Store submission automation: effects of user centred design on organizational learning

Samu Mörsky

Master's Thesis

May 2020

DEPARTMENT OF FUTURE TECHNOLOGIES
UNIVERSITY OF TURKU

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

Samu Mörsky: Store submission automation: effects of user-centred design on organisational learning
Pro gradu -tutkielma 54 s.
Tietojenkäsittelytiede
Toukokuu 2020

Tässä tutkielmassa perehdytään yrityksen sisäiseen käyttäjäkeskeisesti suunniteltuun automaatiotyökaluun, jonka tarkoituksena on lähettää sovellustiedostoja ja markkinointiassetteja Applen sekä Googlen digitaalisiin sisältöpalveluihin. Tarkoituksena on selvittää, onko käyttäjkeskeisellä suunnittelulla vaikutusta organisaation oppimiseen.

Käyttäjäkeskeisessä suunnittelussa lähtökohtana on käyttäjien tarpeiden ymmärtäminen sekä vaatimusten määrittely. Organisaation oppiminen puolestaan viittaa organisaation sisällä tapahtuvaan informaation tai osaamisen kasvamiseen, joko ryhmässä tai henkilökohtaisella tasolla. Näiden kahden kohtaamisesta ei lähdehaun perusteella löytynyt aikaisempaa tutkimusta. Tutkielmassa käydään läpi yksityiskohtaisesti organisaation rakenne, digitaalisten sisältöpalveluiden asettamat vaatimukset sekä miten toteutettiin työkalu vastaamaan tämän luomaa kysyntää, jotta aiheen ja haastatteluiden tulkitseminen tässä kontekstissa olisi mahdollisimman yksityiskohtaista.

Tämä tutkielma käyttää lähtökohtanaan laadullisen tutkimuksen menetelmäsuuntausta. Tutkimuksen yhteydessä haastateltiin käyttäjiä yrityksestä, jolle kyseinen työkalu toteutettiin. Haastatteluiden perusteella ei löydetty suoraa korrelaatiota käyttäjäkeskeisen suunnittelun ja organisaation oppimisen välillä. Pieniä viitteitä niiden välillä kuitenkin löytyi, jonka perusteella lisätutkimus aiheesta olisi paikallaan.

Asiasanat:
Käyttäjäkeskeinen suunnittelu, Organisaation oppiminen, Automaatio, Digitaaliset sisältöpalvelut

UNIVERSITY OF TURKU
Department of Future Technologies

Samu Mörsky: Store submission automation: effects of user-centred design on organisational learning
Master's Thesis 54 p.
Computer science
May 2020

In this thesis, we study an automation tool implemented using user-centred design-paradigm. The aim of this thesis is to study how the user-centred design affects organisational learning. The tool is used for uploading application packages and marketing assets to the Apple and Google digital distribution services.

User-centred design focuses on understanding user's tasks and requirements. Organisational learning is used to describe the learning that happens inside an organisation on individual or group level, which helps the organisation to accumulate long lasting knowledge. In the initial literary search, there was no earlier research focusing on this particular question. In this thesis, we will go through in detail the organisational structure, the requirements set for the digital distribution services and the implementation of the automation tool for this case. This will enable us to scrutinize the interviews and results in this context.

This thesis was carried out using qualitative research methodology. Interviews were conducted with users from the company for which the tool was implemented. These interviews seem to bear no strong correlation between organisational learning and user-centric design. However, the results indicate for the research question that further inspection to the subject could be worthwhile.

Keywords:
User-centric design, Organisational learning, Automation, Digital distribution services

Contents

1. Introduction

In modern world where humans have to interact with increasing amount of digital services and devices, it is important to design interfaces that are, first and foremost, easy to use but also designed for the tasks it must accomplish. User-centric design (UCD) aims to achieve just that. However, it is important to consider what happens to the knowledge that users used to have before digital devices and services were made easy to use. For example, in the early 1990s in order to be able to use computer, user had to obtain knowledge how to use Microsoft MS-DOS (Microsoft Disk Operating System) in order to operate Windows. Nowadays, less and less people need to know how to operate command-line interfaces. With this in mind, it is important to consider what happens to the knowledge that used to be required for accomplishing certain tasks.

Automation is used nowadays to help us humans to achieve tasks that required a lot of manual work or calculating capacity. While the knowledge to accomplish these tasks is required in order to automate them, the users that used to do them might not be required to have this knowledge. In such case, it is fair to assume that the knowledge that many used to possess is transferred to the few that is in charge of designing and implementing the automated system. This in turn might have some effects on organizational learning (OL) that is used to describe the learning that happens inside an organisation. Considering this, I wanted to research the question that does implementing these automated systems using UCD paradigm, have some kind of effect on retaining knowledge regarding a task and organizational learning.

To reach conclusions we will go through the theoretical background regarding UCD and OL. In Chapter 3, we will go through the organizational structure and how the process was before automation for it was implemented to understand the need for this tool. Chapter 4 gives insight to the Apple and Google digital distribution services, and what requirements they have for uploading application packages and marketing assets. Additionally, we will go through the tools available for automating these tasks

in Chapter 5. Chapter 6 describes the user interface, as well as, the tool implemented for automating asset uploading tasks. Finally, we will go through the interviews and reach the conclusions in the Chapters 7 and 8.

2. Theoretical background

In this chapter, we will go through the theoretical background for human-centred
design and organisational learning. We will also look at the inclusion criteria for the interviews conducted with the users of the automated submission system of this thesis.

2.1 Human-centred design

Human-centred design (HCD), also known as user-centred design (UCD), is a problem-solving approach for interactive systems, where the goal is to develop a solution to a problem by involving human (user) perspective to every step of the problem-solving process. Contrary to earlier when design was widely driven by the paradigm that users can adapt to emerging technology, now designers and developers are paying closer attention to building intuitive and easy to use systems. Although UCD and HCD are nowadays used interchangeably, some consider there to be few semantic differences. While UCD, a term coined by Donald A. Norman in his book *User-Centred System Design: New Perspectives on Human-Computer Interaction*, is focusing on the target users and their tasks, HCD is considered to be more general way of interacting with the human nature and finding solutions that conform not only to particular users, but humans as a whole. [29] With that acknowledged, in this thesis these terms are used as synonyms.

The aim of human-centred design is to make the system as usable as possible for the end-users. This is enhanced by interacting with the end-users, brainstorming and going through the requirements with them. This includes gathering feedback from users while development may still be going on. An essential requirement for designing HCD solutions is to define the context of use of the feature or product

under design. [1] In practice, this means documenting the potential users and their tasks. Human-centred approach is usually integrated into other development processes, such as rapid prototyping, so that it becomes a seamless part of development cycle. Overall, the goal of integrating HCD is to constantly ensure good usability of the output. Important thing to note is that UCD is not just asking users for features, and then implementing them. Users usually have only partial understanding of their needs and how to design human computer interaction. [31]

The usability of a system that is not used frequently or every day is even more critical than those that are used all the time. It should be easy to remember how to use a program even if there is a long delay between the sessions. Additionally, when designing systems for certain age groups, it is essential to know that their needs are different. For example, older users might have problems with the same user experience (UX) standards that are widely used in modern applications. [28] As Stara et al. [30] suggest, it is especially important in this age group to try and facilitate the needs of the audience, as bad user experience can dissuade them from using technology. Moreover, mood of the user can affect their experience and actions while using a system. For example, a person feeling stressed or anxious is inclined to prefer a familiar experience using a system knowing what will happen given a certain action. In comparison, a person with a relaxed and happy mood is more likely to be more open minded about the situation and willing to be more tolerant about their experience.

There are of course different points of views for the human-centred design paradigm. One worrying aspect of human centred design, as Norman [2] proposes, is that focus on individual people or group of people tends to make the usability of the system worse for other people who might come to use it. It does make the usability better for the target group, but it might also be a good idea to consider what happens, if the user group changes or evolves during the lifetime of the system, as it usually is for successful products. Norman also proposes that users for whom the system was designed usually get so proficient in using it, that they might require different things from it than they did when they were beginners. The habit of listening to your

customers is always a good thing for developers but abiding too much for the requested features might end up making the system overly complex.

User experience of automated systems is vital. Understanding users' tasks is essential to automate them. While automation might remove certain workloads from the users, it also has a tendency, if not carefully planned, to create new communication and workloads for the end-users. [32] For this reason, it is beneficial to be mindful about the task under automation and consider what effects it might have for the users and their future tasks.

2.2 Organizational learning

Broadly speaking organizational learning (OL) refers to the information and knowledge that is accumulated inside an organization by the people who work there. [33] Part of it usually is managing that knowledge and trying to transfer it so that the information is not lost when a key employee leaves the company. Organization improves over time as it gains experience and through that experience it can gain and create knowledge. And if the knowledge management is not considered in an organization, it may lose knowledge when people leave company or change positions.

The most important thing about OL is not that the company can gain static knowledge but rather to learn continuously by generating new knowledge which the people of the organization can apply and transfer from one to another. OL is not only a group-wide learning, but also something that an individual inside this organization learns is considered learning by the organization. That individual can then choose to share that knowledge with other people. One of the key aspects to enable a culture of learning, within an organization, is to learn from managerial experiences. To promote OL, management must provide an infrastructure capable of supporting and documenting the learning inside an organization. [34] Common mistakes, made by organisations, are that they are not capable to learn from their past project failures. A beneficial practice for such cases is process called post-mortem where teams collect

data from project that was not successful and share it with the organisation accumulating knowledge for the whole organisation.

