jamk

DevOps-elinkaari aloittavalle indie-peliyritykselle

Kim Eriksson

Opinnäytetyö, AMK
Lokakuu 2022
Liiketalouden ala
Tradenomi, tietojenkäsittelyn tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Eriksson, Kim

DevOps-elinkaari aloittavalle indie-peliyritykselle

Jyväskylä: Jyväskylän ammattikorkeakoulu. Lokakuu 2022, 46 sivua.

Liiketalouden ala. Tietojenkäsittelyn tutkinto-ohjelma. Opinnäytetyö AMK

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

DevOps on laajalti tunnettu sovellus- ja web-kehityksen puolella. DevOps yhdistää tuotteen kehityksen ja operaatiot, mahdollistaen paremman kommunikaation tuotteen kehittäjien kesken sekä loppukäyttäjän kanssa. Sen periaatteita voi soveltaa myös pelikehityksessä, mikä vähentää crunchiin, eli ylityöhön turvautumista projektin lopussa.

Kehittämistutkimuksen tarkoituksena oli luoda aloittavalle indie-peliyritykselle ilmainen sekä helposti ylläpidettävä DevOps-elinkaari, mikä kattaa CI/CD automaation, ChatOps-alustan, seurannan ja raportoinnin.

Asetettujen kriteerien ja hankitun teorian pohjalta suunniteltiin DevOps-elinkaari, mitä rakennettiin iteratiivisesti laajemmaksi kokonaisuudeksi käyttäen laadullisia menetelmiä.

DevOps-elinkaaren todettiin tuovan kehitystä ja hyötyä verrattuna lähtötilanteeseen.

Avainsanat (asiasanat)

DevOps, Unity, CI/CD, Jenkins, testaus, indie, pelikehitys

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Eriksson, Kim

Developing DevOps-pipeline for an upcoming indie game studio

Jyväskylä: JAMK University of Applied Sciences, October 2022, 46 pages.

Business administration. Degree programme in business information systems. Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

DevOps is more known and used in software and web development, it brings together Development and Operations, enabling fluent communication inside the company and with the end users. It could be used in game development, since it could reduce the usage of crunch in a project.

The task was to find and implement an free and easy to use DevOps lifecycle for an upcoming indie game studio. The lifecycle must include an CI/CD pipeline, ChatOps platform and tools to monitor the project and gather feedback.

With the gathered knowledge and predefined criteria, an DevOps lifecycle was planned and build in a iterative approach till a working setup was achieved using qualitative research methods.

The DevOps lifecycle was deemed worthy by all set criterias and by practical usage, compared to the baseline.

Keywords/tags (subjects)

DevOps, Unity, CI/CD, Jenkins, testing, indie, gamedev

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Keskeiset käsitteet

ChatOps – Läpinäkyvä yhteistyömalli jossa yhdistetään automaatio, ihmiset, prosessit ja työkalut.

CI/CD – Continuous integration / Continuous Delivery (tai Continuous Deployment riippuen automaation määrästä), eli jatkuvan integraation ja julkaisun yhdistäminen automaatio putkeksi.

Crunch – Yleisesti pelialalla käytettävä termi, jolla viitataan projektin loppusuoralla tehtyyn ylityöhön. Tämä on yleensä ilmentymä teknisesta velasta.

Docker – Palvelutuote, mikä mahdollistaa virtuaalisten konttien ajon. Sen etuja on, että ajettu kontti on aina sama, riippumatta sitä ajavasta ympäristöstä. Kontti on yleensä sovellus ja sen ympäristö, mikä sisältää kaiken, mitä se tarvitsee.

Itch.io – Yksi suurimpia indie-peleihin erikoistuva verkkosivusto, jossa voidaan myydä, antaa ja ostaa, pääasiassa peleihin liittyvää sisältöä. Sivusto mahdollistaa myös pelijamien pitämisen.

LTS – Long Term Support. Tuote, jolla on pitkä tuki valmistajalta.

Pakettivarasto – (engl. repository) Tietovarasto, missä voi olla tiedostoja. Näitä tiedostoja voidaan ladata verkon välityksellä.

Unity engine – Unity Technologiesin kehittämä pelimoottori, jolla voidaan julkaista pelejä usealle eri kohdealustalle.

Webhook – Automatisoitu tapa kommunikoida eri palveluiden välillä.

Tekninen velka – Tekninen velka voi olla tahattomia virheitä tai suunniteltua riskinottoa, jossa pelataan jollain yrityksen resurssilla, yleensä aika ja/tai raha.

1 Johdanto

Pelialalla oli pitkään ollut hiljainen hyväksyntä niin kutsutulle Crunch-kulttuurille, joka oli mahdollistanut epärealistisen asetelman luvatuille odotuksille (Coster 2020; Tyagi 2021). Crunchiin turvautuminen voi olla yksi teknisen velan ilmentymä (Schmitt 2022). DevOps on nykyään yksi suosituimmista ja kehittyvimmistä toimintamalleista sovellus- ja webkehityksessä (Todorov 2022). DevOpsin tuomat hyödyt voisivat helpottaa pelifirmoja tekemään enemmän realistisia arvioita projekteistaan (Coster 2020).

Yrittäjyys on aina kiehtonut minua henkilökohtaisesti ja päätin jo koulutuksen ensimmäisenä lukuvuotena, että perustan sivutoimisen pelialan yrityksen koulusta valmistumiseni jälkeen. Toisen lukuvuoden ohjelmistotuotanto-kurssilla kävimme lävitse mitä DevOps on. Tämä herätti mielenkiinnon siitä, miten sitä voisi soveltaa pelikehitykseen. Kun lähdin tutkimaan DevOpsista pelinkehityksessä, löytyi siitä liian vähän tietoa suhteessa sovelluskehitykseen. Päätin tehdä opinnäytetyöni aiheesta ja lähteä selvittämään, miten se tapahtuu käytännössä ja mitkä olisivat parhaat käytänteet. Tämä palvelee suoraan suunnitelmiani yrityksen perustamisesta.

Tämän kehittämistutkimuksen tarkoituksena on annettujen kriteerien perusteella rajata ja tehdä helppokäyttöinen DevOps-elinkaari. Tämän elinkaaren avulla tuleva yritykseni voi keskittyä itse pelien tekemiseen vaivattomasti. Tärkein kriteeri suunniteltavalle elinkaarelle on sen hinta, koska käytännössä aloittavalla yrityksellä ei ole liioin ylimääräistä rahaa. Rajaus tehtiin koskettamaan vain ilmaisia vaihtoehtoja, pois lukien Unity-pelimoottori.

Vaikka työllä ei ole virallista toimeksiantajaa, koska itse yritystä ei ole vielä perustettu, toimin silti itse työn niin kutsuttuna toimeksiantajana ja olen määritellyt valitsemani kriteerit jo etukäteen. Tämä siksi, koska olen jo ennen opinnäytetyötä päättänyt esimerkiksi kohdejulkaisualustan ja kehittämistavan. Tämän takia tutkimuksen pääasiallinen hyötyjä on tuleva yritykseni, koska annetut kriteerit ovat spesifit minun käyttötarpeilleni. Tutkimuksesta ja sen löydöksistä voi silti olla hyötyä niille, jotka ovat saman tyylisillä ajatuksilla perustamassa indie-pelifirmaa, tai joita ylipäätänsä kiinnostaa, miten DevOpsia voisi käyttää pienen tiimin pelikehityksessä.

