before embarking into mobile development.

The difficulty of implementing the in-app payment varied some between the different APIs. For PayPal the implementation was very easy and even the most inexperienced developers should be able to implement it. You basically just had to write a few lines of code and hard-code some of the GUI. With this easy implementation you lose a lot of flexibility and you should have a back-end server as well. The payment API was implemented and tested in only a few hours, and the number of lines of code is just about 200 including the hard-coded GUI.

With Google, when using API version 2.0, there was a lot more functionality and configurations that had to be implemented in order to fully support in-app payment. Even with the all documentation it's a lot more difficult than PayPal, but you don't need to have a back-end server. To complete the implementation we spent around 20 hours, though a lot of this was time spent waiting and testing. We would say that this API requires some more knowledge as it's more complex. In total we ended up with 5 classes and 500-600 lines of code. A lot of the code was from the documentation and guide which we used and then customized for our app.

We had only made one mobile application beforehand, a simple Android application, but without any possibilities for in app purchase. Because of this, we had an easier time developing for Android than we did for iOS since we had previous experience with Java but no experience with Objective C. After we spent several days working out how to get a simple application up and running on iOS and understanding how Objective C is compared to Java, we could focus on the in-app purchase implementation. Writing the code for the application to fetch the product information from Apple is mostly done by copying code already written and replacing the necessary lines with your own unique application Id's, so this part isn't exactly hard. The difficult part is writing code which allows a purchase to unlock some feature or how the products shall be displayed within the application. The in-app purchase implementation took us about 15 hours, but most of this time was spent waiting for a product to appear after adding it on Apple's systems and testing in general. We added two files to handle the in-app purchase, one header and one implementation file that contained around 200

	Difficulty	Knowledge/experience	Lines of code	Hours spent
iOS	2	2	200	15
Android	3	2	500-600	20
PayPal	1	1	200	4

Table 11: Development rating for the service providers

lines of code. If we would've added functionality to the app such as unlocking features with a purchase, there would've been more code. But we decided due to shortage of time to only test the communication between the app and Apple's service to retrieve the purchase success message.

When comparing difficulty and knowledge/experience we will rate each one from 1 to 5 where 1 is low difficulty and little knowledge/experience recommended, whereas 5 is the opposite.

We can see here that we clearly spent the least amount of time and effort implementing PayPal's solution. Not only did we spend the fewest hours implementing it, but in our opinion it is also the least difficult. But there is a good reason why PayPal is easier to implement than the other two; they have far more complex APIs with greater focus on security, integration with the device itself and more convenience for the developer such as tracking sales and linking each app to each user. Adding the in app purchase products to Apple's and Google's services took several hours before they became available to fetch information from and that is one of the key reasons why we spent more hours completing our task.

10.2.3 Testing

Starting with PayPal, to test the PayPal payment you use the PayPal Sandbox. All you have to do to get ready for the testing is to register an account, create some mock merchant and customer profiles, and change a few lines of code in your application to set it to use the sandboxed testing environment. This works in both the simulator and on a device. It's quick and easy to set up and we were testing in about 30 minutes.

Testing with Google can be quite a problematic task as you probably want to have an extra device to test on, since you can't use the emulator or purchase items from your own account Google account for this. Also, if you have to make some changes in your code or items in your store you have to wait for Google Play to update the changes. This can take up to a

day. However, when all is in order the testing itself is easy. You can easily simulate different scenarios and errors to find flaws in your application.

With Apple you can test in the simulator which is included in the SDK. This makes the testing process a whole lot easier. You can also test with device, but this requires some more time and work. There's also some waiting here as well when doing some changes in the in-app store etc.

Again PayPal is the simplest solution. It's was the quickest and easiest process when compared to the others, and you can without any hassle, test on both a simulator and a device. Apple also supports testing on simulator and a device, but the process of creating a "testing device" can be quite confusing the first time. You can also add test-users/devices with Google, but the process is much easier, except if you want to use your own device for testing. You will also spend a lot of time waiting for updates on Google Play.

10.2.4 Payment

With Apple and Google you link your credit or debit card to your account. With Apple the billing information you type in is stored with your Apple ID and with Google it is stored in the Google Wallet which is linked with your Google account. The way purchases work is very similar since you are already logged into your account in order to get access to the applications on your device. But while Apple only allows you to accept the payment by clicking okay with the one credit/debit card you have added Google lets you choose between several credit/debit cards if you have more than one and even the possibility to pay with your mobile carrier if it is supported. With Google it also displays the permissions the application requires along with terms and conditions which has to be agreed upon in order to finalize the purchase.

PayPal is different since when you press to purchase an item you need to enter your PayPal account username/password and accept the purchase. If your account has sufficient funds it will process your sale and the item will be available when the sale is confirmed.

Apple and Google offer similar ways for the developer to offer their application or in-app products for a price. Apple has iTunes Connect where you have to enter your information in order to receive the money which goes through Apple and after they have taken their cut you get the remaining. Google uses a similar service called Checkout which also works as an agent between yourself and the customer. The developers that use PayPal gets their cut transferred directly to their own PayPal account.

When it comes to customization of the payment, PayPal is the only

provider offering something that differs from the others. You can choose for the payment to be made out to one single receiver, several receivers or to one single which then divides the payment to several other receivers. Getting refunds for an application that you aren't satisfied with or bought on accident is an easier task on the Google Play Store since it allows you to get a full refund if you delete the application within 15 minutes of purchase. If you wish to do this with Apple you have to apply for a refund via iTunes and fill out a form, but you will most likely search the web for a guide on how to get a refund. Refunds on PayPal are a bit different since you have to contact the seller directly and ask for a refund, which then can be granted manually.

When comparing all the providers payment solutions and the way they have handled it in general, Google stands out as more user-friendly when it comes to refunds, more versatile as it offers mobile carrier billing and the possibility of adding several credit cards which can be helpful for company employees that wish to use the company card to purchase necessary work-related products.

10.2.5 Ease of use

With all of the service providers you bind a credit card to your account when you create it. The only one not requiring this is PayPal, but with them you have to transfer funds to your account if you don't bind a credit card to it. Since every in-app purchases go through their official app stores for Apple and Google, the users seldom have to worry about creating an account and giving his credit card information to a third party. The transactions instead go through the account that the mobile device is bound to. This makes the process of completing an in-app purchase very easy. As we discussed earlier, some are complaining that this is too easy, resulting in little kids purchasing items for hundreds of dollars without their parents knowing. PayPal will most likely not have this problem since it requires a login and a few more steps than just confirming through a pop-up box.

So without a doubt the ones that are easiest to use are Google and Apple. There are some downsides to this since it is so easy to purchase, children with access to their parents devices might purchase content without knowing that it bills their parents and might end up spending thousands of dollars. Windows has a feature that aims to prevent this called Kid's Corner and it allows you to add what apps, music, videos and games that should be available and this also turns of in app purchase by default.

10.2.6 Documentation

All of the APIs had guides on how to use and implement them. However, there some small differences in the quality. If we start with PayPal, since the implementation itself is such an easy task, the guide is also very short simple, and probably the easiest to use. Almost anyone will be able to implement it. Google's API is also very well documented. But here the implementation is a lot more complex and requires more work, resulting in a more advanced and comprehensive guide and documentation. It takes some time to get a good understanding of what you need to do the first time, but once you reach this point the guide is of good help towards getting a high quality implementation. Google also has the reference page where a developer can easily look up methods, messages, etc. We found Apple's documentation to be of high quality, but it can be overwhelming. There is so much information, and it could be a problem to find what you're looking for. During our development there were some information we couldn't find and had to look for other sources. So for someone just starting out developing for iOS it could be troublesome. The one that gave us the worst experience was Windows. We had a hard time finding the information we were looking for. There was neither an overview page nor a particularly easy way to navigate the site. One could argue that this could be because the API is quite new and probably will improve their documentation in the future.

10.2.7 Supported technology

First we will give an overview of what kind of languages are supported with each provider as we can see in table 12 on page 90.

The only provider supporting almost all programming languages is PayPal, only missing the implementation of .NET based languages. It is no wonder that no providers support this kind of cross-platform possibility since they usually stick to one native programming language and their own platform.

Besides the native tools that the providers recommend, there are other options out there that make cross-platform development possible. One example is Mono, a open source .NET development framework designed for cross-platform development which makes creating iOS and Android apps possible if you wish to write in C#.