One major issue in OL comes from being able to convince individuals to share their knowledge. In hierarchical company structures, hoarding knowledge can be seen as a power move, as possessing certain knowledge can help individuals to further their careers, or at least cement their place inside an organization as an essential person. [35] For this reason it is of utmost importance for an organization to try to promote and reward knowledge sharing within its ranks. As Antunes et al. [36] conclude “an organization’s ability to use and leverage knowledge is highly dependent on its human resources, which effectively create, share, and use that knowledge.” Human resource management is of paramount importance in this ability to translate existing individual knowledge to results for organization.

The concept of OL is still not absolutely clear as there are many different descriptions of it. While that may be the case, it is widely agreed that OL is a process and the differences in the definitions may stem from the fact that there are multiple different kinds of organisational learning taking place on different levels of the organisation. [33] However, Sunassee and Vernon [33] propose following definition: “Organisational Learning is the way in which individuals in an organisation learn, from the approach they take to addressing a task-related challenge, to their understanding of how they should learn.”

2.3 Inclusion criteria for interviews

For the interview presented later in this thesis, I chose the interviewees on the basis that they have some experience in submitting new versions of games to the application marketplaces. Moreover, they preferably have been doing submissions before and after the submission tools were implemented. One of the interviewees was part of the central submission team responsible for all the submissions. The rest had assumed the responsibility of submissions when the organization shifted the process

to game teams. I also wanted to have some insight from people that are not that versed in using the submission tools. My aim was to try to gather some differing points of view from those who are using and from those who are not using the tools for submission to figure out why it was not used by all and how to get everyone to use them.

The goal of the interviews was first and foremost to gather feedback on how the usability of the subject automation system is currently working from the user perspective, and how it could be improved to better suit the needs of its user base. Also, we wanted to learn how the change in submission process introduced by automation has affected the organization and the transfer of the knowledge regarding submissions when the organization decided to shift the submission responsibility to game teams from central submission team. Additionally, we are also trying to find out how well tools that were designed with human-centred philosophy has retained its usability when the group of users it was designed for has changed.

3. Organization and the processes before the automation

In this chapter, we will go through the process of manual labour regarding the now automated workflow. We will also see the organizational model before the tool was developed. At the same time, this reflects how the company structure and workflow regarding submissions has evolved to this day.

3.1 Process before the automation

The submission of a new game package is usually orchestrated by the game producer. They would have the relevant people to provide necessary assets to the central submission team for the upcoming release. Assets were usually provided through Slack (cloud-based proprietary instant messaging platform) sending URLs and file system paths to where the assets were to be found. A central submission team would then add the submission to their queue of submissions. When prior submissions were done, the submission specialist would proceed to upload these assets to the marketplaces by hand. In practice, for the screenshot submission this would mean dragging images one at a time from the source folder to the image field in App Store Connect or Google Play console. This would be repeated for every size and language localization. Typical App Store Connect image submission can consist of 10 language localizations with up to 10 images for each of the 5 sizes, which would result to 300 images to submit. [3] According to the user interviews this process could take up to a whole working day, as each image would then be processed by the marketplace taking some time before the interface would be ready to process the next screenshot.

Text asset processing was considered a bit faster as there is only need for one localisation for every type of text. However, there are still multiple different texts to

submit. For Apple App store, one has the option to submit a description text that is used to summarize the application and its features for the user, keywords for better search optimization and release notes to describe what is new in the latest version of the application. For Google Play Store, one can similarly choose to provide a description text and release notes. Google Play also has an option for a short description text that is displayed in the listing and can be then expanded to see the longer main description. Still this process was described in the interviews as cumbersome, because the users had to do a lot of copy-pasting between the store console and the source material. The process of submitting texts is considerably faster than submitting screenshots but it is still a substantial amount of repetitive work for the users to do by hand.

In addition to previously mentioned assets, one would also have to submit an updated build for the application. In some cases, the producer might decide to change to a new icon, for example, to reflect some on-going seasonal campaign. Changing the icon is done from the web user interface (UI) in the respective marketplace and does not require any effort for localizations and different sizes. A build can be uploaded to the App Store Connect by sending it with XCode or Apple's application loader app which requires a Mac (OS X) computer to be used. For Google Play, uploading a build is more straightforward as you can upload it from the browser within the Google Play console. The build of course must be signed correctly and must conform to Google and Apple's guidelines. We are going to investigate those guidelines with more depth in Sections 4.1.1 and 4.2.1.

As mentioned earlier, according to the user interviews, this process of submitting assets manually could take a whole business day depending on the amount of assets. Often the assets would arrive late which, in turn, would cause the submission team to upload assets in a hurry. There was often confusion with the correct game build and asset specifications which would result in them getting rejected during submission or in the review process of the marketplace. This was an unnecessary long feedback cycle as then the marketing department would have to modify their assets and developers to create new package with correct specifications.

3.2 Organizational model

The organizational model in the framework of this thesis consists of game teams, technical team, support team and marketing team. Game teams are naturally responsible for developing new games and maintaining old titles. Typical game team in this organization has an executive producer, game producer, quality assurance (QA) lead, multiple game programmers, artists and designers. Additionally, the team has an assigned marketing specialist from the marketing team, a data analyst and a customer support specialist. Game producer is responsible for overseeing the game development and schedule. They act as a liaison between the game team and upper management. Their responsibilities include also communicating the progress of the game under development to other stakeholders. Usually, the producer is the one to schedule a submission of a new updated game package in cooperation with other team members. QA lead is responsible for making sure that the new game package is up to standards and as free of bugs as possible. They orchestrate the testing efforts with third-party testing companies and send the packages to them for testing. Programmers, designers, level designers and artists are working in tandem with each other developing the game by combining their talents to match the vision of designers and producer. Artists are responsible for the visual presentation of the game, while programmers combine the art with the game logic they have developed.

Technology team develops and maintains tools and services for the game teams to use in the game development. These tools vary from automating simple time-consuming tasks to offering services for tracking and analysing the game builds for submission compatibility. Technology team consists of multiple smaller teams responsible for different services they maintain and develop. Additionally, a support team handles communication between the game teams and other game-related functions. Submissions being one of these functions, there were multiple submission specialists whose responsibility was to submit new game builds and assets for the game teams in the manner described earlier. However, there was a shift in the

organizational model that led to submission team downsizing and shifting the submission responsibilities to game teams themselves. Now, instead of ordering a submission and delivering assets for the submission team, game teams must manage that individually. Based on the interviews it was apparent that this organizational shift led to some challenges in the communication and execution of said submissions.

4. Ecosystem

A submission consists of multiple different assets and steps. For a submission, one needs an application package that has the latest features, pictures that show the current state and the new features of version under submission and the texts that can provide a bit deeper overview of the game and its features. These assets can then be submitted to the marketplaces. In many ways both Google Play store (GPS) and Apple App Store Connect (ASC) are very similar when it comes to requirements and submissions. Both use mainly similar asset types, but there are, of course, differences in how and what are displayed to the user. Additionally, stores require different things from the packages that are submitted to stores. In this chapter, we will go through what kind of requirements and at the same time go through the differences and what in common these stores have.

4.1 Google Play store

Google Play store is a Google's digital distribution service for Android operating system. It allows users to download applications developed with Android SDK (Software Development Kit). Google Play store serves other kind of digital media as well, but we will be focusing on the application market and what requirements it has.

4.1.1 Application packages

For Google Play Store, application packages are distributed in .apk or .aab format. APK stands for Android application package and AAB stands for Android app bundle. Developers can choose in which format they want to upload packages to the store for users to download. Android application package is an archive format, similar to ZIP or JAR formats, which contains all the code and resources used by the

application. [4] These resources may vary from audio files to art used by the application. Google Play requires that the compressed APK size should be no more than 100 megabytes. In some cases that is plenty of space for an application, but for bigger applications, such as games, developers can decide to use APK expansion files. [5] It is possible to have two expansion files for an application, each up to two gigabytes of size. The expansion file can be any format, but in the end, Google Play will convert the file extension to Opaque Binary Blob files (.obb). There are two types of expansion files: main and patch. An application can have one of both types of expansion active at the same time. Main expansion file contains the primary additional resources your application requires. Patch instead is used for updates regarding the main expansion file.

While APK is one package meant for all users with different devices, developers might choose to upload AAB instead for better optimization for users. Android app bundle is similar to APKs with the format so that it contains the code and resources, but it defers the APK generation and signing to Google Play instead. This means that Google Play is able to serve APKs better matching the user's device configuration. Using Android app bundles will also increase the app's maximum compressed download size to 150 megabytes without any expansion files. While the download size is increased, app bundles do not allow any additional expansion files. Android app bundle is different from android application package in that it cannot be deployed straight into a device. It is rather an upload format for your application containing all the code and resources. This means that AAB applications cannot be sideloaded onto a device in the same manner as you could do with APK.

Application packages that are uploaded to the Google Play store must be signed with Google signing certificate. This signing key has a public certificate that is used by devices and services to verify that the application is from trusted source. [12]

4.1.2 Visual assets

In Google Play developer can choose to upload screenshots showcasing their application to the potential users in the store as shown in Figure 1. It is possible to add up to 8 screenshots for each supported device type. [6] These device types include: Phone, 7- and 10-inch tablets, Android TV and Wear OS by Google. To be eligible for publishing an application in Google Play developer must upload at least two screenshots for the application. These screenshots have to be in JPEG (Joint Photographic Experts Group) or PNG (Portable Network Graphics) format. Alpha channels are not allowed in these images, which means that in practice screenshots cannot contain transparency values. Submitted screenshots dimensions must be between 320 and 3840 pixels, keeping in mind that the maximum dimension cannot be more than twice the size of the minimum dimension. In addition to screenshots, the developer can optionally choose to upload a video displaying the core mechanics of the game. This video is a YouTube video URL that is embedded to the same container with the screenshots. As shown in Figure 1, the promotional video is displayed first with screenshots following.