Tässä opinnäytetyössä tutkittiin mitä DevOps on, miten sitä voi soveltaa pelinkehityksessä sekä selvitettiin mitä työkaluja on saatavilla ilmaiseksi. Löydösten perusteella luotiin toimiva DevOps-elinkaari, jota vertailtiin lähtöasetelmaan.

2 Tutkimusasetelma

Tutkimuksen lähtöasetelmana on tilanne, jossa aloittava yritys tarvitsee DevOps-elinkaaren sisältäen vähintään automaatioputken, ChatOps-alustan, seurannan ja raportoinnin. Tarkoituksena on helpoittaa kehitystä automatisoimalla mahdollisimman monta vaihetta, sekä luomaan ympäristö, jota on helppo ylläpitää itsessään.

2.1 Tutkimuksen tavoite sekä työn rajaus

Kehittämistutkimuksen tavoitteena on ensin etsiä sopivat kriteerien täyttävät sovellukset. Tämän jälkeen suunnitellaan iteratiivisesti DevOps-elinkaari ja tutkitaan ratkaiseeko suunniteltu elinkaari tutkimusasetelman. Työn rajauksena on tulevan yrityksen asettamat tarpeet ja ehdot, joten seuraavat kriteerit on asetettu tutkimukselle:

1. Kaiken on oltava ilmaista. Maksullisia sovelluksia voidaan harkita, jos sovelluksesta löytyy ilmainen pienen tiimin versio.
2. Julkaisualusta on Itch.io. Tämä on ehdoton kriteeri, koska sinne julkaiseminen ei maksa mitään.
3. Uuden kehittäjän integrointi projektiin on oltava helppoa.
4. Kehitystyössä on alustavasti suunniteltu käytettävän testivetoista kehitystä.
5. Järjestelmän on oltava skaalautuva ja helposti muokattavissa.

2.2 Tutkimuskysymykset

Tutkimusongelmana on, miten tehdään tyhjästä aloittavalle yritykselle DevOpsia hyödyntävä automaatioputki. Noudattaen annettuja rajauksia, muodostui tutkimuskysymyksistä seuraavanlaiset:

1. Millaisia DevOps työvälineitä on saatavilla ilmaiseksi?
2. Millainen DevOps-elinkaari soveltuu parhaiten tarpeisiini?

2.3 Tutkimusmenetelmät

Tutkimusmenetelmäksi valikoitui kehittämistutkimus, koska tavoitteena on toteuttaa toimeksiannon tavoitteet, sekä samalla parantaa pelikehitystä ja saada aikaan muutosta (Pernaa 2013). Kehittämiskutkimus tapahtuu sykleittän. Jokaisen syklin jälkeen anysoidaan tilanne, jossa kehitystä verrataan edelliseen sykliin (Pernaa 2013). Tässä kehittämistutkimuksessa kehityssyklejä on vain yksi, mikä koostuu alasykleittäin rakentuvasta DevOps-elinkaaresta.

Kehittämistutkimuksessa yhdistellään erilaisia kvalitatiivisia ja kvantitatiivisia tutkimusmenetelmiä sekä lisäksi kehittämistutkimukseen kuuluu myös ratkaisun löytäminen ongelmaan. (Kananen 2015, 39–42.) Tämä tutkimus toteutetaan kvalitatiivisena, eli laadullisena tutkimuksena, vertaillen löydettyjä tuotteita annettujen kriteerien mukaan. Tuotteiden paremmuutta yrityskulttuuripohjaiseen menetelmään, eli DevOpsiin, ei voi määritellä määrällisen tutkimuksen pohjalta.

2.4 Tiedonhaku

Tässä tutkimuksessa etsitään tietoa pääsääntöisesti internetistä, sillä painettu media vanhenee suhteellisen nopeasti tietoteknisellä alalla. Lähteitä etsitään muun muassa toisien opinnäytetöiden lähteistä, YouTubesta, sekä tunnettujen sovellusvalmistajien dokumentaatio-sivustoilta. Tiedonlähteinä käytetään myös Janet/Finna-verkkokirjastoa hakusanoilla DevOps*, DevOps* AND Game dev*, pelikehitys AND devops.

Koska DevOps on myös suurelta osalta työkulttuuriin liittyvää, on vaikea antaa suurille julkaisijoille enemmän painoarvoa verrattuna pienempiin, koska kulttuurillisesti heidän näkemyksensä asiasta on yhtä pätevää.

3 Teoriatausta

Tämän luvun tarkoituksena on perehtyä lyhyesti DevOpsiin liittyvään teoriataustaan. Lisäksi perehdytään testivetoiseen kehittämiseen luvussa 3.5, mikä on yksi tämän kehittämistyön asetetuista kriteereistä. Viimeiseksi luvussa 3.6 perehdytään yksikkötestaukseen, mikä sitoo DevOpsin testivetoiseen kehittämiseen.

3.1 DevOps lyhyesti

Mitä DevOps on?

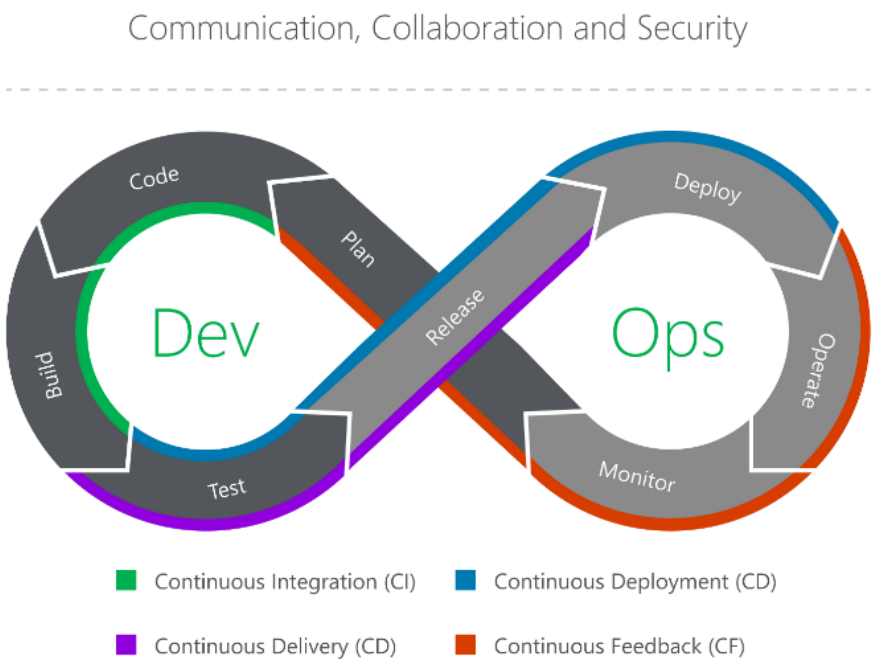
DevOpsista on tehty muun muassa artikkeleita, tutkimuksia, seminaareja, opinnäytetöitä ja opetusvideoita, joissa on pyritty kuvaamaan mitä DevOps on. Esimerkiksi opinnäytetyössään Forsell (2020, 9) toteaa: *”DevOpsin määrittelyn kannalta laajinta suosiota nauttii määritelmä, jossa DevOps on organisaatiokulttuurin muutoksena, jossa kommunikaatiota haittaavia raja-aitoja kaadetaan.”* Atlassianin nettisivuilla on määritelty myös, mitä DevOps heidän mielestään on:*” DevOps is more than just development and operations teams working together. It’s more than tools and practices. DevOps is a mindset, a cultural shift, where teams adopt new ways of working.*” (DevOps: Breaking the development-operations barrier 2022). Kuten edellä mainituista väitöksistä käy ilmi, DevOps on yleisesti toimintamalli, jossa yhdistetään kulttuurifilosofia, käytänteet sekä työkalut, jotta sitä käyttävä organisaatio voi toimittaa sovelluksia ja palveluita nopealla tahdilla.

