jamk.fi

Mobiilipelin versioinnin automatisointi

Peliprojektin vienti Unitystä iOS-alustalle

Tuomas Kyttä

Opinnäytetyö
Maaliskuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Kyttä, Tuomas	Opinnäytetyö, AMK	13.03.2017
	Sivumäärä 76	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Mobiilipelin versioinnin automatisointi
Peliprojektin vienti Unitystä iOS-alustalle

Tutkinto-ohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Jouni Huotari, Paavo Nelimarkka

Toimeksiantaja(t)
Zaibatsu Interactive Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli toteuttaa Zaibatsu Interactive Oy:lle järjestelmä, joka automatisoisi mobiilipelin testiversioiden kääntämisen, jakamisen ja raportoinnin. Lisäksi opinnäytetyössä tutkitaan vastaaville järjestelmille mahdollisia testauksen automatisointitapoja.

Projektin aikana tutkittiin erilaisia työkaluja Unityn, iTunes Connectin ja Google Developer Consolen integroimiseen. Projekti toteutettiin rajaamalla mahdolliset vaihtoehdot ja selvittämällä mitkä olisivat toimeksiantajan kannalta parhaat.

Projektin lopputuloksena luotiin järjestelmä, joka ylitti toimeksiantajan odotukset ja täytti vaatimukset. Järjestelmä on hyvin modulaarinen. Suorittavia järjestelmän paloja voi lisätä ja poistaa käytöstä tarpeen mukaan. Järjestelmä on julkaisuhetkenä aktiivisessa käytössä toimeksiantajalla.

Täysin kattavaa järjestelmää ei projektin aikarajoissa saatu aikaan ja osa jäi prototyyppitasolle. Prototyyppitasolle jääneet ominaisuudet eivät olleet alkuperäisessä toimeksiantajan vaatimusmäärittelyssä ja olivat prioriteettilistalla alimmaisena.

Projektin aikana havaittiin, että joidenkin ohjelmistotalojen vastaavissa systeemeissä oli myös automatisoitu testaamisen eri osia. Projektin aikana ei haluttu käyttää aikaa automatisoitujen testien tekemiseen, mutta mielenkiinnosta aihetta kohtaan sitä tutkittiin ja mahdollisesti lisätään projektin lopputulokseen tulevaisuudessa.

Avainsanat (asiasanat)
Mobiilipeli, Testaus, Google, Apple, Android, iOS, Jenkins

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Kyttä, Tuomas	Bachelor's thesis	13.03.2017
		Language of publication: Finnish
	Number of pages: 76	Permission for web publication: x

Title of publication

Automatic versioning for mobile games

Game project from Unity to iOS

Degree programme

Software Engineering

Supervisor(s)

Huotari, Jouni. Nelimarkka, Paavo

Assigned by

Zaibatsu Interactive Inc.

Abstract

The purpose of the thesis was to develop a system for Zaibatsu Interactive Inc. to automate building test versions of mobile games, sharing versions to testers and reporting on what to test. Additionally, it was also researched how to possibly automate parts of testing in this kind of a system and how some game companies might already have carried this out.

During the project, different tools for integrating Unity3D, iTunes Connect and Google Developer Console were researched. The project was implemented by limiting possible choices of tools and deciding which would best suit Zaibatsu Inc.

The product of the project was a system that all but surpassed original expectations and fulfilled all the demands set on the product. The system is modular, meaning that different parts can be added or removed depending on the project. During the creation of this document, the system is already in active use at Zaibatsu.

A fully inclusive system was not created due to time constraints, and parts of system are still of prototype quality. The parts left to prototype quality were not listed as demands at the start of the project and were therefore prioritized less.

It was found out during the project that some software companies use the same kind of automation systems to automate parts of testing. There was no time to create a system to automate testing in this system during the project. Due to the interest in the subject, it was researched how to create automated tests and a system to support running them. However, it might be implemented in the future.

Keywords/tags (subjects)

Mobile game, Testing, Google, Apple, Android, iOS, Jenkins

Miscellaneous

Sisältö

Kuviot

Termit ja käsitteet

Android	Google kehittämä käyttöjärjestelmä mobiililaitteille
API	Ohjelmointirajapinta (Application Programming Interface)
APK	Tiedostomuoto suoritettaville Android sovelluksille
App ID	Objekti jolla tunnistetaan yksi tai useampi sovellus
App Store	Applen sovelluskauppa iOS-laitteille
Bash	Unixin komentotulkki
Dropbox	Pilvipalvelu tiedostojen tallentamiseen, jakamiseen ja synkronointiin eri koneiden välillä
Git	Versiohallintajärjestelmä
Google Play	Googlen sovelluskauppa Android-alustoille
GPDC	Google Play Developer Console. Googlen palvelu, jolla voi julkaista pelejä ja hallita niiden testaamista.
HTTP	Hypertext Transfer Protocol. Tiedonsiirtoprotokolla Web-palveluille
iOS	Applen kehittämä käyttöjärjestelmä Applen mobiililaitteille
IPA	Tiedostomuoto pakatuille iOS tiedostoille.
iTunes Connect	Applen ylläpitämä palvelu, jossa jaetaan, hallitaan ja testataan sovelluksia.
OAuth	Standardi, jolla käyttäjä voi valtuuttaa sovelluksen käyttämään tietojaan, ilman salasanaa
TestFlight	Applen omistama palvelu sovellusten asentamiseen ja testaamiseen. Osa Applen kehittäjäohjelmaa.
Unity	Pelimoottori jolla tuki monialustaiseen sovelluskehitykseen
Xcode	Applen julkaisema ohjelmointiympäristö
Wildcard App ID	App ID jonka Bundle ID päättyy ”*” merkkiin.

1 Johdanto

1.1 Tavoitteet

Pelin kehityksessä aikaa kuluu hyvin paljon pelin iteratiiviseen testaamiseen kohdealustoilla. Peliin sovellettavista ideoista ei voi tietää, ovatko ne hyviä tai toimivia ennen testaamista. Testaamatta oikealla laitteella ei koskaan saada täyttä varmuutta rikkoiko uusi ominaisuus pelin jollain laitteella. Testaamisen suorittamiseksi koodi käännetään suoritettavaksi ohjelmaksi ja levitetään testattavaksi. Unityllä työskennellessä operaatio on hyvin suoraviivainen Androidille, mutta ajoittain tuskallinen iOS-alustoille. Versiointi koettiin aikaa vieväksi ja ongelmalliseksi operaatioksi ja ongelmaa päätettiin tutkia mahdollisen automatisoinnin kannalta.

Ratkaisuksi päätettiin luoda systeemi, jolla voidaan automatisoida iOS-versioiden tekeminen, levittäminen iTunes Connectin kautta testaajille ja muutosraportin kirjaaminen Trelloon. Sekundaarisiin tavoitteisiin projektissa kuului version valmistumisesta ilmoittaminen Flowdock-kanavalle ja Android-versioiden kääntäminen ja levitys.

Projektin aikana tutkittiin myös testaamista, sen automatisointia, miten sitä on toteutettu peliteollisuudessa ja kuinka testaamisen voisi automatisoida projektin aikana luodussa systeemissä. Tässä työssä ei oteta kantaa, kuinka testaus tulisi toteuttaa tai mikä on tehokkain tapa toteuttaa automatisointi.

1.2 Toimeksiantaja

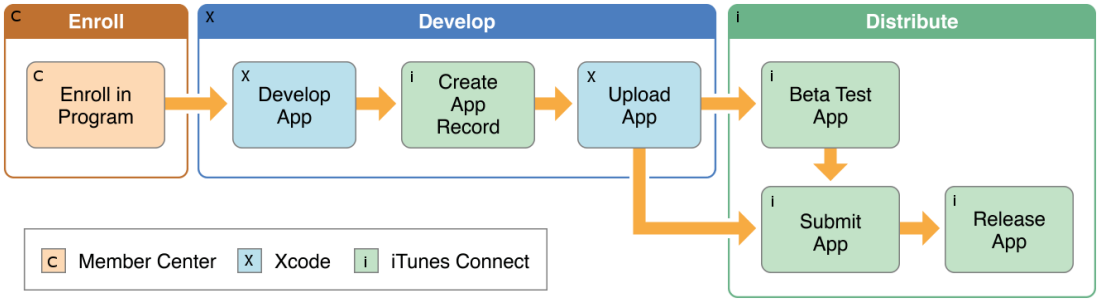
Zaibatsu Interactive Inc. on Jyväskylässä sijaitseva huhtikuussa 2014 perustettu peliyritys, joka tekee myös asiakasprojekteja. YLE on julkaissut Zaibatsun toteuttaman pelin Elder Goo (Möllit) Suomessa 2015. Maailmanlaajuista julkaisua Elder Goolle ennustetaan vuodelle 2017. Asiakasprojektit ovat olleet mm. pelillistämistä ja pelejä. (Zaibatsu press kit. n.d.)

2 Tutkitut sovellukset & tekniikat

Projektin alussa tutkittiin monia vaihtoehtoja versioinnin automatisoinnin tekemiseen. Sen lisäksi oli välttämätöntä tutkia ympäristöjä, joissa testiversioita voidaan jakaa. Applen ja Googlen dokumentaatioita oli välttämätöntä tutkia, koska suurin osa mobiilipeleistä julkaistaan näiden tahojen ylläpitämillä alustoilla.

2.1 Sovelluskehitys Applen ehdoilla

iOS-alustoille kehittäminen on Android alustoille verrattuna monimutkaista, mutta Apple on tehnyt kattavat dokumentaatiot helpottamaan kehittämisen aloitusta. Kuvio 1 kuvaa miten ohjelmiston kehitys etenee eri tasoissa. Alussa rekisteröidytään kehitysohjelmaan. Kehitetään sovellusta, luodaan siitä tallenne ja laitetaan iTunes Connectiin, jonka kautta testataan. Kuvion 1 Develop- ja Distribute-vaiheita toistetaan niin kauan, että päästään Beta Test App -kohdasta eteenpäin.



Kuvio 1. kehitysketju Applen järjestelmässä (About iTunes Connect. n.d.)

Kehitysohjelmia on kahdenlaisia: Apple Developer Program ja Apple Developer Enterprise Program. Ensimmäinen maksaa 99 $ vuodessa ja on tarkoitettu yrityksille tai yksityishenkilöille, jotka tekevät sovelluksia iOS-alustoille. Erona on yritystunnuksen vaatiminen ja yritys voi lisätä useita käyttäjiä yhdelle lisenssille. Jälkimmäinen on tarkoitettu suuryrityksille, jotka julkaisevat sisäiseen käyttöön sovelluksia ja isolla skaalalla, ja maksaa 299 $ vuodessa. Yksityishenkilöt voivat ilman lisenssiä tehdä omilla laitteillaan toimivia sovelluksia signeeraamalla ne Apple Id:llä. (Apple Developer programs. n.d.)

Xcode on Applen kehittämä ilmainen integroitu ohjelmointiympäristö Mac:lle ja iOS:lle, joka sisältää kaiken tarvittavan sovellusten kehittämiseen Applen alustoille. Xcode on saatavilla ainoastaan OS X -järjestelmille. (Klosowski. 2014.)

2.1.1 Developer Portal ja iTunes Connect

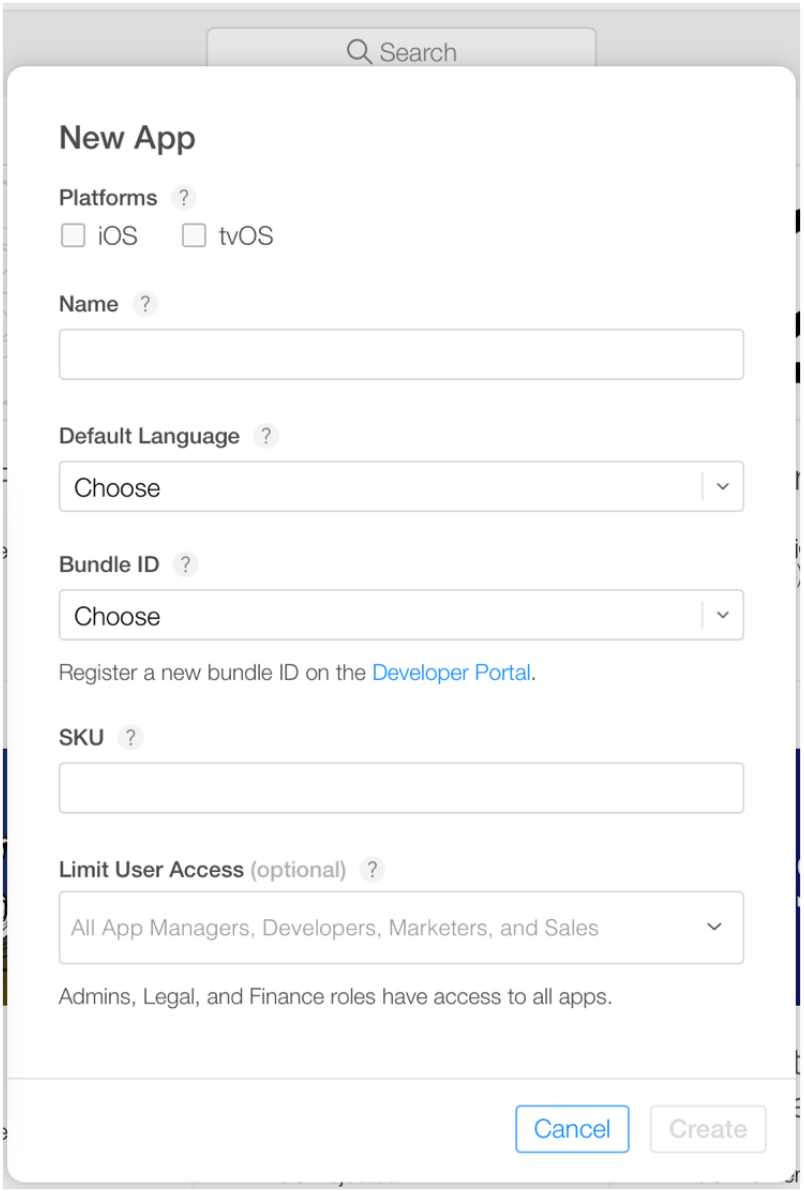
Applella on kaksi tärkeää hallintapaneelia, joita käytetään ohjelmien kehityksessä ja julkaisussa: Developer Portal ja iTunes Connect. Kummatkin järjestelmät vaativat kehittäjätilin, joka kuuluu johonkin kehittäjäohjelmaan. Developer Portaalissa käyttäjä voi hallinnoida kaikkea koodin signeeraamiseen liittyvää (Szalay 2016). iTunes Connect on Apple Inc:n ylläpitämä palvelu, jossa kehittäjät voivat hallita sovelluksiaan, sopimuksia Applen kanssa, raha-asioita, myyntiraportteja ja muita ominaisuuksia, jotka liittyvät sovellusten myyntiin ja levittämiseen (About iTunes Connect. n.d).

Nykyään Xcode tekee monet asiat automaattisesti, mitkä piti ennen asettaa Developer Portaalissa. Näihin operaatioihin kuuluu tunnistajien asettaminen, laitteiden ja profiilien asettaminen, joilla sovellusta voi käyttää. Developer Portaalissa voi kuitenkin tehdä omia provision profiileja testaamista varten, korjata kumotut profiilit ja paljon muuta. mm. Unity Clould Buildia käytettäessä provision profiilit on haettava Mac OS -koneen puuttuessa Developer portaalista. (Maintaining Identifiers, Devices, and Profiles. n.d)

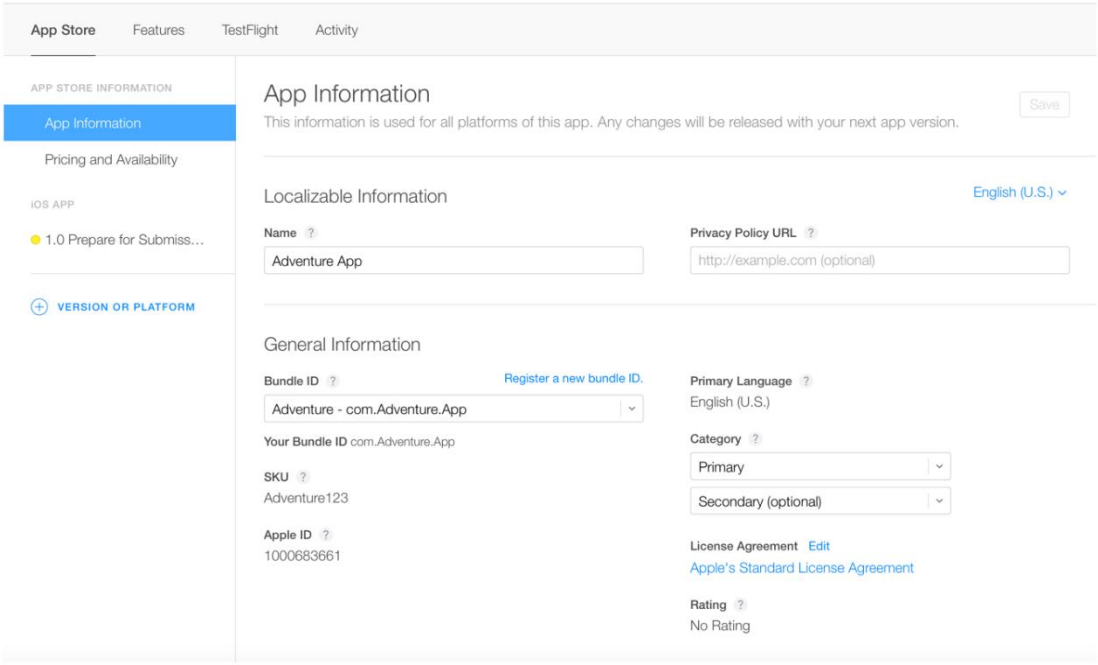
Ennen sovelluksen lähettämistä Applen sovelluskauppaan, sovellus on kirjattava palveluun. Operaatio vaatii käyttäjältä Admin-, Technical- tai App Manager-oikeudet. Operaatio on hyvin suoraviivainen. iTunes Connectiin kirjautumisen jälkeen valitaan kolmesta näkyvästä nappulasta ”My Apps” ja painetaan isoa plussaa ylävasemmalla. Tämän jälkeen näkyy kuviossa 2 esitetty lomake, joka pitää täyttää. Alusta, nimi ja oletuskieli ovat melko selviä. Bundle ID on tunnus, joka yksilöi sovelluksen, ja se määritellään yleensä Xcodessa. Unity-projektin tapauksessa Unityssä. SKU on Stock Keeping Unit. Se on uniikki jokaiselle iTunes Connectissa olevalle sovellukselle ja sitä käytetään toistaiseksi vain Applen generoimissa raporteissa. Käyttäjiä voidaan myös rajata, jolloin vain tietyt henkilöt näkevät sovelluksen iTunes Connectissa. Käyttäjät seuraavilla oikeuksilla näkevät kaikki sovellukset rajauksista riippumatta, koska heillä

ei voi olla rajattua näkyvyyttä: Admin, Technical, Finance tai Reports. (Creating an iTunes Connect Record for an App. n.d.)

Sovelluksen rekisteröinnin jälkeen My Apps valikkoon ilmestyy nappula luodun sovelluksen nimellä ja mahdollisella kuvalla. Klikkaamalla uutta painiketta avautuu kuvio 3:n näköinen valikko jossa voi hallita sovelluksen asetuksia, hintoja, näkyvyyttä ja muuta.



Kuvio 2. Lomake uuden sovelluksen rekisteröinnisssä iTunes Connectiin (Creating an iTunes Connect Record for an App. n.d.)



Kuvio 3. Sovelluksen hallintapaneeli (Creating an iTunes Connect Record for an App. n.d.)

2.1.2 Testiversioiden lähettäminen

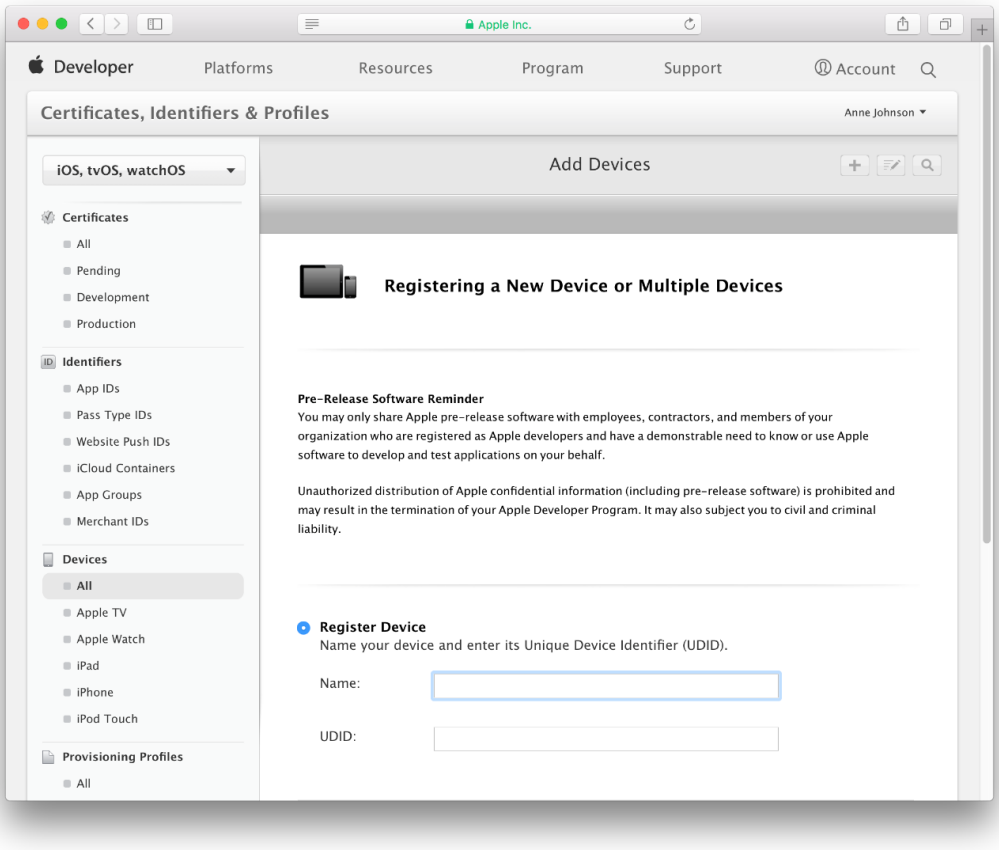
Testiversioiden jakamisen vaihtoehdot iOS-alustoille ovat hyvin rajalliset. Tunnetuimmat ovat julkinen Applen sovelluskauppa, iOS Developer Enterprise Program, Ad Hoc Disribution ja TestFlight (Lamppa. 2012).

Applen sovelluskaupassa testiversioita voi jakaa asettamalla sovellukseen kirjautuminen josta vain ennalta määritellyt henkilöt pääsevät läpi. Asettamalla pienen summan voidaan välttää turhia latauksia henkilöiltä, joilla ei ole oikeutta käyttää sovelluksia. (Lamppa. 2012.) Vaikka tämä on mahdollinen tapa, niin se on kenties huonoin mahdollinen. Sovellus altistuu lievälle teollisuusvakoilulle ja päivitysten tulee läpäistä Applen tarkastus joka nykyisin vie iOS-alustoille n. yhden päivän (Average App Store Review Times. n.d).

iOS Developer Enterprise Program on ohjelma, jonka osana yritys voi julkaista ja jakaa omia sisäisiä sovelluksiaan. Ohjelma maksaa 299 $, ja tavallinen iOS Developer

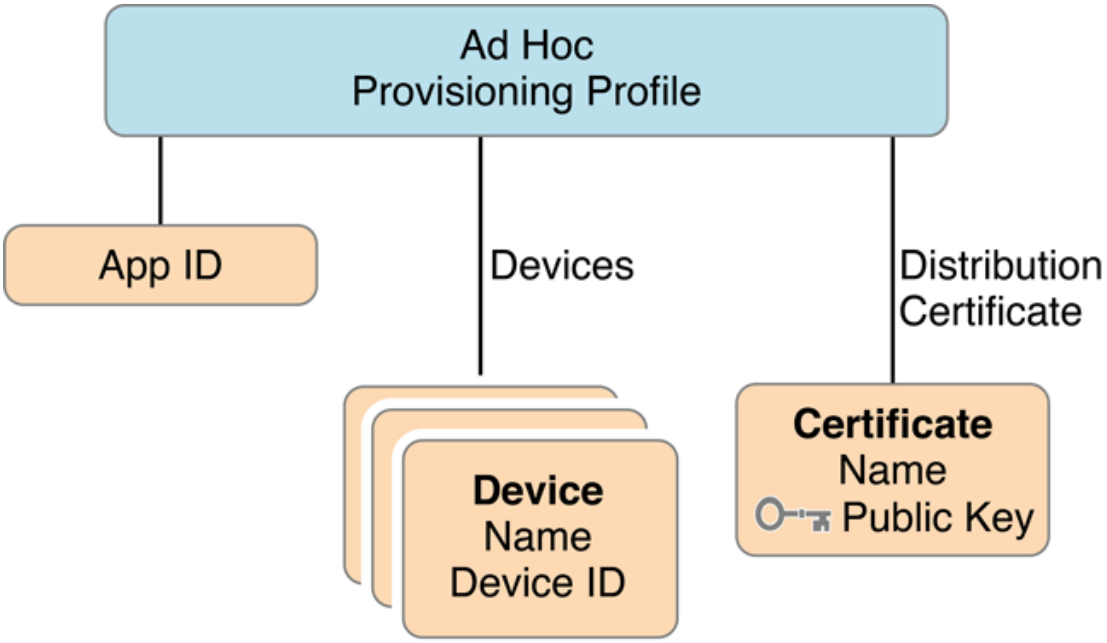
lisenssi maksaa 99 $. Huonona puolena on se, että yrityksen on itse järjestettävä sovellusten levitys ja monet Enterprise-ohjelman osat eivät ole olennaisia pienelle peliyritykselle. (Lamppa, D. 2012.)