Figure 1. Rendition of Google store listing that is shown for the user. (Image source: https://support.google.com/googleplay/android-developer/answer/1078870?hl=en)

In addition to the screenshots, an icon and feature graphic for the application is required. The given icon does not replace the application icon displayed on the device but is rather displayed in various places in the application's store listing page. The icon must be in PNG format with 512×512 pixel dimensions and under 1024 kilobytes. Feature graphics are used to show static screen over promotional video with a play button overlaying the graphic. When clicked, the promotional video is shown. This means that for feature graphic to be displayed on the application's store listing page you also need to submit promotional video. This graphic is shown anywhere in Google Play where application might be featured by Google. The feature graphic must be of PNG or JPEG format without alpha channels and with 1024×500 pixel dimensions. [6]

4.1.3 Text assets

There are three main text components for the application's store listings: description, short description and title. Title is the application's name in the Google Play store, and it should be under 50 characters long. Description or the main description text is for the developer to highlight and tell about their application and try to incentivize users to download said application. Main description should be under 4000 characters long. Short description is the first text description that users see when they access an application's store listing. It should try to capture the attention of users with 80 characters or less.

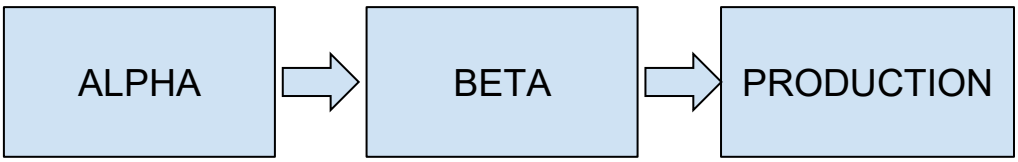
In addition to the aforementioned texts, there is also texts for release notes informing users what is new in the latest update of the application. Submitting release notes is a bit different from other text assets. While the rest of the text assets are associated with the application itself, release notes are linked to update releases of the application. The release notes are attached to application's tracks that hold information about specific releases. When preparing a new release of application, the developer might push the application package to a beta track, and then attach release

notes for that track. Release notes for the track are stored in kind of XML (Extensible Markup Language) format with language localization in tags and the appropriate text inside those tags. An example of this could look like:

```
<en-US>
Example text for new release
</en-US>
```

4.1.4 Track and rollout systems

In Google Play, the releases are handled with mainly three tracks. These tracks are production, beta and alpha. Developers may create and name their own custom tracks, but the workflow will remain similar regardless. With the intended workflow from Google for these tracks the developers would initially upload a build to closed alpha track for internal testing. [7] When all the critical issues have been tested, the track could be promoted to beta, where more people who have signed into the beta program would have the new build available for them. Finally, when everything seems to be in order developers can promote the beta track to production, so that the new update would be available for all the users. [7]



Additionally, for these tracks there is an available system called staged rollout. The rollout system is a way for the developers to release new updates in stages. [8] In practice this means that the developers can choose a percentage of users that the new update will affect. For example, the developer can define that they want to test the waters with 5% rollout for their upcoming release. They would then set the

rollout value as 0.05 and set the production track status to “inProgress”. After the initial rollout if there are no issues with the new release, the rollout value can be increased, for example, to 0.2 to allow more people on the new patch and so on until the new release is deployed for the entire user base.

4.2 Apple App Store

Apple App Store is similar distribution service to Google Play. However, Apple App Store focuses, as the name suggests, on distributing applications, and for other kinds of media Apple has iTunes. Apple App Store Connect is a service where developers can manage their applications that are listed in the App Store. Every action related to publishing an application goes through App Store Connect while App Store is rather meant to act as a storefront for customers.

4.2.1 Application package

For iOS platform, packages from the Apple App Store are distributed to the users in .ipa (iOS App Store Package) format. IPA format is similar to Google’s archive package format, where both can be decompressed with unzipping them. IPA files contain all the assets and code in a specific structure for iTunes and App Store to recognize. Apple has a 4-gigabyte size limit for overall uncompressed app size, but additional downloads can be done from the application. [10] For an application to be eligible for being downloaded over the cellular network, its download size has to be lower than 200 megabytes. If the application is over the limit, it may require to be downloaded over Wi-Fi. [11]

For an application package to be eligible for submission to Apple App Store, it must be signed with correct signing certificates issued by Apple. [13] Code signing ensures that the application to be installed on user’s device is from a known source and has not been changed after signing. This means that Apple does not allow any

application with a wrong certificate or without a certificate to be installed on an iOS device. Provisioning profiles are a part of the signing process and defines whether the application package is intended to store submission or testing. There are two primary provisioning profiles that are used for testing: development and AdHoc. These profiles define for which devices the application package can be installed. Consequently, the test devices must be added to the correct provisioning profile in order for them to be able to install the development packages.

4.2.2 Visual assets

For App Store Connect, the main visual assets are similar to Google Play in a way that you can upload images, icons and preview videos. However, the requirements are a bit different. [14] For screenshots, application has to have at least one, and maximum of 10 images for each screen resolution. Phone screenshot resolutions vary from 1242×2688 pixels for the newest 6.5-inch screen to 640×920 pixels for the 3.5-inch screen used by the iPhone 4s. If the appearance of the application is the same regardless of the device resolution, the resolutions can be scaled down from larger resolutions. [15] For iPad tablet devices, the screenshot sizes vary from 2048×2732 pixels for 12.9-inch 3rd generation tablet to 1536×2008 pixels used by iPad mini. It is worth mentioning, that even though the 12.9-inch 2nd generation iPad uses the same resolution as the 3rd generation, it requires its own screenshots to be uploaded separately. The allowed image formats to be submitted to the App Store Connect are the same PNG and JPEG as they are for Google Play. The screenshots for App Store Connect also do not allow alpha channels in images to be submitted.

Preview videos or App previews as they are called in ASC are uploaded directly to the store as opposed to Google Play where they are hosted by YouTube. App previews follow the same format with resolutions as screenshots, apart from the smallest device iPhone 4s not being supported. Preview videos are optional, but store listing allows up to 3 videos for every size. Supported extensions for app previews are

.mp4, .mov and .m4v. Additionally there are other specifications to validate, as shown in Figure 2, before videos are eligible for submission to the app store connect.

	H.264 format	ProRes 422 (HQ only) format
Target bit rate	10-12 Mbps	VBR ~220 Mbps
Video characteristics	Progressive, up to High Profile Level 4.0	Progressive, no external references
Max frame rate	30 frames per second	30 frames per second
Audio	• Stereo • Codec: 256kbps AAC • Sample Rate: 44.1kHz or 48kHz • All tracks should be enabled Stereo configuration: • 1 track with 2-channel stereo (1st channel L and 2nd channel R) • 2 tracks with 1-channel stereo (1st track L and 2nd track R)	• Stereo • Codec: PCM or 256kbps AAC • Bit Depth (for PCM): 16-, 24-, or 32-bit • Sample Rate: 44.1 or 48kHz • All tracks should be enabled Stereo configuration: • 1 track with 2-channel stereo (1st channel L and 2nd channel R) • 2 tracks with 1-channel stereo (1st track L and 2nd track R)
Supported extensions	.mov, .m4v, .mp4	.mov

Figure 2. App preview specifications for two different formats. (Source: https://help.apple.com/app-store-connect/#/dev4e413fcb8)

4.2.3 Text assets

Similarly to Google Play store, developers can submit texts providing information about the application to the App Store Connect. There is, however, a difference in how these stores consider the texts. In App Store Connect, most of the texts are linked in the platform version of the application. [16] In practice this means that newly submitted texts show up to the users when the new version is approved and promoted to production, while in GPS the texts are updated as soon as they are saved in the Google Play console.

For ASC, the options for texts are description, promotional text, keywords, release notes and subtitle. Description text is used to detail the applications features and functionality with limitation of 4000 characters. Promotional text enables the

developer to inform users of any current app features without having to update the application. The character limit for promotional text is 170 characters. Keywords help the users to discover the application. Keywords are listed and separated with commas and can have up to 100 bytes of content. Additionally, any company or app names is not allowed in the keywords section. The subtitle is a maximum of 30 characters long summary of the application and, similarly to the promotional text, can be updated without submitting an update to the application. [17] Finally, release notes, which are labelled as “What’s new in this update” in the ASC, are again used to inform the users of the new features for the new update. What’s new in this update section is also limited to 4000 characters of length.

4.3 Localizations and ratings

All of the above assets can be localized to different languages. Google Play store offers a possibility to localize your application and listing to over 70 different languages. [9] For App Store Connect, the developers can choose from over 40 languages. [18] Usually it only makes sense to localize to those regions that are seen as a good market opportunity. As such developers might consider localizing the application to those markets. However, it is possible to set a default language for the application. With this all the regions and languages without their own localization will be shown the default language in the store listing as well as the in-game content.

In Google Play store, content ratings vary between countries and continents. Most of the time they follow the standard guidelines for that region. North and South America follow ESRB (Entertainment Software Rating Board) content rating system, while most of Europe and Middle East follow PEGI (Pan European Game Information) standards. Germany, Brazil, Australia and South Korea have their own rating systems differing from the main standards used by their region. [24] This means that the application can have different ratings depending on the territory of the rating authority. In addition to the aforementioned rating standards, all the applications should also comply with Google Play Developer Program Policies. [26]

These policies aim to ensure that applications offered for download in GPS should have at least minimum functionality and are not doing anything inappropriate or illegal. App Store Connect on the other hand is a more closely curated environment. Each application submitted is reviewed individually by experts. [25] As such they do not have set guidelines for each localization and age rating, but rather every case is evaluated by the reviewer. In both Google Play-store and App Store Connect cases, content and ratings guidelines are different for distributing the application in China.