There isn't any real comparison that can be made out of this since each provider has its own technology stack and it wouldn't make any sense to figure out which is better. PayPal does support several implementations, but then again it is not a native operative system such as iOS, Android or Windows Phone. One thing that is possible however is to look at what your

	Apple(iOS)	Google(Android)	PayPal	Windows
Java		x	x	
Objective C	x		x	
.NET(C#, C++, VB.net)				x

Table 12: The languages which are supported by each service provider

options are when choosing a provider based on what operating system you are programming with and if you have the correct developer kits.

10.2.8 Support and community

Each service provider has its own support site and help center where developers can find the help they need or they'll be redirected to where they can find it. For code-level support, the support is mostly in form of a forum where developers can ask other developers about whatever problems they might have. StackOverflow[46] seems to be the place where most developers come to seek help. This seems to be a trend for most of the platforms.

PayPal clearly has the smallest community, and has the least activity. On the official forums a lot of post goes unanswered. On StackOverflow PayPal has only 4000 questions where 1500 are unanswered. PayPal's pages can also be hard to navigate when looking for something specific. Windows Phone's developer community seems to be larger than PayPal when looking at the activity on the different support sites and forums. You can also easily find links to other resources besides the official forums.

Apple and Google of course by far have the largest communities. Although one difference is that with Apple the official forums are very active compared to the others, where much of the activity is on unofficial forums like StackOverflow. And with Apple and Windows Phone the developers have the option to write support tickets for additional technical support.

10.2.9 Security

Of the four in-app payment providers PayPal is the only one that isn't integrated in its own operating system and is therefore not equal when comparing. There haven't been many reports on targeted at the payment APIs. As we mentioned in the section on PayPal, they have had some flaws in the past, but none related to the mobile API. Our biggest concern with PayPal is that there is nothing about security in their guides, which could potentially

lead to a weakness in the app. This is something that Apple and Google has taken seriously and have included in their documentation.

As for threats to the other platforms, the number of malware and spyware is quite extensive. The worst one by far is Android. When looking at figure 13 you can see how big the difference is in malware based on platform. This is the result of the open mindset Android has and it being the most used operating system on the marked. It makes for the most attractive target for hackers and other wrongdoers. If we take a look at total number of threats percentage on figure 14 we see that Android is the worst here as well. Compared to iOS and Windows Phone the difference is quite large. So we can safely say that Android is the most exposed of them all.

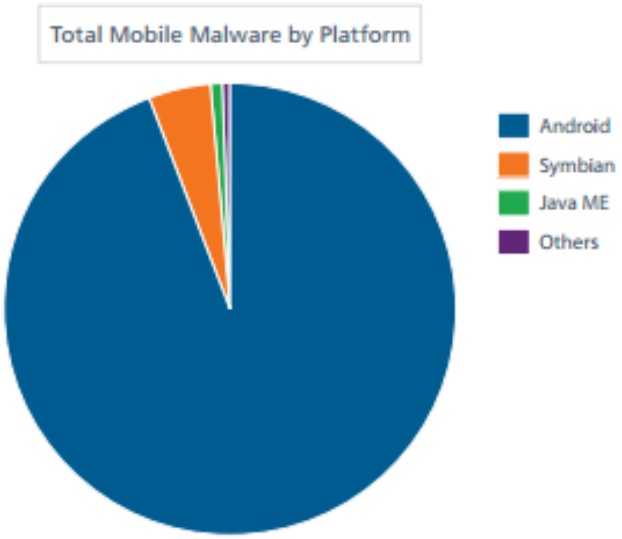

Figure 13: Malware by platform, 2011-2012 [1]

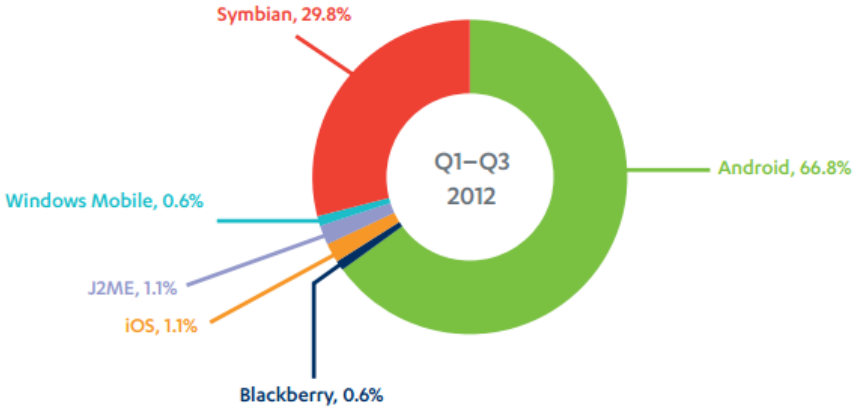

Figure 14: Mobile threats by platform, 2011-2012 [2]

Apple is responsible for both the OS and the devices that they sell, whereas Google offer their OS to hardware manufacturers such as Samsung, HTC, Sony. Google also have their own devices which run Android but Apple does not give permission for outsiders to use their iOS. This allows for device manufacturers to create their own version of Android and that means own security implementations. Devices with Windows Phone 8 are manufactured by Nokia, HTC, Samsung and Huawei but all use the same OS. Since Apple is in charge of both device and software this allows them to do security exactly the way they want to.

When submitting applications to Apple and Microsoft they go through a process where it is checked for malicious code, quality and security issues among others. Since this isn't done manually by Google, some bad applications manage to slip by the automated checks and into the store. These applications are often swiftly removed when they get reported by users before they infect too many devices. The work that Apple and Microsoft put into preventing these applications from reaching the market in the first place gives them more trust among users.

(Another security feature that Apple, Google and Microsoft has is the infamous kill switch which allows for deletion of applications without user consent.) To conclude, Apple's closed source and strict app submitting process puts them at the top when it comes to the lack of security issues. Windows is similar, but Apple is so much larger when it comes to market size and applications available that it is impressive that there have been so few security scandals. Even though Google's security practices are good, there have been too many malicious applications and attempts towards Android

users for it to compete with Apple and Microsoft

In the next section, we will present the user survey which we conducted in order to get some real data from the developers to help us understand what they consider as the most important aspects of in-app payment.

11 User survey

We wanted to conduct a survey in form of a questionnaire to help us better understand exactly what properties most hobby developers prioritized when thinking of introducing a store within their app. By getting this knowledge we could compare it up against the properties of each of the different in-app purchase service providers and see which one best matches the developers preferred properties and from that see what would be the best choice.

11.1 Defining the survey

When selecting the types of questions we wanted to ask, and what we wanted to get out of our survey in general, we looked at our comparison model to help shape the survey. Since we would use the result of the survey to help answer some of our research questions, using the comparison model as a baseline for the survey questions felt natural. We had some hypotheses regarding some of the topics in our survey and we wanted to see if we were somewhat correct or not. Our hypotheses are the following:

1. Those who have implemented the in-app purchase prefer it as a business model.

2. Potential profit from application development would be ranked very high, and even higher for those with more programming experience.

3. Good documentation and an easy development process is very important for developers with less experience.

4. Most people have used and prefer the Android platform.

We used an online survey tool called SurveyMonkey [72] to generate and distribute our survey. It had some limitations, such as number of questions and logical rules that couldn't be applied unless we paid for a premium membership. This premium membership would've also given us fewer restrictions when analyzing the data. The fact that we couldn't ask as many questions as we wished might just have been a good thing for us, since we didn't want our participants to spend too much time answering them - this often ends in people getting tired and not completing it. We then had to be more careful when selecting 10 questions which was the maximum amount of questions available for free.

We started our survey with a simple way to get to know our participants, since it is interesting to see what kind of influence the years of programming experience they have and what current occupation they are in affects their

opinions. We also wanted to map what kind of mobile platforms they have developed for and which they preferred before digging into the important questions about in app purchase.

The first thing we wanted to find out was what kind of business models in mobile development (free, paid, ads, in-app payment and subscriptions) they have implemented themselves and which one they preferred. Now that we had all these topics covered, we could focus the rest of the survey on in app purchase related questions.