Ad Hoc -sovellukset on tarkoitettu testaamista varten. Ad Hoc -sovelluksen tekemiseen tarvitaan Apple-tili, joka kuuluu johonkin Applen maksulliseen ohjelmaan. Käytännössä kehittäjätilille rekisteröidään iOS-laite, jolla sovellusta testataan, laitteen ID:llä Applen Developer sivustolla. Kuviossa 4 on esitetty laitteen rekisteröinti. Kun sovellus käännetään suoritettavaksi, se signeerataan provision profiililla, joka kuuluu käytetylle tilille. Tämä on nopein tapa testata sovellusta yksittäisellä laitteella. Useammalla laitteella testatessa jokainen laite pitää lisätä käsin tai antamalla tiedoston, joka sisältää tiedot testilaitteista. Ad Hoc on rajoitettu 100 laitteeseen per tili, ja laitteita voi vaihtaa vuoden välein. Sovellus toimii maksimissaan vuoden, koska sen toiminen valtuutetaan provision profiililla ja ne vanhenevat vuoden välein. (Lamppa. 2012.)



Kuvio 4. Laitteen rekisteröinti kehittäjätilille. (Maintaining Identifiers, Devices, and Profiles. n.d)

Applen järjestelmässä on kahdenlaisia levittämiseen tarkoitettuja provision profiileja. Toinen on tarkoitettu iTunes Connectissa julkaisuun ja toinen Ad Hocille. Ad Hocin provision profiilin tarkoitus on estää testiversioiden leviäminen luvattomille käyttäjille. Ad Hoc profiilia luodessa sille asetetaan App ID, joka voi vastata yhtä tai useampaa sovellusta, testilaitteet ja yhden jakeluvarmenteen, kuten kuviossa 5 näytetään.



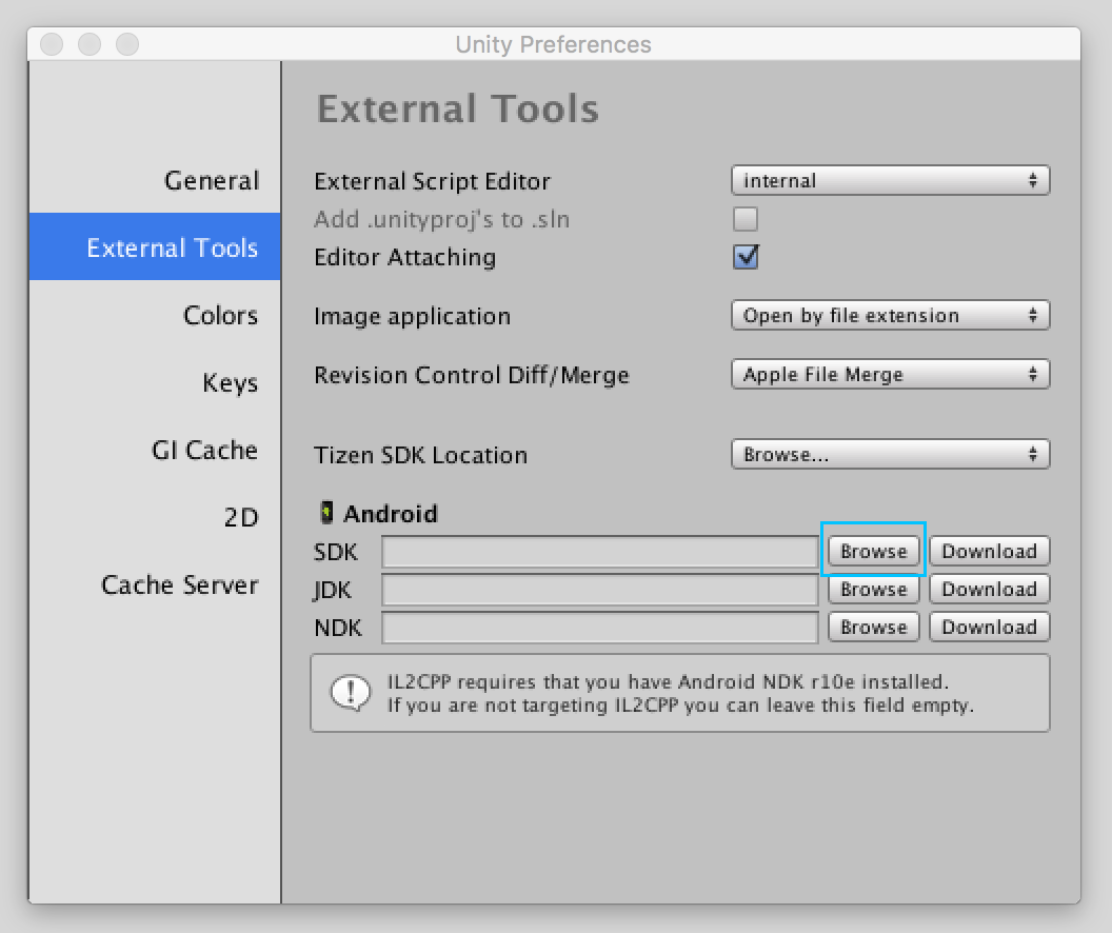
Kuvio 5. Ad Hoc provision profiilin koostumus. (Exporting Your App for Testing. n.d.)

TestFlight on järjestelmä, jolla levitetään testattavia sovelluksia testaajille. TestFlightin käyttö vaatii kehittäjätunnukset ja sitä käytetään iTunes Connectin kautta (TestFlight Beta Testing. N.d). TestFlightia käytettiin projektissa toimeksiantajan päätöksestä. TestFligtiin lähettäessä käytetään samanlaista provision profiilia kuin sovelluskauppaan lähettäessä, joten se voidaan generoida Xcoden kautta automaattisesti tai ladata Developer Portaalin kautta. Erona Ad Hoc provision profiiliin on testilaitteiden puute.

2.2 Julkaisu Androidille

2.2.1 Android sovellus Unity3D:llä

Android-versioiden kääntäminen ja testaaminen on helpompaa verrattuna iOS-versioihin. Android-versioiden ulos saamiseksi on tehtävä hieman enemmän esivalmistelua. Unity3D käyttää kahta ulkoista työkalua Android-versioiden tekemiseen: Java Development Kit (JDK) ja Android Software Development Kit (SDK).
JDK:n voi ladata Oraclen sivuilta. Asennus tapahtuu ladatulla ohjelmalla. Asentamisen jälkeen Unity saattaa löytää asennuksen sijainnin. JDK:n sijainnin voi kertoa Unityn editorista Edit → Preferences → External Tools. Kuviossa 6 on kuvattu mihin tiedostopolku kirjoitetaan ja millaiseen ikkunaan pitäisi päätyä aikaisemmilla ohjeilla. (Building your Unity game to an Android device for testing. n.d.)



Kuvio 6. Android työkalujen asettaminen (Building your Unity game to an Android device for testing. n.d.)

SDK:n voi ladata Android Developer sivustolta. Sivustoilla tähän voidaan viitata komentorivityökaluiksi. Kun ladattu paketti on purettu, voidaan sen tiedostoista valita ”tools” niminen kansio, jossa on tiedosto nimeltä ”android”. Android nimistä tiedostoa painamalla voidaan asentaa Android kehitykseen vaadittavia paketteja. SDK:n sijainti on kerrottava kuviossa 6 näkyvään sijaintiin Unityn editorissa. (Building your Unity game to an Android device for testing. n.d.)

2.2.2 Google Play Developer Console

Google Play Developer Console on Googlen ylläpitämä palvelu, johon kehittäjät voivat ladata Android-sovelluksia julkaistavaksi tai testattavaksi. Sovelluksia voi testata, julkaista, hallita hintoja ja tutkia statistiikkaa käyttäjistä yms. Google Play Developer Consolen käyttämiseen tarvitaan tili Googlen palveluihin. Tämän lisäksi käytöstä pitää maksaa 25 $. (How to use the Google Play Developer Console. n.d.)

Uuden sovelluksen lähettäminen tapahtuu Google Play Developer Consolessa Kaikki sovellukset -valikosta valitsemalla ”Luo uusi sovellus”. Tässä vaiheessa sovellukselle asetetaan nimi ja oletuskieli. Seuraavaksi valitaan ”Upload APK” APK-tiedostoa lähettäessä voi valita onko sovellus tuotanto-, beeta- vai alfa-vaiheessa.

Lähetettävän version versionumeron on kasvettava aina lähettäessä. Toistaiseksi maksimi on kaksisataa miljoonaa, jonka jälkeen uusia versioita samasta sovelluksesta ei voi lähettää. Lähettävä versio on oltava signeerattu sertifikaatilla. Android-ympäristössä sertifikaatit ovat Keystore-tiedostoja. Keystore-tiedostot ovat binääritiedostoja, jotka sisältävät yhden tai useamman privaatin avaimen. (Upload an app. n.d.) Tämän lisäksi, jos APK-tiedoston koko on yli 100 MB, tulee sovellus jakaa kahteen osaan. Jako tapahtuu Unityn editorissa valitsemalla Player-asetuksista ”Split Application Binary” valituksi. Käytännössä Unity3D tekee APK-tiedoston, joka sisältää kaiken mitä sovellus tarvitsee käynnistyäkseen, ja OBB-tiedoston, mihin säilötään kaikki muu. OBB-tiedostoa käytettäessä projektiin tulee liittää koodi, joka hakee tarvittavat tiedot käyttäjälle tiedostosta pelin käynnistyessä. Unityn Asset-kaupasta löytyy myös ilmainen liitännäinen, joka tekee tämän. (Unity: Support for Split Application Binary. n.d.)

Alfa ja beeta ovat eri tasoisia testausvaiheita. Googlen dokumentaatiossa suositellaan aloittamaan ensin pienellä alfa-ryhmällä ja siirtymällä myöhemmin laajempaan beeta-testiin. Suljetussa testiryhmässä versioita jaellaan sähköpostilla tai Google+ ryhmillä. Developer Consoleen voi rekisteröidä sähköpostit, joille lähetetään testiversiot aina uuden version ilmestyttyä. Testejä voi suorittaa, vaikka sovelluksesta olisi olemassa julkaistu versio. Versiokoodi pitää alfa-versioissa olla isoin, jotta niitä voi olla samaan aikaan kuin beeta-versioita. Jos beeta-versio, jolla on isompi versionumero kuin alfalla, lisätään palveluun, alfa-versiot lakkaavat toimivasta. Samoin, jos tuotantoversio julkaistaan isommalla versiokoodilla, lakkaavat testiversiot toimimasta. (Set up alpha/beta tests. n.d.)

2.3 Koodista peliksi

2.3.1 Yleiskatsaus

Unity projektin kääntäminen suoritettavaksi sovellukseksi käsin tapahtuu seuraavasti iOS-alustoille: Unity3D:llä aloitetaan pelin kääntäminen ja saadaan Xcode projekti, projekti avataan Xcodella ja arkistoidaan, arkistoitu projekti voidaan lähettää Xcoden kautta iTunes Connectiin. Lähetyksen alussa Xcode luo arkistosta IPA-tiedoston ja signeeraa sovelluksen. Toimeksiantajan mukaan operaatio voi pettää lähes kaikissa vaiheissa. Unityyn on voinut jäädä vääriä tietoja, jotka ilmenevät vasta lähettäessä. Xcode voi kieltäytyä lähettämästä IPA-tiedostoa, mutta satunnaisella kerralla lähettää sen.

Android-alustoille kääntäminen on suoraviivaista: Unityllä valitaan kohdealustaksi Android ja varmistamattoman APK-tiedoston saa suoraan ulos. APK-tiedoston voi lähettää Android-laitteelle, asentaa ja testata suoraan. Julkaistavissa sovelluksissa, jotka halutaan lähettää Googlen Play-kauppaan, tulee signeerata erillisellä keystore tiedostolla ja sen salasanoilla.

2.3.2 Unity3D

Unity3D on Unity Technologiesin omistama, suosittu pelimoottori, jolla voi tehdä 2D- ja 3D-pelejä monelle eri laitealustalle. Unityllä voi ohjelmoidessa käyttää C#:a tai UnityScriptiä. UnityScript on muunnelma JavaScriptistä. Ohjelmoinnissa käytetään komponenttimallia. (Unity: Creating and using scripts. n.d.)

Yksi tutkituista vaihtoehdoista koodin kääntämiseen oli kirjoittaa editorin suorittama skripti, joka käyttäisi BuildPipeline-luokan buildplayer-metodia ja loisi Xcode-projektin seuraavia vaiheita varten. Tämä Xcode-projekti voitaisiin kääntää, signeerata ja pakata Xcodella, mutta tarvittaisiin jokin ulkoinen skripti tai ohjelma hallinnoimaan operaatiota.

Etuja tällä ratkaisulla olisi korkea kustomointi ja mahdollisesti helpompi ylläpidettävyys. Toimeksiantajalla oli tarkasti määritelty nimeämisperiaate, ja iTunes Connectiin lähettäessä versionumeron on oltava edellistä suurempi. Tällä ratkaisulla sekin voidaan automatisoida, eikä koodaajan tarvitsisi itse muistaa kasvattaa lukua Unityn sisällä asetuksista.

2.3.3 Unity Cloud Build

Unity Cloud Build on Unity Technologiesin tarjoama palvelu, jolla voi kääntää, testata ja jakaa käännöksiä Unity-projekteista. Unity Cloud Build toimii seuraavasti: Kirjaudut palveluun, luot uuden projektin, määrität projektin versiohallinnan sijainnin. Cloud Buildin voi asettaa tekemään käännöksen aina muutoksen sattuessa tai manuaalisesti asettamalla. Käännöksen valmistuessa projektin jäsenet saavat käännöksen latauslinkin sähköpostiinsa.

Unity Cloud Buildissa voi mukauttaa käännöstä määrittelemällä sille Unity- ja Xcode-version käsin. Xcoden käyttämät provision profiilit on ladattava palveluun iOS-käännösten tekemiseen. Toinen yleinen tapa mukauttaa projektin kääntämistä on käyttää PostProcessBuild-skriptejä. PostProcessBuild-skripteillä voi määrittää operaatioita, jotka toteutetaan käännöksen lopussa. Tavallisesti PostProcessBuild-skripteissä määritellään projektiin lisättäviä kirjastoja ja projektiasetuksia.

Projektissa harkittiin Unity Cloud Buildin käyttämistä, mutta suurin havaittu este esitutkimuksen aikana oli käännettävän projektin kokoraja. Tämä kokoraja muutettiin kesken projektin, ja Unity Cloud Build otettiin uudelleen harkintaan. Lieväksi esteeksi koettiin Xcoden provision profiilien vuosittainen lähettäminen ja jatkuvan versionumeroinnin automatisointi. Unity Cloud Build myös rajoittaa palvelun käyttöä riippuen käyttäjätilin lisenssistä. Esimerkiksi 6.1.17 testattuna ilmaisella lisenssillä projektissa joutuu odottamaan tunnin käännösten välissä. Unityn omiin julkaisuihin tämä latausaika on joko jätetty mainitsematta tai piilotettu todella hyvin. Jos maksullisissa versioissa on vastaavanlainen latausaika, niin se johtaisi kiusallisiin tilanteisiin Applen päivittäessä järjestelmiään. Käännöksessä epäonnistuminen asettaa projektin myös latausajalle ennen seuraavaa käännöstä. Aikaisempien kokemuksien perusteella aina kun Apple tekee muutoksia, käännös epäonnistuu muutaman kerran epäselvien virheilmoituksien takia.

2.3.4 uTomate

Unity-projektiin voi lisätä ulkoisia lisäosia, ja niitä myös myydään Unity Technologisin ylläpitämällä Asset Store -palvelussa. uTomate – Automation Solution on 50 $ maksava Ancient Light Studion julkaisema lisäosa. uTomatella pystyy automatisoimaan lähes kaikki Unityn toiminnot ja jopa käännösten lähettämisen tietyille jakamisalustoille. (Unity asset store: uTomate. 2013.)

uTomaten komentoja voi ajaa komentoriviltä, ja sille voi tehdä omia kustomoituja operaatioita. Kustomoiduilla komennoilla voisi tehdä esimerkiksi operaation, jossa tietyt kentät lisätään projektiin, niille luodaan valot, ajetaan testit yms. Valikot, joilla uTomatea käytetään, löytyvät Unity-editorin yläpalkeista tai klikkaamalla hiiren oikealla näppäimellä jotain relevanttia kohdetta, joihin kuuluvat mm. kansiot ja Unityn prefab objektit. (uTomate manual. n.d.)

2.3.5 Xcode

Xcode on Applen integroitu ohjelmointiympäristö Mac:lle ja iOS:lle. Kesäkuussa 2016 julkaistiin Xcode 8. Xcode sisältää kaiken mitä kehittäjä tarvitsee kehittääkseen sovelluksia Applen laitteisiin. Xcodeen sisältyviä työkaluja ovat mm. Interface Builder, jolla voi suunnitella käyttöliittymän kuin se olisi kohdelaitteella, editori lisäkkeet joilla voi muokata omaa koodausympäristöä, Swift 3, ajoaikaisten ongelmien tunnistajan, jolla voi metsästää hankalasti löydettäviä ongelmia samalla kun ohjelmaa ajetaan ja paljon muuta. Suurin merkittävä muutos Xcode 8:ssa verrattuna aiempaan versioon on sovellusten signeeraamisen vaadittavien avainten hallinta ja käyttö. Ennen Xcodessa oli pakko määrittää millä provision profiililla sovellus signeerataan, mutta nyt Xcode voi päätellä oikean provision profiilin lähes automaattisesti. (Xcode. n.d.)

Unity3D tekee pelistä Xcode-projektin, kun sillä aloittaa käännöksen iOS-alustalle. Tämä Xcode-projekti tulee arkistoida, pakata, signeerata ja lähettää iTunes Connectiin, jos sitä halutaan levittää testaajille käyttäen TestFlightia.

Xcode tukee jatkuvaa integraatiota. Prosessi vaatii Xcode-palvelimen, jonka saa OS X -palvelimen osana. Paikallisella Mac-koneella on mahdollista tehdä botteja, joilla voi automatisoida kääntämistä, arkistointia ja lähettämistä. Botit lähetetään palvelimelle, jonka jälkeen ne suorittavat niille määriteltyjä toimintoja käyttäen versiohallinnasta löytyvää koodia. (About Continuous Integration in Xcode. N.d.)

Xcodeen on integroitu sovellus nimeltä Application Loader. Sen tarkoituksena on tarjota nopea lähettäminen, vakaat yhteydet lähettäessä ja aikaiset varoitukset mahdollisista virheistä. (Using Application Loader. N.d.) Application Loaderilla voisi saada varmemmin lähetettyä pakatun sovelluksen. Empiirisesti testattuna näyttäisi, että Unity-projekteilla yleisesti käytetty Archive → Send archive ei toimi isoilla projekteilla. Vaikuttaisi, että lähettäessä avataan yhteys iTunes Connectiin ja aloitetaan IPA-tiedoston pakkaaminen. Jos pakkaamisessa ja siihen kuuluvassa signeerauksessa menee liian kauan, yhteys ehtii katketa ja lähetys epäonnistuu.

2.3.6 Shenzhen

Shenzhen on osa isompaa avoimen lähdekoodin projektia nimeltä Nomad. Nomad on kokoelma komentorivityökaluja, joilla helpotetaan iOS-alustoille kehittämistä. Shenzhenillä pakataan kehitysversioita ja lähetetään pakatut käännökset eri jakelualustoille. (Thompson, M. 2014.) Shenzhenin käyttö on yksinkertaista ja sen asentaminen onnistuu Rubyn gem komennolla tai suoraan Githubista projektin versiohallinnan säilöstä.

2.3.7 Fastlane

Fastlane sai alkunsa Felix Krausen sivuprojektina 2014. 2015 Fastlane otettiin osaksi Fabric nimistä projektia. Fabric on Twitterin kehittämä kehitysalusta mobiilisovelluksille. Nykyään Krause työskentelee Twitterillä Fastlanen parissa täyspäiväisesti. (Krause, F. 2015.) Huhtikuussa 2016 Fastlanen oli laskettu säästäneen yli miljoona tuntia sovelluskehittäjien työajasta (Krause, F. 2016).

Fastlane on kokoelma työkaluja, joilla julkaistaan iOS- ja Android-sovelluksia. Fastlane on avoimen lähdekoodin projekti, joka on jaossa Githubissa (Automation done right. n.d.). Osa Fastlanen sovelluksista on tehty korvaamaan Nomad projektin osia (Fastlane. 2016). Opinnäytteen yhteydessä toteutetussa projektissa meitä kiinnostivat erityisesti Pilot ja Gym nimiset työkalut.

Pilot on ohjelma, jolla lähetetään ja hallitaan pakattuja sovelluksia sekä hallitaan testaajia (Pilot readme. N.d). Gym kääntää, pakkaa ja signeeraa IPA- tai APP-tiedoston. Gym on tarkoitettu olemaan helppo peruskäytössä, mutta tarjoaa myös säätömahdollisuuksia projektikohtaisesti. Gym tarvitsee Xcoden ja käyttää sitä uusimmilla rajapinnoilla, joista keskeisimmät ovat xcodebuild ja xpretty. (Gym readme. n.d)

2.4 Jakamisalustat

2.4.1 TestFlight

TestFlight on Apple Inc:n tarjoama palvelu, jolla voi jakaa testattavia sovelluksia testaajille palautteen saamiseksi ja julkaisun valmistelemiseksi maksimissaan 100:lle sovellukselle yhtä aikaa. TestFlightissa on kahden tyyppisiä testaajia: sisäisiä ja ulkoisia. Sisäisen testaajan tulee olla osa ryhmää, joka jakaa kehittäjälisenssin jolla sovellus on signeerattu. Sovelluksesta tarvitsee antaa huomattavasti vähemmän metadataa, kun sitä testataan sisäisillä testaajilla. Sovelluksilla, joita testaavat ulkoiset testaajat, pitää olla kerrottuna mm. mitä testataan, kuvaus sovelluksesta ja sähköposti palautteen annolle. Sovelluksien, joita halutaan ulkoisten testaajien testaavan, tulee läpäistä Applen tekemä tarkastus. Tässä tarkastuksessa menee versiosta ja muutoksista riippuen enemmän aikaa, kuin jos sovellusta testattaisiin pelkästään sisäisillä testaajilla. Ulkoisia testaajia sovelluksella voi olla maksimissaan 2000 kappaletta. Testaajat kutsutaan sähköpostilla ja yksittäisen version testaus on voimassa 60 päivää kutsun hyväksymisen jälkeen. (TestFlight Beta Testing. N.d.)

2.4.2 Google Play Developer Console

Google Play Developer Console on Googlen tarjoama palvelu, jonka kautta kehittäjät voivat lähettää Android sovelluksia Googlen Play kauppaan, levittää testiversioita, määritellä sovellusten hintoja, tutkia dataa sovellusten käytöstä ja paljon muuta. Palvelu vaatii rekisteröitymisen ja siihen kuuluvan maksun 25 $. (How to use the Google Play Developer Console. N.d.) Google Play Developer Consolea voisi verrata Androidin iTunes Connectiksi.

Testiversioita voi levittää käyttäjille kolmella tavalla käyttäen Google Developer Consolea:

- Sähköpostilla
- Play kaupalla
- Google+:lla

Suljettuja testejä voi tehdä sähköpostilla ja Google+:n ryhmillä. Google Developer Consoleen voi tehdä 50 sähköpostilistaa per taso (beeta ja alfa) ja jokaiselle listalle

voi asettaa kaksi tuhatta testaajaa. Google Developer Consolen dokumentaatiossa Google+ ryhmälle testiversioiden jakamiseen ei ole mainittu rajaa käyttäjä määristä. (Set up alpha/beta tests. n.d.)

Avoimia testejä voi suorittaa suoraan Play kaupassa. Sovelluksille joilla ei ole tuotantoversiota ennestään olemassa sovellus näkyy julkisena beeta-versiona hakukenttää käyttämällä. Jo julkaistuille sovelluksille käyttäjät voivat asentaa beeta-version sovelluksen julkisesti näkyvästä kauppasivusta. (Set up alpha/beta tests. n.d.)

Projektin alkuperäisissä tavoitteissa Android käännösten tekeminen ja levittäminen oli priorisoitu toissijaiseksi, koska Android-versioiden testaaminen oli suoraviivaisempaa kuin iOS-versioiden.