5. Technical overview of the tools

In this chapter, we will go through the selection of tools that can be used currently to upload different assets to the stores. There are multiple tools that developers can choose to use, but most of them have some pitfalls that can affect the usability and reliability of the submission. We will also see why we should choose to use these particular solutions over the other available tools.

5.1 Fastlane

Fastlane is an open source platform to simplify and automate many of the tasks involved with releasing an application in the Google Play and Apple App Store. [27] It can be used to automate screenshot capturing to code signing applications with its built-in tools. However, in the context of this thesis we are going to focus more on following tools in particular: Deliver and Supply. Deliver is a tool used to upload assets to the App Store Connect while Supply is geared towards Google Play-store.

Deliver uses another Fastlane tool called Spaceship to perform actions to App Store Connect. Spaceship in turn uses API endpoint “du-itc.appstoreconnect.apple.com” to upload assets to the ASC, and for uploading application packages deliver uses the command line tool ITMSTransporter. Supply on the other hand uses the Google Play Developer API to manage all the communication with the Google Play-store.

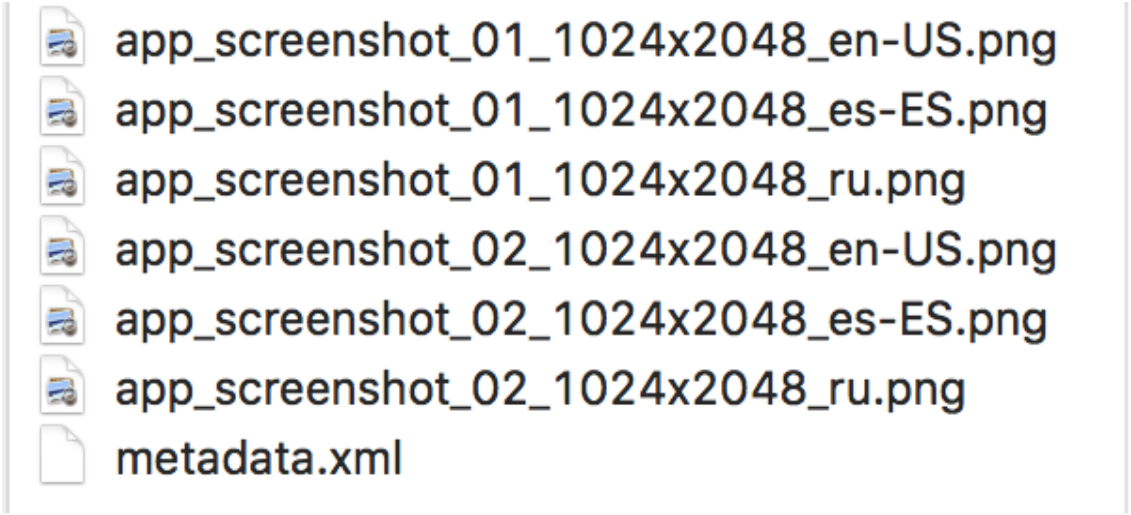
As all of the Fastlane’s tools are run from the command line, the problem with using it for our project was that we had to invoke subprocesses for running Supply or Deliver. Additionally, the problem with the earlier mentioned API, used by the Deliver for uploading assets, is that there was no public documentation for it. Every

time Apple decided to change how the API works, there was usually a short delay before the fix was merged and released. This meant that parts of the tool could be unusable for a short period of time. For this reason, it was decided to change the strategy to use the iTunes Music Store Transporter and Google Play Developer API directly.

5.2 iTunes Music Store Transporter

Assets can be delivered to the Apple App Store Connect using iTunes Music Store Transporter (ITMSTransporter). ITMSTransporter is Apple's Java-based command line tool to validate and deliver assets to the store. [19] ITMSTransporter can be used for delivering all assets to the app store or iTunes store. By default, ITMSTransporter is delivered with XCode application, but can be installed separately as well. XCode can only be installed on Mac computers, but for Linux and Windows a download option for the tool is also available.

The metadata is delivered to the App Store Connect through the App Store Package format (.itmsp). An App Store Package is a directory marked with .itmsp extension. It contains a metadata.xml (Extensible Markup Language) file where all the delivered content is described and files to be delivered with the metadata-file. For example, a directory where screenshots are uploaded to the ASC would look like:



An example metadata-file is provided in Appendix A. This example follows the specification defined in the app metadata specification. [20] Before assigning the delivery content in the metadata-file, identifying information for the application is required. App Store Connect provider, team id and vendor id must all be defined at the beginning of the metadata-file, corresponding to the application the metadata will be uploaded to. These are marked with "<provider>", "<team_id>" and "<vendor_id>" elements. These elements define the application that the to be delivered metadata is linked with in the App Store Connect. Version element contains short version string of the application version in the store where the metadata will be assigned. This version must correspond to an existing version in the store. Locales inside the version element are defined with the RFC 5646 specification (https://tools.ietf.org/html/rfc5646). The en-US locale defined in the metadata-file would then be applied English-speaking residents of the United States.

```
<locale name="en-US">
  <title>Test Application</title>
  <subtitle>Testing-filled application</subtitle>
  <description>Application for testing.</description>
  <promotional_text>On sale for a limited time.</promotional_text>
  <keywords>
    <keyword>testing</keyword>
```

```
    <keyword>educational</keyword>
  </keywords>
  <version_whats_new>Fixed a bug.</version_whats_new>
```

Inside the “<locale>”-elements is defined the localized content for delivery. Inside these tags are also defined all the assets explained in Chapter 4. While the text assets are described inside their corresponding elements as a plain text, screenshots and app previews require additional content for delivery.

```
<software_screenshot display_target="iOS-iPad" position="1">
  <file_name>screen-en-US-ipad-1.png</file_name>
  <size>286243</size>
  <checksum type="md5">3cefb7c5c37f6c868c0f4c46dc16c415</checksum>
</software_screenshot>

<app_preview display_target="iOS-6.5-in" position="1">
  <preview_image_time format="24/999
1000/nonDrop">00:00:17:01</preview_image_time>
  <data_file role="source">
    <file_name>app_preview.mp4</file_name>
    <size>33558000</size>
    <checksum
type="md5">cd12fca6b5858985fdbe10a422ade6c3</checksum>
  </data_file>
</app_preview>
```

The “<software_screenshot>” element contains information about the display target of the screenshot. These display target names are consistent with the resolutions mapped to the device names in the screenshot specification (Section 4.2). The position argument next to the display target indicates the desired position for screenshot in relation to other provided screenshots for that display_target and locale. Name correlating to the file stored inside the App Store Package must be supplied in

the “<file_name>” element. Additionally, the file size in bytes and calculated MD5-checksum must be provided in their own elements under the “<software_screenshot>” or “<app_preview>” elements. The app preview is described similarly, except there is possibility to define a poster frame for the app preview. Poster frame is shown to the user when the app preview is not playing. The source file is provided inside “<data_file>” elements, as opposed to a more straightforward definition of software screenshots.

For submitting assets automatically to the App Store Connect. there is realistically no other feasible way besides using ITMSTransporter. For example, Fastlane, uses Apple’s REST API (Representational State Transfer Application Programming Interface) to upload assets to the store platform. The problem with this approach is that the API used by Fastlane is neither documented nor officially supported. This could provide some major problems down the road if Apple decides to discontinue or change how the API works. With this in mind, the only practical solution for automating submissions to ASC is to use ITMSTransporter offered by Apple. There are, however, some limitations and inconveniences with using Transporter. Most notable is the delivery format that requires multiple file and system IO-operations, as well as, parsing and generating .xml documents from templates. Secondly, running command line tool from inside the program introduces some challenges for parsing and reporting the output.

5.3 Google Play Developer API

Google offers an extensive API for a wide variety of operations to different Google-platforms. One of these is Google Play Developer API which can be used to send assets and application packages to Google Play-store. [21] API is, as defined by Red Hat, a set of definitions and protocols for building and integrating application software. In practice this means that an application can communicate with other services without knowledge of how it is implemented. API defines the input and output of the endpoints and with this information the developers can build their software to communicate with the API. The benefits of the Google Play Developer API compared to the Transporter offered by Apple is that the API offers the developer more control for interacting with the service. For example, uploading a single asset is more straightforward with calling an API endpoint than invoking a subprocess for the Transporter command line tool.

Authentication to Google Play Developer API can be handled with service account or with OAuth client to authenticate with personal credentials. For the implementation of the tool of this thesis, service account was selected due to the reason that not all users had appropriate level of access rights to do all the operations required to submit assets to the Google marketplace.

Google Play Developer API implements a resource called Edits that allows preparing multiple changes to the application through a series of methods before committing them all at once. [22] For interacting with the Google Play Developer API, there is a client library for Python. [23] Appendix B provides an example file for simple Android Application Package upload implementation. From this example we can see that first by calling `edits().insert()` method with empty payload we are creating an open edit in the application. After that by supplying the returned edit identifier to consequent requests it is possible to make multiple changes for the same session, before finally committing them in batch. It is however worth noting that this implementation would not be enough by itself to publish this application package to any of the applications' tracks. In order to accommodate this feature, the file

presented in Appendix B would need some additions. An update call to the edits().tracks() resource would have to be made with request body defining the application packages' Android versioncode.

```
payload = {
        'track': 'beta',
        'releases': [{
                'versionCodes': [<application_versioncode>],
                'status': 'completed'
        }]
}
```

For automating tasks to Google Play-store, the Google Play Developer API seems to be the only practical way. This is, however, a non-issue as the Google Play Developer API is extensive and well documented. Changes to it are communicated well in advance and there is support for multiple different programming languages.