We then asked our participants to grade a series of questions on a scale of importance. This was presented as a list where they selected how important they felt each issue was to them. The five choices were: "Don't care", "Not important", "Less important", "Important" and "Very important". The questions we used were:

7. How would you grade the following statements on a grade of importance?
   - I want to pay as little as possible to develop and sell my app.
   - I want to sell my app in the official app store.
   - I want to make money on my app.
   - I want an easy development process.
   - I want good guides and documentation.
   - I want an active community to give/receive help.
   - I want to be able to receive high quality support from official technicians.
8. How would you grade the following statements on a grade of importance?
   - I want to spend the least amount of time implementing the in-app payment.
   - I want a fast and easy way to set up and configure my in-app purchases.
   - I want to be able to quickly test my in-app store/purchases.
   - I want to be able to test in an emulator.
   - I wish for my customers to be able to pay over carrier billing/phone bill.
   - I want an easy way to handle refunds.

- I want my customers to have an easy process completing a purchase.
- I want there to be mandatory security mechanisms in place(Authentication, encryption, access control etc)
- I want to easily configure the security in my app(Authentication, encryption, access control etc).

In order to avoid that all participants answered that everything is equally important or equally not as important, the last part of the survey asked them to rank some of these properties or topics in order of importance. You can view the full survey in Appendix A. By doing this we hoped to get some good data on what the developers considered as the most important aspect of them all and what kind of influence their experience and occupation had on this.

11.2 Data collection

When conducting the survey we wanted to reach as many people as possible to get a more precise result, but to be realistic we set a goal of 50 responses where the respondent's experience would be somewhat equally divided among the groups we set. But with our limited time and resources we decided to go for random sampling when collecting data.

Our target group consist of people who develop in their spare time on private projects, not work related, i.e. hobby developers who do small-scale development. We also wanted people who have experience in developing apps for smart phones on any platform and preferably some experience with in-app purchases, though that was not a requirement.

To distribute and collect data we sent the questionnaire to contacts via e-mail. Some of them continued to distribute it on their Intranet and such so we would reach a few more people. We also used social medias like Facebook to distribute the questionnaire.

In the end we ended up with a sample size of 47, a bit short of what we wanted. We understand that there are some issues with having such a low sample size, but it should give an insight nonetheless. We realize that for example the question regarding preferred mobile platform would most likely look different if our target audience was a larger group of programmers, say 500, since a lot of our respondents are students and are likely to have more Java experience than Objective-C experience due to costs and programming courses. We want to mention that all of the respondents were from Norway, meaning that we have to be careful and consider that the result can be local for Norway and might you get some different results internationally.

Even though we wanted more input from people with low experience, some of the questions might strike them as intimidating due to their unfamiliarity with some of the concepts around mobile development.

It's a hard task to reach the people that are willing to complete surveys, and it makes it even harder since we offer the respondents no compensation for taking their time to complete it, except our humble gratitude for helping out.

11.3 Findings

Before we go too deep into the data collected we will give you a quick overview. As we said earlier we had 47 responses to our questionnaire. Their occupation was divided between Students (24) and employed with an IT-position (23), with very variable experience, as you can see on table 15.

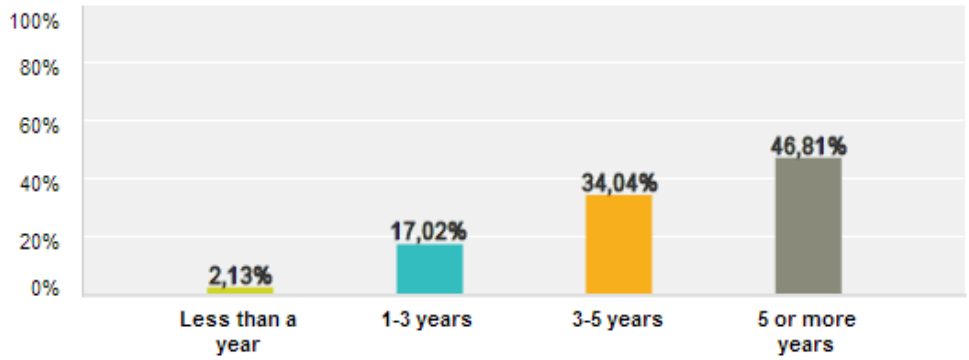

Figure 15: Question 2: "How many years of experience do you have with programming?"

It is regrettable that we didn't collect more data from people with less than one year experience, but we couldn't find an effective way to reach a large amount of young IT-students with the time and resources we had left. We can see that almost half of our respondents have five or more years of programming experience, which is not unexpected since almost half of them are employees within the IT-business. The second largest group belongs at the 3-5 years experience level, which is where we are at and most of our acquaintances are as well. Later we will look deeper into how the different experience levels and current occupation might affect their opinions on in-app purchase.

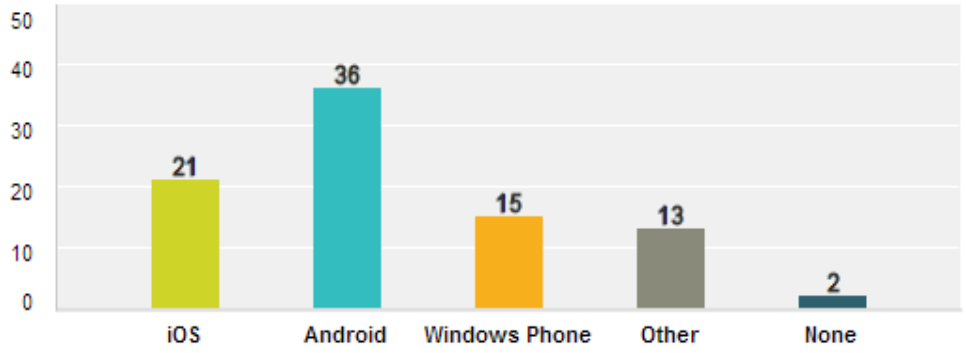

Figure 16: Question 3: "Which of these platforms have you developed for?"

When looking at which platforms the participants have developed for it's not surprising to see Android on top. This may be because it's free to develop and the Android platform itself is very popular. We will discuss this later in the paper. iOS came as the second most used platform in our survey, with a total of 45% having developed an app. Next we have Windows Phone with 32%. Under the "other" category we have platforms such as Symbian, Blackberry, etc.

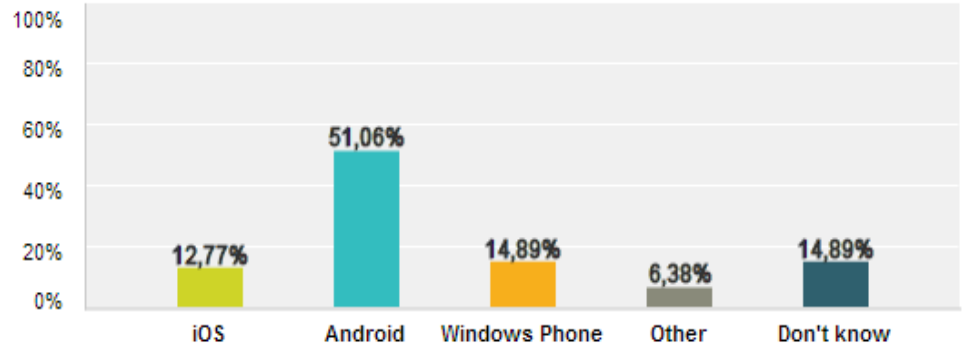

Figure 17: Question 4: "Which platform do you prefer to develop for?"

We also asked the participants which platform they preferred to develop for. The result was that 51.1% said they preferred Android over the other platforms. Our assumption that Android was the most experienced and preferred platform was correct. It's somewhat surprising that iOS only got 12.8%. But given the fact how easy it is to get a hold of the tools that's necessary for Android development compared to iOS in terms of price and hardware requirement, we didn't expect the small-scale developers to have that much experience with iOS in general.

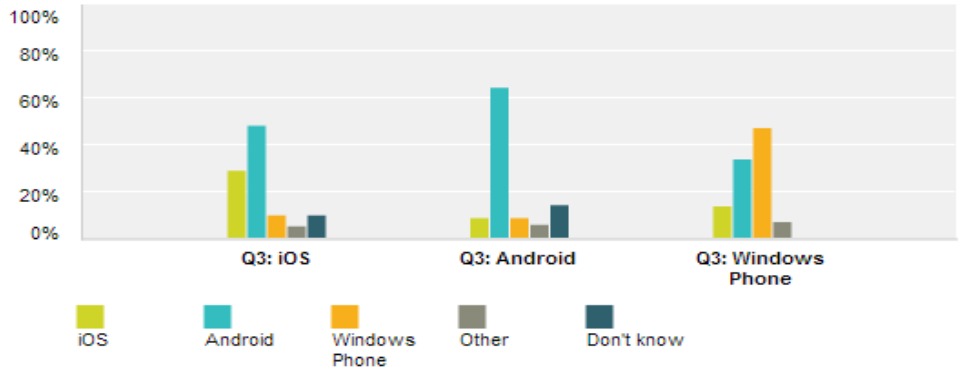

Figure 18: The people who have developed for each platform, and which one they preferred

The people who have experienced developing apps for iOS still preferred Android over iOS. What's also quite interesting is that almost 50% of people having experience with Windows Phones prefer this over the other platforms. In table 18 you can see which platform the participants prefer based on what on platform they have developed for.