Historia

DevOps kehitettiin halusta yhdistää ohjelmistokehitys ja palveluntarjonta, koska niiden koettiin taistelevan toisiaan vastaan, aiheuttaen heikkoja tuloksia (Abildskov 2021; Buchanan n.d). Ideaalitilanteessa palvelua tai sovellusta kehittävä tiimi ymmärtää asiakkaan tarpeen ja yrityksen tiimien välistä yhteistyötä helpotetaan (DevOps: Breaking the development-operations barrier 2022).

3.2 DevOps-elinkaari ja vaiheet

DevOps-elinkaari koostuu useista eri vaiheista, jotka ovat havaittavissa kuviossa 1. Kuviosta saa sellaisen käsityksen, että DevOps olisi suoraviivainen elinkaari. Todellisuudessa DevOpsin elinkaaren osaset ovat kaikki riippuvaisia toisistaan ja yleensä käynnissä yhtäaikaisesti. Lisäksi DevOpsissa käytettävät sovellukset voivat hoitaa useamman vaiheen kuvion yksi vaiheista, kuten esimerkiksi Jenkins-sovelluksella voidaan orkestroida koko CI/CD-automaatioputki. Unityä voidaan myös käyttää useissa vaiheissa.



Kuvio 1. DevOps-elinkaari ja vaiheet (Pennington 2019)

3.2.1 Jatkuva tuotanto

Jatkuva tuotanto (engl. Continuous Development) pitää sisällään tyypillisesti suunnittelu- ja koodausvaiheen (Alexander 2018). Tässä vaiheessa esimerkiksi suunnitellaan työt käyttäen sovelluksia kuten JIRA, Trello tai ClickUp. Koodia luodaan erilaisilla työvälineillä kuten Visual Studio, IntelliJ, PyCharm. Versionhallinnan käyttö on yleensä ensimmäinen edellytys DevOpsille, yleisin näistä on nykyään git-versionhallinta. Koodia säilytetään yleensä pakettivarastoissa kuten GitHub, GitLab tai Bitbucket.

3.2.2 Jatkuva integraatio

Jatkuva integraatio (engl. Continuous Integration, CI) on työnkulku, jossa sovellusta kehittävät työntekijät työskentelevät yhden ajantasaisen version pohjalta. Jatkuvan integraation tavoitteena on vähentää yhteensopivuusongelmia projektissa, minkä parissa työskentelee mahdollisesti useita kehittäjiä. Yleensä kehittäjät integroivat työnsä päivittäin, eli käytännössä yhdistävät koodinsa pää-pakettivarastoon. (Fowler 2006.) Yleisempiä jatkuvan integraation työkaluja on Jenkins, Circle CI, TeamCity ja Travis CI.

3.2.3 Paketointi

Tässä vaiheessa rakennetaan lähdekoodista uusi paketti (engl. Build). Paketointi automaatioputkessa aktivoituu yleensä koodimuutoksen jälkeen. Jos käytössä on jatkuva testaaminen, paketin luomisen yhteydessä voidaan myös tehdä halutut testit automaattisesti. Mikäli yksikin näistä testeistä epäonnistuu, koko paketointi epäonnistuu. (Pennington 2022.)

Paketti voi olla esimerkiksi niin kutsuttu ”Nightly Build”, joka paketoidaan joka päivä aina automaattisesti nimensä mukaisesti öisin. Tämän tyyppiset paketit ovat yleensä testaus- tai demotarkoituksiin, ja niissä ovat pienimmätkin muutokset.

3.2.4 Jatkuva testaaminen

Jatkuva testaaminen koostuu yleensä paketointi- ja testausvaiheista (Alexander 2018). Jatkuvan testaamisen tavoitteena on varmistaa tuotetun koodin toimivuus eri vaiheissa DevOpsin elinkaaressa (Fowler 2006; What is DevOps? 2022). Jotta päästään etenemään seuraavaan vaiheeseen, tulee kaikkien testien olla onnistuneita. Testaamista on suotavaa suorittaa useammassa vaiheessa DevOps-putkea (Alexander 2018). Wickramasinghe (2021) toteaa, että jatkuvaa testaamista on monentyyppisiä ja yleisimpiä näistä on yksikkö-, saavutettavuus-, funktionaalinen-, hyväksyntä- ja integraatiotestaus.

Yksi jatkuvan kehittämisen kehityskäytänteistä, mikä tukee jatkuvaa testausta, on Test Driven Development (TDD), jonka avulla yksikkötestit (engl. Unit Tests) kirjoitetaan ennen varsinaisen koodin kirjoittamista. (Fowler 2006; Hamilton 2022a; Alexander 2018.). TDD:stä kerrotaan lisää luvussa 3.5.

3.2.5 Jatkuva julkaisu

Jatkuva julkaisu (engl. Continuous Deployment) mahdollistaa sovellusten jakamisen halutuille kohdealustoille automaattisesti ilman ihmispäätöstä. Tämä on kehittyneempi käytänne Continous Deliverysta, jossa viimeisen päätöksen tuotantoon viemisestä vastaa yleensä ihminen. (Pittet, n.d.)

3.2.6 Jatkuva valvonta ja palaute

Valvontaa (engl. Monitoring) on suoritettava, jotta kehittäjät ja palveluntarjoaja saavat kriittistä tietoa telemetrian avulla mahdollisista virhetilanteista, tai esimerkiksi siitä, miten kehitettävä tuote käyttäytyy eri tuotantoympäristöissä. Tämä tieto voi olla ei-teknistä tietoa tai esimerkiksi jotain mitä sovellus lähettää toimintojensa yhteydessä. (Kim, Humble, Debois, Willis & Allspaw 2016, luvut 14 ja 15; Raza 2021).

Jokaisen iteraation jälkeen kehitystiimin tulee saada palautetta (engl. Feedback), jotta he voivat kehittää prosesseja ja tuoda uusia ideoita seuraavaan iteraatioon (DevOps: Breaking the development-operations barrier 2020). Jatkuvan palautteen ansiosta kehitystiimit kykenevät tuottamaan turvallisempaa käyttöönottoa tuotantoon (Kim ym. 2016, luku 16). On tärkeää muistaa, että jatkuva valvonta ja palaute on sellaista toimintaa, mitä tapahtuu koko DevOps-elinkaaren aikana.