6. Technical solution

In this chapter, we will go through the implementation of the automated tools for uploading assets to the store marketplaces. There are two main components for this implementation: BuildTracker and Shipit. The handling and ordering of assets for the submission is done from the BuildTracker and Shipit is a microservice responsible for submitting the assets. For the programming language for this service python was chosen based on the fact that other surrounding tools in the technology team stack were also done with it. In the beginning it brought some problems when using Fastlane because it is implemented using ruby programming language.

6.1 Buildtracker

Buildtracker is an internal Django application developed for keeping track of all the application packages built in the company. Every application package built by automated build tools are sent to the Buildtracker. All the packages that are sent to Buildtracker are analysed and the contents and information about the packages are displayed for users. From the application package page users can also download the build, re-sign the package with different signing certificate and send it for external testing companies. From this page the application package can be uploaded to the corresponding marketplaces as shown in Figure 3.

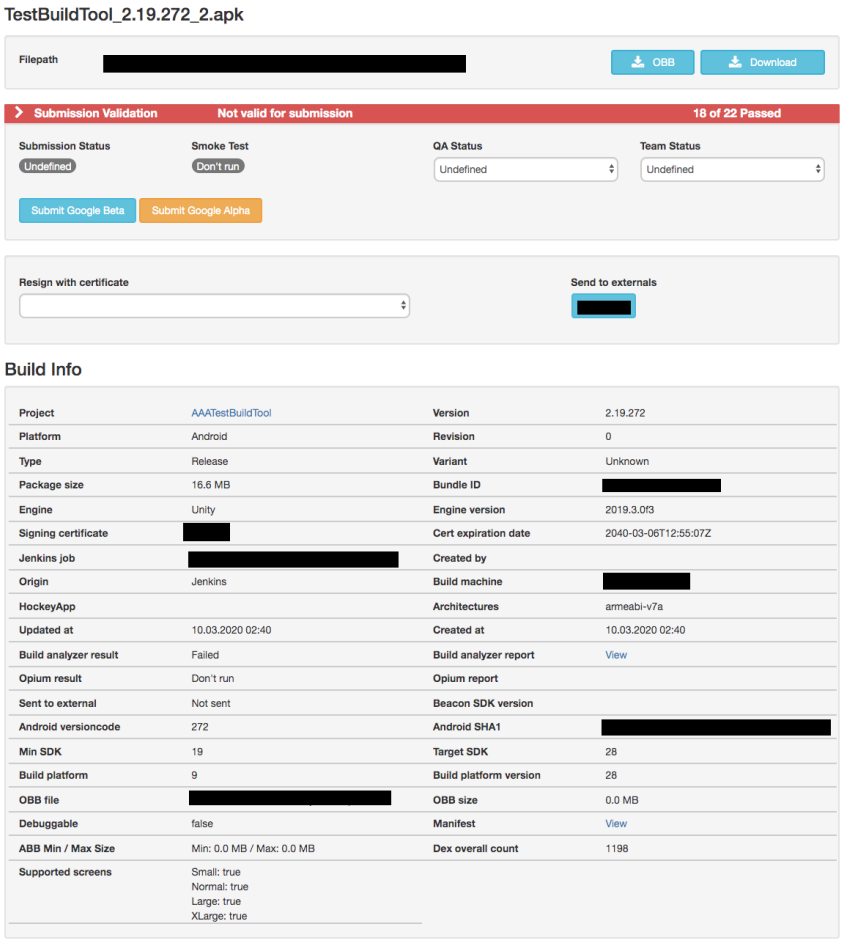


Figure 3. Application package info page.

Screenshots and app preview videos are validated from their own tab. Users can create new validation from the button shown on the page in Figure 4. From there the project and version is chosen for the validation. Then URL to Google Drive folder containing images or videos is defined. If images or videos are valid, the preview for it is created automatically after validation. On the other hand, if some of the assets are not valid, an error message containing offending assets with which of the validation steps failed are shown in the message field.

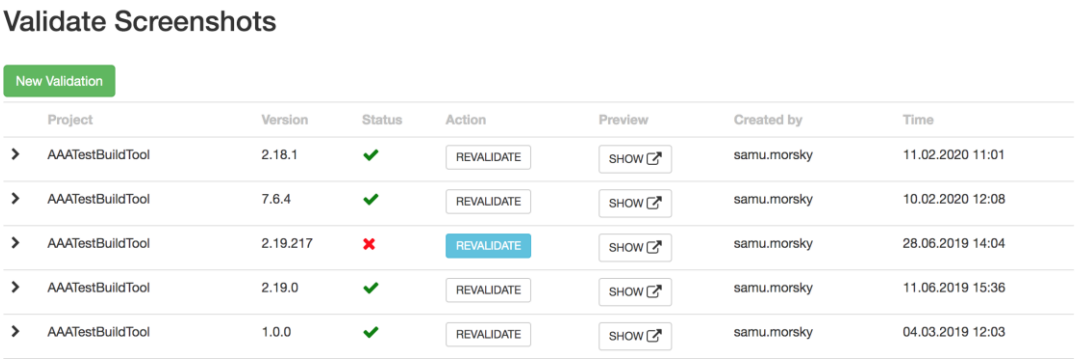


Figure 4. Screenshot validation page.

From the Project homepage users with relevant permissions can create a "submission task" for the project. This includes deciding what assets are needed for the new submission. Users have all the asset options presented in Chapter 4 excluding the icon upload. In this request, a new application package is automatically required for the new submission as every version should add something new to the game. This design choice was made during the design with the users. Stores themselves allow to change assets without updating the application package itself, and it is possible to accommodate this with submission automation but that means users will have to deviate from the intended workflow.

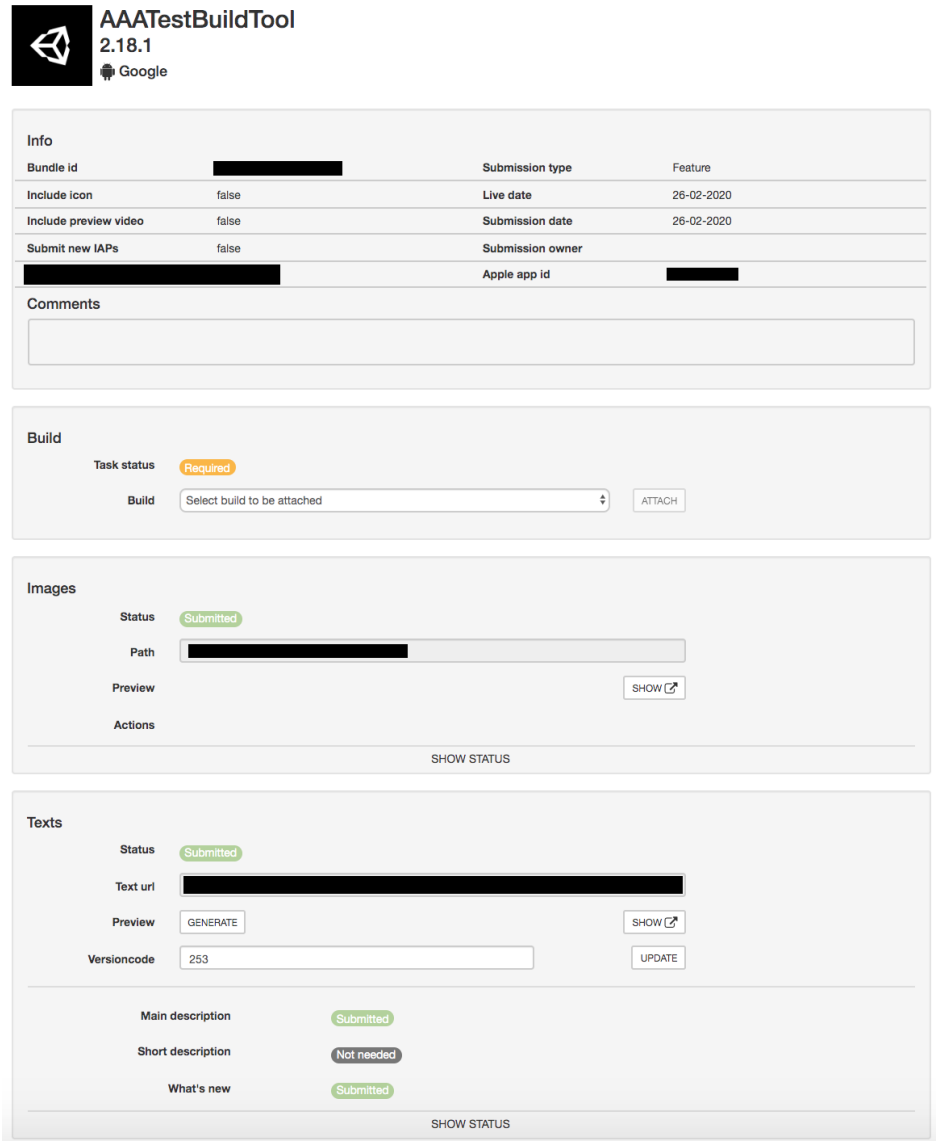


Figure 5. User interface for submitting assets.