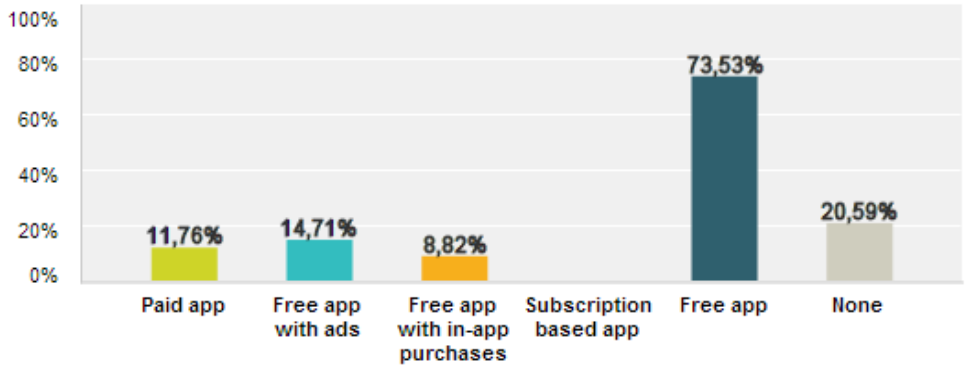

Figure 19: Question 5: "Which of the following business models have you implemented in your app?"

When uncovering what kind of different business models the developers had implemented, only 34 out of 47 people answered the question and 7 people selected none. There was an option to select multiple choices on this question, and the result can be seen in table 19. Those who didn't answer this question or selected "none" might have done some mobile development for educational purposes or for fun and have not released their application to the public, thus thinking that they didn't implement one of these business models. We can see that the most used business model here is the free

application model, and none had any experience with a subscription based app.

Out of the remaining three business models, the in app purchase model was the one that our participants had least experience with, only three people had implemented it in their application. When we select these three people and compare them to other questions, we get some interesting finds. We had hoped that additional people would've had experience with in-app purchase since this would lead to an even stronger and relevant sample size. Reasons for this low number could be that in-app purchases is mostly used in games, which require a great amount of work and experience. Implementing in-app purchases in itself also requires more work than just making the app free or a simple paid app, which could be why many choose to not to implement it. The three people all had Android as their preferred platform to develop on, this might be coincidental due to the how many of our participants that have Android experience and the low response on this question. Two of these persons with in-app purchase experience also had this business model as their preferred business model. This gives us a small indication that our assumption about in-app purchase in hypothesis number 1 was correct. When looking into questions 7 and 8 where the participants are asked to grade some statements on a scale of importance, one statement stands out as more important than the others. "I want my customers to have an easy process completing a purchase" has a score of 4.67(ranging from 1 to 5, where 1 is don't care and 5 is very important) between the three persons with in-app purchase experience. Four people had implemented paid applications and five had done free apps with advertisement.

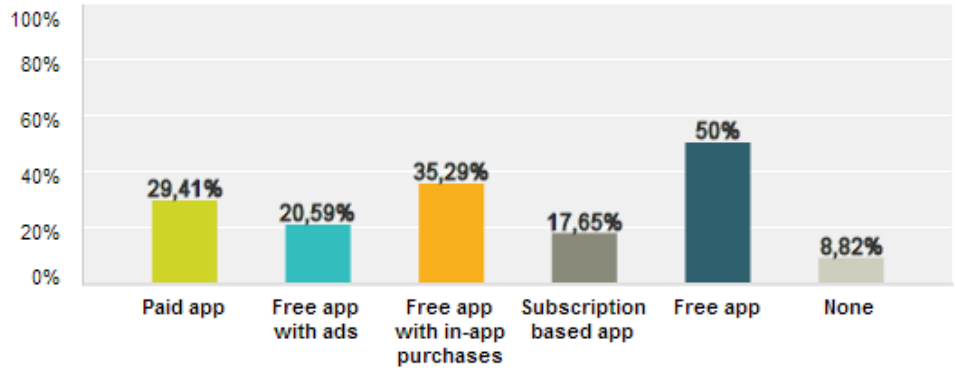

Figure 20: Question 5: "Which of the following is your preferred business model in mobile apps?"

After seeing what experience the developers have in both platforms and business models, we wanted to know if they had made up their mind about

what business model they themselves prefer. Here the alternatives to choose from were every common available business model and ”none”. The result can be seen in table 20. It’s not shocking that the ”free app” model was the most popular for hobby developers. This could be because most hobby developers are motivated by just developing an app for enjoyment, not profit. Another reason we’re seeing this could be that most of the developers have only used the ”free” model, as we learned in question 5. Then we have in-app purchases as the second most popular business model with 35%, and following are the paid apps. When comparing this result to which platforms(Android, iOS, Windows) they have experience in developing for and prefer the result is roughly the same. Weirdly most of those who chose in-app purchases do not have experience with implementing it. This could be because we did not specify in the question that they should have experience with the business model the choose. But this still indicates that many like and prefer the freemium model in apps in general.

	Don't care	Not important	Less important	Important	Very important	Average Rating
I want to pay as little as possible to develop and sell my app.	5.88%	11.76%	23.53%	47.06%	11.76%	3.47
I want to sell my app in the official app store.	5.88%	2.94%	5.88%	44.12%	41.18%	4.12
I want to make money on my app.	2.94%	8.82%	38.24%	35.29%	14.71%	3.50
I want an easy development process.	5.88%	2.94%	5.88%	38.24%	47.06%	4.18
I want good guides and documentation.	2.94%	2.94%	8.82%	35.29%	50%	4.26
I want an active community to give/receive help.	2.94%	2.94%	14.71%	41.18%	38.24%	4.09
I want to be able to receive high quality support from official technicians.	11.76%	35.29%	35.29%	14.71%	2.94%	2.62

Figure 21: The table which shows how important each property/feature is to the participants

With question 7 and 8 we hoped to find exactly what features and properties are important to the developers when choosing which platform and API to use. As we mentioned earlier the question was based on several statements in which the participants had to rank the statements based on how important they are to them. Based on this data we can see the statement that got the highest score was ”I want good guides and documentation”, which means that to the average developer in our survey this is the most important property when developing an app. The least important statement was support from official technicians. Other properties that scored quite high were easy development and being able to sell their app in an official app store.

We wouldn't have guessed that making money would have such a low score, but as hobby developers they might favour the learning experience and other reasons instead of making money.

One interesting thing to look for in this question is to compare how important these issues are based on their years of programming experience. Unfortunately some of our participants skipped the last questions, so at question 7 we only had 4 people with 1-3 years of experience, 12 people with 3-5 years of experience and 18 people with 5 years or more of experience. The wish for an easy development process was favoured most by the people with 3-5 years of experience with a score of 4.33, while not surprisingly the people with 5 years or more didn't favour it as much, but still gave it a pretty high score of 4.06. We also found out that good documentation and guides is more important the more experience you've got as we can see in figure 22. This was the opposite of we expected, since we thought the people with less experience would consider good documentation as more important.