3.3 DevOps-työkalut

DevOpsin elinkaaressa käytetään laajasti erilaisia työkaluja ja osalla työkaluista voidaan suorittaa useita DevOpsin vaiheita. Taulukossa 1 on löytämiäni DevOps-elinkaaren eri vaiheiden tunnettuja ilmaisia DevOps-työkaluja ja mihin ne sijoittuisivat DevOps-elinkaaressa. Ilmaisia sovelluksia on valtavasti, joista osa täysin ilmaisia, osa rajoitettuna pienille tiimeille tai rajoitetuilla toiminnallisuuksilla.

Taulukko 1. DevOps-työkalut

Työkalu	Plan	Code	Build	Test	Release	Deploy	Operate	Monitor
Trello	x						x	x
Draw.io	x							
GitHub	x	x	x				x	x
Taiga	x							x
GitHub Actions			x	x	x	x		x
GameCI			x	x	x	x		
Jenkins			x	x	x	x		x
Unity		x	x	x				x
SonarQube Community Edition				x				x
Butler (itch.io)					x	x		
File browser	x	x						x

Grafana								x
DevLake								x
Docker						x		x
Portainer CE						x		x
Yacht						x		x
Discord	x						x	x
Rocket.Chat	x						x	x
Mattermost	x						x	x

Varsinkin erilaisia ChatOps-sovelluksia kuten Rocket.Chat tai Mattermost on saatavilla paljon. Projektinhallinta sovelluksissa löytyy mielenkiintoisia ilmaisia sovelluksia kuten Taiga ja Trello. Pelkästään seurantaan sopivia sovelluksia löytyi oikeastaan vain kaksi, Grafana+Prometheus ja DevLake.

3.4 Itch.io -julkaisualusta

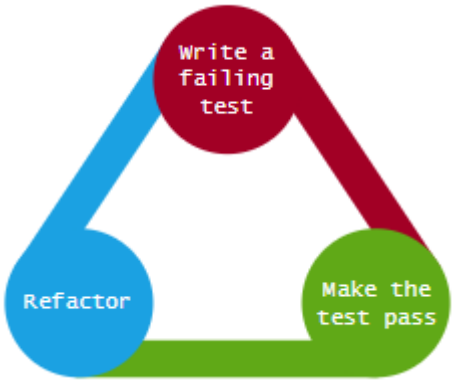
Itch.io on avoin markkinapaikka sisällöntuottajille. Itch.io sivustolla voi siis myös julkaista pelejä joko myyntiin tai ilmaiseen jakeluun. Alusta tukee myös ”maksa mitä haluat”- periaatetta, millä asiakkaat voivat maksaa tai lahjoittaa haluamansa summan, tämä summa voidaan myös jakaa kehittäjän ja julkaisualustan välille. (About Itch.io 2022.)

3.5 Testivetoinen kehitys

Testivetoinen kehitys (engl. test-driven development, TDD) on kehitystekniikka, jossa ohjelmointia tuetaan tekemällä ensin yksikkötestit, ja vasta tämän jälkeen tehdään tarvittava koodi (Hamilton 2022a; Koutifaris 2018). Lisäksi Hamilton (2022a) toteaa, että myös hyväksyntätesti (engl. Acceptance Test) toimii syötteenä TDD:lle. Kun TDD:tä käytetään, ajan myötä rakentuu kattava valikoima automaattisia yksikkötestejä, joita voidaan ajaa milloin vain (Hamilton 2022a). Tämä siis tukee myös DevOpsin vaihetta: jatkuva testaus. Yksikkötestaukseen perehdytään tarkemmin luvussa 3.6.

Hamilton (2022a) ja Koutifaris (2018) kuvailevat TDD:tä hieman erilaisesti toisistaan, mutta he kuvailevat silti samaa asiaa. Koutifariksen (2018) kuvaamassa TDD:ssä käytetään niin kutsuttua ”red/green/refactor”, eli RGR-sykliä. Kuviossa 2 havainnollistetaan RGR-sykli. Hamilton (2022) puolestaan kuvailee TDD:tä myös kolmen vaiheen mallina, mutta hieman laajemmin erillisvaihein ja eri

termeillä. Tutkimuksessa tullaan perehtymään tarkemmin Koutifariksen malliin sen selkeyden takia.



Kuvio 2. TDD:n red/green/refactor eli RGR sykli.

Ensimmäisessä vaiheessa Koutifariksen (2018) mukaan asetutaan asiakkaan saappaisiin ja suunnitellaan yksikkötesti, jolla on merkitystä. On erityisen tärkeää muistaa, että tässä vaiheessa ei tule miettiä, miten kirjoitetaan tuotantokoodia. Testejä kirjoittaessa mietitään, mitä tarvitaan nyt, eikä mitä saatetaan tarvita. Koska testi kirjoitetaan ensin, tulee testi epäonnistumaan.

Toisessa vaiheessa Koutifariksen (2018) sekä myös Hamiltonin (2022a) mukaan kirjoitetaan vain minimimäärä koodia, jolla saadaan ensimmäisen vaiheen epäonnistunut testi onnistumaan. Koodi voi olla siis tässä vaiheessa vielä rujoa tai laadullisesti huonompaa. Ylimääräisiä funktioita tai algoritmeja, mitkä eivät liity suoraan yksikkötestin läpivientiin, tulee välttää, sillä ne ovat ylisuorittamista, eivätkä ole enää osa TDD:tä.

Kolmannessa vaiheessa tarkoituksena on refaktoroida tuotantokoodia, rikkomatta testejä (Koutifaris 2018). Sekä Koutifaris (2018) että Hamilton (2022a) ovat samaa mieltä siitä, että refaktorointivaiheen yksi tärkeimmistä tavoitteista on vähentää koodissa itsensä toistamista (engl. Code duplication).

3.6 Yksikkötestaus

Yksikkötestaus on lasilaatikko, eli white-box -testaamista, jossa kehittäjälle on pääsy koko koodiin (Hamilton 2022b). Yksikkötestausta voidaan käyttää osana TDD:tä ja DevOpsia. Kuviossa 3 esitetään Microsoftin Visual studio -dokumentaation esimerkki tyypillisestä yksikkötestin rakenteesta.

```
[TestMethod]
[Timeout(2000)] // Optional timeout in milliseconds
public void Withdraw_ValidAmount_ChangesBalance()
{
    // arrange
    double currentBalance = 10.0;
    double withdrawal = 1.0;
    double expected = 9.0;
    var account = new CheckingAccount("JohnDoe", currentBalance);

    // act
    account.Withdraw(withdrawal);

    // assert
    Assert.AreEqual(expected, account.Balance);
}
```

Kuvio 3. Esimerkki yksikkötestistä (Unit test basics 2022)

Yhtenä yksikkötestien tarkoituksena on testata luokkien metodien toimivuutta. Yleisesti yksikkötestit rakennetaan AAA:n periaatteella eli Arrange, Act, Assert. On tärkeää miettiä miten yksikkötestejä rakentaa, jotta testillä saadaan testattua halutulla tavalla. (Kralj 2022; Unit test basics 2022.) Yksikkötestit tulisi myös nimetä yleisen käytänteen mukaisesti helposti luettavaan muotoon. Yksi yleinen nimeämistapa on nimetä testit kolmiosaisesti **TestattavanMetodinNimi_testiSkenaario_oletettuTulos**. (Lynch 2018; Unit test best practices 2021.) Seuraavissa kappaleissa perehdymme AAA:n jokaiseen vaiheeseen.