The UI (User Interface) for submissions as shown in Figure 5 allows users to submit the assets defined in the "Request submission" phase. "Build submissions" are done from the application package page as described earlier, but in this submission page you can "attach" the final build to the current submission task to show which package was actually made available for the users. In the images section, submission users can send screenshots to the store. Screenshots are made available automatically and a preview for the images is generated when artists validate that the screenshots are compatible with the store requirements. From this preview, users can make sure that all the desired screenshots are in the right order and belong to the correct

localisation. Additionally, for App Store Connect submissions video section is also available. The UI for it works in the same manner as for screenshot submission. For text submissions the preview is generated by user action. Then a request is sent to the microservice to look up texts for the desired version number from a Google sheet defined for each individual project. After this is done, a HTML page is generated and the found texts are displayed for the user to make sure that everything is in order. If everything is correct, the users can then submit these texts to the store.

There are two main ways to access the individual submission page. First is through the submission calendar view showing all the upcoming company submissions and the second is through individual project home page showing only the submissions relevant to that particular project. The submissions are displayed in the project page in a submission tab and companywide submission calendar, which contains ongoing and upcoming requested submissions for every project. When all the planned sections are done for a submission, it is marked live. Submission calendar shows every submission for two weeks and on the top, there is a section for all the submissions that are late.

The user interface in Buildtracker was designed with users to promote and make their workflow more straightforward. The intended workflow and roles can be seen in Figure 6. One of the main themes when designing this workflow was to discover and report possible errors and blockers of submissions as early as possible. For example, when artists validate screenshots the errors would be reported directly back to them and the images would not appear in the submission before all checks passed. The same would happen when a new package is uploaded, a submission validation would be run for the package and users would then be able to see which of the compliance requirements, if any, were not as it should be. The end-users wanted to make it easier to communicate between teams, by collecting the links to assets and submissions into a centralised system. Finally, as is the case most of the time with automated systems, users wanted to reduce the amount of manual labour and errors associated with submissions.

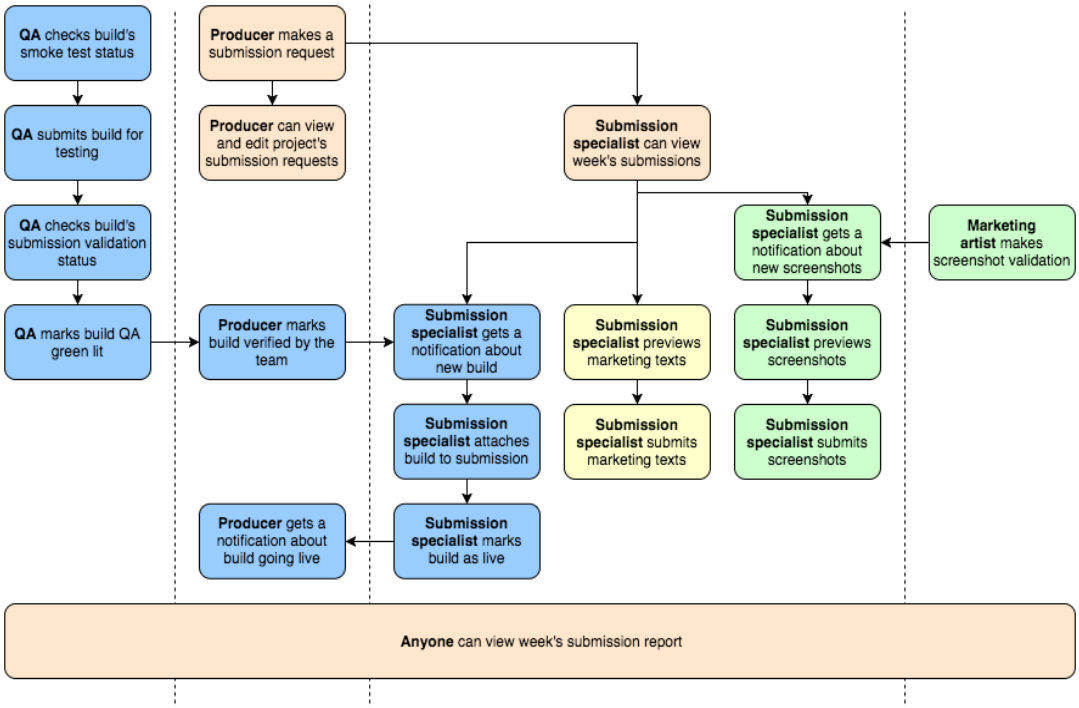


Figure 6. Intended workflow for users.

Initially the results were good, and submissions were used as designed. However, after some time when the submission team downscaled and finally ceased to exist, some problems with the UI were found that did not conform to the needs of the new users. When the submission responsibilities shifted from the submission team to individual game teams, users found the intended submission workflow a bit stiff and restricting. Submission requests that were intended to be used as a sort of a ticket system for the submission team, became a more a way of creating an on-demand submission with extra steps. The submission calendar that was created for the submission team became irrelevant because game teams submitting their assets are mostly invested only in their own projects' submissions. As a result, the submission calendar became bloated with incomplete tasks for the reason that submitted builds were not attached to the correct submissions for them to be marked as complete.

6.2 Shipit

Shipit is the project name for a set of microservices made with Python programming language. The microservice consists of multiple processes running in a process control system. Every asset type has their own process running and polling new submissions from Buildtracker API for each submission type. Additionally, there are processes for validators as well. Images and videos are validated by a set of functions that each are validating their own specification. The errors are then collected and formatted to as human readable format as possible. The reports are then sent back to Buildtracker for artists to go through. After successful validation, thumbnails from the actual images are generated and sent to AWS S3 bucket from where Buildtracker can display them for the preview.

Image submission consists of Google and Apple submission modules. For a submission to ASC images or videos, metadata file and information about the application is required. The required information consists of user credentials, vendor, team and application identifier. The vendor and team identifiers are fetched from the App Store Connect based on the application identifier defined in the project's Buildtracker configurations. The metadata file for upload is generated from templates conforming to the specifications. When setting up the submission, a Python dictionary containing all the information needed for the submission is created and passed to the template to populate the place holder spots. The Python dictionary is a data structure very similar to JSON (Java Script Object Notation) format in the sense that both consist of key-value pairs. The assets are downloaded from Google drive using Google Drive API which is very similar to Google Play Developer API explained in Section 5.3. The downloaded files are analysed to determine the language, display target and position for the template. The obtained resolutions are then compared to the resolutions defined by Apple to determine the device target. The module is only searching for three main resolutions in the source folder, because all the other device sizes can be scaled from them automatically in the App Store Connect. These resolutions are, as defined in Section 4.2.2, 1242×2688 for 6.5-inch iPhone, 1242×2208 for 5.5-inch iPhone and 2048×2732 for iPads. Screenshot

positions on the other hand are determined from names as they are sorted alphabetically. This means that the screenshots must be sorted in the right order already in the source folder. Once the screenshots are copied to the .itmsp-package, the metadata-file is also generated and situated in the folder. Finally, a subprocess is used to call the iTMSTransporter command line tool to start the submission. This process is followed by a process monitor module which reports the progress and result of the submission to the Buildtracker.

Google image submission module in contrast only requires the images and the application package name for a submission. The images are downloaded through the same API as for the App Store Connect submission. The same screenshots are used as for ASC submission, for the reason, that Google has very liberal requirements for the screenshot resolutions. The screenshots are then divided based on the language and device targets in a dictionary. The dictionary is then iterated over using the `edits.images.upload` method. References for building these requests can be found in Appendix B.

Text submissions work in the same vein as they work for the corresponding platforms in the image submission module. The texts are fetched from the Google sheets using a Google sheets utility library. The worksheet is organized by version number and text type, so that it is more machine readable. The worksheet is then searched for rows containing the desired version number and text type. The dictionary containing the texts are finally sent to the metadata template for Apple submission, or iterated over, using the same technique as for the screenshot submissions.

The communication between Shipit and Buildtracker is handled through the Buildtracker API. Consequently, this means that the Buildtracker does not interact with the Shipit services. The communication flows from Shipit to Buildtracker, as shown in Figure 7, by polling new tasks and sending results to different API routes.

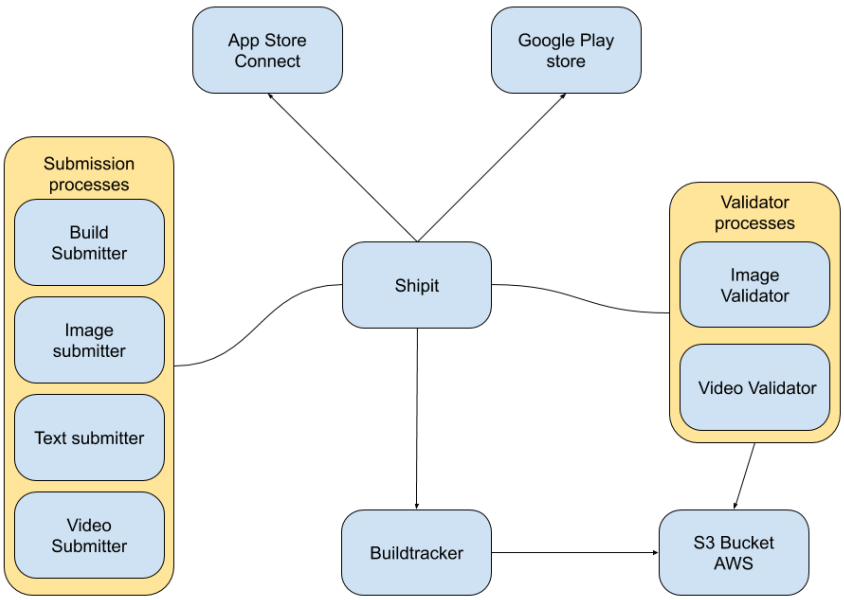


Figure 7. Communication flow chart between Shipit and other platforms.