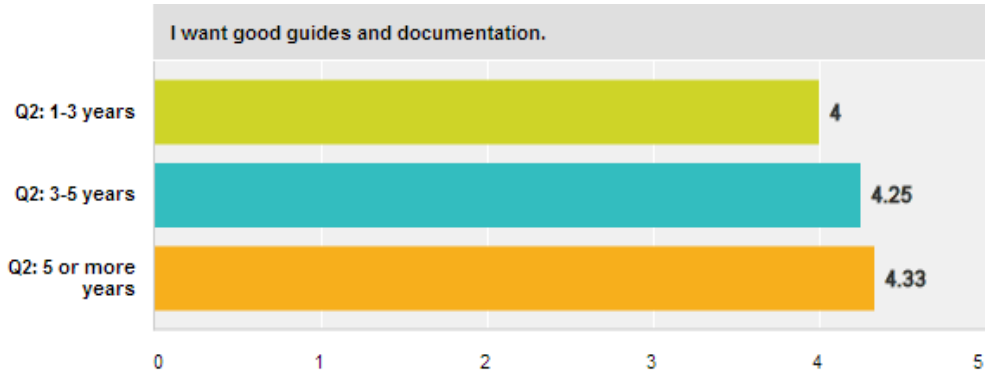

Figure 22: How important good guides and documentation are for the different programmers

	Don't care	Not important	Less important	Important	Very important	Average Rating
I want to spend the least amount of time implementing the in-app payment.	14.71%	5.88%	14.71%	50%	14.71%	3.44
I want a fast and easy way to set up and configure my in-app purchases.	14.71%	0%	20.59%	47.06%	17.65%	3.53
I want to be able to quickly test my in-app store/purchases.	11.76%	0%	5.88%	64.71%	17.65%	3.76
I want to be able to test in an emulator.	11.76%	11.76%	20.59%	26.47%	29.41%	3.50
I wish for my customers to be able to pay over carrier billing/phone bill.	26.47%	23.53%	11.76%	26.47%	11.76%	2.74
I want an easy way to handle refunds.	14.71%	2.94%	17.65%	44.12%	20.59%	3.53
I want my customers to have an easy process completing a purchase.	8.82%	0%	0%	26.47%	64.71%	4.38
I want there to be mandatory security mechanisms in place(Authentication, encryption, access control etc)	8.82%	2.94%	2.94%	41.18%	44.12%	4.09
I want to easily configure the security in my app(Authentication, encryption, access control etc).	8.82%	2.94%	14.71%	32.35%	41.18%	3.94

Figure 23: The table which shows how important each property/feature is to the participants, questions more focused on in-app purchase

Question 8 was similar to Q7, except it focuses more on features and properties for in-app payment. So based on this question we want to find what properties are most important when picking an in-app payment API, and from it we can derive the most common requirements. When looking at the ones that got the highest average score we see that there are a few that stand out. The highest ranked was ”I want my customers to have an easy process completing a purchase” with a score of 4.38, ”I want there

to be mandatory security mechanisms in place (Authentication, encryption, access control etc)" with 4.09 and "I want to easily configure the security in my app(Authentication, encryption, access control etc)" with 3.94. The statement with the lowest score was "I wish for my customers to be able to pay over carrier billing/phone bill" with 2.74.

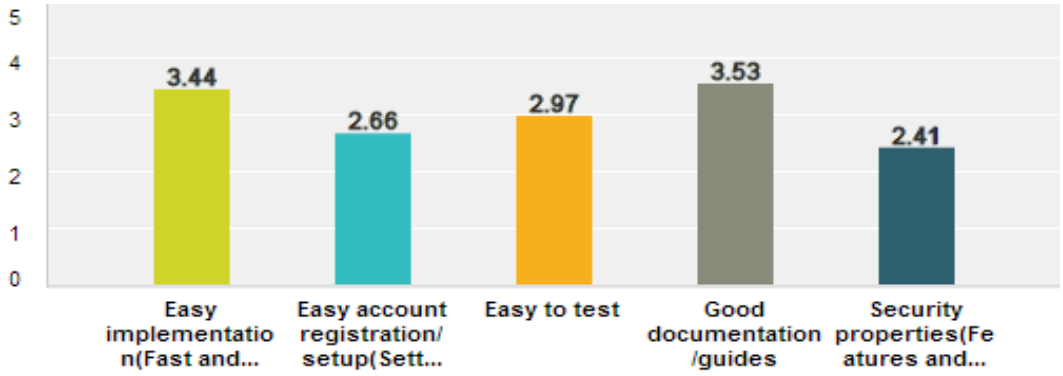

Figure 24: The chart which displays the important properties for the developers

With these questions answered, we begin to see what most hobby developers prefer in terms of requirement and properties. Now, to get an indication of what is the most important, we ask the participants to rank and prioritize some statements. Based on what we learned on Q7, there was no surprise that the statement that ranked the highest was Good documentation and guides, with an average ranking of 3.54. Following right behind is Easy implementation with 3.44. What's interesting is that even though security properties were ranked as one of the most important features in Q8, it was the lowest prioritized in Q9. You can see the rest of the results from Q9 in figure 24.

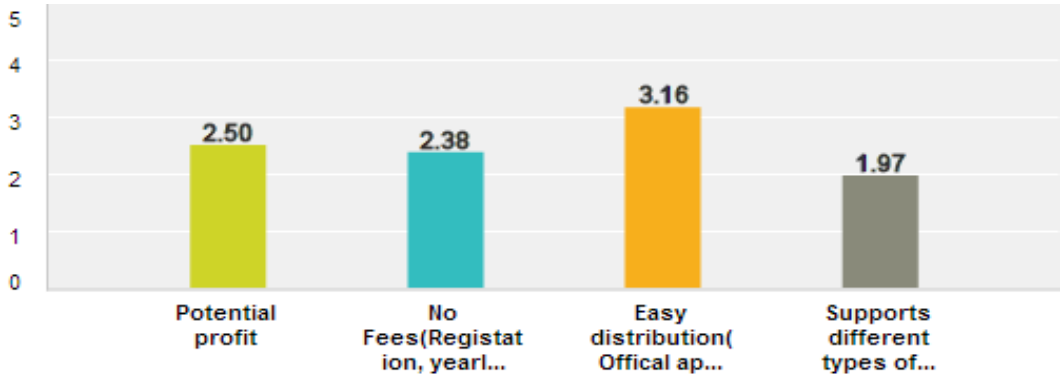

Figure 25: The chart which displays the important properties for the developers

In question 10 we wanted to find out a bit more about the thoughts on the economic aspect. So here we wanted the participants to prioritize which economic property was the most important. We defined the properties as potential profit, fees, distribution and support for different business models. As you can see on figure 25 highest prioritized property was easy distribution through official app stores. We expected to see no fees ranked high, but instead potential profit was ranked higher, even though with just a small margin.

We wanted to see if any of these findings changed if we took a deeper look into Q7-10 based on experience. Here we uncovered some interesting changes. For some reason people with 1-3 years of experience have a lower priority on good documentation and guides as we can see in figure 26, and a much higher priority on security properties as shown in figure 27. This could because of the low sample size, as we can't seem to find any reasonable reasons for this change. One would think that documentation is more important for developers with less experience. We also see a less focus on potential profit and fees. You can see this in figure 28 and 29. For experience and preferred platform we didn't see much difference in the data.

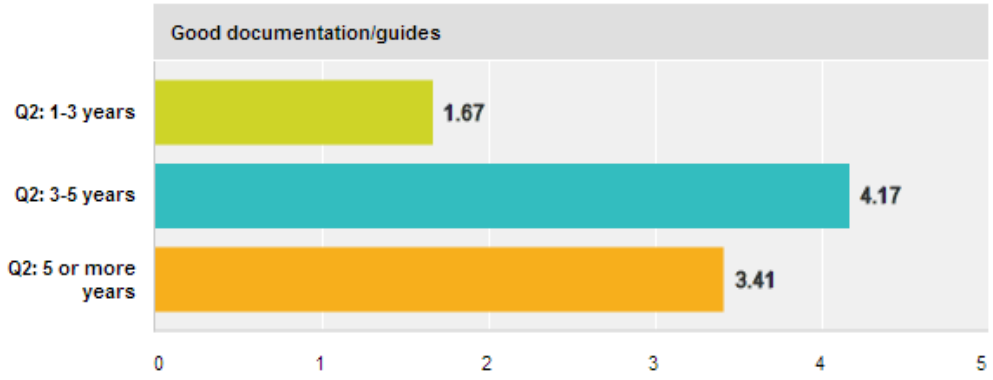

Figure 26: How highly prioritized good guides and documentation are for the different programmers