Arrange-vaiheessa valmistellaan metodi, mitä halutaan testata, luomalla yleensä uusi instanssi testattavana olevasta luokasta. Tässä vaiheessa yleensä myös asetetaan muuttujille arvot. (Kralj 2022.) Yksikkötestauksessa tulisi vältellä käytettävän nk. taikalukuja (engl. Magic numbers), sillä ne eivät myös kerro mikä ja miksi kyseinen luku on (Lynch 2018).

Act-vaiheessa tehdään testin toimenpiteet. Tämän vaihe pitäisi yleensä olla vain yhden rivin pituinen ja siinä ajetaan Arrange-vaiheessa kutsutun luokan metodi syöttämällä sille arvoja, joita voi

olla myös määritetty Arrange-vaiheessa. Saatu tulos tallennetaan muuttujaan. (Kralj 2022.) Myös tässä vaiheessa tulee myös välttää taikalukuja, jotta tiedetään mitä lukuja testataan.

Assert-vaiheessa vertaillaan oletettua tulosta Act-vaiheessa saatuun tulokseen. Yleisesti tulosta verrataan esimerkiksi muodoissa **Assert.AreEqual(x,y)** joissa arvojen **x** ja **y** tulee olla sama, jolloin testi menee lävitse. Toisaalta voidaan käyttää myös **Assert.AreNotEqual(x,y)**, jolloin testi menee lävitse vain, jos **x** ja **y** ovat eri arvoja. (Kralj 2022; Unit test basics 2022.)

4 Tutkimuksen toteutus

Tässä luvussa käydään lävitse peliprojektin pystytys, mikä tapahtui iteratiivisesti lisäten ja testaten uusia osia. Tässä opinnäytetyössä ei oteta kantaa esimerkiksi siihen, mitä koodieditoria kehittäjät käyttävät tai miten esimerkiksi yksittäisiä sovelluksia asennetaan, pl. oleelliset sovellukset, mitkä ovat automaatioputkessa keskeisissä osissa.

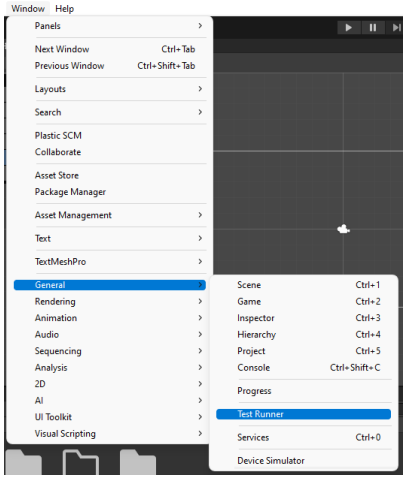
4.1 Demo-peliprojekti

Demo-peliprojektin sisältö ei ole tässä tutkimuksessa keskiössä, joten projektina käytän vain uusien projektien mukana tulevaa SampleSceneä. Tähän sceneen laitetaan vain minimimäärä tavaraa, jotta voidaan toteuttaa vaatimusten mukainen TTD:n käyttö ja todeta automaatioputken toimivan.

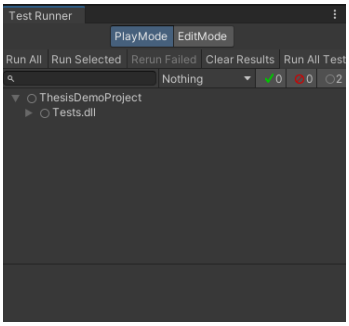
Projekti aloitetaan luomalla uusi yksityinen pakettivarasto GitHub-sivustolle, jonka jälkeen luodaan Unityssa uusi 2D-projekti. Tämän jälkeen ajetaan komentokehotetta käyttäen ”git init”-komento peliprojektin kansiossa. Seuraavaksi lisätään projektikansion juureen .gitignore-tiedoston, jonka tarkoituksena on poissulkea tiettyjä tiedostoja, tiedostomuotoja sekä kansioita päätymästä GitHub -pakettivarastoon. Viimeisenä vaiheena lisätään GitHub-pakettivaraston osoite projektin git config -tiedostoon ja työnnetään (engl. push) uudet muutokset GitHub-pakettivarastoon.

Kuvio 4. Projektitiedostot GitHubissa

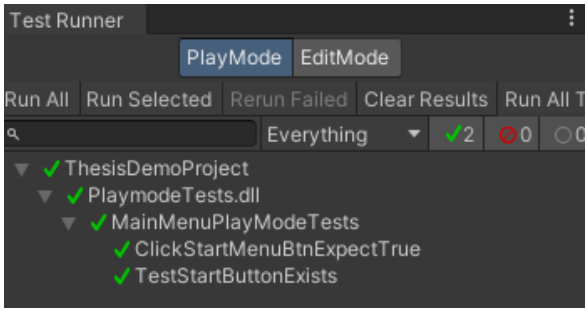
Jotta voidaan testata että Unityn yksikkötestit toimivat, luodaan kaksi yksikkötestiä SampleSceneen. Näiden testien on tarkoitus testata, löytyykö scenestä nappi nimeltä ”start” ja että toimiiko se. Yksikkötestit löytyvät liitteestä kaksi. PlayMode-testit suoritetaan, kun peli on päällä editorissa. Näitä testejä voi myös suorittaa komentokehotteelta Linux-serveriltä, kuten tässä putkessa on tarkoituksena. Kuviosta 7 näkyy että PlayMode-testit ovat menneet läpi.