The submission automation tools are working for the most part as intended. Most problems in submissions are still occurring from user errors. These errors mostly stem from the fact that the instructions and usage of these tools are not so straightforward for a new user. This problem could be alleviated with extensive documentation or training made readily available for users.

7. Interviews and results

In this chapter we will go through the interviews and collect the results for each question. The objective for these interviews was to gather user experiences for the achieved automation. From the company perspective, we wanted to gather knowledge on the user's perceived experience of the service to assist us in redesigning the user interface to better match the current and future needs of the users conducting the submissions. On academic level, we wanted to focus on how, if at all, the automated submission service designed with UCD-paradigm affected the organization's ability learn and retain information relating to the automated domain. (TODO conclusions: It has to be mentioned, that in hindsight the user perspective might have had even too much weight on the end product. As we established in Section 2.2, users might not always know how to best implement their wishes.)

7.1 User's role before automation

Question: *What is your individual work process regarding submission?*

Users' individual work process seems to be all around pretty same, as first build is submitted for testing, and after that gathering required assets and submitting them or coordinating submission for someone else. iOS builds are usually submitted well in advance as it takes longer time for them to be approved. Submissions for Google Play are more or less instant.

Question: *What is the update process in your team?*

From these answers we can gather that people who usually make submissions are more or less in charge of the whole submission process. Sometimes the process of uploading the assets is given to the marketing people, but the decision making is the responsibility of the person who makes the submission.

Question: *What was submission like before automation?*

Submissions before were tedious and a lot of manual work. When the central team oversaw the submissions, it would lead to situations where game teams ordering the submission were not necessarily aware of all the things needed for the submission. For example, some image sizes could be missing, or the images might contain alpha channels which are not allowed by apple. That would lead to hurry as artists would need to modify images and they needed to be submitted again. Apple would require the submission to be made from apple hardware so in the worst case the submitter would need two use two computers. Also, the time from providing assets to the submission team and them going to the store could take some time as the submission team would have queue for submissions.

7.2 User experiences

Question: *What did/do you expect from automation?*

The main expectation is mostly convenience across all answers. Some interviewees said that it is hard to expect or ask for anything as they are not thinking on the same technical level as developers, so it is hard for them to try and imagine what is possible to do. Some things that might sound simple for them would prove to be impossible to implement, and this goes the other way around as well. One interviewee also expressed that they did not even believe that it would be possible to automate submissions. I think there is something to be looked in to in how the difference in users and developers' technical abilities affect the design of the software.

Question: *Have you used submission automation? If not why?*

All interviewees have used most parts of automation. Video submission is, for some reason, not used that much. It might be because videos are not that frequently submitted. That could affect the "confidence" to use the tool to submit them when there is a long delay between sessions.

Question: *How would you describe your experience learning to use submission automation tools? How could the learning process be improved?*

Overall, learning to use submission automation was not perceived as hard. But some people expressed that there are parts where you do not remember what to do. For this reason, the usability could be improved. Moreover, some people said that the UI is "a bit engineery" and some parts of the tool are outdated as the user group has changed. Additionally, when users have longer periods when they are not doing submissions, coming back to the tool can be like learning to do it again. Improvements wise more straight forward submissions were requested. This means for example, getting rid of the request submission form, described in Section 6.1, and doing the submission from one page containing all required action points. Additionally, the concept of versions was hard to grasp for part of the interview group. This confusion stems from the fact that sometimes builds follow slightly different versioning from the submission version. For example, an application package having version 1.1.0.1, which in this case consists of major-, minor-, patch- and commit number. All the while, the version under submission would be in this example 1.1.0. Some added complexity for the users offers the fact that Google Play store do not even require any versioning for assets other than the application packages.

Question: *How did the introduction of submission automation change your process regarding submission?*

One interviewee who was part of the central submission team expressed that the hardest part was to get teams to use the submission tool. From the user perspective, the expected convenience was achieved, which is why the change was

for the better across all interviewees. Also, it was interesting to notice that people would push back the submission date when the required time to do them was reduced. Before the tool was introduced, people would start to prepare for submission week in advance. After the tool was in use, the submission assets could be delivered as late as the day the game was supposed to go live. Additional benefit for the users was that this enabled multitasking as the submitter could perform the submission and do other things while the automation tool was uploading assets. The interviewees also expressed that less errors would occur, as the validation process for assets were introduced as part of the submission tools.

> Question: *How did the change in submission responsibilities change your submission process?*

There was a bit of division among the answers. Some users had always been doing their submissions even before automation, while other people observed that the whole process “felt a lot more on me”. Some said that at the moment there is no one who is on top of all the submission related things and there should be someone who supervises and helps people with matters related to stores and submissions.

> Question: *How did the knowledge regarding submissions transfer from submission team to game teams?*

Among these answers, some said that it is hard to say how the knowledge transferred, as the old submission team was a kind of outgoing by nature and most of the interviewees were already their good friends and helping each other even before the responsibilities shifted. Consequently, the transfer of knowledge was quite seamless between those parties. The central submission team would do the submission with a person from the game teams and instruct them for how to do it by themselves. In conclusion, interviewees suggested that no significant amount of knowledge was lost during this phase, because the knowledge for submissions were actively shifted to more people.

7.3 User evaluation

Question: *How would you evaluate the effectiveness and reliability of submission automation?*

One observation was that the way Buildtracker and submission automation went on to develop changed the whole concept of submission in the company, which in turn reduced the importance of the central submission team, as was said that "anyone could do submissions with a press of a button". There was no real need for one team to do all the submissions anymore. (As a side note, pretty soon after the tools were in use, the submission team size reduced; it would be interesting to know if this had something to do with the tools, or if it is just a coincidence.) The consensus amongst all interviewees is that the tool is effective as a whole. There are some concerns with the reliability, and most of the time it is really hard to guarantee that store side servers are functioning. Some users do not trust the automation system because of those cases. Moreover, sometimes the assets take long time to appear in Apple App Store: at most, some builds could take as long as 10 hours to be visible. Users expressed that if there would be some errors, they would contact the development team and the issue would be resolved in timely manner.

Question: *How would you improve submission automation?*

Common improvement among these answers was: faster submissions and more promotional asset types, for example, when Apple considers a game for a store featuring. Additionally, it was suggested that getting rid of submission calendar that was developed for the old submission team to track submissions would make the feel of the submissions more straightforward. Users also expressed that, while not urgent, the UI is dated and could be somewhat improved. One interviewee hoped for new store features to be integrated faster, for example, adding new localisation options and image sizes would be helpful on the day they are allowed in appstore. One idea was also, to have automated email to be sent out when a version for a game is live.

Question: *How can the submission automation tool be better integrated to your update cycle?*

Multiple users said that more agile way to do submission would be better way to integrate the submission to teams' ways of working. The current way of requesting submissions is a bit stiff, which is why a page that covers all the needs with intuitive and agile user experience would be ideal for users. When it is time to submit, just "punching" in the assets would make the process more straightforward.

Question: *How important role submission plays in your team's development processes?*

Most people were of the opinion that the submission is a really important part of the development process. It is, of course, the whole point of development to get the packages to users and as core business as it gets. One differing perspective was that it is not really important anymore as the tooling has made the team to not be so concerned about the submission as it is already pretty painless. Hence, the producer or QA can just put everything into the store and no other team members really need to be concerned with the process.

8. Conclusions

This thesis aimed to identify whether an automated system made with user centred design had some effect on organizational learning. Considering the fact that this thesis only focused on one system matching the description, it is hard to come to any definitive conclusions in this field. Still, the results were indicative of there being some help retaining information inside this organization even though core competence regarding submissions left the company. As stated in the interviews, the core submission team would instruct new users on how to use the automated system. Spreading the submission knowledge throughout the game teams will help the organization learn, but the automation would leave users with only shallow knowledge of the underlying store platform systems. Furthermore, the training of the new users would be performed by the previous user, who might not have that good of an understanding of the system themselves. In my opinion it would be beneficial for new users in the future to get the required training from people with good understanding of the system. It is worth mentioning that, even though the knowledge regarding the submissions might be shallow for the users using the automation, the team responsible for implementing the automated system has to have a deep understanding of the submissions. In this sense, the core competence has shifted from the submission specialists to the technology team implementing and documenting the system resulting in increase in the overall and long-term submission knowledge for the organization.

Whether this increase in organizational learning had anything to do with the automated system being designed with UCD can be debated. The UCD part of the design may have been a bit too inclusive of the user feedback. Considering this fact, the resulting system had some features that could not keep up with the changing organization. As stated earlier, when designing the system, the end-user was consulted perhaps even too much about their preferences for the user interface. For example, some of the small details about submissions were good ideas in theory, but in practice they just ended up being steps that brought no real value for the user. One

such example was the QA and team status for game packages considered for submission seen in Figure 3. The idea would be that both team and QA lead would go to the package page and greenlight it when it gets approved for submission on their part, then a submission specialist would see that this package can now be submitted. This would rarely happen in practice, instead the package for submission would be communicated using other channels. Consequently, if this research would be continued in the future, I would suggest paying more attention in the UCD process to be applied in the design phase of the system.