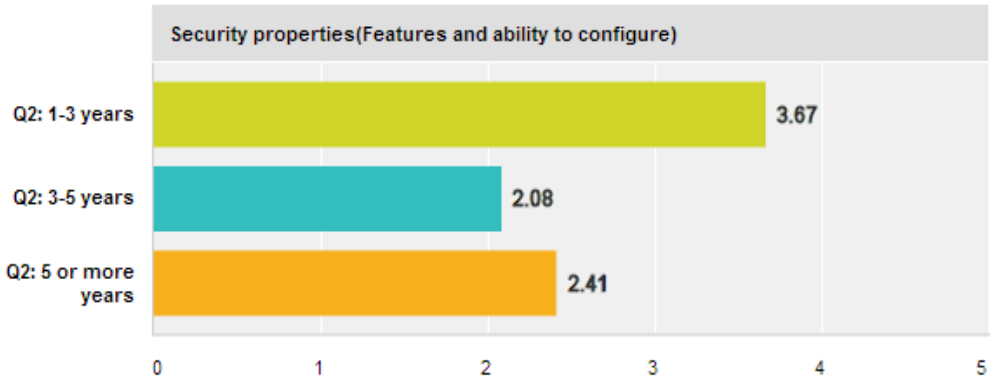

Figure 27: How highly prioritized security properties are for the different programmers

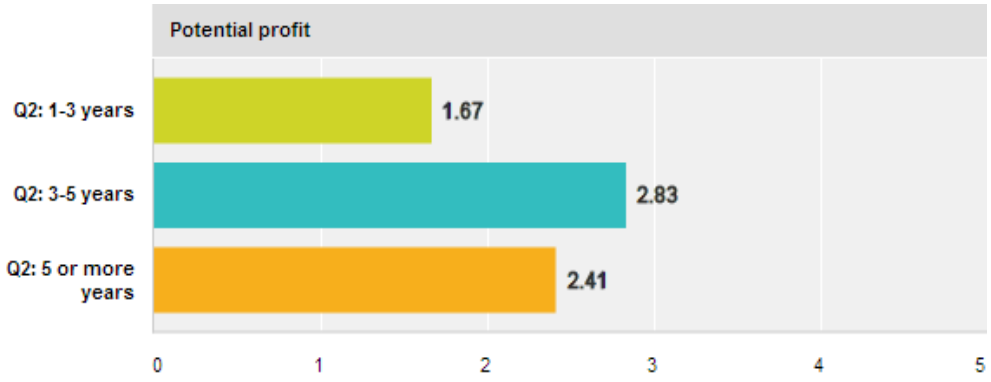

Figure 28: How highly prioritized potential profits are for the different programmers

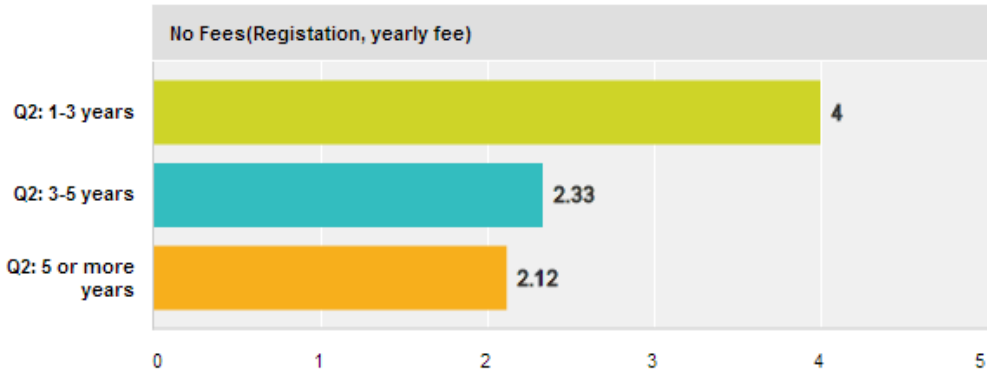

Figure 29: How highly prioritized no fees are for the different programmers

12 In-app purchases in the future

Earlier in this paper we mentioned ABI Research's forecast for in-app purchases and how they predict an increase in revenue from in-app purchases. Research company Gartner said in a press release in September 2012 that they also foresee an increase in revenue. Their forecast is that in-app purchases will by 2016 be responsible for 41% of all revenue in the app stores based on today's market, and that 93% of all app downloads will be from free apps [73]. These numbers are for several large app stores across different platforms.

We share the belief that there will still be an increase of in-app purchases. We also expect to see a move from mostly games to other categories of apps using the freemium model. An example is news apps. Many newspapers have implemented a "premium" feature on their websites where readers pay a small sum to get access to premium-only articles. Similar services will be offered in apps based on subscriptions and in-app purchases. But we should mention that there's been an increasing discontent with freemium model on the web. Some consumers don't like the fact this business model is designed to make people purchase additional content and drop some coins in your in-app store. How this will affect the future of in-app payment is uncertain, but if it should gain enough supporters it could lead to some changes.

Another much talked about technology is HTML5 and its capability to produce cross-platform applications. How this will affect the native apps and app market is still highly uncertain. This can change the whole balance of the world of applications. Separation from the native apps open a lot of new possibilities for handling in-app purchases as well, and might knock the big companies down from their throne. This can be the opening third-party payment providers like PayPal need to really enter the app market. However, Apple, Microsoft and Google won't sit idle by and wait for this change. Currently the native apps have a very strong foothold and offer features and performance which HTML5 can't match. So most likely the balance won't change in the nearest future, but what the future holds is hard to say.

There is one more player on the playing field which we haven't talked much about yet. BitCoin has been in the wind lately and have gotten much attention from the media as an alternative currency in digital form. There are already apps that use BitCoins as currency, but since BitCoins is not that easy to use yet, and very few non-technical or persons who are especially interested in it have an account or use it, we don't see it being used in many apps quite yet. Though it is an interesting thought, and maybe it will grow in popularity over the next few years.

13 Conclusion

When we started our thesis we first made ourselves familiar with the topic of in-app payment, looking for articles and collecting information about the subject in general. We then wrote about Google, Apple, PayPal and Windows and gathered all information we could use in preparation of our comparison model. We took a look at the business aspect of in-app payment where we listed some of the business models that are available today along with some thoughts about which one you should choose depending of the scope of your application and ambitions. It didn't take us long before we started developing apps for the different platforms, in order to gain some experience and in-depth knowledge on the development processes and how it is to implement in-app purchase for an application. We decided to split our comparison model up into two models. One technical part and one business part. We then proceeded to evaluate the chosen service providers and their in-app purchase services. In section 10 we finally compared the service providers based on the results from our evaluation in section 9. We needed some real input on what the developers themselves considered as the most important aspects when choosing a platform and a in-app payment solution, so we created a survey with questions based on our comparison model. We took a deeper look into the data that this survey gave us, and found out a couple interesting things which could help us conclude our research questions.

We discovered throughout our survey what kind of properties the developers wanted and prioritized when choosing a platform and a provider of in-app payment solutions. We gave the participants a series of statements they had to give a score. After that we asked them to prioritize them, which gave us some insight into what's most important if they had to chose one. Based on this we could see some indications of what the average hobby developer found to be the most important properties. Easy distribution, good documentation and guides along with an easy development process was some of the features which the participants of our survey prioritized as the most important ones. Some properties such as security and testing were also important, but when given the choice to prioritize one over the other, they fell short in being at the top. Google offers the simplest and most effective way to distribute your app since you can easily use the official app store and at the same time due to Android's open-source platform, be able to quickly get your application on device without any restrictions. It's hard to chose a winner when it comes to documentation and guides, Google and Apple's guides are easily navigable and of high quality in general. The easiest implementation was without a doubt PayPal's solution, there wasn't much code to write in order for it all to work and it was easy to test, sadly it can't be distributed through official

app stores.

The process around the in-app purchase itself makes it hard to give the winning title to a single provider, we have to look at it from a customers point of view in terms of user-friendliness but also from the developers perspective. Purchasing content is simple on all platforms except PayPal where you have to manually enter your information each time, but Google supports an easier refund process alongside with the support of carrier billing which makes it more flexible for the customer. Testing in-app purchase is simpler on iOS since you can do it via the emulator. They all have pros and cons, but when choosing a platform and a provider, one must first look at what's important and based on that which solution is best suited based on those prioritizations.

Based on the result of our survey, Google's solution fits the average hobby developer the best - it's low cost, the documentation is good, it's familiar for students who learn Java first and overall it just ended up ahead on the issues which the survey participants deemed as the most important ones when compared to our comparison model. Apple's has the biggest potential for the developer to make a profit, their documentation is extensive and their focus on security is solid, but the high fees and expenses in addition to the slightly more difficult development process puts it just behind Google. PayPal's low cost, easy implementation and high profit per sale is great, but this doesn't make up for the lack of an easy way to distribute your app through an official channel. Windows Phone can also be costly, lacks in documentation and the market size for potential profit is lower when compared to Apple and Google. However, based on the survey the people who have developed for Windows Phone seem to prefer it.