Kuvio 5. Test Runnerin lisääminen projektiin



Kuvio 6. Unity Test Runner lisättynä projektiin

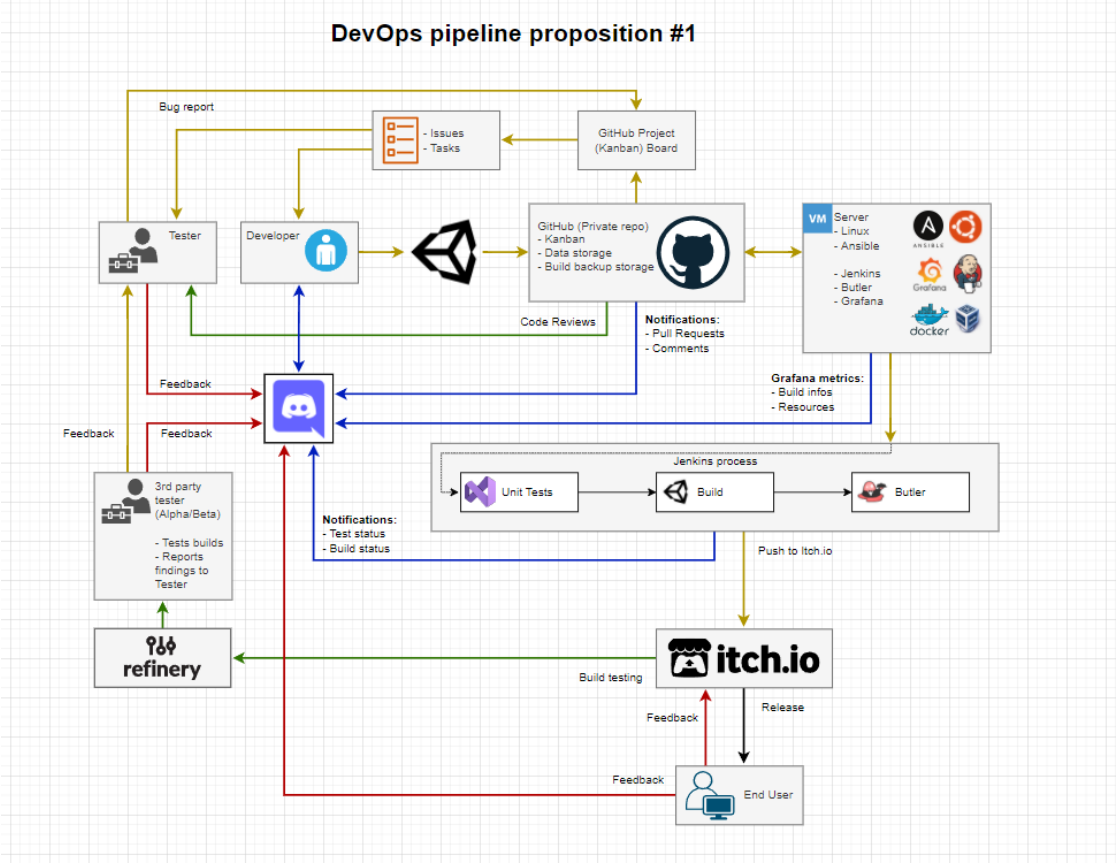


Kuvio 7. SampleSceneen tehdyt PlayMode-testit

4.2 DevOps-elinkaaren rakentaminen

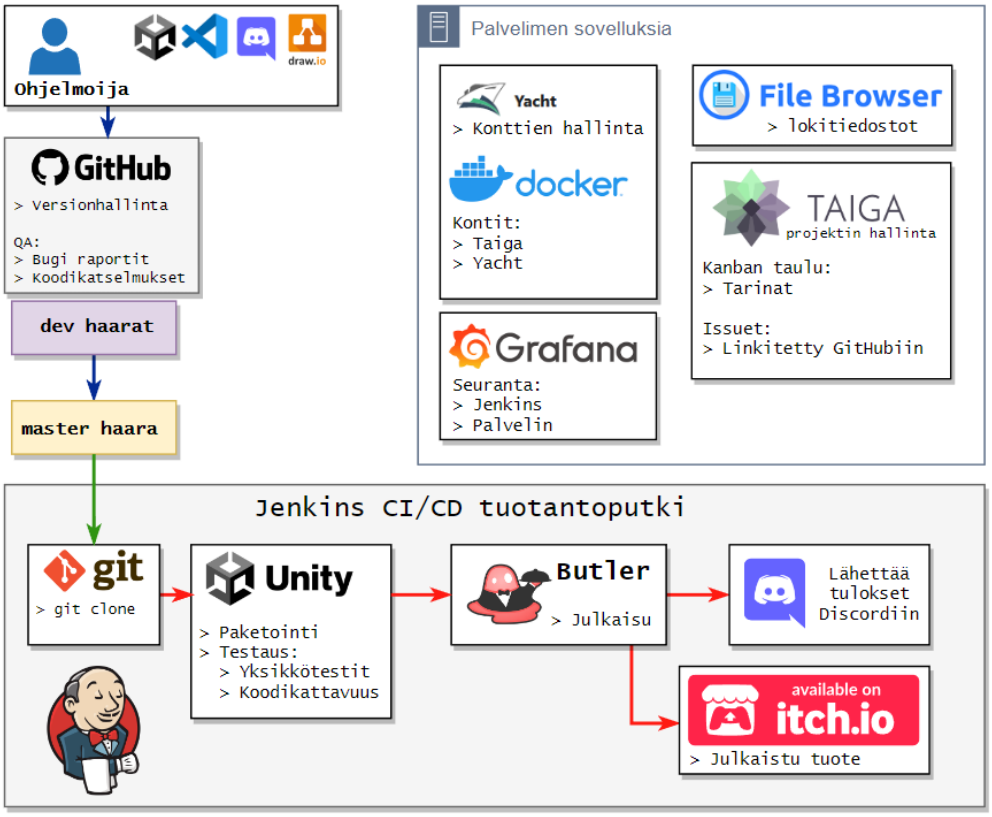
Tämä elinkaari perustuu ajatukseen, jossa käytössä on Linux-palvelin, joka pyörii joko fyysisenä tai virtuaalisena. Tässä ratkaisumallissa elinkaaren keskiössä on Jenkins, avoimen lähdekoodin sovellus, joka mahdollistaa CI/CD-automaatioputkien luomisen sekä mahdollistaa seurannan.

Kun ensimmäiseksi aloitin tekemään suunnitelmaa (ks. kuvio 8) ja rakentamaan automaatioputkea, päätin kokeilla tehdä kaiken virtuaalipalvelimeen. Valitettavasti kesken automaatioputken rakentamista törmäsin Unityn kanssa seinään, nimittäin Unity ei suostunut toimimaan oikein VirtualBoxissa ajettavassa Linux-käyttöjärjestelmässä, lukuisista uudelleenasenteluista huolimatta.



Kuvio 8. Alkuperäinen DevOps-elinkaari suunnitelma

Lopulta useiden suunnitelman muutosten jälkeen tutkimuksessa päädyin käyttämään vanhaa tietokonetta, johon on asennettu Ubuntu 22.04 Linux -käyttöjärjestelmä. Viimeisen version suunnitelmassa (ks. kuvio 9) kuvataan vain ohjelmoijan osuutta ja on pyritty kertomaan vain mitä sovelluksia palvelimessa on, sekä selitetään Jenkins-automaatio/tuotantoputki.



Kuvio 9. Viimeistelty DevOps-elinkaarisuunnitelma

4.2.1 Jenkins

Jenkins valikoitui automaatiotyökaluksi, koska se on avoimen lähdekoodin sovellus, jossa on laajasti saatavilla erilaisia lisäosia. Työkalun nauttiva laajamittainen suosio vaikutti myös valintaan, koska siihen on saatavilla paljon tukea. Käytän automaatioputken luomiseen Jenkinsfile-tiedostoa, jolloin myös sen saa versionhallinnan alle. Valmis Jenkinsfile-tiedosto löytyy liitteistä.