Considering the qualitative nature of this study, the initial expectations for this study were met reasonably well. As stated earlier, the knowledge of the submission and its processes are now spread more in the organization rather than residing in one team. Even though the aim was to see if UCD and OL would overlap in this case, there was more evidence to support automation affecting OL rather than UCD being responsible for it. The user interviews indicated that even though there is no one on top of all the submissions like before, the knowledge regarding the submissions are to be found in more places and perhaps not even needed as much, because the process is more streamlined from automated validation to uploading of the assets. Additionally, the feedback from learning to use the system was generally good, and users had no problems getting support when encountering situations which they could not resolve themselves. Also, from the quantitative perspective the overall submission times were considerably faster. As users stated in the interviews, work that could take them at worst the whole business day, could now be done with a push of a button allowing them to do other work in tandem while the submission was processing in the background.

At the organizational level, we got a lot of valuable insight on the state of the automated submission tools. Feedback from the users will help us understand better what needs to be changed or improved in the next iterations of the user experience for the system. While feedback from users is most certainly valuable, it is important to not take it at face value, rather it is better to try and really understand the users and their tasks. From the interviews and my gathered insight for the subject, I would

surmise that convenience takes priority over functionality when automating tasks, even if they are work heavy in the first place. If the user experience of the system is tedious and hard to learn for the users, they might choose to do the original tedious work that is at least familiar to them. One concern from the interviews was that some of the users were not aware of all the existing features for automated submissions. Some interviewees suggested a feature only to discover that it already existed. For this reason, it is really important to onboard and educate people to get them to use the existing systems as soon as possible before they develop their own habits that could make them impervious to new ones.

In conclusion, the results were not indicative of there being any significant learning due to the UCD, but overall this study gave direction for new research to be made on the subject. The observed OL could be attributed to general automation and resulting change in the work environment. The results from the system were overall positive as the interviews suggested. It provided valuable insight on the subject of designing systems for specific user group and how new users brought by changing organizational structure changed the demand for the features of the same system.

References

[1] Nigel Bevan & Ian Curson, Planning and implementing user-centred design. In CHI EA '99: CHI '99 *Extended Abstracts on Human Factors in Computing Systems*, (May 1999), Pages 137–138

[2] Norman, Donald A. "Human-centered design considered harmful", *Interactions*, Volume 12, Number 4 (2005), Pages 14-19

[3] App Store icon, app preview, and screenshots overview, (2020), https://help.apple.com/app-store-connect/#/dev910472ff2

[4] Google, Application Fundamentals, (27.12.2019), https://developer.android.com/guide/components/fundamentals

[5] Google, APK Expansion Files, (27.12.2019), https://developer.android.com/google/play/expansion-files

[12] Google, Use app signing by Google Play, (2020), https://support.google.com/googleplay/android-developer/answer/7384423?hl=en

[6] Google, Graphic assets, screenshots, & video, (2020), https://support.google.com/googleplay/android-developer/answer/1078870?hl=en

[7] Google, Set up an open, closed, or internal test, (2020), https://support.google.com/googleplay/android-developer/answer/3131213?hl=en

[8] Google, APKs and Tracks, (18.5.2018), https://developers.google.com/android-publisher/tracks

[9] Google, Translate & localize your app, (2020), https://support.google.com/googleplay/android-developer/answer/3125566?hl=en

[10] Apple, Maximum build file sizes, (2020), https://help.apple.com/app-store-connect/#/dev611e0a21f

[11] Apple, Reducing Your App's Size, (2020), https://developer.apple.com/documentation/xcode/reducing_your_app_s_size

[13] Apple, Code Signing, (2020), https://developer.apple.com/support/code-signing/

[14] Apple, App Store icon, app preview, and screenshots overview, (2020), https://help.apple.com/app-store-connect/#/dev910472ff2

[15] Apple, Screenshot specifications, (2020), https://help.apple.com/app-store-connect/#/devd274dd925

[16]Apple, Platform version information, (2020), https://help.apple.com/app-store-connect/#/devf29afbb74

[17] Apple, App information, (2020), https://help.apple.com/app-store-connect/#/dev219b53a88

[18] Apple, Prepare for a Global Audience, (2020), https://developer.apple.com/internationalization/

[19] Apple, Transporter User Guide 2.0, (2019), https://help.apple.com/itc/transporteruserguide/en.lproj/static.html

[20] Apple, Basic App Metadata Annotated, (2020), https://help.apple.com/asc/appsspec/#/itc6e4198248

[21] Google, Google Play Developer API, (05.11.2019), https://developers.google.com/android-publisher

[22] Google, Edits, (05.11.2019), https://developers.google.com/android-publisher/edits

[23] Google, GitHub google-api-python-client, (13.3.2020), https://github.com/googleapis/google-api-python-client

[24] Google, Apps & Games content ratings on Google Play, (2020), https://support.google.com/googleplay/answer/6209544?hl=en

[25] Apple, App Store Review Guidelines, (04.03.2020), https://developer.apple.com/app-store/review/guidelines/

[26] Google, Developer policy center, (2020), https://play.google.com/about/developer-content-policy/

[27] Felix Krause, Fastlane, (2019), https://fastlane.tools/

[28] Vera Stara, Richard Harte, Mirko Di Rosa, Liam Glynn, Monica Casey, Patrick Hayes, Lorena Rossi, Anat Mirelman, Paul M.A.Baker, Leo R. Quinlan & Gearóid ÓLaighin, Does culture affect usability? A trans-European usability and user experience assessment of a falls-risk connected health system following a user-centred design methodology carried out in a single European country, *Maturitas*, Volume 114, August 2018, Pages 22-26

[29] The International Organization for Standardization, ISO 9241-210:2019(en), *Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*, (2019)

[30] Pieter M.A.Desmet, Haian Xue & Steven F.Fokkinga, The Same Person Is Never the Same: Introducing Mood-Stimulated Thought/Action Tendencies for User-Centered Design, She Ji: *The Journal of Design, Economics, and Innovation*, Volume 5, Issue 3, Autumn 2019, Pages 167-187

[31] Mica R. Ensley & Debra G. Jones, *Designing for situation awareness*, Second edition, 2004

[32] N.B. Sarter, D. D. Woods & C.E. Billings, Cognitive Systems Engineering Laboratory, The Ohio State University, AUTOMATION SURPRISES, *Handbook of Human Factors & Ergonomics*, second edition, G. Salvendy (Ed.), Wiley, 1997

[33] Nakkiran N. Sunassee & Vernon Haumant, Organisational Learning versus the Learning Organisation, SAICSIT '04: *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, October 2004 Pages 264–268

[34] Michael G. Harvey, Jonathan W, Palmer & Cheri Speier, Intranets and organizational learning, SIGCPR '97: *Proceedings of the 1997 ACM SIGCPR conference on Computer personnel research*, April 1997 Pages 110–116

[35] Jim Q. Chen, Ted E. Lee, Ruidong Zhang & Yue Jeff Zhang, "Systems requirements for organizational learning", *Communications of the ACM*, December 2003

[36] Helder de Jesus Ginja Antunes & Paulo Gonçalves Pinheiro, Linking knowledge management, organizational learning and memory, *Journal of Innovation & Knowledge*, 31 May 2019

Appendix A: Apple App Store metadata-file example

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://apple.com/itunes/importer"
version="software5.11">
  <provider>ApplicationDeveloper</provider>
  <team_id>ABCD1234</team_id>
  <software>
    <vendor_id>VENDOR1D</vendor_id>
    <software_metadata>
      <versions>
        <version string="1.1.1">
          <locales>
            <locale name="en-US">
              <title>Test Application</title>
              <subtitle>Action-filled combat game</subtitle>
              <description>Application for
testing.</description>
              <promotional_text>On sale for a limited
time.</promotional_text>
              <keywords>
                <keyword>testing.</keyword>
                <keyword>educational.</keyword>
              </keywords>
              <version_whats_new>Fixed a
bug.</version_whats_new>
              <software_url>http://www.testapp.com/
</software_url>
              <privacy_url>http://www.testapp.com/privacy/
</privacy_url>
              <privacy_policy_text>Privacy
policy</privacy_policy_text>
```

```
              <support_url>http://www.testapp.com/support/
</support_url>
              <app_previews>
                <app_preview display_target="iOS-6.5-in"
position="1">
                  <preview_image_time format="24/999
1000/nonDrop">00:00:17:01</preview_image_time>
                  <data_file role="source">
                    <file_name>app_preview.mp4</file_name>
                    <size>33558000</size>
                    <checksum
type="md5">cd12fca6b5858985fdbe10a422ade6c3</checksum>
                  </data_file>
                </app_preview>
              </app_previews>
              <software_screenshots>
                <software_screenshot display_target="iOS-iPad"
position="1">
                  <file_name>screen-en-US-ipad-
1.png</file_name>
                  <size>286243</size>
                  <checksum
type="md5">3cefb7c5c37f6c868c0f4c46dc16c415</checksum>
                </software_screenshot>
              </software_screenshots>
            </locale>
          </locales>
        </version>
      </versions>
    </software_metadata>
  </software>
</package>
```

Appendix B: Google package upload script example

```
from googleapiclient.discovery import build
from google.oauth2 import service_account

# Create credentials from service account json-file.
credentials =
service_account.ServiceAccountCredentials.from_json_key_f
ile(
    json_key_path,
scopes=['https://googleapis.com/auth/androidpublisher']
)

# Create service to make calls to the API
service = build(
    'androidpublisher',
    'v3',
    credentials=credentials
)

# Create edit request and save edit id
edit_request = service.edits().insert(body={},
package_name=application_package_name).execute()
edit_id = edit_request['id']

# Call upload method for the .apk
service.edits().apks().upload(
    editId=edit_id,
    package_name=application_package_name,
    media_body=path_to_apk,
```

```
    media_mime_type='application/octet-stream'
).execute()

# Finally commit changes
service.edits().commit(editId=edit_id,
package_name=application_package_name).execute()
```