13.1 Future work

As for the validity of this paper we believe we're lacking the data in terms of sample size and diversity in the survey to be able to fully generalize our results, and for future work a more comprehensive user survey should be conducted. We added Windows Phone at a later stage in our writing, and since we didn't have time to create an app for it due to time restriction and simply not owning a Windows Phone device, we felt it didn't have the same basis to evaluate it as we did with the others. The platform is still on the rise and should be considered for future work.

One could also take a look at in-app payment and what results you get from surveys conducted on developers outside of Norway. The potential of carrier billing might also be interesting to look into, countries whose inhabitants might lack the means to pay with credit cards might open up a big market for new customers if they are able to pay for apps and in-app content

with their phone bill.

References

[1] M. Labs, "Mcafee threats report: Third quarter 2012." http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2012.pdf?cid=BHP012, 2012. Accessed: 2013-05-14.

[2] "F-secure mobile threat report q3 2012." http://www.f-secure.com/static/doc/labs_global/Research/Mobile%20Threat%20Report%20Q3%202012.pdf, 2012. Accessed: 2013-05-14.

[3] G. Goggin, *Cell phone culture: Mobile technology in everyday life*. Routledge, 2012. ISBN: 978-0415367448.

[4] "World has about 6 billion cell phone subscribers, according to u.n. telecom agency report." http://www.huffingtonpost.com/2012/10/11/cell-phones-world-subscribers-six-billion_n_1957173.html, 2012. Accessed: 2013-05-15.

[5] "Gartner says asia/pacific led worldwide mobile phone sales to growth in first quarter of 2013." http://www.gartner.com/newsroom/id/2482816, 2013. Accessed: 2013-05-16.

[6] "Nielsen - the mobile consumer, a global snapshot." http://www.nielsen.com/content/dam/corporate/us/en/reports-downloads/2013%20Reports/Mobile-Consumer-Report-2013.pdf, 2013. Accessed: 2013-05-15.

[7] "Smartphone shipments forecast to rise on growth in asia-pacific." http://focustaiwan.tw/news/aeco/201212240014.aspx, 2013. Accessed: 2013-05-15.

[8] "App store tops 40 billion downloads with almost half in 2012." http://www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html, 2013. Accessed: 2013-05-14.

[9] M. Lytras, E. Damiani, and P. de Pablos, *Web 2.0: The Business Model*. Springer Science+Business Media, LLC, 2009. ISBN: 9780387858951.

[10] ABIresearch, "Mobile application business models." http://www.abiresearch.com/research/product/1009105-mobile-application-business-models/. Accessed: 2013-05-14.

[11] A. Insider, "In-app purchases from 'freemium' titles account for 71% of iphone app revenue." http://appleinsider.com/articles/13/03/29/in-app-purchases-from-freemium-titles-account-for-71-of-iphone-app-revenue, 2013. Accessed: 2013-05-14.

[12] O. W. Linzmayer, *Apple Confidential: The Real Story of Apple Computer, Inc.* No Starch Press, 1999. ISBN: 1-886411-28-X.

[13] D. E. Dilger, "1990-1995: Why the world went windows." http://www.roughlydrafted.com/RD/Q4.06/3EC02E78-FD4D-4CDF-92A0-9C4CBDFAB3D2.html, 2006. Accessed: 2013-05-16.

[14] A. Cantrell, "Apple's remarkable comeback story." http://money.cnn.com/2006/03/29/technology/apple_anniversary/?cnn=yes, 2006. Accessed: 2013-05-16.

[15] A. P. Info, "iphone premieres this friday night at apple retail stores." http://www.apple.com/pr/library/2007/06/28iPhone-Premieres-This-Friday-Night-at-Apple-Retail-Stores.html, 2007. Accessed: 2013-05-16.

[16] R. Flandez, "Programmers jockey for iphone users at apple site." http://online.wsj.com/article/SB121789232442511743.html?mod=googlenews_wsj, 2008. Accessed: 2013-05-16.

[17] B. Elgin, "Google buys android for its mobile arsenal." http://www.webcitation.org/5wk7sIvVb, 2005. Accessed: 2013-05-14.

[18] "Gartner says worldwide mobile phone sales declined 1.7 percent in 2012." http://www.gartner.com/newsroom/id/2335616, 2013. Accessed: 2013-05-14.

[19] A. D. Blog, "In-app billing version 3." http://android-developers.blogspot.no/2012/12/in-app-billing-version-3.html, 2012. Accessed: 2013-05-14.

[20] "Paypal finacials." https://www.paypal-media.com/mediacenter.cfm, 2013. Accessed: 2013-05-16.

[21] "The world's biggest public companies." http://www.forbes.com/global2000/list/#page:1_sort:0_direction:asc_search:_filter:Software%20%26%20Programming_filter:All%20countries_filter:All%20states, 2013. Accessed: 2013-05-14.

[22] "The history of windows ce." http://www.hpcfactor.com/support/windowsce/wce3.asp, 2012. Accessed: 2013-05-14.

[23] J. Evers, "Microsoft to phase out pocket pc, smartphone brands." http://www.infoworld.com/d/hardware/microsoft-phase-out-pocket-pc-smartphone-brands-232. Accessed: 2013-05-14.

[24] D. Koh, "Q&a: Microsoft on windows phone 7 series." http://asia.cnet.com/qanda-microsoft-on-windows-phone-7-series-62061278.htm. Accessed: 2013-05-14.

[25] "Windows phone store." http://www.windowsphone.com/nb-no/store. Accessed: 2013-05-14.

[26] B. Bishop, "Samsung reports q4 2012 financials:$8.27 billion in operating profit on $52.45 billion in revenue." http://www.theverge.com/2013/1/24/3912972/samsung-reports-q4-2012-financials-8-27-billion-in-operating-profit, 2013. Accessed: 2013-05-16.

[27] J. Kincaid, "Amazon's android app store launches: Test drive apps directly from your browser." http://techcrunch.com/2011/03/22/amazon-android-app-store-3/, 2011. Accessed: 2013-05-16.

[28] H. Koekkoek, "Distimo publication full year 2011." http://www.distimo.com/publications/archive/Distimo%20Publication%20-%20Full%20Year%202011.pdf, 2011. Accessed: 2013-05-14.

[29] R. Kim, "Freemium app revenue growth leaves premium in the dust." http://gigaom.com/2012/10/26/freemium-app-revenue-growth-leaves-premium-in-the-dust/. Accessed: 2013-05-14.

[30] "Apple terms and condotions." http://www.apple.com/legal/itunes/us/terms.html. Accessed: 2013-05-14.

[31] R. Srinivasan, "Freemium has run its course." http://gigaom.com/2012/07/21/freemium-has-run-its-course/, 2012. Accessed: 2013-05-14.

[32] M. Lytras, E. Damiani, and P. de Pablos, *The Business of iPhone and iPad App Development*. Apress, 2011.

[33] D. A. Kristina Shampan'er, "Zero as a special price: The true value of free products." http://web.mit.edu/ariely/www/MIT/Papers/zero.pdf, 2007. Accessed: 2013-05-14.

[34] R. Srinivasan, "Let us do expected value math on $0 price." http://iterativepath.wordpress.com/2010/11/17/let-us-do-expected-value-math-on-0-price/, 2010. Accessed: 2013-05-14.

[35] D. Lamppa, "5 options for distributing your ios app to a limited audience (legally)." http://mobiledan.net/2012/03/02/5-options-for-distributing-ios-apps-to-a-limited-audience-legally/, 2012. Accessed: 2013-05-27.

[36] "Apple updates ios to 6.1." http://www.apple.com/pr/library/2013/01/28Apple-Updates-iOS-to-6-1.html, 2013. Accessed: 2013-05-14.

[37] G. J. Spriensma, "Distimo 2012 - year in review." http://fortunebrainstormtech.files.wordpress.com/2012/12/distimo-publication-full-year-2012.pdf, 2012. Accessed: 2013-05-14.

[38] T. Brant, "In app purchase: A full walkthrough." http://troybrant.net/blog/2010/01/in-app-purchases-a-full-walkthrough/, 2010. Accessed: 2013-05-14.

[39] T. . G. Mylonas, Dritsas, "Smartphone security evaluation - the malware attack case." http://www.aueb.gr/users/amylonas/docs/secryptShort.pdf, 2011. Accessed: 2013-05-27.