2.4.3 Dropbox

Dropbox on palvelu, johon käyttäjä voi tallentaa tiedostonsa, jakaa niitä ja synkronoida niitä eri laitteilta eri käyttäjien toimesta, riippuen asetuksista. Tiedostoja voi käyttää ja tarkastella selaimen, tietokoneen ja puhelimen kautta. Dropboxissa voi jakaa yksittäisen tiedoston tai kansion. Jakamisen monipuoliset vaihtoehdot takaavat Dropboxin hyödyllisyyden yhteisille projekteille työelämässä ja vapaa-ajan käyttöön esimerkiksi jakamalla yksittäisiä kuvia ystäville. Droboxin mukaan yli 500 miljoonaa ihmistä ympäri maailmaa käyttää Dropboxia auttamaan erilaisissa projekteissa. (The Dropbox Tour. n.d)

Testattavien Android-sovellusten jakamiseksi suunniteltiin Dropboxin käyttöä, koska niiden testaaminen Android-alustoilla onnistuu suoraan asentamalla sovellus puhelimelle APK-tiedostosta.

Toimeksiantaja käytti Dropboxia jo ennestään Android sovellusten sisäiseen testaamiseen. iOS sovellusten testaamiseksi Dropbox ei sovellu helposti. Testilaitteet tulisi rekisteröidä Applen kehittäjätilille, jolla on rajana 100 laitetta (Exporting Your App for Testing. n.d). Sisäiseen testaamiseen Dropbox ja iOS siis soveltuisivat, mutta eivät ulkoiseen testaamiseen tai asiakkaalle luovutettavan testiversion jakamiseen.

2.5 Käännöksen aloittaminen

2.5.1 Jenkins

Jenkins on monialustainen jatkuvan integraation (Continuous Integration) työväline. Jenkinsiä käytetään sovellusten kääntämiseen ja testaamiseen automaattisesti helpottaen kehittäjien työtaakkaa ja auttamaan testaajia saamaan aina ajan tasalla olevan version testattavasta sovelluksesta. Jenkinsin etuihin lukeutuu mm. helppo asennus, helppo ja kattava konfigurointi, laaja valikoima valmiita lisäosia (plugin) ja kyky lisätä useita koneita ns. orjakoneiksi tekemään testejä tai käännöksiä. (Meet Jenkins. 2016.)

Vaikka Jenkins on dokumentaationsa mukaan jatkuvan integraation työväline, sillä voi toteuttaa myös helposti järjestelmän jatkuvalle toimitukselle (Continuous Delivery), jossa kohdesovellusta käännetään ja testataan tiiviiseen tahtiin. Jenkins on niin säädettävissä, että sillä voi automatisoida lähes mitä tahansa suoritettavia komentoja.

> *Jenkins serves as the workflow engine to manage this CI/CD pipeline from source to delivery, Labourey says, but along the way many different tools may be called upon to perform different functions.* (Knorr, E. 2016.)

2.5.2 Etähallinta puhelinsovelluksella

Esitutkimuksen jälkeen toimeksiantaja tiedusteli mahdollisuutta toteuttaa puhelinsovellus, jolla voisi aloittaa uuden testiversion tekemisen. Vaatimuksena esitettiin sovelluksen toimimista Android- ja iOS-puhelimilla. Jenkinsiä käyttäessä olisi mahdollista käyttää töiden aloittamiseen Jenkinsin rajapintaa (Jenkins: Remote access API. n.d).

Puhelinsovellus arvioitiin myös käteväksi tavaksi helposti hallinnoida etänä tapahtuvaa testiversion tilaamista. Puhelinsovelluksella voisi myös näyttää arvion kauanko version tekemiseen menee aikaa ja ilmoittaa mahdollisista virheistä operaation aikana.

2.6 Raportointi

2.6.1 Trello

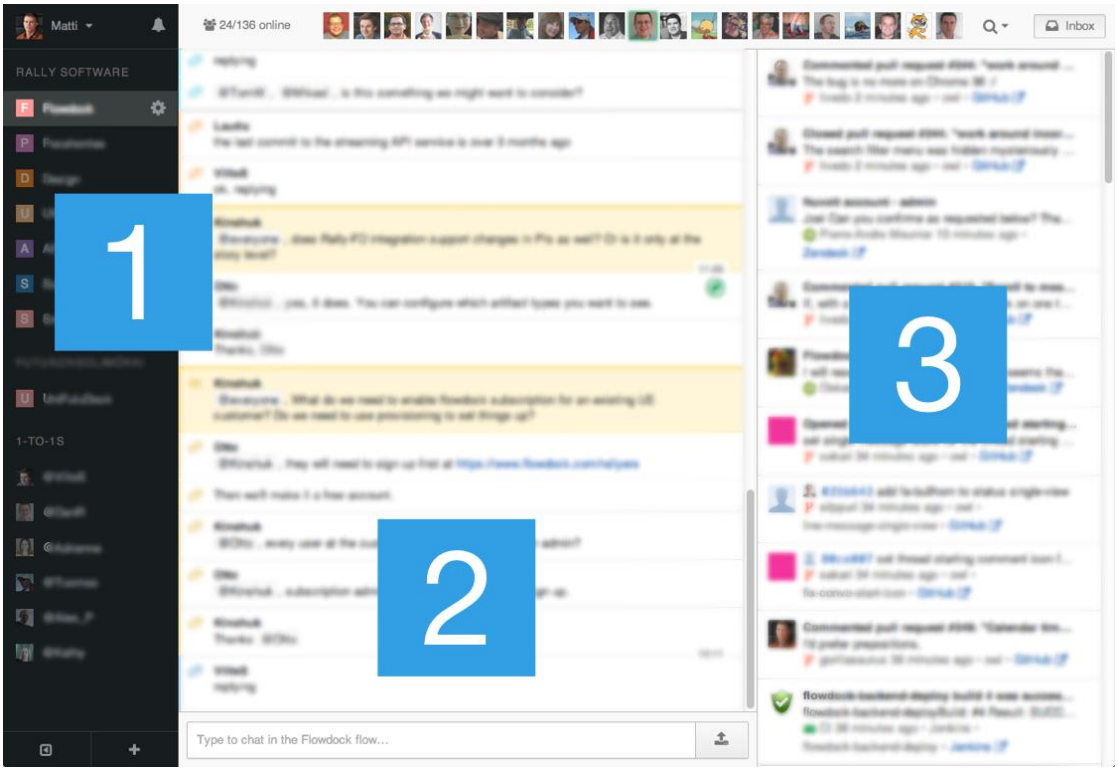
Trello on selaimella tai puhelimella käytettävä ilmainen sovellus, joka jakaa projektit erilaisiksi tauluiksi. Tauluihin voi lisätä kortteja, ja kortit voi järjestää erillisiin tasoihin kuten ”To-Do”, ”Doing” ja ”Done”. Korteille voidaan asettaa henkilöitä, jotka tekevät kortilla määriteltyä tehtävää, kortille voi kommentoida, lisätä sisäisiä tehtävälistoja yms. (Trello tour. n.d.)

Toimeksiantaja käytti Trelloa projektienhallintaan ja samalla myös kevyenä dokumentaatioalustana. Kaikki tehtävät, jotka on tehty, joita tullaan tekemään ja joita tehdään näkyvät Trellossa kanban tyylisesti. Toimeksiantaja oli kokeillut parissa projektissa systeemiä, missä aina kun versiohallintaan lähettiin koodia, kirjattiin, mitä oli lähetetty. Projektille asetettiin tavoitteeksi automatisoida tämä kirjaussysteemi käyttäen Trellon julkista rajapintaa.

Trellolla on julkinen rajapinta, jolla voi muokata tauluja, kortteja, taulun käyttäjien oikeuksia ja paljon muuta. Rajapinnan käyttö vaatii avaimen, jolle on annettu jonkun käyttäjän oikeuksia. Rajapintaa käytetään lähettämällä http-protokollan mukaisia pyyntöjä rajapinnalle dokumentaation määrittelemään osoitteeseen määritellyillä parametreilla. Trello on tehnyt oman kirjaston rajapinnan käyttämiseen, mutta sitä ei tässä projektissa käytetty. (Start Building with Trello. n.d.)

2.6.2 Flowdock

CA Flowdock on CA Technologies:n omistama viestintäsovellus. Flowdockissa keskustelu käydään kanavien sisällä. Jokainen viesti kanavassa on osa isompaa viestilankaa. Tämä tarkoittaa sitä, että Flowdockin chatti näyttää perinteiseltä chattiruudulta, mutta klikkaamalla viestiä, pääsee näkymään missä langan, johon viesti kuului, viestit näkyvät. Tämä aluksi epäselvä systeemi on pyritty havainnollistamaan kuviossa 7.



Kuvio 7. Flowdockin UI. 1 on kanava valikko, 2 chatti ja 3 kanavan postilaatikko (Flows. n.d).

Toimeksiantaja käytti Flowdockia sisäiseen viestintään. Projekteilla oli omat kanavansa, johon tulivat projekteihin liittyvät viestit. Jokaisella projektilla on oma viestiketjunsa, johon ilmoitettiin, koska oli lähetetty testattava versio iTunes Connectiin, tai mihin laitettiin linkki Android version lataamiseksi. Projektissa päätettiin, että systeemi ilmoittaisi vanhoihin lankoihin uusia viestejä aina, kun operaatio onnistuu.

Flowdockilla on rajapinta, jolla voi lähettää viestejä ja toteuttaa muita toimintoja. Rajapinnasta on tämän työn tekohetkellä kolme erilaista vaihtoehtoa: Push API, REST API ja Streaming API. Push API on poistumassa ja sitä ei suositella enää käytettäväksi. Sitä käytetään käyttäjätunnuksen omistamalla avaimella ja pääsääntöisesti viestien lähettämiseen ilman käyttäjän tunnistamista. REST API:lla käyttäjä tunnistetaan ja sillä voi lähettää, vastaanottaa ja muokata tietoja kanavista ja viesteistä. Streaming API:lla voi kuunnella kanavalla lähetettyjä viestejä reaaliaikaisesti. (The CA Flowdock API. N.d.)

2.7 Git ja Bitbucket

Git on versionhallintatyökalu. Git sai alkunsa Linus Torvaldsin kehitellessä korvaavaa systeemiä Bitkeeperille vuonna 2005. Projektissa, missä käytetään Gittiä, jokaisella käyttäjällä on oma paikallinen kopio. Paikallisiin kopioihin tehdään muutoksia ja paikalliset kopiot lähetetään palvelimelle, missä pääversio sijaitsee. Palvelimella eri versiot nidotaan yhdeksi kokonaisuudeksi. Nykyään Git on laajassa käytössä niin avoimen lähdekoodin projekteissa kuin kaupallisissa sovelluksissa. (What-is-git. n.d.)

Bitbucket on Atlassian omistama versiohallintapalvelu. Käytännössä palvelin, johon voi asettaa omia projektejaan ja niiden versiohallinnan. Bitbucket tukee Git:n lisäksi VCS:ää, muttei toistaiseksi SVN:ää. Bitbucket tarjoaa palvelussaan paitsi paikan säilöä koodia ja versiohallinnan ainakin seuraavia ominaisuuksia: Pull requestit, rajattomasti yksityisiä repoja, oksien vertailun, koodin historian, graafinen työkalu SourceTree ja monia valmiita integraatioita. (Github-vs-bitbucket. N.d.)

3 Automatisaatio ja pelinkehitys

> *Automation is so useful in game development. It will take long, drawn out and error prone processes and make them fast, repeatable, reliable and has the potential to free up your time for the things that really matter. This extra time can help you concentrate on making a better game.* (Davis & Single. 2016)

3.1 Automatisointi

Kääntäminen ja testaaminen ovat usein toistuvia operaatioita, vievät paljon aikaa ja ovat siksi usein hyviä kohteita automatisoinnille. Automatisointi vie kuitenkin aikaa pois pelinkehityksestä ja siksi on tärkeää miettiä, mitä automatisoidaan ja onko sille oikeaa tarvetta. Automatisoimalla ei voida myöskään korvata täysin käsin tehtäviä operaatioita.

Automatisoimalla pyritään tehostamaan ajankulutusta. Testaamisen automatisointi antaa mahdollisuuden suorittaa testausta esimerkiksi yön aikana. Aamuisin kehittäjät voivat tutkia testien tuloksia ja tehdä tarvittavia korjauksia. Jatkuva testaus oikein

suoritettuna myös takaa dokumentoimattomien epätoivottujen ominaisuuksien ilmenevän aikaisemmin kehitysketjussa. (Helppi. 2015.)

3.2 Testauksen tarkoitus

Mitä testauksella yritetään saavuttaa? Testatun ohjelmistosovelluksen tulisi:

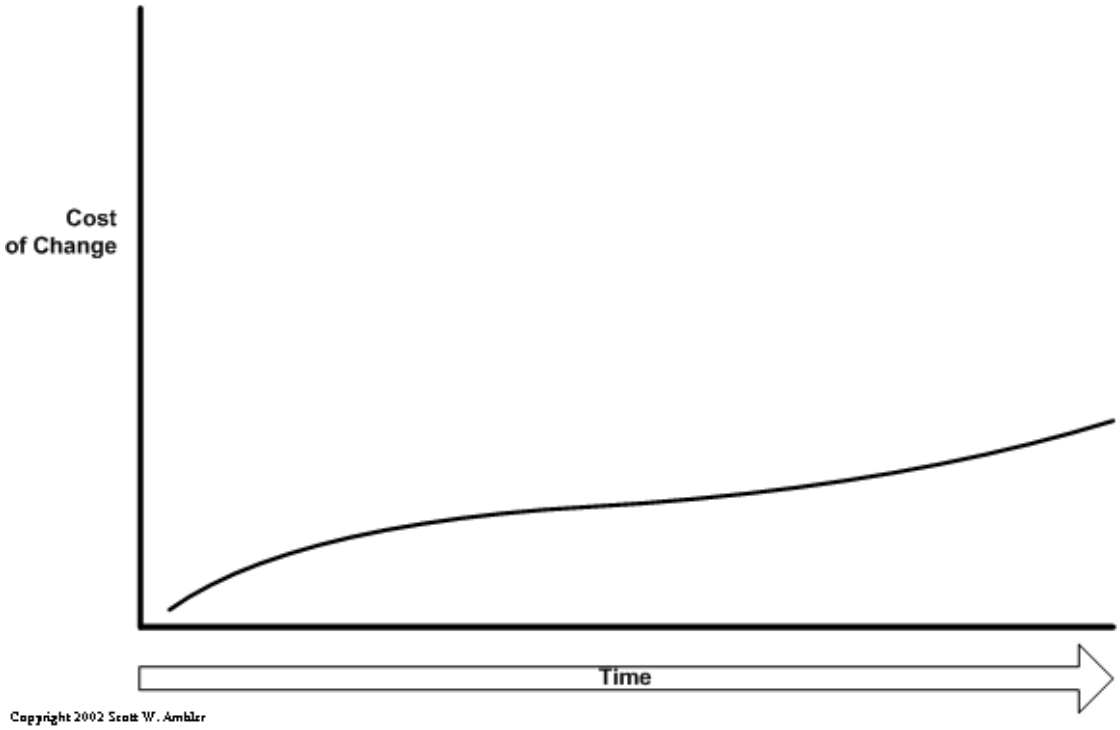
- Täyttää sille asetetut vaatimukset
- Toimia virheettä
- Toimia minimi resursseilla
- Olla vakaa
- Pelien tulisi näiden lisäksi olla hauskoja

Kaikkea näitä ei voi saavuttaa täysin testaamalla. Aikarajat ja resurssien puuttuminen rajoittavat testauksen ja kehittämisen kattavuutta. Hauskuus on subjektiivinen käsite ja siksi vaikea testata. Automatisoidulla testauksella ei voi mitenkään testata peli-idean hauskuutta, mutta ollakseen toimiva, pelin on toimittava. Toimivuuden testaamista voidaan automatisoida. (Davis & Single. 2016.)

Ohjelmistotuotteen kehittämisen kustannukset voidaan lajitella kolmeen osioon:

- Resurssien (Asset) tuottaminen (Musiikki, 3D-mallit yms)
- Koodin tuottaminen
- Koodin korjaaminen ja ylläpito

Pelit elävät jatkuvasti kehityksen aikana. Erilaisia ominaisuuksia testataan, kunnes oikea määrä hauskuutta löytyy. Koodin korkea ylläpidettävyys nousee hyvin tärkeäksi. Kuviossa 8 kuvataan, mikä on realistinen muutoksen hinta suhteessa aikaan, ketteriä menetelmiä käytettäessä ohjelmistoprojektissa. Automatisoimalla testausta ongelmat pyritään löytämään mahdollisimman ajoissa ja muutoksiin kuluvat resurssit matalana. Testauksella ja sen automatisoinnilla pyritään säästämään rahaa. Testauksen automatisointi on ohjelmistoalalla ollut yleisesti hyväksytty työkalu siihen jo kymmeniä vuosia. (Carucci. 2009)



Kuvio 8. Realistinen kuvio muutoksen hinnasta ketterillä menetelmillä (Ambler. 2002)

3.3 Testaus peliprojekteissa

Pelien testaus tapahtuu usein käsin siihen erikseen palkattujen, tai valikoituneiden vapaaehtoisten, ihmisten toimesta. Testaajan työ koetaan stressaavaksi, pitkäveteiseksi ja vain pääsytienä muihin tehtäviin pelialalla (Schreier. 2017).

> *Video game quality assurance (QA), as testing is called, is often perceived as "playing games for a living" but might better be described as breaking them. It's a low-paying, occasionally rewarding, often frustrating job that has both more and less to do with the quality of today's games than you might expect.* (Schreier. 2017.)

Testaaminen on kuitenkin olennainen osa peliprojekteja. Pienellä yrityksellä harvemmin on erillistä QA-osastoa ja kaikki työntekijät ottavat osaa pelien testaamiseen. Kuviossa 9 on kuvattu arvio pelinkehitykseen käytetystä ajasta ja miten eri testaustavat kuuluvat siihen ja mihin aikaan. Todellisuus vaihtelee projekteittain ja yrityskohtaisesti.

Pelin testaaminen alkaa varhaisessa vaiheessa nopeasti tehdyillä prototyypeillä. Protovaiheessa on hyvä aloittaa Fokus-testaamisella. Fokus-testaamisella selvitetään varhaisessa vaiheessa, onko idea, teema tai tyyli sopiva pelille. Proton kehittyessä

voidaan siirtyä käytettävyystestaamiseen, jossa selvitetään peli intuitiivisuus ja ymmärtääkö pelaaja tavoitteet sekä kaikki ominaisuudet. Käytettävyystestausta kannattaa toteuttaa jo melko varhaisessa vaiheessa. Tällä vältytään menemästä väärään suuntaan kesken kehityksen. Tasapainottaminen on oma osionsa testaamisessa missä selvitetään ovatko pelin osat kuinka vaikeita tai ovatko tietyt ominaisuudet liian ylivoimaisia. Ihannetilassa tasapainottamista tehdään, kun kaikki pelattavuuteen vaikuttavat ominaisuudet ovat valmiita. Vikojen etsimisessä yritetään löytää virheitä ja ominaisuuksia, jotka eivät toimi halutuilla tavoilla. Vikoja etsitään aina uuden ominaisuuden valmistuessa. Yhteensopivuus-testauksella testataan, miten peli toimii eri laitteilla. (Smith. 2008)

Kuvio 9. Skaalautuva arvio pelinkehitykseen käytetystä ajasta (Smith. 2008.)

3.4 Automatisoidun testaamisen tyypit

Yksikkötestaus testaa koodinpätkiä erillään muusta koodista. Yksikkötestien tekeminen oikein vaatii taitoa, työtä, huolenpitoa ja sillä on suuri vaikutus koodin suunnitteluun. Yleensä hyvä vaikutus ja johtaa modulaariseen koodiin, mutta vie enemmän aikaa. Yksikkötestaus kannattaa jättää koodille, jonka on pakko olla lähes virheetöntä pelin toiminnan kannalta. (Davis & Single. 2016.)

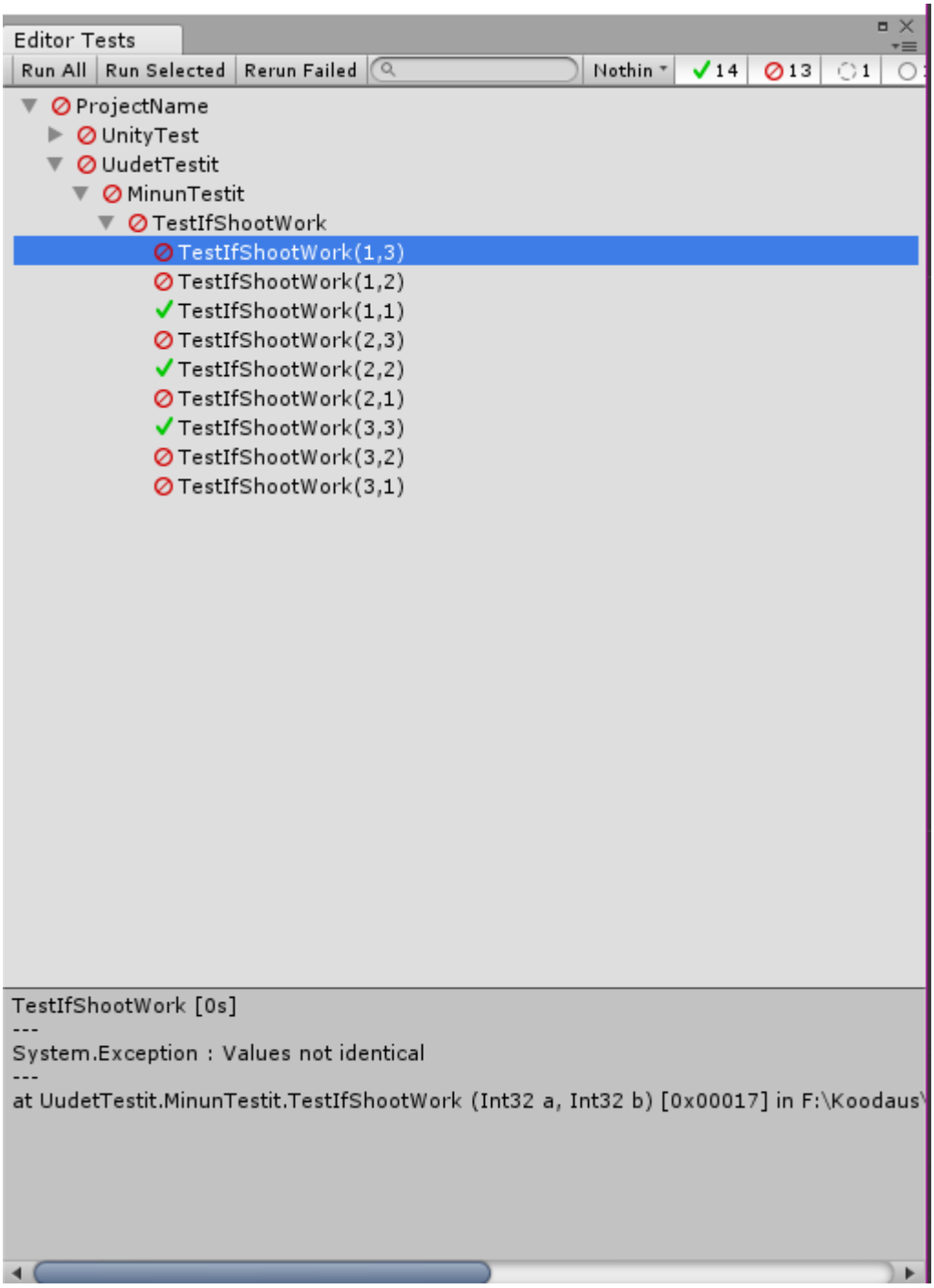
Integraatiotesteillä pyritään selvittämään, miten sovellus käyttäytyy reunatapauksissa (Geig. 2014). Integraatiotestaus on helpompi toteuttaa verrattuna kattaviin yksikkötesteihin, koska testeissä ei ole pakko pitää osia erillään. Integraatiotestejä voi myös toteuttaa jälkikäteen valmiiksi olevalle koodille, koska se ei vaadi koodin suunnittelua etukäteen testauskelpoiseksi samalla tavalla kuin yksikkötestaus. (Davis & Single. 2016.)

Savutestauksella (Smoke Tests) testataan kokonaisia systeemin paloja, tai koko peliä. Esimerkiksi pelin eri kenttiä voidaan ladata eri kohdista ja etsiä mahdollisia kaatumispisteitä. Pelin kohtia tai kenttiä voidaan ajaa skriptoilla läpi tai pakottaa skriptillä pelille erilaisia komentoja ja tutkia mitä tapahtuu. (Carucci. 2009)

Automatisoitu testaus voidaan toteuttaa käsinkirjoitetuilla testiskripteillä tai määrittelemällä manuaalisesti tallenteita (Record-Playback), joita ajetaan automaattisesti testaten ominaisuuksia. Testiskriptien tekeminen on paras vaihtoehto, kun testit tehdään ohjelmointiin kykenevän ihmisen toimesta. Testien luontiin on valtava määrä työkaluja ja niillä voi testata täsmälleen sitä, mitä milloinkin halutaan. Tallenteiden tekeminen on vähemmän virhealtista manuaalisen toteutuksen takia. Tallenteilla testaus on rajoittuneempaa kuin skripteillä testaaminen. Tallenteilla testataan yleensä käyttäjän mahdollisia toimintoja ja niiden toimimista erilaisissa ympäristöissä, kuten esimerkiksi eri puhelimilla. (Helppi. 2015.)