Jotta Jenkins voidaan asentaa, tarvitsee se Javan toimiakseen. Javan saa asennettua Linuxiin komennolla: ”**sudo apt install openjdk-11-jdk**”. Javan asentamisen jälkeen testataan, että se toimii.

```
kimmokissa@kimmokissa-Legion-Y540-15IRH:~$ java --version
openjdk 11.0.16 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
```

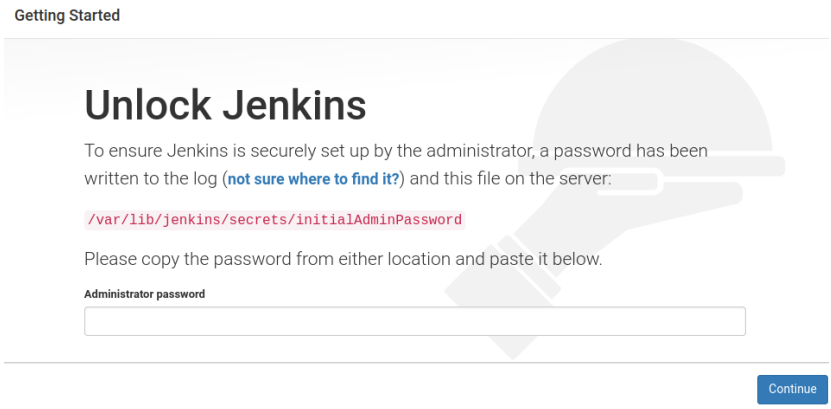
Kuvio 10. Jenkins-asennus: OpenJDK11 toimii

Seuraavaksi asennetaan Jenkins LTS -versio. Koska Jenkinsistä ei löydy suoraan Ubuntun omista pakettivarastoista, lisätään jenkins.io stable -pakettivarasto koneelle. Tämän jälkeen päivitetään kaikki pakettivarastot ja asennetaan Jenkins. Asentamisen jälkeen palvelu käynnistyy automaattisesti. Kuviossa 11 on havainnollistettu asennusprosessissa käytettävät komennot.

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Kuvio 11. Jenkinsin asentaminen Linuxilla (Jenkins User handbook 2022)

Kun Jenkins on asennettu ja palvelu on onnistuneesti käynnistynyt, mennään internetselaimella osoitteeseen localhost:8080, ja kuten kuviossa 12 ilmenee, asennus on onnistunut.

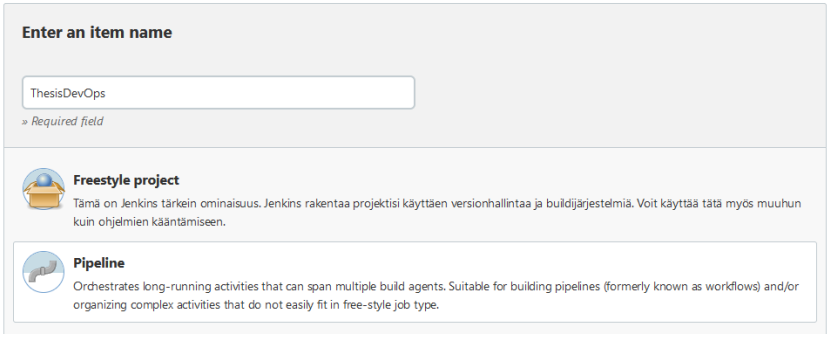


Kuvio 12. Onnistunut Jenkins-asennus

Koska Jenkins tarvitsee oikeudet ajaa kaikkia sovelluksia mitä sille annetaan käyttöön, on suositeltavaa muokata jenkins-käyttäjäprofiilia. Jenkins-käyttäjäprofiilille vaihdettiin asetus, jossa järjestelmä ei kysele jatkuvasti tunnuksen salasanaa (ks. kuvio 13). Uusi projekti luotiin käyttämällä Pipeline-valintaa, tämä mahdollistaa Jenkinsfilen käyttämisen.

```
cd /etc/
sudo visudo sudoers
# Lisää seuraava rivi sudoers tiedostoon.
jenkins ALL=(ALL) NOPASSWD: ALL
```

Kuvio 13. jenkins-käyttäjätilin muokkaaminen



Kuvio 14. Uusi Pipeline projekti Jenkinsissä

4.2.2 Testaus: Unity

Yksikkötestit

Kappaleessa 4.1 lisättiin demopeliprojektiin yksikkötestejä. Näitä yksikkötestejä voidaan nyt ajaa Jenkinsin avulla komentokehotteessa kuvion 15 mukaisesti. Näistä **-batchmode** on pakollinen automaatioratkaisuissa.

```
stage("Unit Tests"){
            // Ensin tehdään playmode testit.
            steps {
                sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -
serial ${UNITY_SERIAL} -batchmode -projectPath ${UNITY_PROJECT_PATH}thesisdemoproject -
runTests -testResults ${UNITY_TEST_LOG_PATH}/PlaymodeTestResults.ver-${BUILD_VER}.xml -logFile
${BUILD_LOG_PATH}/TestLog-ver-${BUILD_VER}.txt -testPlatform PlayMode -nographics'
            }
        }
```

Kuvio 15. Yksikkötestit Jenkinsfilessä

Parametreista **-username** ja **-password** ovat tunnukset, joilla kirjaudut unityn palveluille. **-serial** on unity plus- tai pro version -tuotekoodi. Huom. pelkästään tämä rajoittaa ylipäätänsä sitä, kuka voi käyttää automaatiota, koska Unity ei tällä hetkellä anna tuotekoodeja personal-tason tunnuksille. **-batchmode** mahdollistaa Unityn ajamisen ilman ihmiskosketusta. **-nographics** mahdollistaa Unityn ajamisen ilman grafiikkaohjainta. **-projectPath** parametriin laitetaan se osoite, mistä Unity-projekti löytyy.

Kun on tarkoitus ajaa yksikkötestejä, käytetään lisäksi myös seuraavia parametreja, joilla ajetaan testit: **-runTests** ja valitaan testialusta: ”**-testPlatform PlayMode**”. Parametri ”**-testResults /sijainti/tuloksiin.xml**” sijoittaa yksikkötestien tulokset haluttuun kansioon.