[40] C. Halbronn and J. Sigwald, "iphone security model & vulnerabilities." http://esec-lab.sogeti.com/dotclear/public/publications/10-hitbkl-iphone.pdf, 2010. Accessed: 2013-05-27.

[41] C. Show, "Parents sue apple over apps that let children spend real money while playing games." http://www.dailymail.co.uk/news/article-2133021/Parents-sue-Apple-apps-let-children-spend-real-money-playing-games.html#ixzz2TH0eHqPw, 2012. Accessed: 2013-05-14.

[42] "Google play developer program policies." http://play.google.com/intl/en/about/developer-content-policy.html. Accessed: 2013-05-14.

[43] "Android marks fourth anniversary since launch with 75.0third quarter, according to idc." https://www.idc.com/getdoc.jsp?containerId=prUS23771812, 2012. Accessed: 2013-05-14.

[44] B. Womack, "Google says 700,000 applications available for android." http://www.businessweek.com/news/2012-10-29/google-says-

700-000-applications-available-for-android-devices, 2012. Accessed: 2013-05-14.

[45] "Implementing in-app billing." http://developer.android.com/google/play/billing/v2/billing_integrate.html. Accessed: 2013-05-14.

[46] "Stackoverflow." http://stackoverflow.com/. Accessed: 2013-05-14.

[47] T. R. M. E. X. W. D. S. Daniel Reynaud, Eui Chul Richard Shin, "Freemarket: Shopping for free in android applications." http://droidblaze.cs.berkeley.edu/freemarket.pdf, 2011. Accessed: 2013-05-27.

[48] H. Lockheimer, "Android and security." http://googlemobile.blogspot.no/2012/02/android-and-security.html. Accessed: 2013-05-14.

[49] "Amazon appstore." http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011. Accessed: 2013-05-14.

[50] "Handster." http://www.handster.com/. Accessed: 2013-05-14.

[51] "Getjar." http://www.getjar.com/. Accessed: 2013-05-14.

[52] "National vulnerability database." http://nvd.nist.gov/. Accessed: 2013-05-14.

[53] B. Weitzenkorn, "Android malware more than doubled worldwide in 2012." http://www.technewsdaily.com/17817-android-malware-doubles.html, 2013. Accessed: 2013-05-20.

[54] "Windows phone store." http://www.windowsphone.com/en-us/store. Accessed: 2013-05-14.

[55] http://msdn.microsoft.com/library/windows/apps/jj193593. Accessed: 2013-05-14.

[56] "Creating trial apps for windows phone." http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff967558(v=vs.105).aspx, 2013. Accessed: 2013-05-14.

[57] B. Zamora, "Increase monetization by adding in-app purchase to your apps." http://blogs.windows.com/windows_phone/b/wpdev/archive/2012/11/09/increase-monetization-by-adding-in-app-purchase-to-your-apps.aspx, 2012. Accessed: 2013-05-14.

[58] "Company app distribution for windows phone." http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206943(v=vs.105).aspx, 2013. Accessed: 2013-05-14.

[59] "Chevronwp7." http://www.chevronwp7.com/. Accessed: 2013-05-14.

[60] T. Brix, "Reflecting on 2012: scale and oppurtunity." http://blogs.windows.com/windows_phone/b/wpdev/archive/2012/12/26/reflecting-on-2012-scale-and-opportunity.aspx, 2012. Accessed: 2013-05-14.

[61] "Windows phone api reference." http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626516(v=vs.105).aspx, 2013. Accessed: 2013-05-14.

[62] "Windows phone community." http://dev.windowsphone.com/en-us/community. Accessed: 2013-05-14.

[63] "Reddit windows phone 7 dev subreddit." http://www.reddit.com/r/wp7dev. Accessed: 2013-05-14.

[64] "Windows phone 8 security and ecryption." http://www.windowsphone.com/en-us/business/security. Accessed: 2013-05-14.

[65] "Apple's app store marks historic 50 billionth download." http://www.apple.com/pr/library/2013/05/16Apples-App-Store-Marks-Historic-50-Billionth-Download.html, 2013. Accessed: 2013-05-21.

[66] S. Mitha, "Google i/o 2013 day 1: All you need to know." http://www.thinkdigit.com/Internet/Google-IO-2013-Day-1-All-you_14664.html, 2013. Accessed: 2013-05-21.

[67] M. Stroh, "Windows phone blog." http://blogs.windows.com/windows_phone/b/windowsphone/archive/2013/05/10/the-wait-is-over-lumia-928-for-verizon-wireless-launches-may-16-for-under-100.aspx, 2013. Accessed: 2013-05-21.

[68] "Gartner says worldwide sales of mobile phones declined 3 percent in third quarter of 2012; smartphone sales increased 47 percent." http://www.gartner.com/newsroom/id/2237315, 2012. Accessed: 2013-05-21.

[69] "Gartner says worldwide sales of mobile phones declined 2.3 percent in second quarter of 2012." http://www.gartner.com/newsroom/id/2120015, 2012. Accessed: 2013-05-21.

[70] "Gartner says worldwide sales of mobile phones declined 2 percent in first quarter of 2012; previous year-over-year decline occurred in second quarter of 2009." http://www.gartner.com/newsroom/id/2017015, 2012. Accessed: 2013-05-21.

[71] "Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth." http://www.gartner.com/newsroom/id/1924314, 2012. Accessed: 2013-05-21.

[72] "Surveymonkey." http://www.surveymonkey.com/. Accessed: 2013-05-14.

[73] Gartner, "Gartner says free apps will account for nearly 90 percent of total mobile app store downloads in 2012." http://www.gartner.com/newsroom/id/2153215, 2012. Accessed: 2013-05-14.

A User Survey

Mobile development and In-app purchases

***1. What's your current occupation?**

- ○ Student
- ○ Employed with an IT-position
- ○ Employed with a non IT-position
- ○ Unemployed

***2. How many years of experience do you have with programming?**

- ○ Less than a year
- ○ 1-3 years
- ○ 3-5 years
- ○ 5 or more years

***3. Which of these platforms have you developed for?**

- ☐ iOS
- ☐ Android
- ☐ Windows Phone
- ☐ Other
- ☐ None

***4. Which platform do you prefer to develop for?**

- ○ iOS
- ○ Android
- ○ Windows Phone
- ○ Other
- ○ Don't know

***5. Which of the following business models have you implemented in an app?**

- [ ] Paid app
- [ ] Free app with ads
- [ ] Free app with in-app purchases
- [ ] Subscription based app
- [ ] Free app
- [ ] None

***6. Which of the following is your preferred business model in mobile apps?**

- [ ] Paid app
- [ ] Free app with ads
- [ ] Free app with in-app purchases
- [ ] Subscription based app
- [ ] Free app
- [ ] None

***7. How would you grade the following statements on a grade of importance?**

	Don't care	Not important	Less important	Important	Very important
I want to pay as little as possible to develop and sell my app.	○	○	○	○	○
I want to sell my app in the official app store.	○	○	○	○	○
I want to make money on my app.	○	○	○	○	○
I want an easy development process.	○	○	○	○	○
I want good guides and documentation.	○	○	○	○	○
I want an active community to give/receive help.	○	○	○	○	○
I want to be able to receive high quality support from official technicians.	○	○	○	○	○

***8. How would you grade the following statements on a grade of importance?**

	Don't care	Not important	Less important	Important	Very important
I want to spend the least amount of time implementing the in-app payment.	○	○	○	○	○
I want a fast and easy way to set up and configure my in-app purchases.	○	○	○	○	○
I want to be able to quickly test my in-app store/purchases.	○	○	○	○	○
I want to be able to test in an emulator.	○	○	○	○	○
I wish for my customers to be able to pay over carrier billing/phone bill.	○	○	○	○	○
I want an easy way to handle refunds.	○	○	○	○	○
I want my customers to have an easy process completing a purchase.	○	○	○	○	○
I want there to be mandatory security mechanisms in place(Authentication, encryption, access control etc)	○	○	○	○	○
I want to easily configure the security in my app(Authentication, encryption, access control etc).	○	○	○	○	○

＊9. Rank the following properties in order of importance to you.

- Easy implementation(Fast and easy development)
- Good documentation/guides
- Security properties(Features and ability to configure)
- Easy to test
- Easy account registration/setup(Setting up SDK, certificates, bank information, etc.)

＊10. Rank the following properties in order of importance to you

- Easy distribution(Offical app store vs. other channels)
- Supports different types of business models(Ads, paid, inapp, etc)
- No Fees(Registation, yearly fee)
- Potential profit