3.5 Automatisointitestaus Unity3D:llä

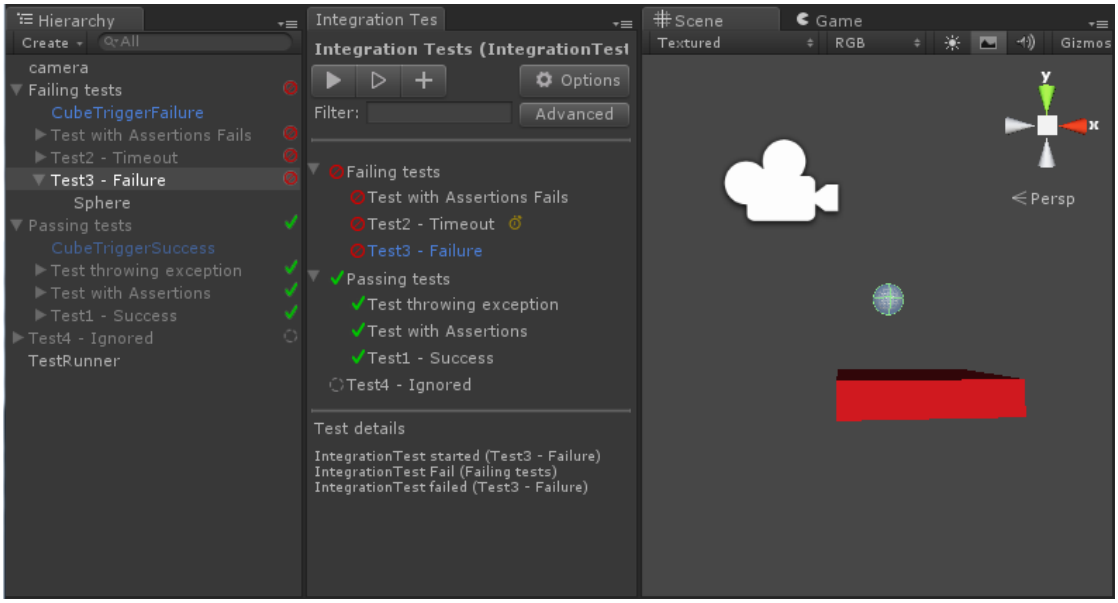
Unity3D:llä työkalu yksikkötestien tekemiseen on ollut editorissa versiosta 5.3 lähtien. Työkalu oli aluksi nimeltään ”Unit Tests Runner” ja Unityn 5.4 versioista se löytyy Unity editorista Window → Editor Test Runner. Työkalu käyttää NUnit-ohjelmistokehystä, joka on yleisesti käytössä oleva testikirjasto .Net kielille. Testit tulee luoda Editor nimiseen kansioon. Testejä voidaan ajaa komentoriviltä tai editorista käsin. Editorilta ajettaessa testien tulokset näkyvät kuviossa 10 näytetyllä tavalla. (Unity Test Tools Documentation. n.d.)



Kuvio 10. Työkalu yksikkötestien ajamiseen Unityn editorissa

Integraatiotestejä varten Unity Technologies on julkaissut ilmaisen lisäosan, joka on saatavilla Unityn Asset Storesta. Integraatiotestit tehdään omiin kenttiinsä ja jokainen testi on erillään toisistaan. Tarkoittaen, että sama kenttä voi pitää sisällään monia integraatiotestejä, mutta ei mitään varsinaiseen peliin kuuluvaa. Testeihin suositellaan käytettäväksi peliä varten luotuja valmiita prefab-objekteja ja testata miten objektit käyttäytyvät erilaisissa tilanteissa. (Unity Test Tools Documentation. n.d.)

Integraatiotestin luomiseksi luodaan tyhjä kenttä. Tämän jälkeen avataan Unityn editorin yläkulmasta ”Unity Test Tools” → ”Integration Test Runner” ja klikataan ”Create” nappulaa. Jokaisella yksittäisellä testillä on objekti, jolla on ”TestComponent” luokan ilmentymä kiinni itsessään. Kaikki testiobjektin lapsiobjektit kuuluvat kyseiseen testiin. Muut objektit ovat yhteisiä kaikille testeille. Näitä yhteisiä objekteja voivat olla mm. lattia, taivas, seinät yms. Integraatiotestaus työkalu ja testien hierarkia järjestelmä näkyvät kuvassa 11. (UnityTestTools: How to use the Integration Test Framework. n.d.)



Kuvio 11. Esimerkki Unityn integraatiotestaus työkalusta (UnityTestTools: How to use the Integration Test Framework. n.d.)

Savutestaus vaatii Unity-projekteissa suunnittelua. Skriptit, joilla testataan pelin toimimista eri vaiheissa pitää tehdä käsin. Pelin koodi täytyy suunnitella siten, että ominaisuuksien ohjaaminen on mahdollista skriptin toimesta. Savutestaus on myös mahdollista suorittaa ulkoisilla työkaluilla, joilla peliä ajetaan kohdealustalla pitkiä aikoja.

4 Työn toteutus

4.1 Yleiskatsaus

Järjestelmä, joka projektissa luotiin on Jenkins-palvelin, joka suorittaa ennalta määritettyjä töitä. Palvelin kuuntelee ainoastaan paikallisen koneen verkkoliikennettä.

Kone, jolla palvelin on, suorittaa skriptiä joka tarkistaa tasaisin väliajoin ulkoisella palvelimella olevia tiedostoja, joissa pidetään kirjaa tilattavista versioista. Järjestelmä on kuvattu liitteessä 1. Liitteessä 2 kuvataan eri tilat, joihin on mahdollista päätyä suoritettavan työn aikana.

Ulkoisella palvelimella on tiedostot, joihin on kirjattu versioitavat pelit, aika joka niiden kääntämiseen menee, päivämäärä milloin viimeisin versio on tehty ja mahdolliset projektikohtaiset virheet. Palvelimella on myös virhetiedosto, johon kirjataan virheet jotka estävät koko järjestelmän käytön. Kaikissa virheraporteissa on selvityksenä viesti, josta käy ilmi missä vaiheessa käännösketjua virhe on tapahtunut.

Järjestelmän ytimenä toimii Jenkins palvelin, johon on määritelty projektikohtaiset työt. Ulkoisten sovellusten ja palveluiden käyttö suoritettiin Bash-skripteillä ja Jenkinsillä suoritettiin kyseisiä skriptejä. Projektikohtaisten töiden lisäksi Jenkins asetettiin suorittamaan tasaisin väliajoin tarpeellisia operaatioita. Näihin operaatioihin kuului Xcoden generoimien tiedostojen tyhjennys ja signeeraamattomien Android-versioiden jakeluun käytetyn Dropbox tilin tyhjennys.

Järjestelmää käytetään selaimella Jenkinsin hallintapaneelista tai projektin aikana luodulla puhelinsovelluksella. Järjestelmän havaitsemat virheet näytetään puhelinsovelluksessa ja estävät virheen aiheuttaneen pelin versioinnin tai koko sovelluksen käytön, kunnes virhe on merkattu korjatuksi.

Uusia Jenkins-töitä, joilla versioidaan pelejä, voi asettaa Jenkinsin kautta. Töiden perusaskeleina toimivat projektin aikana luodut skriptit, jotka on pyritty nimeämään siten, että niistä käy ilmi niiden tarkoitus. Projektin aikana tehtiin mm. seuraavanlaisia skriptejä:

- Build
- BuildAdroid
- BuildChecker
- ClearDeliveredData
- ClearXcodeArchives
- EditTrellocard
- EmptyDB
- GetCommitMessages
- RemoveRepository

- ReportBack
- SendAndShareViaDB
- SendMessageToFlow
- SendToGooglePlay

Osaa skripteistä ei käytetä Jenkins-töissä ja suurin osa on vapaaehtoisia asettaa työhön. Tällöin pelit, jotka ovat prototasolla, eivät tarvitse esim. taulua Trellosta tai kanavaa Flowdockista.

4.2 Jenkins

Jenkinsiä käytettiin hallitsemaan kääntämisen eri vaiheita ja yhdistämään eri työkaluja. Jenkinsin käyttöä Mac-koneilla tutkittiin ja päädyttiin Jenkins.app ratkaisuun. Jenkins.app on yksinkertainen AppleScript-sovellus, joka lataa Jenkis.war-tiedoston, pyytää käyttäjää tarkentamaan komentorivi kehotteet ja suorittaa Javalla Jenkins.war-tiedoston. (Tikka. 2014)

Normaalisti Jenkinsin käynnistyy erillisen käyttäjän taustaprosessina. Tällä käyttäjällä ei normaalisti ole pääsyä Keychainiin, mihin tallennetaan koodin signeeraukseen käytetyt tunnukset. Jenkins.app suorittaa Jenkinsin kirjautuneen käyttäjän prosessina, jolla on samat oikeudet Keychainiin kuin käyttäjällä. (Tikka. 2014)

Jenkins.app ladataan joko make-työkalulla tai selaimella. Ladattu tiedosto puretaan ja Jenkins.app on asennettu. Kun Jenkins.app käynnistetään ensimmäisen kerran, se asentaa Jenkinsin koneelle. Jenkinsin asennus avaa muutaman ikkunan, joissa muokataan asennusta. Tärkein tässä vaiheessa on komentorivin kehotteen muokkaus. Komentorivin kehotteen muokkauksella voi asettaa mm. Voiko Jenkinsiin ottaa yhteyttä muilta koneilta. Asennuksen jälkeen Jenkinsiin pääsee käsiksi selaimella. Oletuksen Jenkinsiin pääsee käsiksi kirjoittamalla hakupalkkiin ”localhost:8080”, mutta tämän voi määritellä haluamakseen. (Tikka. 2014)

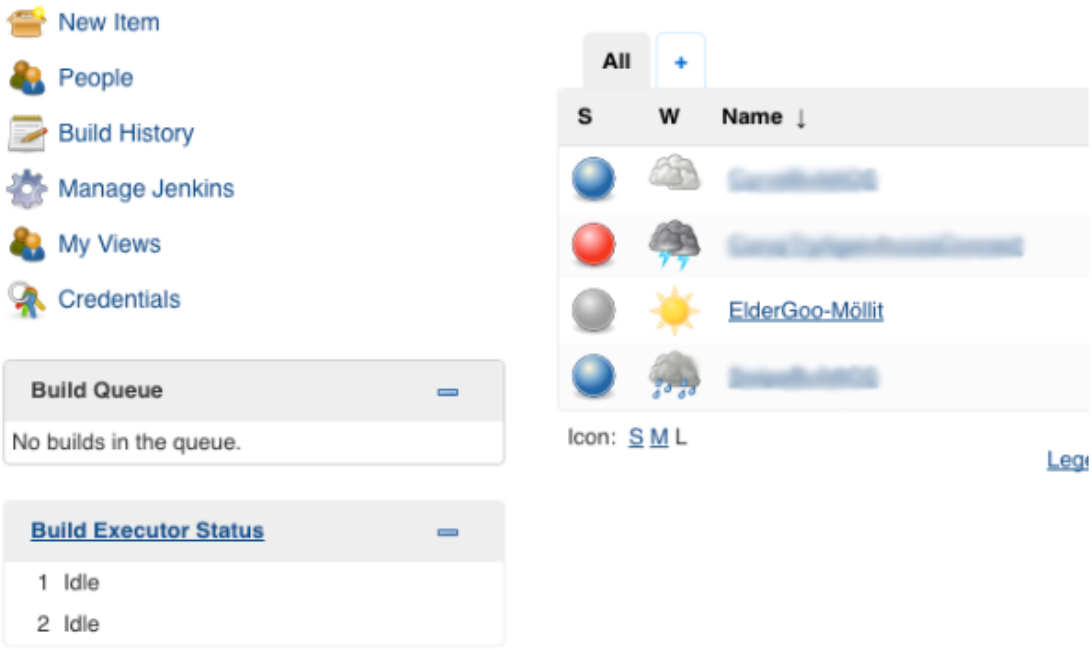
Jenkins asentaa oletuksena joitain lisäosia, mutta järjestelmää tehdessä asennettiin erikseen seuraavat lisäosat:

- Conditional buildstep
- Parameterized trigger

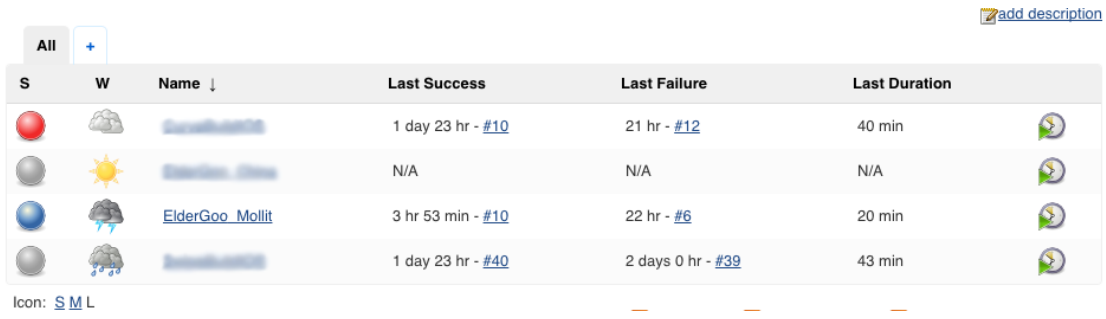
- Envlnject

4.2.1 Uuden työn asettaminen

Kuviossa 12 on kirjautumisen jälkeen näkyvä valikko. Pallukat vasemmalla kuvastavat viimeisimmän työn onnistumista. Sää kuvastaa miten hyvin edelliset työt ovat onnistuneet. Last success, last failure ja last duration ovat yksiselitteisiä ja näkyvät kuviossa 13. Viimeisenä on nappula jolla työn voi ajaa.



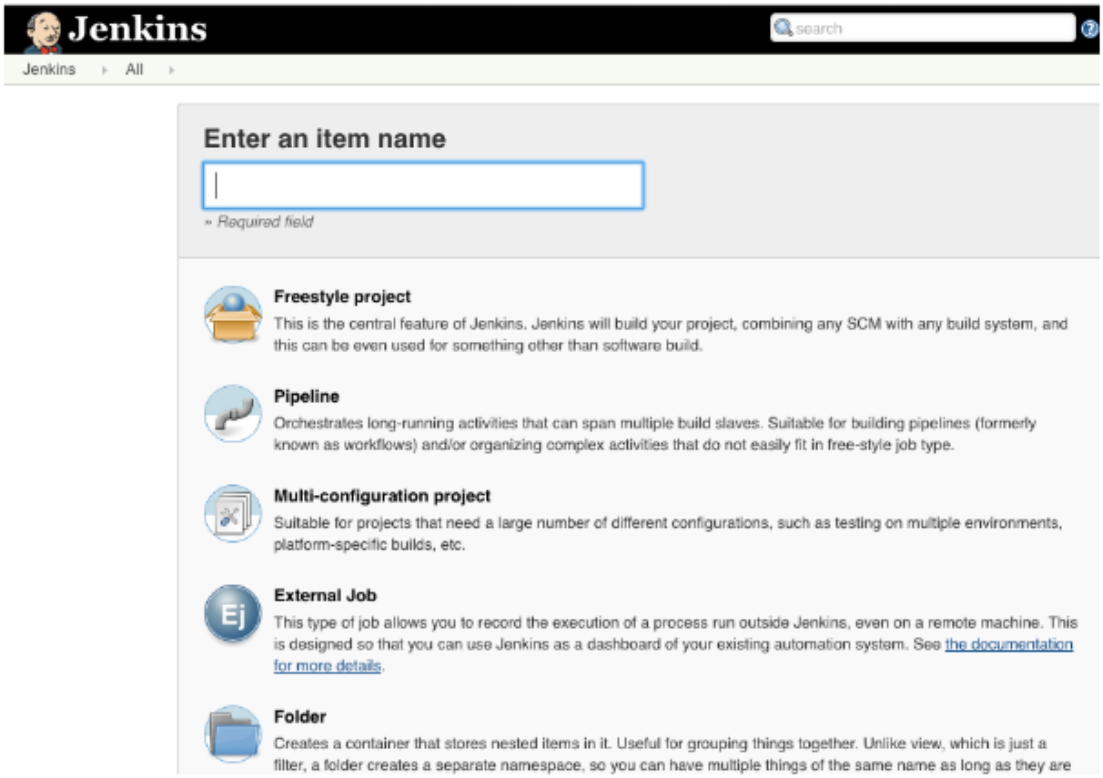
Kuvio 12. Jenkinsin päävalikko projektin alkuvaiheina



Kuvio 13. Projekteista näkyvät tiedot projektin alkuvaiheina

Uuden työn asettamiseksi käyttäjä valitsee kuviossa 12 vasemmalla näkyvän painikkeen ”New Item”. Avautuvasta valikosta voi määritellä minkälaisen pohjan haluaa työlle ja haluaako käyttää jotain jo ennalta olevaa työtä pohjana. Vaihtoehdot on

näytetty kuviossa 13. Tässä projektissa käytettiin pelkästään Freestyle tyylisiä projekteja.



Kuvio 14. Valikko uuden projektin alussa

Jokaisesta projektista on mahdollista tehdä iOS- tai Android-versio. Jokaisessa työssä, jossa käännetään peli, on kaksi parametria: buildIOS ja buildAndroid. Ne asetetaan kuviossa 15 näkyvällä tavalla.

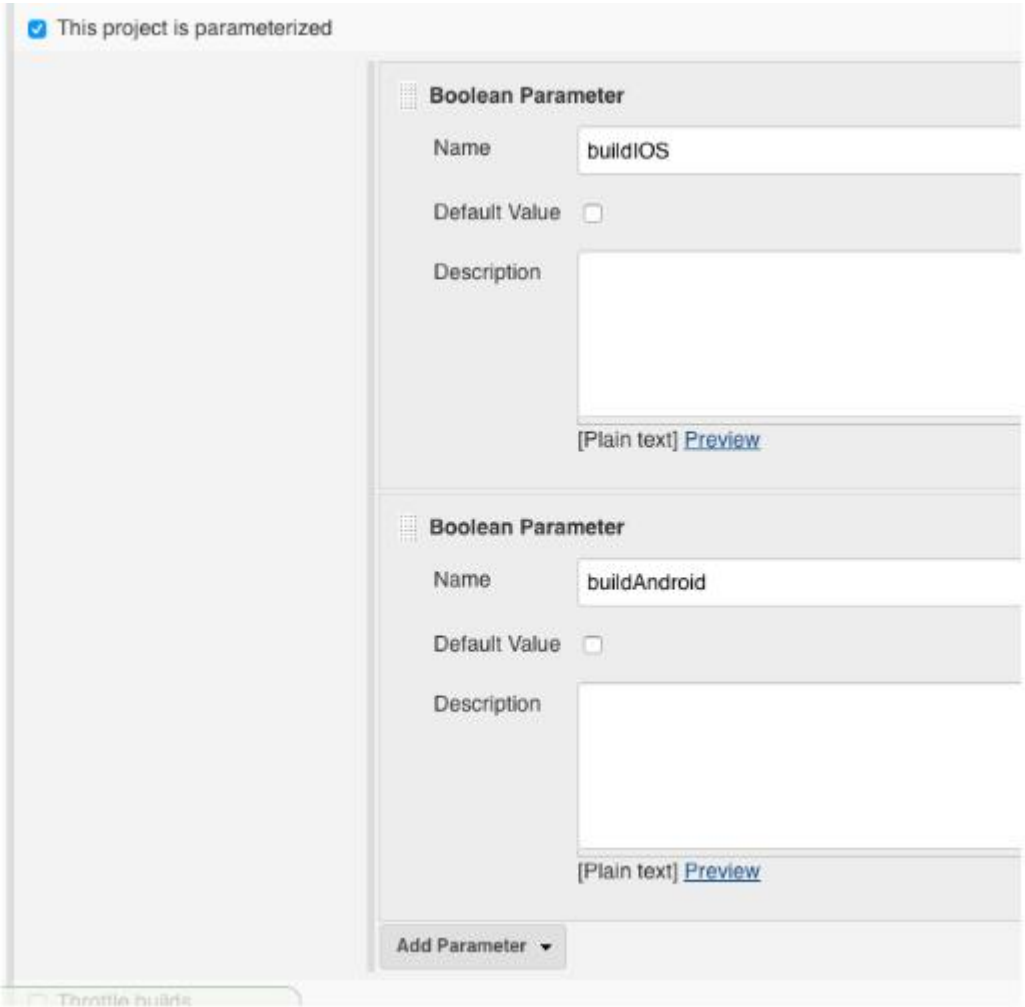
Työt aloitetaan joko selaimelta tai Jenkinsin rajapintaa käyttäen puhelimella käytettävältä sovellukselta seuraavalla tavalla:

```
# Get acces to trigger Jenkins job
crumb=$(curl -s --user $user:$password \
    $server/crumbIssuer/api/xml?xpath=concat\(//crum-
bRequestField,%22:%22,//crumb\))

# Start jenkins job and before that remove quotes
curl -s -v -H "$crumb" -X POST -u $user:$password
"$server/job/$name /buildWithParameters?build-
IOS=$buildTriggerIOS&buildAndroid=$buildTrig-
gerAndroid"
```

Järjestelmän Jenkins-palvelin käyttää ”Prevent Cross Site Request Forgery exploit” asetusta ja tämän vuoksi pyyntöihin tulee lisätä http-otsakkeeseen CSRF varmenne. Ylempi komento on esimerkki varmenteen saamiseksi. Alemmassa on tavanomainen pyyntö aloittaa parametrisoitu työ. Muuttujat joiden edessä on $-merkki ovat seuraavanlaisia:

- Crumb: CSRF varmenne.
- user: Jenkins käyttäjä
- password: Jenkins salasana
- server: Jenkins url
- name: Aloitettavan työn nimi
- buildTriggerIOS: True tai False arvoinen muuttuja
- buildTriggerAndroid: True tai False arvoinen muuttuja



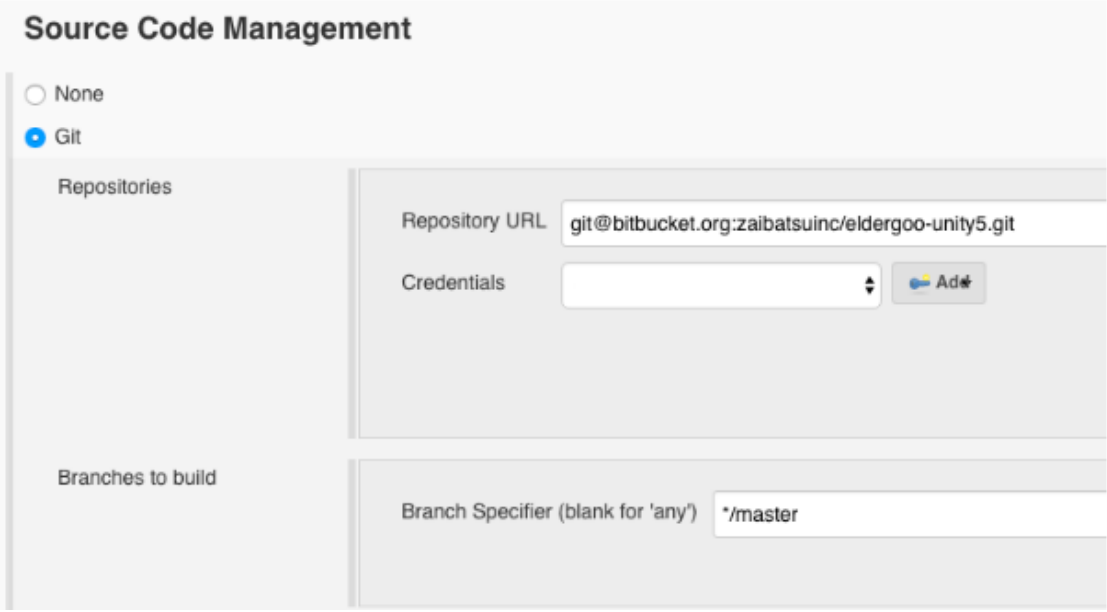
Kuvio 15. Parametrien asettaminen töille

Lähdekoodin kloonaaminen ja päivittäminen tapahtuvat Gitillä ja SSH-avaimen avulla. Kuviossa 16 näytetään, kuinka Repository URL asetetaan SSH-avainta käyttäen ja

kloonattavan oksan voi asettaa. SSH-avaimen luominen on melko helppoa ja tapahtuu seuraavasti Mac-koneelle:

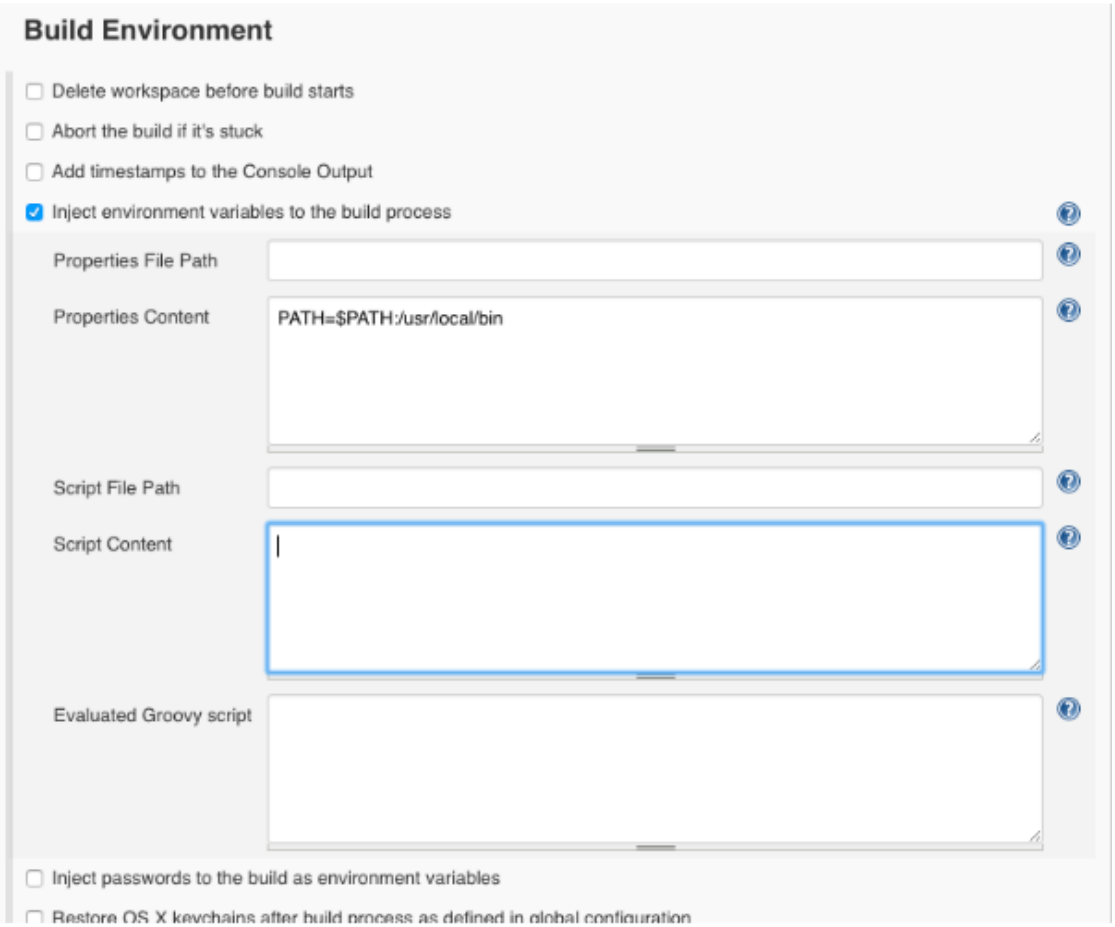
1. Avaa komentorivi
2. Tarkista, että koneella on SSH asennettuna. ssh -V komennolla saa näkyviin version asennetusta SSH:sta, jos se on koneella.
3. Avaa ~/.ssh:n sisältö komennolla: ls -a ~/.ssh. Jos koneelle on asetettu oletus identiteetti näet kaksi id_* tiedostoa ja voit käyttää id_rsa.pub tiedoston sisältöä avaimena.
4. Jos identiteettiä ei ole määritelty niin ssh-keygen voidaan generoida avain. Noudata ruudulle tulevia ohjeita. Tässä projektissa passphrasen jätettiin tyhjäksi.
5. cat ~/.ssh/id_rsa.pub komennolla saadaan avaimen sisältö komentoriville. Kopioi se.
6. Bitbucketissa käyttäjätilin hallinnassa on kohta ”SSH Keys” ja sinne pitää kopioida kohdassa 5 kopioitu sisältö.

Projektissa SSH-avain asetettiin toimeksiantajan luomalla Bitbucket tilille, jolla oli oikeudet kaikille relevanteille projekteille, eikä tunnuksia käytetty muuhun. (Set up SSH for Git. n.d.)



Kuvio 16. SSH-avaimen asettaminen

Jenkins-työssä on nyt määritelty projektin nimi ja versiohallintatyökalut. Varsinaiset suoritettavat askeleet tulisivat seuraavaksi. Projektissa oleellisimmaksi muodostui Shell-operaatioiden suorittaminen. Jenkinsillä ei normaalisti ole oikeuksia juuri mihinkään sitä pyörittävällä koneella. Bash-skriptien ajaminen kaatui alussa oikeuksien puuttumiseen ja nopean tutkimisen jälkeen tilanne päätettiin ratkaista EnvInject-lisäosalla. Lisäosan tarkoituksena on muokata työympäristöä eri töissä. Tässä projektissa sitä käytetään oikotienä ympäristömuuttujiin kuten kuviossa 17 näkyy. (EnvInject Plugin. n.d)



Kuvio 17. EnvInject lisosan käyttö

4.2.2 Kääntäminen ja jakaminen

Ensimmäisessä Shell operaatiossa tarkistetaan, tehdäänkö Android-versiota ja suoritetaan BuildAndroid.sh annettujen parametrien avulla. Skriptin suorittamiseen annetaan parametreina polku työn työtilaan ja projektin nimi. Kuviossa 18 annetut parametrit näkyvät sinisellä värillä. Tämä tarkoittaa, että ne ovat Jenkkisin omia muuttujia, jotka ovat projektikohtaisia. Suoritettavan skriptin oleellisimmat toiminnot ovat liitteessä 3. Skriptin alussa joko tyhjätään tai luodaan ”Builds” niminen kansio projektin juureen. Tämän jälkeen käynnistetään Unity3D komentorivi komennolla ja ajetaan CustomBuild luokan staattinen metodi PerformAndroidBuild. CustomBuild-luokan sisältö kuvataan omassa kappaleessaan.

Seuraava askel on vaihtoehtoinen, eikä näy kuviossa 18, mutta on Execute shell -operaatio, jossa suoritetaan SendAndShareViaDB.sh. Askel on tarkoitettu Android-version jakamiseen Dropboxin kautta, eikä sitä toteutettu kaikille projekteille. Skripti on seuraavanlainen:

```
# Send stuff to Dropbox
curl -X POST https://content.drop-
boxapi.com/2/files/upload \
  --header 'Authorization: Bearer '$key'' \
  --header 'Content-Type: application/octet-stream' \
  --header 'Dropbox-API-Arg: {"path":"/'$pro-
ject'/'$fileName'","mode":{".tag":"add"},"autore-
name":false}' \
  --data-binary @''$fileName''
```

Skriptin alussa lähetetään luotu APK-tiedosto Dropboxiin järjestelmää varten luodulle tilille käyttäen Dropboxin julkista rajapintaa.

```
# Create sharing link and save it to variable
link=$(curl -X POST https://api.drop-
boxapi.com/2/sharing/create_shared_link_with_settings
\
  --header 'Authorization: Bearer '$key'' \
  --header 'Content-Type: application/json' \
  --data '{"path":"/'$project'/'$fileName'","set-
tings":{"requested_visibility":{".tag":"public"}}}' |
jq ".url" | sed 's/^"\(.*\)"$/\1/')
```

Dropboxin rajapintaa käyttäen luodaan linkki lähetetylle tiedostolle. Tämä linkki lähetetään myöhemmin Flowdockkiin liitteessä 3 olevan skriptin mukaisesti.

Järjestelmässä pidetään erillisessä tiedostossa kirjaa mahdollisista virheistä ja virhe paikoista. Monessa suoritettavassa komennossa tarkistetaan, voidaanko kyseinen komento suorittaa. Jos komentoa ei voi suorittaa, kirjataan ongelmakohta virhetiedostoon ja lopetetaan skriptin suorittaminen eri palautusarvolla kuin 0. Palautusarvo 0 tarkoittaa, että skripti suoritettiin onnistuneesti, kun taas muut arvot tarkoittavat virhettä. Virhe lopettaa Jenkinsin operaation, ellei ole määritelty erikseen aina ajattavia askeleita.

Kolmannessa Shell-operaatio askeleessa tarkistetaan, tehdäänkö iOS-versio. Projektin alkuvaiheessa tämä oli ainoa vaihtoehto ja sen vuoksi suoritettavan skriptin nimeksi jäi Build.sh. Skripti vaatii seuraavat parametrit toimiakseen:

- Polku työtilaan
- AppleID
- BundleID
- Pelin nimi

Skriptissä on hyvin paljon samanlaisuutta Android-version tekevän skriptin kanssa. Poikkeuksena on Unity3D osuuden jälkeen Gym työkalun käyttäminen komennolla ”Gym” ja komennolle tehty virhe tarkistus. Gym työkalun käyttämisen jälkeen skripti lähettää IPA-tiedoston iTunes Connectiin Pilotilla seuraavasti:

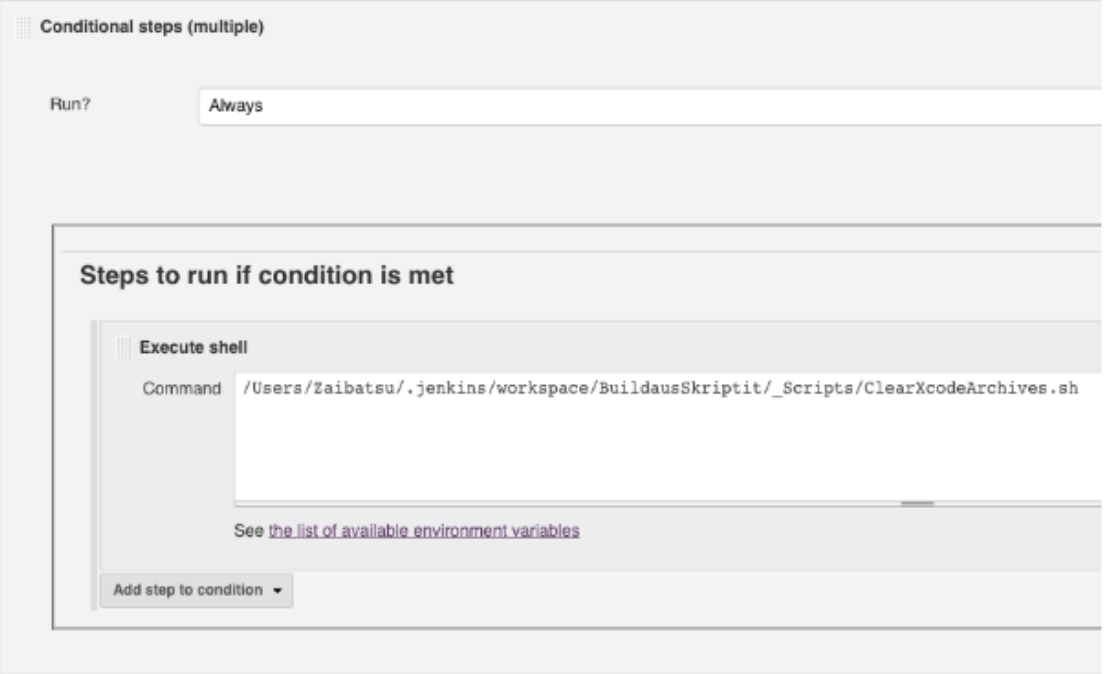
```
pilot upload -u "$user" -a "$bundleId" -p "$appleId"
--distribute_external false -z true;
```

Parametreina annetaan käyttäjätunnus, jonka tiedot on tallennettu ennalta koneen Keychain-ohjelmaan talteen, BundleID, AppleID. Lopussa oleva ”-z true” tarkoittaa, ettei odoteta vastausta iTunes Connectista onnistuiko lähetys.

Liitteessä 4 on Build.sh:n varhainen versio. Liite 4 kuvastaa hyvin virhetarkistuksen toteutusta. Liitteessä 4 myös näkyy miten muuttuva projekti ja sen skriptit ovat olleet.

Kuvio 18. Ensimmäiset askeleet

Näiden askelien jälkeen suoritetaan aina askel, jossa suoritetaan ClearXcodeArchives.sh niminen skripti. Kuviossa 19 näytetään, kuinka askel pakotettiin aina suoritettavaksi, riippumatta työn onnistumisesta tai epäonnistumisesta, käyttäen Conditional BuildStep pluginia. Xcodella aina, kun tehdään Xcode-projektista IPA-tiedosto, jää Xcoden Archive kansioon tiedosto operaatiosta. Luodun tiedoston koko vaihtelee projekteittain. Nämä tiedostot kuitenkin veivät tilaa merkittävän määrän, joten päädyimme ratkaisuun, jossa kansio tyhjennetään tiedostoista, jotka ovat yli päivän vanhoja. Nämä arkistotiedostot ovat turhia Unity projekteissa, ellei samaa versiota aiota lähettää useaan kertaan. Muussa tapauksessa luotua arkistotiedostoa käytetään vain kerran.



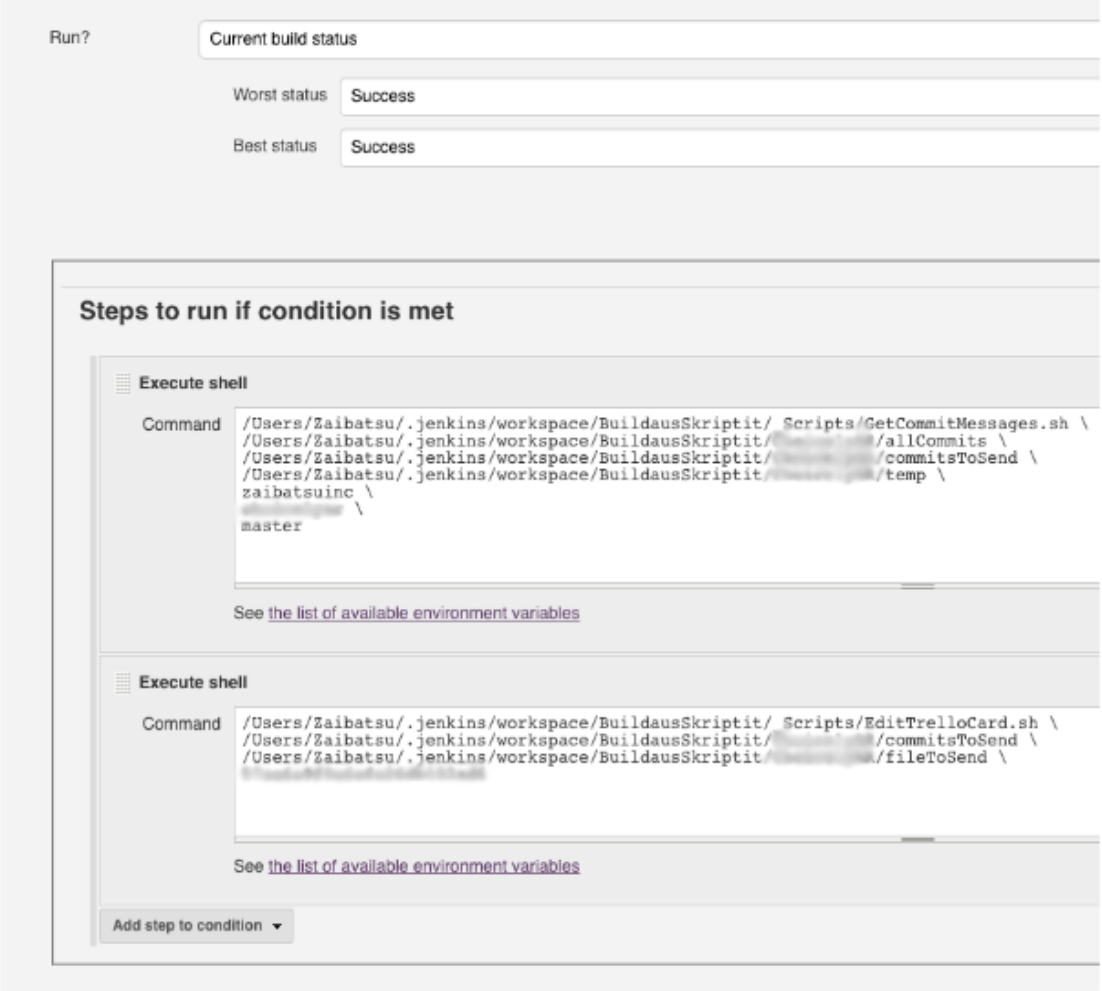
Kuvio 19. Clear Xcode Archives

4.2.3 Raportointi

Kääntämisen ja lähettämisen jälkeen systeemi raportoi, mitä uudessa versiossa on uutta. Tämä on kolmiosainen tapahtumaketju ja on kuvattu kuvioissa 20 ja 21. Ensimmäinen vaihe on hakea Bitbucketin julkisella rajapinnalla tietoa versioidusta pelistä. Tässä systeemissä käytettiin toimeksiantajan pyynnöstä commit-viestejä. Commit-

viestit haetaan versioidusta oksasta ja niistä parsitaan uusimmat tai ennalta määritelty maksimi määrä.

Commit-viestit lähetetään Trelloon ennalta määritellylle kortille ja ennen lähetystä muotoillaan skriptissä Trellon käyttämän Markdownin mukaisesti edustavimmiksi. Trelloon kirjataan commit-viesti, aika ja linkki Bitbuckettiin kyseiselle commitille.

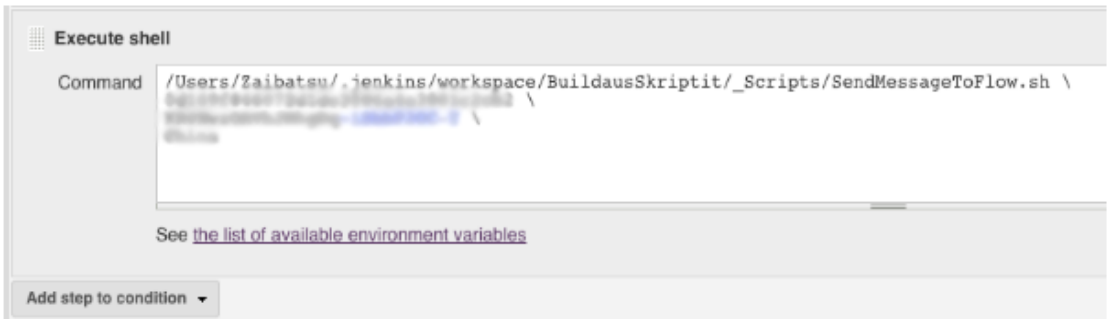


Kuvio 20. Takaisin raportointi

GetCommitMessages.sh ottaa parametreiksi kolme polkua tiedostoihin, mihin säilötään dataa, organisaation nimen, jonka nimissä versiohallinnassa sijaitseva koodi on, projektin nimen ja oksan, jonka commit-viestit halutaan.

EditTrellocard.sh tarvitsee parametreina tiedoston, missä on lähetettävät viestit, tiedoston mihin lopullinen lähetettävä sisältö säilötään ja ID:n kortille, johon viesti kirjoitetaan.

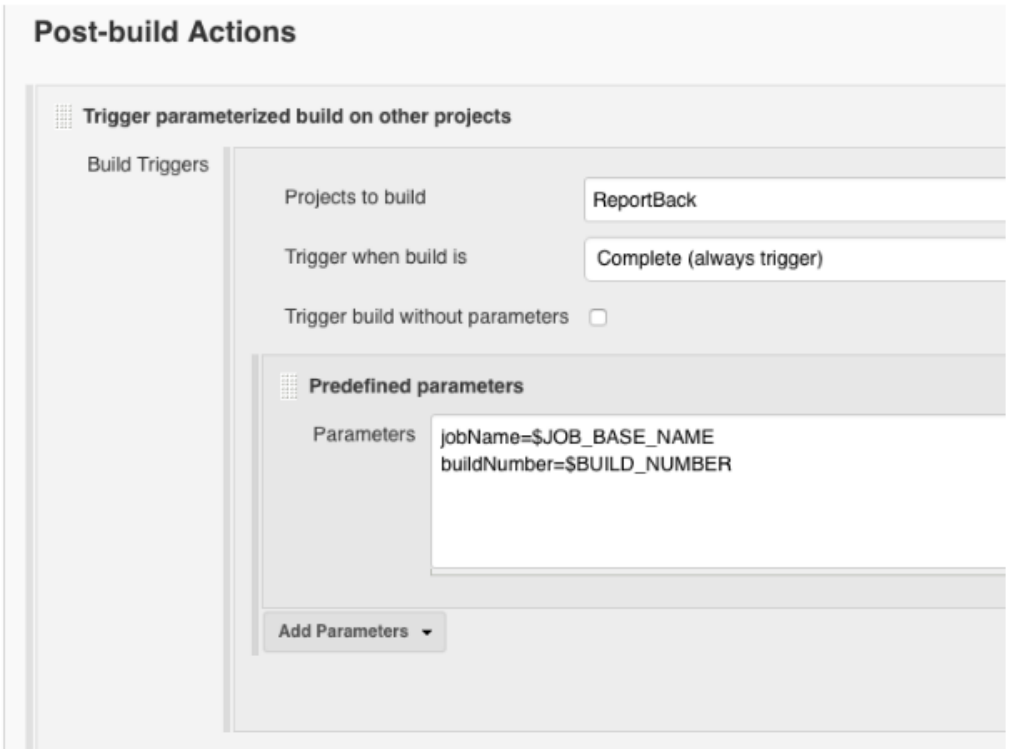
Viimeisessä vaiheessa lähetetään viesti versioidun pelin Flowdock kanavalle ennalta määriteltyyn viestiketjuun. Tätä toimintaa ei toteuteta kaikille töille. Sitä ei myöskään suoriteta, ellei työssä käännetä iOS-versioita, koska Android-version kääntämisessä lähetetään viesti Flowdockkiin SendAndshareViaDB-skriptin suorittamisen aikana. Viestin sisältö vaihtelee, riippuen käännetäänkö pelkkä Android- vai myös iOS-versio.



Kuvio 21. Viestiminen Flowdockiin

SendMessageToFlow.sh ottaa parametreina kanava tokenin, langan ID:n ja juuren mihin versioidun pelin koodi on säilötty.

Jokaisen käännöksen jälkeen suoritetaan Jenkins-työ nimeltä ”ReportBack”. Tämä työ raportoi työn tilan tiedostoon, jota mobiilisovellus lukee. Tarkoituksena on saada mahdollisimman nopeasti tieto, mikäli käännös epäonnistuu ja missä vika on. Työ aloitetaan ”Parameterized Trigger” pluginilla kuvion 22 mukaisesti. Muuttujat määritellään suoritettavassa työssä ja niihin sijoitetut muuttujat ovat Jenkinsin omia muuttujia.



Kuvio 22. Viimeinen työn jälkeen suoritettava askel

4.3 Unity

Projekteihin, joiden kääntäminen halutaan automatisoida, lisätään editori skripti Asset kansion juureen “Editor” nimiseen kansioon. Skriptiin tehdään staattisia metodeja, jotka käyttävät Unityn BuildPipeline-luokan BuildPlayeria hyväkseen. Metodeissa pyydetään osa projektin asetuksista suorittavalta koneelta, määritellään osa ja asetetaan ne. Asetusten asettamisen jälkeen aloitetaan kääntäminen kuin se olisi tehty käsin Unityllä.

Projekteihin lisättyjä skriptejä ja niiden staattisia metodeja voidaan ajaa komentoriviltä ja niihin voi antaa parametreja joilla säädellään Unityn käyttäytymistä. Järjestelmässä komento ajetaan osana Jenkinsissä määriteltyä työtä.

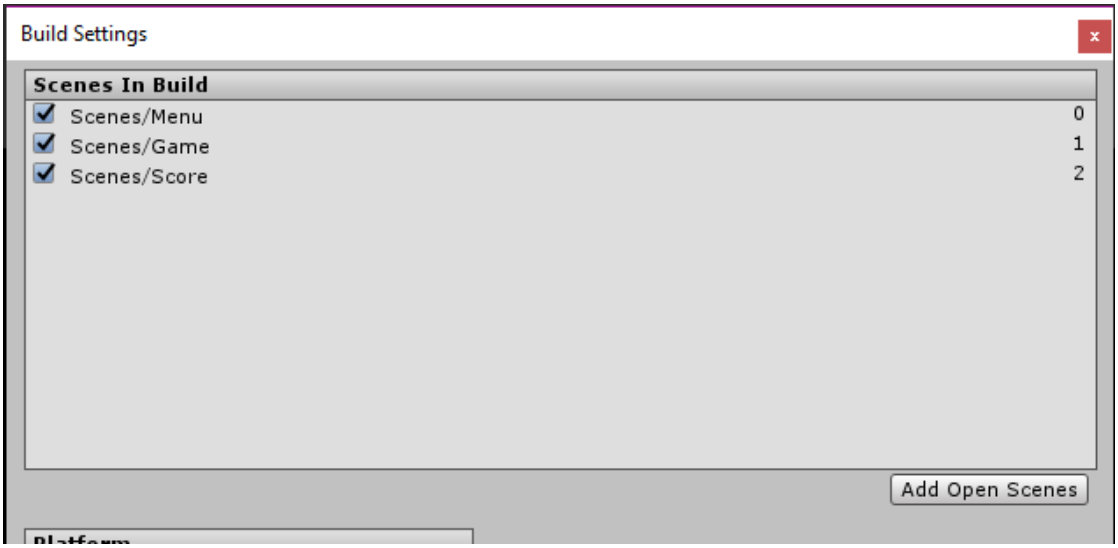
```
C:\Programming\Unity\editor\Unity.exe -batchmode
-quit -projectPath C:\Programming\Zaibatsu\projectA
-executeMethod CustomBuild.PerformIOSBuild
```

Komennon alussa käynnistetään Unity3D. Seuraava muuttuja avaa Unityn tilassa, jossa varmistetaan, ettei käyttäjän tarvitse klikata nappuloita, jotta Unity voisi jatkaa toimintaansa. -quit komento sammuttaa Unityn toiminnan jälkeen.

CustomBuild-skripti on laitettava käännettävän projektin ”Editor” nimiseen kansioon. Skriptin toiminnallinen sisältö alkaa määrittelemällä kentät, jotka asetetaan käännettävään versioon.

```
/* Method to set levels to build */
private static string[] FillLevels()
{
    return (from scene in EditorBuildSettings.scenes
    where scene.enabled selectscene.path).ToArray();
}
```

Yllä oleva metodi tutkii Unity-projektin editorista asetetut kentät ja lisää ne taulukkoon, jota käytetään seuraavassa metodissa. Kentät tulee asettaa käsin Unityn editorissa File → Build Settings -valikosta pelin kehitysvaiheessa kuvion 23 tapaan.



Kuvio 23. Kentän lisääminen Unity editorissa käsin File → Build Settings asetuksista

```
/* Method to build iOS */
static void PerformIOSBuild()
{
        string[] levels = FillLevels();
        string buildNumber;
        buildNumber = VersionNumber();

        PlayerSettings.iOS.buildNumber = buildNumber;
```

```
        string buildCount = builNumber.Split('.')[1];
        buildNumber = buildNumber.Split('.')[0];

        PlayerSettings.showUnitySplashScreen = false;

    BuildPipeline.BuildPlayer(levels, "Builds/” +
    projectName + “_" + buildNumber + "_" +
    buildCount, BuildTarget.iOS, BuildOptions.None);
}
```

Metodissa määritellään aluksi versioon tulevat kentät käyttäen aiemmin esiteltyä metodia ”FillLevels”. Tämän jälkeen määritellään toimeksiantajan määrittittelyn mukainen versionumero, joka asetetaan julkaisuasetuksen buildNumberiksi. BuildPipelinen BuildPlayer metodilla käännetään peli. Parametreina metodi ottaa string-taulukon kentistä, tallennussijainnin, nimen ja mahdolliset lisäasetukset.

Projektin aikana Xcode 7.3 vaihdettiin 8.1. Tämä muutti hieman logiikkaa, jolla Xcode tunnisti ja asetti provision profiilit Xcode-projektille, mikä saatiin ulos Unitystä käännöksen lopuksi. Xcode 8 tunnistaa oikeat profiilit projektin asetuksista tiimin tunnuksesta. Tämän vuoksi projektin Editor-kansioon tulee lisätä skripti, joka on sisällöltään liitteen 5 kaltainen. Käytännössä skripti asettaa projektin asetuksiin käännettyyn Xcode-projektiin ”development team” -asetukseen toimeksiantajan ryhmän tunnuksen. Tunnus löytyy Applen Developer -sivustoilta.

Unity-projekteihin voidaan asettaa ulkoisia lisäosia (plugin). Osa näistä lisäosista ei tue Bitcodea. Bitcode on systeemi, joka mallintaa ohjelmaa, ja jolla Apple voi optimoida lähetettyä sovellusta ilman, että kehittäjän pitää lähettää uutta versiota (App Thinning. n.d). Jos pelissä käytetään lisäosia, jotka eivät tue Bitcodea, tulee editor kansioon lisätä skripti, joka post build processia käyttäen asettaa Bitcoden pois päältä. Suorittava osuus on kuvattu liitteessä 6.

Testiversioissa Bitcoden puuttumisella ei havaittu olevan mitään vaikutusta, mutta lopullisen version lähettämisessä sovelluskauppaan se olisi hyvä olla. Projektin kääntäminen IPA-tiedostoksi epäonnistuu, mikäli asetuksissa on laitettu ”Käyttää Bitcodea” ja projekti ei tue sitä.

4.4 Fastlane

Fastlanesta järjestelmä käyttää kahta eri ohjelmaa ja niiden mukana tulevia apuohjelmia. Pilot, Gym ja CredentialsManager. CredentialsManager on käytössä useimmissa Fastlanen työkaluissa ja on tarkoitettu rajapinnaksi macOS Keychainin ja Fastlanen välille.

Järjestelmässä on tallennettava iTunes Connectiin rekisteröidyn käyttäjän tunnukset koneen Keychain-sovellukseen käyttäen CredentialsManageria. Järjestelmää tehdessä käytettiin tunnusta, jolla oli App manager -oikeudet. Sovelluksen käyttö tapahtuu komentorivillä

```
Fastlane-credentials add --username asd@asd.com
```

Tämän jälkeen komentoriville tulee pyynnöt salasanasta. Operaation jälkeen luotu tunnus löytyy Keychainista avaimet osiosta.

4.4.1 Gym

Jenkinsissä määritellyssä työssä Unity kääntää projektin Xcode-projektiksi, joka kääntää ja pakata IPA-tiedostoksi. Gym tekee tämän käyttäen erilaisia Xcoden rajapintoja. Gym käyttää koneella tallennettuja tunnuksia, jotka näkyvät myös Xcodessa.

Gymin asennus tapahtuu komentorivillä

```
$ sudo gem install gym
```

Järjestelmän luovutusvaiheessa Gymiä käytetään Build.sh:ssa komennolla ”Gym”.

Tarvittavat tunnukset asetetaan Xcodeen seuraavasti Applen ohjeiden mukaisesti (Adding Your Account to Xcode. n.d.):

1. Avaa Xcode
2. Klikkaa ”Preferences”
3. Avautuvan ikkunan yläosasta klikkaa kohtaa ”Accounts”
4. Ikkunan alaosassa on plusmerkki. Klikkaa sitä.
5. Valitse ”Add Apple ID”
6. Kirjoita Apple ID ja salasana niihin kuuluviin kenttiin. Tunnuksen pitää olla validi Apple Developer portaaliin. Tunnuksien pääsy iTunes Connectiin ei ole välttämätöntä.
7. Nyt aikaisemmassa Accounts-valikossa pitäisi näkyä äsken lisätty tunnus. Valitsemalla tunnuksen käyttäjä näkee mihin kehittäjä ryhmiin tunnus kuuluu. valitse haluamasi ja klikkaa ”View Details” nappulaa.

8. Klikkaa iOS Development ja halutessasi Distribution tekstien päässä näkyvää ”Create” painiketta. Jos tästä tulee virheilmoitus tunnusten olevan käytössä jossain muualla, varmista muun ryhmän kanssa onko niiden pakko olla. Jos kyllä, niin harkitse uuden tunnuksen luomista, tai tunnusten kopiointia vanhalta koneelta uudelle.

Viimeinen osio saattaa aiheuttaa ongelmia. Tässä järjestelmässä päädyttiin ratkaisuun, missä järjestelmälle luotiin oma tunnus Applen Developer ohjelman tilille. Tämä tili oli ainoastaan järjestelmän koneella käytössä. Tämä johtuu siitä, että ohjelmaan signeeraamiseen käytetyt tiedot koostuvat avainpareista, joista toinen on julkinen ja toinen privaatti.

> *If you want to code sign your app using another Mac, you export your developer profile on the Mac you used to create your certificates and import it on the other Mac. You can also share distribution certificates among multiple team agents using this feature. (Team members should not share development certificates.)* (Maintaining Your Signing Identities and Certificates. n.d.)

4.4.2 Pilot

Osana käännösketjua käytetään Pilottia lähettämään Sovelluksen iTunes Connectiin. Pilotin asennus tapahtuu komentorivillä komennolla:

```
$ sudo gem install pilot.
```

Pilottia käytetään osana Build.sh-skriptiä, joka ajetaan Jenkinsissä seuraavasti:

```
$ pilot upload -u "$user" -a "$bundleId" -p "$ap-
pleId" --distribute_external false -z true.
```

- -u on käyttäjätunnus iTunes Connectiin. Tämä tunnus on oltava Keychainissa aikaisempien ohjeiden mukaisesti.
- -p on appleid, joka määritellään projektille iTunes Connectissa.
- -a on bundleid, joka määritellään projektille iTunes Connectissa.
- --Distribute_external on Pilotin boolean muuttuja, joka määrittää lähetetäänkö käännöstä ulkoisille testaajille. Tämän voi asettaa iTunes Connectista versio kohtaisesti ja päädyimme pitämään tässä vaiheessa arvon epätotena.
- -z on muuttuja siitä, että odotetaan iTunes Connectin varmistusta perille menemisestä. True tarkoittaa, ettei odoteta.

4.5 Bitbucket

Toimeksiantaja käyttää Bitbuckettia versiohallintapalvelunaan kaikissa projekteissaan. Toimeksiantaja oli luonut aikaisemmin toimintamallin, missä projektien etenemistä seurattiin commit-viestien perusteella. Koodaajat olivat tehneet skriptin, jolla

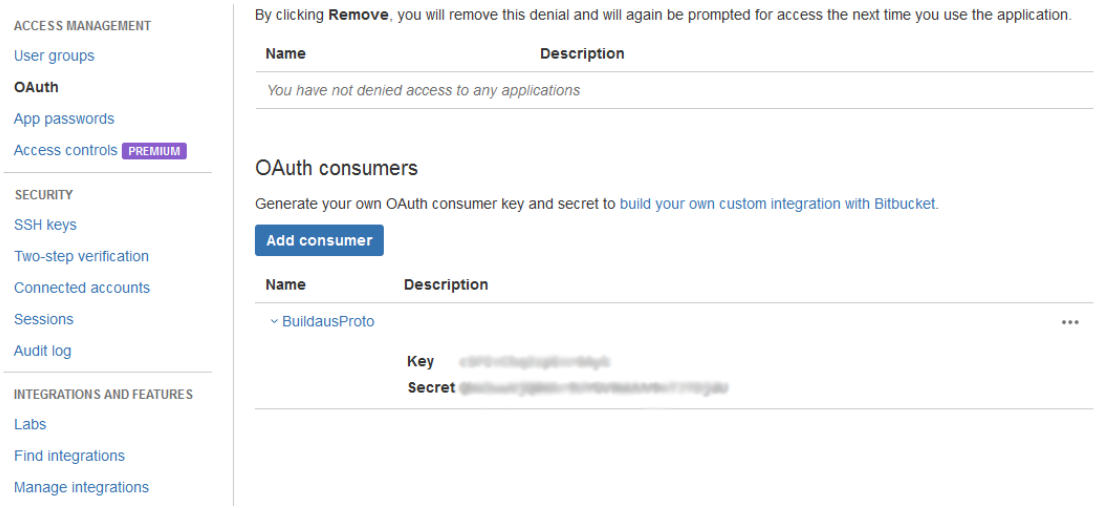
olennaiset viestit lähetettiin automaattisesti Trelloon Git push -komennon jälkeen. Projektissa päätettiin korvata systeemi siten, että aina käännöksen alkaessa katsottaisiin uudet viestit käännettävästä Git-juuresta. Useimmissa tapauksissa tämä juuri on Master.

Raportointisysteemin toteuttamiseen käytettiin Bitbucketin julkista rajapintaa. Bitbucketin rajapinta käyttää tunnistamiseen OAuthia. OAuthissa käytetään käyttäjän tunnistamiseen erilaisia tapoja salasanan sijasta. Bitbucketin käyttämässä systeemissä vaihtoehtoina on neljä RFC-6749 nimisen kehyksen määrittelemää myöntämistapaa. Tässä projektissa käytetään Client credentials granttia.

> *This grant is similar to the resource owner credentials grant except only the client's credentials are used to authenticate a request for an access token. Again this grant should only be allowed to be used by trusted clients.*
> *This grant is suitable for machine-to-machine authentication, for example for use in a cron job which is performing maintenance tasks over an API. Another example would be a client making requests to an API that don't require user's permission.* (Bilbie. n. d.)

Clien credential grants soveltuu hyvin tunnistustyypiksi, jota käyttää kone. Grantilla saadaan access- ja refresh-tunnus. Access tunnuksella suoritetaan varsinainen tunnistautuminen ja se on voimassa yhden tunnin ajan. Tunnin jälkeen refresh-tunnuksella pyydetään uusi. Järjestelmässä joka käännöksen alussa pyydetään uusi access-tunnus.

Järjestelmää varten on luotava Bitbucket tilillä OAuth consumer. Tämä on mahdollista kirjautumalla Bitbuckettiin halutulla tunnuksella, klikkaamalla oikeassa yläkulmassa tunnuksen kuviota ja valitsemalla valikosta ”Settings”. Asetuksista valitaan kuvion 24 mukaisesti Access managementin alta OAuth ja klikataan ”Add Consumer”. Consumer nimetään halutun mukaiseksi ja sille asetetaan halutut oikeudet. Järjestelmä ei tarvitse kuin lukuoikeuden. Toiminnan jälkeen saadaan käyttöön avain ja salainen tunnus, jota voidaan käyttää tunnuksina rajapinnassa.



Kuvio 24. OAuth consumerin luominen

Avaimella ja salaisella tunnuksella saadaan tunnukset, joilla suoritetaan rajapintakutsuja seuraavasti:

```
curl -X POST -u "{Key}:{Secret}" \
    https://bitbucket.org/site/oauth2/access_token \
 -d grant_type=client_credentials
```

Pyyntö palauttaa seuraavanlaisen vastauksen ja tässä systeemissä skripti tallennetaan sen omaan tiedostoonsa talteen:

```
{"access_token": "PITKÄKOODI", "scopes": "repository", "expires_in": 3600, "refresh_token": "TOINENPITKÄ KOODI", "token_type": "bearer"}
```

Access-tunnus muuttuu toimimattomaksi tunnin kuluttua sen myöntämisen jälkeen. Uuden acces-tunnuksen saa rajapinta kutsulla seuraavasti:

```
curl -X POST -u "Key:Secret" https://bitbucket.org/site/oauth2/access_token -d grant_type=refresh_token -d refresh_token={refresh_token}
```

Tunnusten hallinnan jälkeen voidaan Bitbucketista hakea commit-viestit seuraavanlaisella pyynnöllä:

```
curl https://api.bitbucket.org/2.0/repositories/{Organisaatio / käyttäjä}/{repon nimi}/commits/{Branchi}?access_token=$acces_token
```

Kuvailtuja rajapintakutsuja käytetään Jenkinsillä ajettavissa skripteissä, joka on osittain kuvailtu raportointi otsikon alla.

4.6 Trello

Toimeksiantaja käyttää Trelloa sisäiseen viestintään, tehtävien hallintaan ja tehtävien jakamiseen. Projektin alkuvaiheessa päätettiin tehdä jokaisen projektin taululle kortti nimeltä ”Build log” ja päivittää sitä Bitbucketista saaduilla viesteillä.

Trellon käytössä käytimme Trellon julkista rajapintaa. Trellolla on omat kehittäjäsivunsa, josta saa nopeasti sovellusavaimen. Sovellusavainta käytetään kaikissa rajapinnan kutsuissa (Start Building with Trello. n.d).

Sovellusavaimella voimme tehdä rajapintapyynnön:

```
https://trello.com/1/authorize?expiration=ne-
ver&scope=read,write,account&response_type=to-
ken&name=Server%20Token&key={SovellusAvain}
```

Ylempi pyyntö antaa sovellukselle luku- ja kirjoitusoikeudet kirjautuneen käyttäjän näkemiin tauluihin ja kortteihin.

Kortteihin kirjoittamiseen rajapinnan avulla tarvitaan niiden ID. ID:n saaminen onnistuu kahdella selaimella tehdyllä GET-pyynnöllä Trellon rajapinnalla seuraavasti:

```
https://api.trello.com/1/members/{käyttäjäTun-
nus}/boards?key={Key}&token={Token}
```

- {KäyttäjäTunnus}: käyttäjätunnus.
- {Key}: Kehittäjäavain.
- {Token}: Valtuutuksen vastauksena saatu tunnus.

Ylempi kutsu näyttää JSON-muodossa kaikki taulut, johon tunnuksilla on pääsy. Valitaan halutun taulun ID ja käytetään sitä seuraavassa kutsussa:

```
https://api.trello.com/1/boards/{Boar-
dId}/cards?fields=name,idList,url&key={Avain}&to-
ken={Token}
```

Ylempi pyyntö palauttaa eritellyn taulun kortit. Listasta valitaan halutun kortin ID. Tätä ID:tä käytetään EditTrelloCard-skriptissä seuraavanlaisesti:

```
curl -i -X POST -H "Content-Type: application/json" -
d '{
"key": {Key},
"token": {Token},
"desc": {CommitMessages}
```

```
}' https://api.trello.com/1/cards/{id}/actions/com-
ments
```

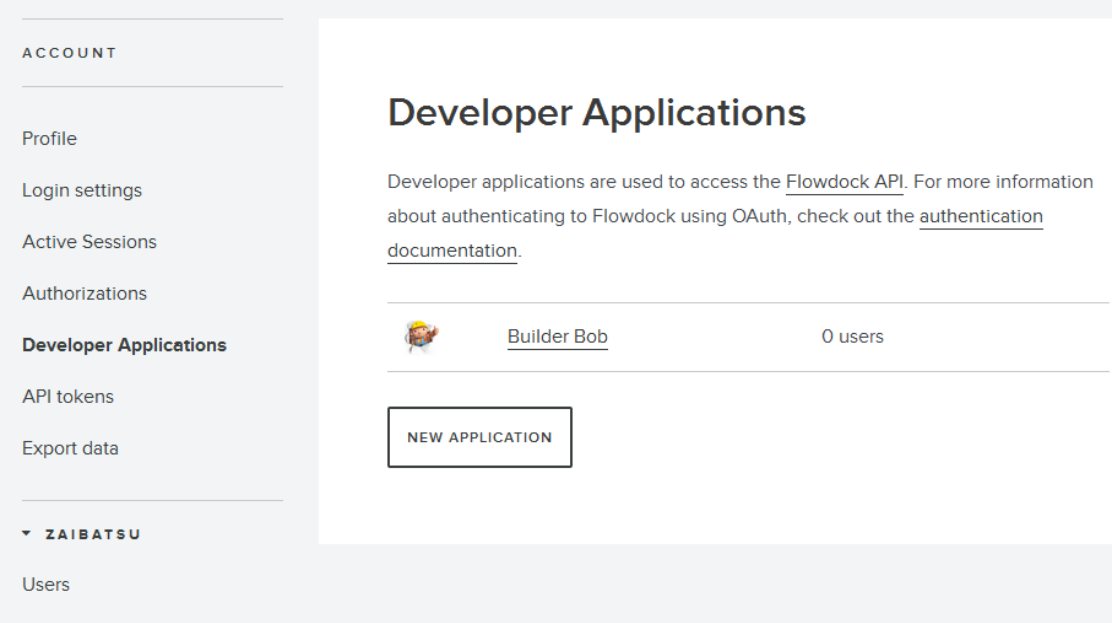
4.7 Flowdock

Toimeksiantaja käytti Flowdockkia sisäiseen viestintään. Jokaiselle projektille on luotu kanava, johon projektille relevantti viestintä keskittyi. Flowdock ja Trello on integroitu Trellon omalla integraatiolla. Projektien kanavissa oli aina “Buildaus-ketju” niminen lanka, johon järjestelmä kävi ilmoittamassa viestillä minkä alustan ja minkä Git-juuren käännös on valmistunut.

Viestien lähettäminen lankoihin toteutettiin Flowdockin REST-rajapinnalla. Autentikointiin käytettiin kanavatunnusta (Flow token). Flowdockkiin rekisteröinyt käyttäjä voi luoda oman integraatiosovelluksen, jonka käyttäjä voi valtuuttaa käyttämään näkemiään kanavia. Kanavatunnuksella sovellus pystyy aloittamaa ketjuja ja kirjoittamaan viestejä kanavassa, jolle se on myönnetty. Kanavatunnus ei lakkaa toimimasta, vaikka sen valtuuttanut henkilö poistuu tai poistetaan kanavasta. Integraatiosovellus, joka käyttää kanavatunnusta, pitää poistaa erikseen kanavalta, mikäli se halutaan poistaa. (The CA Flowdock API Authentication. n.d).

Oman integraatiosovelluksen tekeminen tapahtuu kirjautumalla Flowdockin sivuille tilinhallinnasta. Tilinhallinnassa valitaan kuviossa 25 näkyvä ”Developer Applications” kohta. New Application näppäimestä aukeaa valikko, missä sovellukselle määritellään asetukset. Tässä järjestelmässä voidaan valita ”Shortcut application”, jolloin määritettäväksi jää ainoastaan nimi, kuvaus, ohjeet ja kuva.

Kun oma sovellus on tehty, ilmestyy se kuviossa 25 näkyvään listaan. Klikkaamalla sovelluksen nimeä tulee näkyviin ohjeet sovelluksen käytöstä ja Flowdockin ohjeet sovelluksen julkaisemisesta. ”Tools for testing” alta voidaan luoda kanavatunnuksia ja samalla lisätä luotu sovellus valitulle kanavalle. Kanavatunnukset on tarkoitettu testaamiseen, mutta soveltuvat hyvin sisäisten integraatiosovellusten tekemiseen, jos niitä ei aiota julkaista muiden tahojen käytettäväksi.



Kuvio 25. Flowdock integraation tekeminen

Luotua kanavatunnusta käytetään skripteissä, jotka lähettävät viestejä Flowdockkiin. Viestin lähettäminen tapahtuu POST-pyynnöllä rajapintaa käyttäen seuraavasti:

```
curl -i -X POST -H "Content-Type: application/json" -
d '{
"flow_token": "{FlowToken}",
 "event": "message",
 "content": "Tahan tulee viesti",
 "thread_id": "threadID"
}' https://api.flowdock.com/messages
```

Thread ID:n saa näkyviin selaimesta avaamalla halutun langan Flowdokissa ja katsomalla selaimen osoitepalkista URL:n viimeisen osio.

4.8 Dropbox

Dropboxia päätettiin käyttää Android käännösten levittämiseen, koska APK-tiedostojen asentaminen ja suorittaminen onnistuvat suoraan puhelimelle. Järjestelmälle luotiin tili, johon säilötään jaettavia APK-tiedostoja. Dropboxin julkisella rajapinnalla tiedostot lähetetään tilille, niille luodaan jakolinkit ja ne poistetaan tietyn ajan jälkeen. Jakolinkit postitetaan Flowdockkiin ennalta määritellyille kanaville.

Dropboxiin voi luoda omia integraatiosovelluksia kirjautumalla Dropboxiin ja klikkaamalla sivun alakulmasta valikosta ”Developers”. Linkistä aukeaa kuvion 26 näköinen

sivu. Valitsemalla ”My apps” voi luoda uuden sovelluksen ja määritellä kuinka paljon haluaa sovelluksen näkevän käyttäjän tilistä. Tässä järjestelmässä käytetään sovellusta, joka näkee tilin kaikki kansiot. Sovelluksen asetuksista generoidaan autentikointitunnus, jolla voidaan tehdä rajapintapyyntöjä omiin tiedostoihin. Toisille käyttäjille autentikoinnissa näin luotu tunnus ei sovellu, mutta muiden käyttäjien ei koettu tarvitsevan oikeuksia systeemiin. (Dropbox Platform developer guide. n.d)

Luotu integraatio sovellus, tai paremminkin sen avaimella, tehdään järjestelmässä kolme pyyntöä. Ensimmäinen on tiedoston lähettäminen:

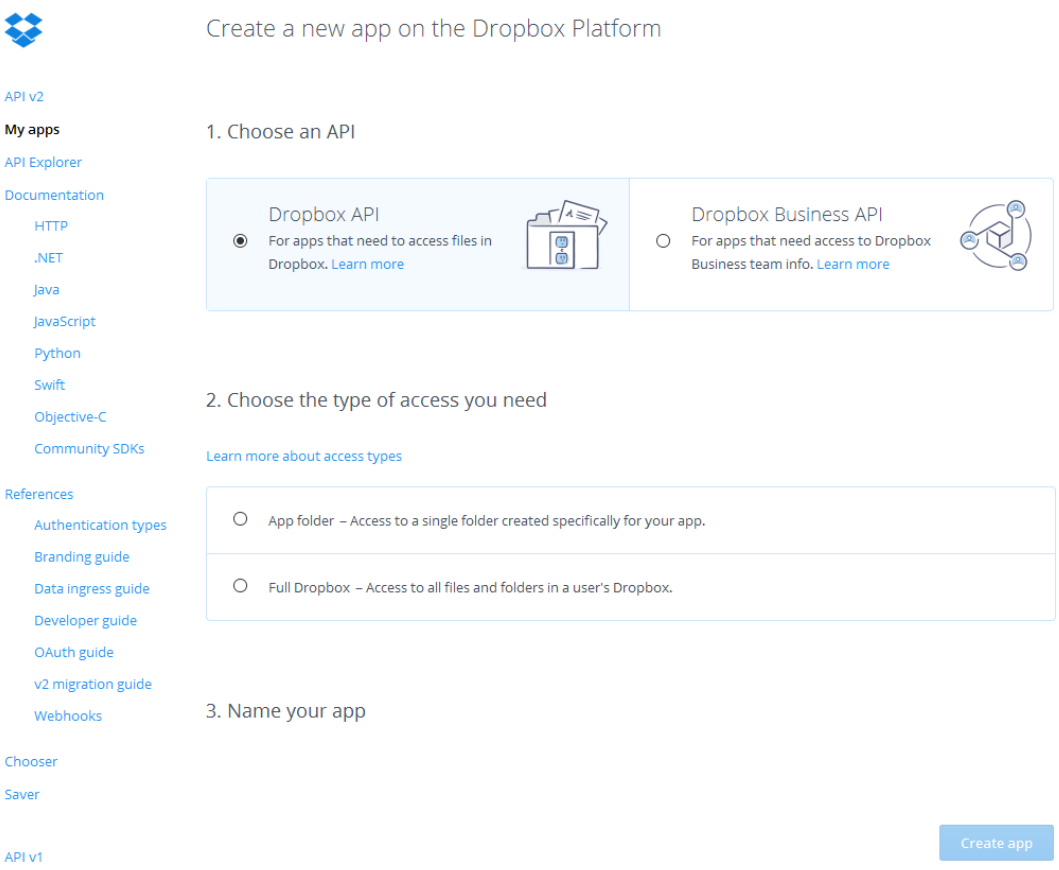
```
curl -X POST https://content.drop-
boxapi.com/2/files/upload \
 --header 'Authorization: Bearer [AccessToken]' \
 --header 'Content-Type: application/octet-stream' \
 --header 'Dropbox-API-Arg:{"path":[PathInDropbox]",
"mode":{".tag":"add"},
"autorename":false}' \
 --data-binary @'[TestFileToSend.txt]'
```

Kun tiedosto on lähetetty, sille tehdään jakolinkki seuraavasti:

```
curl -X POST https://api.dropboxapi.com/2/shar-
ing/create_shared_link_with_settings \
 --header 'Authorization: Bearer {AccessToken}' \
 --header 'Content-Type: application/json' \
 --data '{"path":"/TestiFolder/asd2.txt","set-
tings":{"requested_visibility":{".tag":"public"}}}'
```

Vanhat versiot poistetaan, kun ne ovat tarpeeksi vanhoja erillisellä skriptillä, käyttäen seuraavanlaista pyyntöä:

```
curl -X POST https://api.dropboxapi.com/2/files/de-
lete \
 --header 'Authorization: Bearer {AccessToken}' \
 --header 'Content-Type: application/json' \
 --data '{"path":"/{Folder}/{File}"}'
```

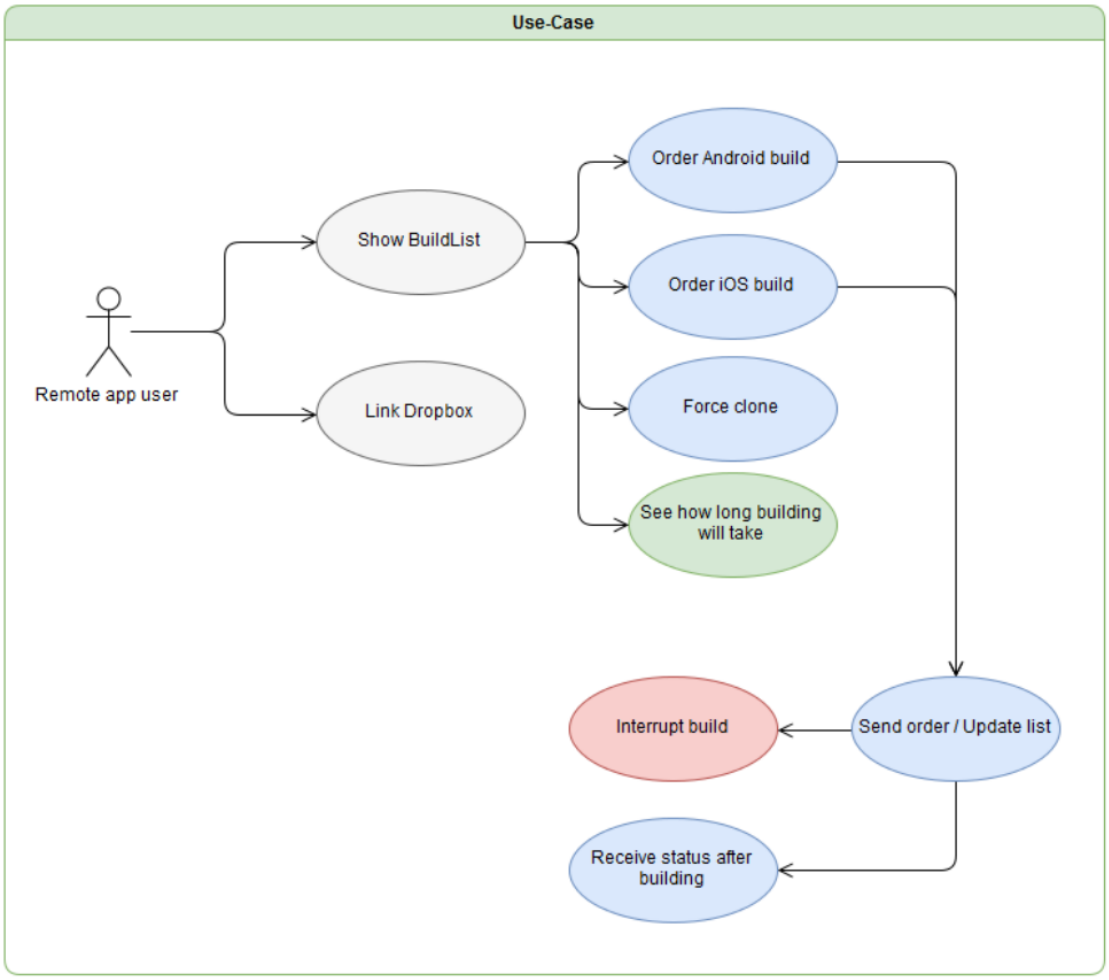


Kuvio 26. Dropbox sovelluksen luonti

4.9 Remote app

Projektin edetessä toimeksiantaja esitti toivomuksen sovelluksesta, jolla voisi tilata käännöksen aloittamisen etänä puhelimella. 2016 heinäkuussa Tarmo Jussila ja Tuomo Oksanen aloittivat sovelluksen toteuttamisen. Toteutuksessa neuvoteltiin Tuomas Kytän kanssa, miten sovellus keskustelisi palvelimen kanssa ja miten järjestelmän tietoja pidettäisiin yllä, mikä niitä muuttaisi ja miten. Myöhempi Remote appin kehittäminen järjestelmän kehittyessä jäi Tuomo Oksasen vastuulle.

Remote app on tarkoitettu toimeksiantajan sisäiseen käyttöön ja sillä voi toteuttaa mm. kuvion 27 mukaisia toimintoja. Tällä hetkellä käyttäjä kirjautuu Dropbox tunnuksilla ja jos käyttäjä kuuluu toimeksiantajan jakamaan kansioon, annetaan käyttäjälle pääsy systeemiin. Remote App on tehty Xamarin.Formsilla, joka generoi sovelluksen ja UI:n C#-koodista iOS:lle ja Androidille.



Kuvio 27. Remote appin use-case kaavio

Koska järjestelmän Jenkins-palvelin kuuntelee vain paikallisen koneen liikennettä, Remote app käytännössä muokkaa ulkoisella palvelimella olevia tiedostoja. Kone, jolla järjestelmä on, tarkkailee jatkuvasti pyörivällä skriptillä muutoksia kyseisissä tiedostoissa ja aloittaa Jenkinsillä määriteltyjä töitä havaittujen muutosten mukaan.

4.10 Tukevat Jenkins-työt

Suurin osa Jenkinsille määritellyistä töistä on töitä, joissa versioidaan peli. Osa on järjestelmää tukevia töitä, jotka suoritetaan säännöllisesti tai osana muita töitä ketjumaisesti.

4.10.1 Vanhojen tiedostojen poisto

Jokaisen Xcode projektin jälkeen Xcoden kansioon nimeltä ”DerivedData” tuli uusia tiedostoja. Derived data -kansion alikansioiden sisältö generoidaan aina, kun Xcodella

käännetään sovellusta ja alikansiot voidaan poistaa iTunes Connectiin lähetyksen jälkeen. Derived data -kansion sisältö oli pahimmillaan projektin aikana 60 Gt kokoinen ja vei suurimman osan koneen tilasta. Tätä varten Jenkinsiin määriteltiin kansion tyhjentävä työ, joka suoritetaan jokaisena maanantai aamuna klo 4:00.

Järjestelmää varten luodulle Dropbox tilille lähetetyt tiedostot pitää poistaa tietyn ajan jälkeen, jotta tili ei täyttyisi turhaan. Sama Jenkins-työ, jossa poistetaan Derived data -kansion sisältö, suorittaa myös skriptin, joka poistaa tililtä kaksi viikkoa vanhat tiedostot.

4.10.2 Google Play Developer Consoleen lähetys

Järjestelmään tutkittiin APK-tiedostojen lähettämistä ja jakamista Google Play Developer Consolen avulla. Jenkinsiin on tehty lisäosa, Google Play Android Publisher Plugin, jolla työn jälkeisenä osana voi lähettää tiedoston Googlen palveluun. Googlen palvelu todettiin kuitenkin toistaiseksi turhaksi toimeksiantajan tarpeille.

Google Play palveluun on järjestelmälle tehty palvelutili (Service account). Tälle palvelutilille pitää antaa projektien oikeudet, joita halutaan sitä kautta lähettää Google Play palveluun (Google Play Android Publisher Plugin. n.d). Ohjeet palvelutilin luomiseen ja sen lisäämiseen Jenkinsiin ovat lisäosan verkkosivuilta ja liitteessä 7.

Lisäosan rajoitteena on, että se voi lähettää tiedostoja vain omasta projektikansiostaan. Toisena rajoitteena Googlen rajapinnassa on se, ettei täysin uutta versiota voida lähettää. Pelistä on siis oltava olemassa versio, jotta rajapinnalla, jota lisäosa käyttää, voidaan lähettää mitään.

Tutkittiin vaihtoehtoa tehdä ehdollinen työn jälkeinen operaatio, jossa käytettäisiin lisäosan ominaisuutta. Tämä ei osoittanut mahdolliseksi projektin aikana, joten päätettiin kiertää ongelma määrittelemällä parametrisoitu työ, joka suoritettaisiin Android-version kääntävän skriptin aikana. Skriptin osa on liitteessä 8.

4.10.3 Tulosten raportointi

Jokaisen käännöksen jälkeen suoritetaan Jenkins-työ nimeltä ”ReportBack”. Tämä työ raportoi työn tilan tiedostoon, jota mobiilisovellus lukee. Kuviossa 21 on näytetty, miten työn aloittaminen asetetaan. Työ itsessään on yksinkertainen: Työn vaatii kaksi parametria ja suorittaa yhden skriptin, johon annetaan parametreiksi työlle annetut parametrit.

5 Tulokset

5.1 Lopputulos

Lopputuloksena saatiin modulaarinen järjestelmä, jolla pystyy automatisoimaan versioinnin iOS- ja Android-alustoille. Uuden projektin asettaminen onnistuu n. vartissa riippuen halutuista ominaisuuksista. Järjestelmää käytetään manuaalisen testauksen apuvälineenä ja se suorittaa automaattisesti aiemmin käsin tehdyt operaatiot säästäen toimeksiantajan aikaa projektista, tai projekteista, riippuen parhaimmillaan 25h viikossa. Säästetty aika riippuu projektien koosta ja kuinka monta projektia on yhtäaikaisesti kehityksessä.

Järjestelmän suorittamissa skripteissä on virheentarkastus ja virheilmoituksissa kirjataan, missä virhe tapahtui. Virhejärjestelmä on myös käytössä järjestelmän puhelin sovelluksessa. Osa virheistä estää tietyn pelin versioinnin ja osa estää koko sovelluksen käytön, kunnes vika on merkattu korjatuksi.

Google Developer Console ei ole käytössä, mutta se testattiin ja sen käyttöönotto on mahdollista korvaamalla nykyiset Android-version tekevät skriptit.

5.2 Jatkokehitys

Tällä hetkellä järjestelmä kuuntelee vain paikallisia yhteyksiä. Koneella pyörivä skripti tarkkailee muutoksia ulkoisella palvelimella olevassa tiedostossa ja suorittaa sen mu-

kaan toimintoja. Ulkoiselle palvelimelle myös kirjataan virheilmoitukset ja tilanne raportit. Tähän päädyttiin toimeksiantajan pyynnöstä. Jenkins palvelin, johon voisi yhdistää sisäverkon ulkopuolelta, oikoisi kehityksessä tulleita mutkia.

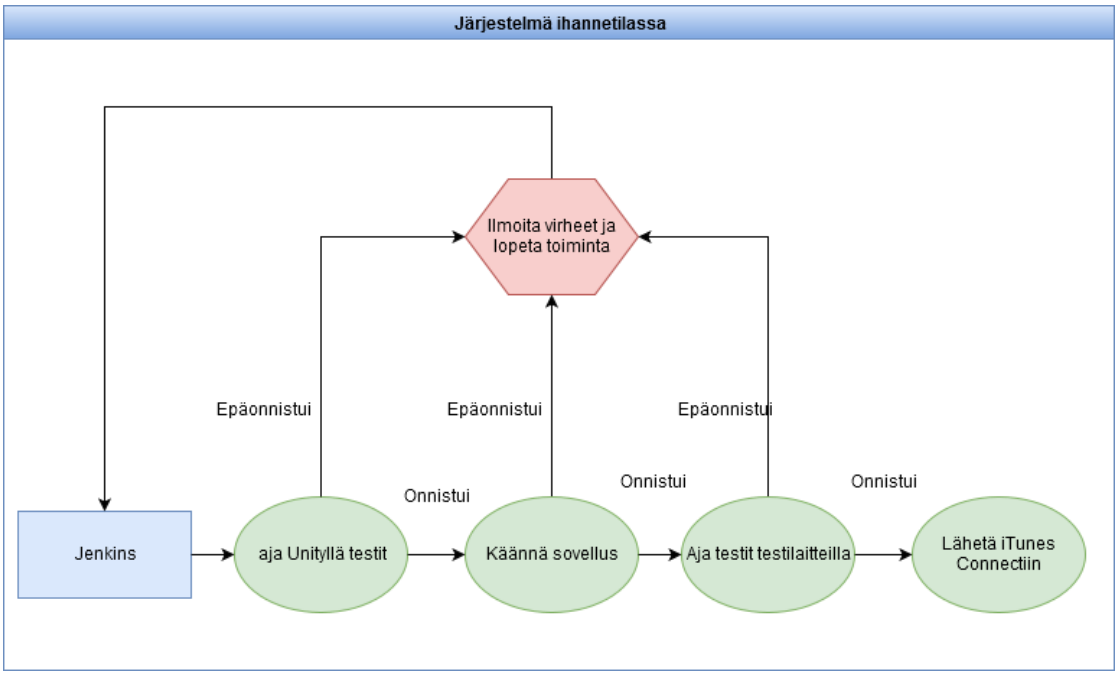
Järjestelmässä luotiin Jenkinsin työtilakansioon kansio, johon säilöttiin suoritettavat skriptit ja jokaiselle pelille omat pelikohtaiset tiedot. Pelikohtasiin tiedostoihin kuuluivat mm. tiedostot johon säilöttiin commit-viestit ja sen hetkinen versionumero. Nämä tiedot olisi voinut yhdistää JSON-tiedostoiksi ja tiedoston nimetä pelin mukaan, jolloin olisi selvennetty tiedostohierarkiaa. Tämä vaatisi muutoksia Unityn ja Jenkinsin suorittamiin skripteihin.

Uuden työn asettamiseen voisi luoda oman skriptin tai oman laajennuksen Jenkinsiin, missä käyttäjä täyttää tarvittavat tiedot ja niistä muodostettaisiin ajettava työ järjestelmään.

Projektin luovutustilaisuudessa pohdittiin toimeksiantajan kanssa mahdollisuutta tehdä skripti, joka virhetilanteessa pistäisi viestiä järjestelmää varten luodulle Flowdock-kanavalle. Tämä nopeuttaisi virheiden havaitsemista ja korjaamista.

Luovutushetkellä järjestelmä ei toteuta testausautomaatiota. Ainoat virheet, jotka testataan, liittyvät käännöksen ja iTunes Connectiin lähetyksen onnistumiseen. Unityllä suoritettujen testien implementointi järjestelmään olisi suoraviivaista, mutta ne pitäisi erikseen tehdä. Tämä vaatisi enemmän työtä koodaajilta, mutta oikein toteutettuna voisi säästää aikaa ja vaivaa. Kuviossa 28 on esitetty mahdollinen ihanne tilanne systeemille.

Laitteilla testaamisen automatisointi, esim. muistin käyttö ja mahdollinen kaatuminen, vaatisivat huomattavia investointeja kattavaan testilaitteistoon tai palvelun ostamista ulkoisilta toimijoilta. Mahdollisia ulkopuolisia palveluntarjoajia ovat mm. Googlen omistama Firebase. Vaihtoehtoisesti testejä voisi suorittaa Appiumilla. Appium on avoimen lähdekoodin projekti ja käytännössä kääntää Seleniumim WebDriver komentoja UIAutomation tai UIautomator komennoiksi kohdealustasta riippuen (Helppi. 2015).



Kuvio 28. Jatkokehityksen ihannetavoite järjestelmälle

5.3 Haasteet

Työn aikana koettiin haasteita, järjestelmä muuttui monimuotoisemmaksi ja monimutkaisemmaksi. Suurin yksittäinen haaste oli tiedon puute. Tieto kuinka Applen järjestelmä toimii, oli hyvin vähäinen projektin alussa. Tieto kuinka toteutettu järjestelmä kannattaisi toteuttaa oli alussa suurin piirtein tiedossa. Lähteet verkossa automatisointiin peliteollisuudessa olivat vähäisiä, mutta niistä sai hyvin suuntaviivaa.

Projekti kehittyi iteratiivisesti yrityksen ja erehdyksen kautta. Onnistumisten jälkeen pohdittiin uusia ominaisuuksia ja se ajoi nopeasti modulaariseen toteutukseen. Mitään yksittäistä operaatiota ei haluttu sitoa liian tiukasti toisiin operaatioihin, koska ei voitu ennalta tietää mitä uusia vaatimuksia tai ideoita tulisi.

5.3.1 Apple ja Xcode

Alkuperäinen päätavoite oli saada iOS-versioiden versiointi ja lähettäminen. Applen systeemit olivat tekijöille lähes tuntemattomia. Tutkiminen, miksi tiedoston lähettäminen iTunes Connectiin epäonnistui, vei huomattavan määrän aikaa projektin alussa. Xcode ja sitä käyttävät työkalut eivät antaneet virheilmoituksia, joista olisi

saanut syitä selville helposti. Osittain tämä johtuu tavasta, jolla Unity3D tekee projektista Xcode-projektin.

Tietämättömyys Applen järjestelmistä myös johti tapahtumaan, missä alussa testasimme järjestelmää jonkun Zaibatsun työntekijän tunnuksilla. Applen omissa dokumenteissa kehotetaan tällaista toimintaa vastaan, mutta sitä ei yksinkertaisesti tiedetty. Tästä seuranneet paikallisten tunnusten ajoittaiset epäämiset kummastuttivat hyvin paljon. Asiaa ei auttanut Xcode 7:n Fix issues -nappula, joka todellisuudessa epäsi kaikki tunnuksen käytössä olevat provision profiilit ja sertifikaatit.

Projektin alussa käytimme Xcode 7.2:sta. Projektin aikana siirryimme käyttämään Xcode 8.1:stä. Xcode 8 julkaistiin 13.9.2016 ja projekti aloitettiin 1.6.2016. Xcode 8 muutti logiikkaa, jolla koodin signeeraukseen käytetyt tunnukset tunnistettiin ja se vaati omia korjausliikkeitään projektin aikana. Kokonaisuudessa Xcode 8 helpotti järjestelmän toteutusta.

6 Johtopäätökset ja pohdinta

Opinnäytetyö oli mielenkiintoinen toteuttaa. Oli mielenkiintoista oppia miten mobiilipelien julkaisuun käytettävät järjestelmät toimivat. Automaatioketjujen suunnittelu ja toteutus oli myös mielenkiintoista. Suurimmaksi osaksi sain vapaat kädet toteuttaa järjestelmän. Jenkins-palvelimen näkyvyys oli toimeksiantajan rajaama rajoite. Kustannustehokkuus oli myös huomioitava. Työ suoritettiin enimmäkseen ilmaisilla avoimen lähdekoodin projekteilla.

Opinnäytetyö sai alkunsa harjoittelun aikaisena tehtävänä, kun kyseenalaistin versioinnin tekemisen käsin. Jyväskylän ammattikorkeakoulussa testauskurssilla olin oppinut Jenkinsistä vähän ja se vaikutti nopeasti tutkittuna hyvältä vaihdoehdolta työn toteuttamiseen. Jenkinsin käytöllä saatiin säästettyä aikaa käyttöliittymän ja hallintapaneelin tekemisessä.

Järjestelmä, joka toimi ja lähetti iOS-version iTunes Connectiin, oli valmis melko nopeasti. Järjestelmä, joka toimi vakaasti ja ilman virheitä, vaati huomattavasti

enemmän aikaa. Osa virheistä oli itse aiheutettuja, johtuen tietämättömyydestä. Järjestelmä ja sen skaala kasvoivat projektin edetessä ja projektista toteutettiin ainakin kaksi toisistaan niin eroavaa versiota, että ne vaativat erillisen dokumentaation.

Johtuen osittain CI-systeemien oppimisesta testauskurssilla aloin pohtimaan kesken projektin miten helppoa olisi lisätä testaus osaksi automaatioketjua. Alussa olin skeptinen kannattaako testauksen automatisointi peliprojekteissa ollenkaan, mutta mitä enemmän tutkin aihetta, tajusin vastauksen olevan varsin yleinen: ”Riippuu tilanteesta”. Jos projektissa on paljon ohjelmoijia, jos osan koodista on pakko toimia ehdottomasti virheettä ja jos projekti elää kohtuullisen kauan kehitysvaiheessa niin testauksesta ja sen automatisoinnista saa hyötyä irti.

Fastlane ja sen sovellukset vaikuttivat hyviltä työkaluilta ja ovat sitä. Niitä olisi tarkoitus ajaa Fastlane-työkalulla, jossa määriteltäisiin tiedostoon mitä työkaluja käyetetään milloinkin. Koska käytin projektissa vain Pilottia ja Gymiä, en kokeut tätä tarpeelliseksi vaan ajoin niitä omina sovelluksinaan. Fastlanen dokumentaatiossa oli myös monia mielenkiintoisia ideoita koodin signeeraukseen käytettyjen tunnusten säilömiseen, mutta ne tuntuivat yliampuvilta projektin tarkoituksiin.

Testaus ja sen automatisointi jäivät suunnittelutasolle. Vaikka joku voisi mieltää ylipäänsä kääntymisen olevan yksi iso testi, se ei takaa sovelluksen toimivuutta. Laitteistojen erilaisuus on etenkin Androidilla iso haaste ja vaikuttaisi, ettei halpoja ratkaisuja sovellusten testaamiseen eri laitteille ole. Testauksen automatiosinti vaatii myös perehtymistä myös kaikilta projektin koodaajilta. Testeistä parhaimman hyödyn saa, kun ne tehdään testattavan ominaisuuden tekijän toimesta.

Lähteet

About Continuous Integration in Xcode. N.d. Apple Inc. dokumentaatio jatkuvasta integroinnista Xcodella. Viitattu 9.11.2016. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/xcode_guide-continuous_integration/

About iTunes Connect. N.d. Apple Inc. Dokumnetaatio iTunes Connectista. Viitattu 10.11.2016. https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/About.html#//apple_ref/doc/uid/TP40011225-CH1-SW1

App Thinning. N.d. Pala Applen dokumentaatiota. Viitattu 31.1.2017. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/AppThinning/AppThinning.html

Apple Developer programs. N.d. Kooste Applen kehittäjä ohjelmista. Viitattu 10.11.2016. https://developer.apple.com/support/compare-memberships/

Adding Your Account to Xcode. N.d. Applen ohjeet tunnuksen lisäämiseen Xcodeen. Viitattu 31.1.2017. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppStoreDistributionTutorial/AddingYourAccounttoXcode/AddingYourAccounttoXcode.html

Ambler, S. 2002. Examining the Agile Cost of Change Curve. Viitattu 7.2.2017. http://www.agilemodeling.com/essays/costOfChange.htm

Automation done right. N.d. Fastlanen sivuilla oleva kooste Fastlaneen kuuluvista työkaluista. Viitattu 9.11.2016. https://fastlane.tools/

Average App Store Review Times. N.d. Sivusto, joka kerää ihmisten lähettämästä datasta Applen tarkastusaikojen keskiarvoa. Viitattu 12.2.2017. http://appreviewtimes.com/

Bilbie, A. n. d. A Guide To OAuth 2.0 Grants. Viitattu 8.12.16. https://alexbilbie.com/guide-to-oauth-2-grants/

Building your Unity game to an Android device for testing. N.d. Unityn dokumentaatio Android-versioiden tekemiseen. Viitattu 5.2.2017. https://unity3d.com/learn/tutorials/topics/mobile-touch/building-your-unity-game-android-device-testing

The CA Flowdock API. N.d. Flowdockin rajapinnan esittely. Viitattu 13.1.2017. https://www.flowdock.com/api

The CA Flowdock API Authentication. N.d. Flowdockin rajapinnan autentikoniti metodit. Viitattu 1.2.2017. https://www.flowdock.com/api/authentication

Carucci, F. 2009. AAA Automated Testing. Viitattu 7.2.2017. http://www.slideshare.net/fcarucci/aaa-automated-testing

Cerney, E. 2015. Setting up Jenkins CI on a Mac. Viitattu 10.11.2016. http://www.cimgf.com/2015/05/26/setting-up-jenkins-ci-on-a-mac-2/

Creating an iTunes Connect Record for an App. N.d. Applen dokumnetaatiota. Viitattu 25.1.2017. https://developer.apple.com/library/content/documentation/LanguagesUtilities/ConceptualiTunesConnect_Guide/Chapters/CreatingiTunesConnectRecord.html

Davis, A. & Single, A. 2016. Testing for Game Development. http://www.what-could-possibly-go-wrong.com/testing-for-game-development/

Dropbox Platform developer guide. N.d. Dropboxin dokumentaatiota. Viitattu 7.2.2017. https://www.dropbox.com/developers/reference/developer-guide#production-approval

The Dropbox Tour. N.d. Dropboxin esittelysivu. Viitattu 13.2.2017. https://www.dropbox.com/tour/0

EnvInject Plugin. N.d. EnvInject-lisäosan esittelysivu. Viitattu 7.2.2017. https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin

Exporting Your App for Testing. N.d. Apple Inc. dokumentaatio netissä. Viitattu 10.11.2016. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/TestingYouriOSApp/TestingYouriOSApp.html

Flows. N.d. Flowdockin dokumnetaatio flowsta. Viitattu 12.1.2017. https://www.flowdock.com/help/flows

Geig, M. 2014. Unity Test Tool -tutoriaalivideo Unity Technologisin tarjoamiin testaustyökaluihin. Viitattu 7.2.2017. https://unity3d.com/learn/tutorials/topics/production/unity-test-tools

Github-vs-bitbucket. N.d. Upguardin sivuilla tehty vertailu bitbucketista ja Githubista. Viitattu 13.1.2017. https://www.upguard.com/articles/github-vs-bitbucket

Google Play Android Publisher Plugin. N.d. Jenkins lisäosan esittelysivu. Viitattu 2.2.2017. https://wiki.jenkins-ci.org/display/JENKINS/Google+Play+Android+Publisher+Plugin

Gym readme. N.d. Gym sovelluksen repossa oleva read me -tiedosto. Viitattu 10.11.2016. https://github.com/fastlane/fastlane/blob/master/gym/README.md

Helppi, V. 2015. The Basics Of Test Automation For Apps, Games And The Mobile Web. Viitattu 18.2.2017. https://www.smashingmagazine.com/2015/01/basic-test-automation-for-apps-games-and-mobile-web/

How to use the Google Play Developer Console. N.d. Googlen dokumentaatio Developer Consolesta. Viitattu 10.1.2017. https://support.google.com/googleplay/android-developer/answer/6112435?hl=en&ref_topic=3450769

Jenkins: Remote access API. N.d. Jenknisin rajapinnan esittely. Viitattu 17.2.2016. https://wiki.jenkins-ci.org/display/JENKINS/Remote+access+API

Klosowski, T. 2014. I Want to Write iOS Apps. Where Do I Start? Viitattu 14.11.2016. http://lifehacker.com/i-want-to-write-ios-apps-where-do-i-start-1644802175

Knorr, E. 2016. Why Jenkins is becoming the engine of devops. Viitattu 15.1.2017. http://www.infoworld.com/article/3046038/application-development/why-jenkins-is-becoming-the-engine-of-devops.html

Krause, F. 2015. fastlane is now part of Fabric. Viitattu 10.1.2017. https://krausefx.com/blog/fastlane-is-now-part-of-fabric

Krause, F. 2016. fastlane has saved over 1 million developer hours. Viitattu 10.1.2017. https://krausefx.com/blog/fastlane-has-saved-over-1-million-developer-hours

Lamppa, D. 2012. 5 Options for Distributing Your iOS App to a Limited Audience. Viitattu 10.11.2016. http://mobiledan.net/2012/03/02/5-options-for-distributing-ios-apps-to-a-limited-audience-legally/

Maintaining Your Signing Identities and Certificates. N.d. Applen dokumentaatiota. Viitattu 31.1.2017. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html

Maintaining Identifiers, Devices, and Profiles. N.d. Applen dokumentaatiota. Viitattu 25.1.2017. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html

Meet Jenkins. 2016. Jenkinsin wikin alkusivu. Viitattu 15.1.2017. https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

OAuth on Bitbucket Cloud. N.d. Atlassian dokumentaatio OAuthin käytöstä. Viitattu 8.12.16. https://confluence.atlassian.com/bitbucket/oauth-on-bitbucket-cloud-238027431.html

Pilot readme. N.d. Pilot sovelluksen repossa oleva read me -tiedosto. Viitattu 10.11.2016. https://github.com/fastlane/fastlane/blob/master/pilot/README.md

Schreier, J. 2017. Quality Assured: What It's Really Like To Test Games For A Living. Viitattu 7.2.2017. http://kotaku.com/quality-assured-what-it-s-really-like-to-play-games-fo-1720053842

Set up alpha/beta tests. N.d. Google Play Developer Consolen dokumentaatioita. Viitattu 5.2.2017. https://support.google.com/googleplay/android-developer/answer/3131213

Set up SSH for Git. N.d. Atlassian dokumentaatio SSH-avaimista Git palveluissa. Viitattu 7.2.2017. https://confluence.atlassian.com/bitbucket/set-up-ssh-for-git-728138079.html

Smith. J. 2008. Making a game "Just Right" through testing and play balancing. Viitattu 7.2.2017. http://www.slideshare.net/julio.gorge/making-a-game-just-right-through-testing-and-play-balancing

Start Building with Trello. N.d. Trellon ohjeet kehittäjille integraatio sovellusten tekoon. Viitattu 12.1.2017. https://developers.trello.com/get-started/start-building

Szalay, R. 2016. Creating a Build-Once iOS Deployment Pipeline: Part 1 – Introduction. Viitattu 9.11.2016. https://blog.richardszalay.com/2016/08/22/ios-deploy-pipeline-1-introduction/

TestFlight Beta Testing. N.d. Apple Inc. dokumentaatio TestFlightista. Viitattu 9.11.2016. https://developer.apple.com/library/content/documentation/LanguagesUtilities/ConceptualiTunesConnect_Guide/Chapters/BetaTestingTheApp.html

Thompson, M. 2014. Nomad. Viitattu 9.11.2016. http://nomad-cli.com/

Tikka, S. 2014. Jenkins in your Dock! Viitattu 14.2.2017. https://github.com/stisti/jenkins-app/blob/master/README.rst

Trello tour. N.d. Trellon esittelysivu. Viitattu 12.1.2017. https://trello.com/tour

Unity asset store: uTomate. 2013. uTomaten myyntisivu Unityn asset storessa. Viitattu 7.11.2016. https://www.assetstore.unity3d.com/en/#!/content/7703

Unity: Creating and using scripts. N.d. Unityn ohjeet scriptien tekoon. Viitattu 9.11.2016. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Unity: Support for Split Application Binary. N.d. Unityn ohjeet obb-tiedostojen hallnitaan Android peleihin. Viitattu 12.1.2017. https://docs.unity3d.com/Manual/android-OBBsupport.html

Unity Test Tools Documentation. N.d. Viitattu 7.2.2017. https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/Home

UnityTestTools: How to use the Integration Test Framework. N.d. Viitattu 6.3.2017. https://bitbucket.org/Unity-Technologies/unitytesttools/wiki/IntegrationTestsRunner

Upload an app. N.d. Google Play Developer Consolen dokumentaatiota. Viitattu 5.2.2017. https://support.google.com/googleplay/android-developer/answer/113469?hl=en&ref_topic=7072031

Using Application Loader. N.d. Apple Inc. ohjeet Application loaderin käytöstä. Viitattu 9.11.2016. http://help.apple.com/itc/apploader/#/apdS673accdb

uTomate Manual. N.d. Ancient Light Studiosin dokumentaatio uTomatesta. Viitattu 10.1.2017. https://ancientlightstudios.com/utomate/documentation/manual.html

Veer, R. 2015. iOS continuous delivery with Jenkins and Fastlane. Viitattu 10.11.2016. https://labs.kunstmaan.be/blog/ios-continuous-delivery-with-jenkins-and-fastlane

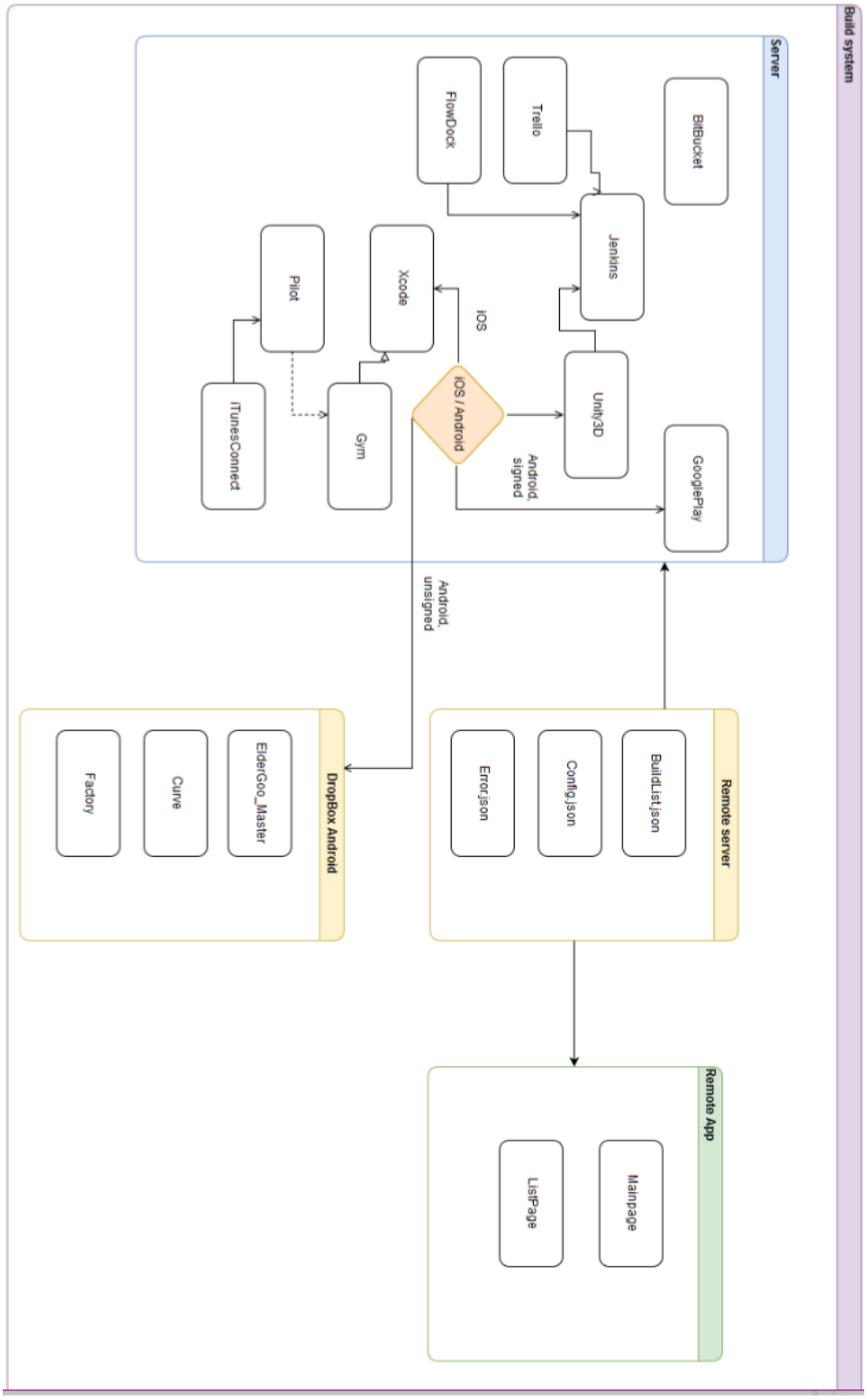
What-is-git. N.d. Atlassianin tekemä ohje Git työkalusta. Viitattu 13.1.2017. https://www.atlassian.com/git/tutorials/what-is-git

Xcode. N.d. Xcoden esittelysivu. Viitattu 9.11.2016. https://developer.apple.com/xcode/

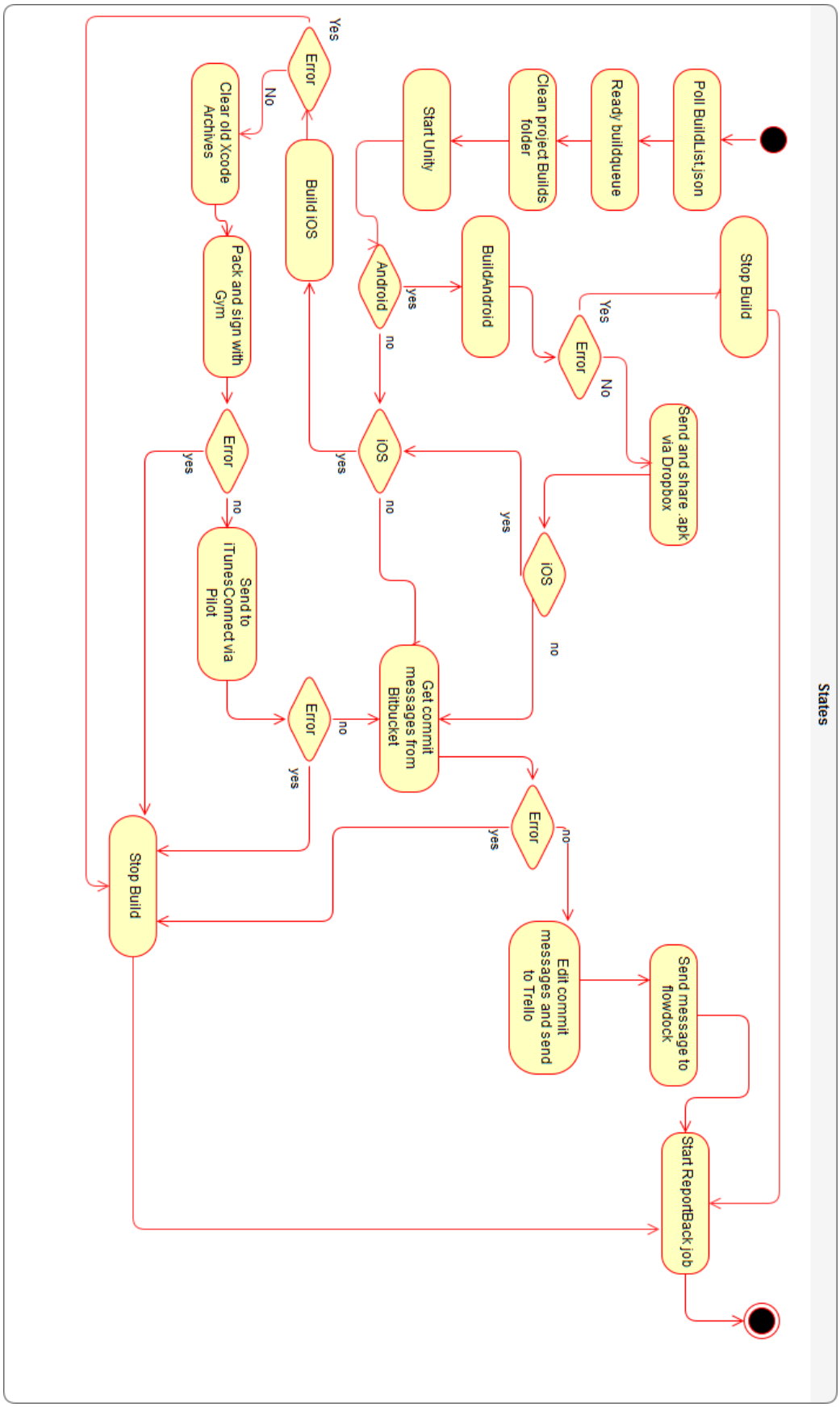
Zaibatsu press kit. N.d. Zaibatsun sivuilla oleva press kit. Viitattu 7.11.2016. http://zaibatsu.fi/press/

Liitteet

Liite 1. Projektin toteutuksen kuvio.



Liite 2. Järjestelmän tilakaavio.



Liite 3. Osa skriptistä, jolla lähetetään APK-tiedoston jakamislinkki Flowdockkiin

```
#  Make a post to flowdock containing link to apk
if ! curl -i -X POST -H "Content-Type: application/json"
-d '{
  "flow_token": "'"$flowToken"'",
  "event": "message",
  "content": "'"$content"'",
  "thread_id": "'"$threadId"'"
}' https://api.flowdock.com/messages; then
echo "Problems with Flowdock"

echo $errorJsonData | jq ".Errors | map(if .Description
== \"Flowdock\" then . + {\"Error\": \"TRUE\"} + {\"Mes-
sage\": \"Problems with sending message to Flowdock.\"}
else . end) | {\"Errors\": . }" > $errorJson
exit 1;
else
echo "Flowdock working as intended"
echo $errorJsonData | jq ".Errors | map(if .Description
== \"Flowdock\" then . + {\"Error\": \"FALSE\"} + {\"Mes-
sage\": \"No problems with sending message to Flowdock
for now.\"}  else . end) | {\"Errors\": . }" > $errorJson
fi
```

Liite 4. Oleellisimmat osat iOS-version luovasta skriptistä

```
#!/bin/bash
# Script to build projects for iOS

#Main variables
config='/Users/Zaibatsu/.jenkins/workspace/BuildausSkriptit/con-
fig.json'
projectpath=$(echo $1)
password=$(cat $config | jq ".ComputerPassword" | sed
's/^"\(.*\)"$/\1/')
user=$(cat $config | jq ".iTunesConnectUser" | sed
's/^"\(.*\)"$/\1/')
appleId=$(echo $2)
bundleId=$(echo $3)
projectName=$(echo $4)
filename=$(cat $config | jq ".BuildList" | sed 's/^"\(.*\)"$/\1/')
errorJson=$(cat $config | jq ".ErrorJson" | sed 's/^"\(.*\)"$/\1/')
workspace=$(cat $config | jq ".Workspace" | sed 's/^"\(.*\)"$/\1/')
folder="BuildausSkriptit/"
unityVersion=0

#Reads all prejects and selects unity versio based on name
ReadUnityVersio(){
unityVersion=$(cat "$workspace$folder$projectName/unityVersion.txt")
}

#Remove old builds
if [ -d $projectpath/Builds ]
then
    rm -frd $projectpath/Builds/*
    echo "Removed existing builds"
else
    mkdir $projectpath/Builds
    echo "Created Builds -folder"
fi

#Go through all projects and select right Unity version
ReadUnityVersio

#This echo is mainly for debugging via Jenkins console
echo "Started Unity"

errorJsonData=$(cat $errorJson)

echo "$unityVersion"

#Trigger Unity editor script to actyally build
if ! /Applications/Unity$unityVersion/Unity.app/Contents/MacOS/Unity
-batchmode -quit -buildtarget ios -projectpath $projectpath -exe-
cutemethod CustomBuild.PerformIOSBuild; then
  echo "Problem with Unity"
  echo $errorJsonData | jq ".Errors | map(if .Description ==
\"Unity\" then . + {\"Error\": \"TRUE\"} + {\"Message\": \"Problems
with Unity during project $projectName\"} else . end) | {\"Errors\":
. }" > $errorJson
  exit 1;
else
  echo "Unity worked as intended"
```

```
  echo $errorJsonData | jq ".Errors | map(if .Description ==
\"Unity\" then . + {\"Error\": \"FALSE\"} + {\"Message\": \"No prob-
lems with Unity for now\"}  else . end) | {\"Errors\": . }" > $error-
Json
fi

#errorInBuildList=$(cat $filename)
#if ! /Applications/Unity$unityVersion/Unity.app/Contents/MacOS/Unity
-batchmode -quit -buildtarget ios -projectpath $projectpath -exe-
cutemethod CustomBuild.PerformIOSBuild; then
#  echo "Problem with Unity"
#  echo $errorInBuildList | jq ".Games | map(if .Name == $projectName
then . + {\"Error\":\"Problems with Unity"} else . end) | {\"Games\":
.}" > $filename
#  exit 1;
#else
#  echo "Gym worked as intended"
#  echo $errorInBuildList | jq ".Games | map(if .Name == $projectName
then . + {\"Error\":\"\"} else . end) | {\"Games\": .}" > $filename
#fi

echo "Unity project done"
#Return to project folder, go to builds folder and cd first one
#Gym will build .ipa and pilot will send it to iTunesConnect
#Password ought to be saved to keychain
cd $projectpath
Security unlock-keychain -p $password
cd Builds/`ls -1 Builds/ | head -1`

errorJsonData=$(cat $errorJson)

if ! gym; then
  echo "Problem with Gym"
  echo $errorJsonData | jq ".Errors | map(if .Description == \"Gym\"
then . + {\"Error\": \"TRUE\"} + {\"Message\": \"Problems with pack-
aging and signing .ipa during project $projectName\"} else . end) |
{\"Errors\": . }" > $errorJson
  exit 1;
else
  echo "Unity worked as intended"
  echo $errorJsonData | jq ".Errors | map(if .Description == \"Gym\"
then . + {\"Error\": \"FALSE\"} + {\"Message\": \"No problems with
.ipa for now\"}  else . end) | {\"Errors\": . }" > $errorJson
fi

errorJsonData2=$(cat $errorJson)

if ! pilot upload -u "$user" -a "$bundleId" -p "$appleId"
--distribute_external false -z true; then
  echo "Problem with Pilot. Network probably went down."
  echo $errorJsonData2 | jq ".Errors | map(if .Description == \
"Pilot\" then . + {\"Error\": \"TRUE\"} + {\"Message\": \"Problems
with sending .ipa to iTunesConnect during project $projectName\"}
else . end) | {\"Errors\": . }" > $errorJson
  exit 1;
else
  echo "Unity worked as intended"
  echo $errorJsonData2 | jq ".Errors | map(if .Description ==
\"Pilot\" then . + {\"Error\": \"FALSE\"} + {\"Message\": \"No prob-
lems with iTunesConnect for now\"}  else . end) | {\"Errors\": . }" >
$errorJson
fi
```

Liite 5. PostProcessBuild osio luokasta, joka asettaa Xcode-projektiin oikean ryhmän

```
[PostProcessBuild]
public static void OnPostprocessBuild(BuildTarget buildTar-
get, string path)
{

   if (buildTarget == BuildTarget.iOS)
   {
        string projPath = path + "/Unity-
        iPone.xcodeproj/project.pbxproj";

        PBXProject proj = new PBXProject();
        proj.ReadFromString(File.ReadAllText(
        projPath));

        string target = proj.TargetGuidByName("Unity
        -iPhone");

        proj.SetBuildProperty(target,
        "DEVELOPMENT_TEAM", "1234567asd");

        File.WriteAllText(projPath,
        proj.WriteToString());
    }
}
```

Liite 6. PostProcessBuild osio luokasta, joka asettaa Bitcoden epätodeksi

```
[PostProcessBuild]
public static void OnPostprocessBuild(BuildTarget
buildTarget, string path)
{
   if (buildTarget == BuildTarget.iOS)
   {
      string projPath = path + "/Unity-iPone.
      xcodeproj/project.pbxproj";

      PBXProject proj = new PBXProject();
      proj.ReadFromString(File.ReadAllText(
      projPath));

      string target = proj.TargetGuidByName(
      "Unity-iPhone");

      proj.SetBuildProperty(target, "ENABLE_BITCODE",
      "false");

      File.WriteAllText(projPath,
      proj.WriteToString());
    }
}
```

Liite 7. Palvelutilin lisääminen Google Play Consoleen ja Jenkinsiin (Google Play Android Publisher Plugin. n.d.)

Create Google service account

To enable automated access to your Google Play account, you must create a service account:

1. Sign in to the Google Play developer console as the account owner
2. Select Settings → API access
3. Click "Create new project"
4. Once created, click "Create Service Account"
5. Follow the link to the Google Developers Console
6. From the "New credentials" drop-down, choose "Service Account Key"
7. From the "Service account" drop-down, choose "New service account"
8. Give the account any name you like, e.g. "Jenkins"
9. Select "JSON" as the key type (P12 works as well, but JSON is a little simpler)
10. Click the "Create" button
11. Note that a .json file is downloaded, named something like "Google Play Android Developer-xxxxxxxxxxxx.p12"
12. You can now close the page

Assign permissions to the service account

1. Return to the Google Play developer console page
2. Click "Finished" on the dialog
3. Note that the service account has associated with the Google Play publisher account
4. Click the "Grant access" button for the account (e.g. "jenkins@api-xxxxxxxxxxxx.gserviceaccount.com")
5. Ensure at least the following permissions are enabled:
    a. Edit store listing, pricing & distribution
    b. Manage Production APKs
    c. Manage Alpha & Beta APKs
    d. Manage Alpha & Beta users
6. Click "Add user"
7. You can now log out of the Google Play publisher account

Add the service account to Jenkins:

1. Navigate to your Jenkins instance
2. Select "Credentials" from the Jenkins sidebar
3. Choose a credentials domain and click "Add Credentials"
4. From the "Kind" drop-down, choose "Google Service Account from private key"
5. Enter a name for the credential — the actual value is not important
6. Choose the "JSON key" type
7. Upload the .json file that was downloaded by the Google Developers Console
8. Click "OK" to create the credential

Jenkins now has the required credentials and permissions in order to publish to Google Play.

Liite 8. Osa Google Play Developer Consoleen tiedoston lähettävästä skriptistä

…

```
# Take the only file in Build folder
fileName=$(ls | sort -n | head -1)

# Send file to SendToGooglePlay work folder
mv "$projectpath/Builds/$fileName" $destination

# Get acces to trigger jenkins job
crumb=$(curl -s --user $user:$password \
    $server/crumbIssuer/api/xml?xpath=concat\(//crumbRe-
questField,%22:%22,//crumb\))

# Trgger SendToGooglePlay and tell which file(s) to send
curl -s -v -H "$crumb" -X POST -u $user:$password
"$server/job/SendToGooglePlay/buildWithParameters?File-
ToSend=$fileName"
```