Koodikattavuus

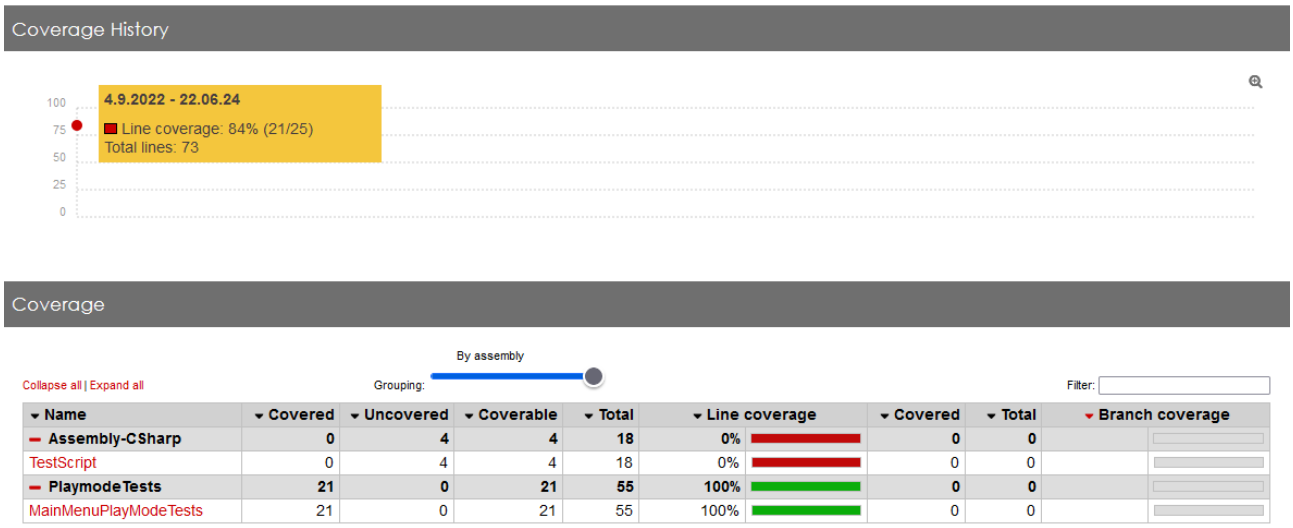
Jotta voitaisiin olla tietoisia siitä, kuinka laajasti projektissa yksikkötestit käyvät lävitse koodia, asennetaan peliprojektiin myös Unityn package managerista löytyvä Code Coverage -työkalu. Tällä

työkalulla on myös mahdollista seurata, kuinka paljon koodista on ajon aikana saavuttamatonta (Unity Code Coverage 2022).

```
stage("Code Coverage"){
    // Code coverage
    steps {
        sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -serial ${UNITY_SERIAL} -batchmode -
projectPath ${UNITY_PROJECT_PATH}thesisdemoproject -debugCodeOptimization -enableCodeCoverage -coverageResultsPath
/var/www/tarinataikomo/buildreports/CodeCoverage/ -coverageOptions
"generateHtmlReport;generateBadgeReport;assemblyFilters:+my.assembly.*" -nographics -quit'
        sh 'sudo cp -R /media/kimmokissa/unity/Projects/thesisdemoproject/CodeCoverage/ /var/www/tarinataikomo/buildreports/'
        sh 'sudo chmod 777 /var/www/tarinataikomo/buildreports/CodeCoverage/'
    }
}
```

Kuvio 16. Koodikattavuus Jenkinsfilessä

Kun koodikattavuusvaihe on mennyt läpi, löytyy luotu raportti intrassa, josta saa visuaalista dataa siitä, miten kattavasti koodia on testattu (ks. kuvio 17).



Kuvio 17. Koodikattavuusraportti

4.2.3 Paketointi: Unity

Paketin (engl. build) teko toimii samalta pohjalta kuin testien teko, eli vähimmäisvaatimukset ja lisätään **”**-buildTarget Win64 -buildWindows64Player /media/kimmokissa/Documents/Builds/ThesisDevOpsProject/game.exe -quit” (ks. kuvio 18). Parametreistä **-buildTarget** kertoo kohdealustan paketille ja **-buildWindows64Player** /polku/peli.exe mihin paketti tehdään.

```
stage("Build"){
    // Onnistuneiden testien jälkeen vasta buildataan.
    steps{
        sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -serial
${UNITY_SERIAL} -batchmode -nographics -buildTarget Win64 -buildWindows64Player
${BUILD_TARGET}  -projectpath ${UNITY_PROJECT_PATH}thesisdemoproject -logFile
${BUILD_LOG_PATH}/BuildLog-ver-${BUILD_VER}.txt -quit'
    }
}
```

Kuvio 18. Paketointi Jenkinsfilessä

Huomattavaa on, että dokumentaatiossa sanotaan **-quit** parametrin piilottavan mahdollisesti virheviestejä (Unity Documentation 2022). Aluksi koitin ajaa paketointia ilman -quit parametria, mutta jostain syystä prosessi jää päälle, eikä Jenkins pääse seuraavaan vaiheeseen, vaikka paketointi olisi onnistunut. Ainoa ratkaisu oli käyttää **-quit** parametria, vaikka se saattaisi hävittää lokitietoja.

4.2.4 Julkaisu: Butler

Butler on Itch.io:n oma sovellus, jolla saadaan ladattua suoraan komentoriviltä halutut tiedostot Itch.io sivustolla sijaitsevaan projektiin. Butlerilla tehdään siis Jenkinsin avulla jatkuvaa toimittamista tuotantoon. Kuviossa 19 on havainnollistettuna ajettavat komennot, jotta Butler saadaan asennettua koneeseen. Asentamisen jälkeen siirsin Butlerin kansioon **/etc/Butler/** sekä testasin ajamalla sen **-V** parametrilla toimiiko Butler oikein. Asennuksen toimivuus on esitetty kuviossa 20.

```
sudo curl -L -o butler.zip https://broth.itch.ovh/butler/linux-amd64/LATEST/archive/default
sudo unzip butler.zip
sudo chmod +x butler
```

Kuvio 19. Butlerin asennus Linuxiin

```
kimmokissa@kimmokissa-Legion-Y540-15IRH:/etc/Butler$ ./butler -V
v15.21.0, built on May 11 2021 @ 21:18:40, ref 6ad656eebbb5b80bf2065644d8bf9e40a15e8276
```

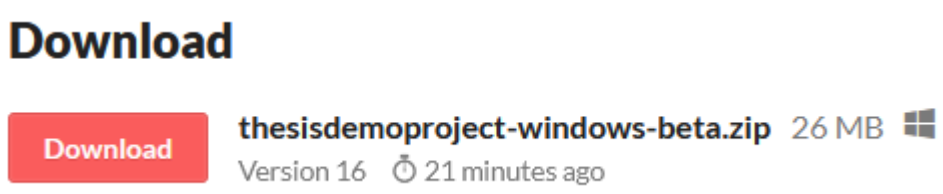
Kuvio 20. Butlerin toimivuus testattuna

Jotta Jenkins voisi käyttää Butleria, pitää Butlerilla suorittaa autentikointi Itch.io-palvelimelle. Ensiksi pitää kirjautua jenkins-käyttäjätunnukselle komennolla **sudo su -s /bin/bash jenkins**, jonka jälkeen komennolla **/etc/Butler/butler login** aloitetaan tunnistautumisprosessi. Komento avaa selaimen, jossa voit kirjautua ja valtuuttaa Butlerin julkaisemaan Itch.io:n sivuille.

Onnistuneen tunnistautumisen jälkeen testataan paketin viemistä Itch.io-sivustolle. Kuviossa 21 näkyvä Jenkinsfilen komento ajaa butler push -komennon, jossa **/media/kimmokissa/Documents/Builds/** kansio on kansio, mistä paketti lähetetään. **kimmokissa/thesisdemoproject** kuvastaa käyttäjänimeä ja projektia Itch.io sivustolla ja **:windows-beta** kertoo julkaisualustan. Julkaistu demo on ladattavissa Itch.iossa, kuten kuviosta 22 huomaa.

```
stage("Deploy to itch.io"){
            steps{
                // Lähettää onnistuneen buildin Itch.ioon.
                sh "/etc/butler/butler push /media/kimmokissa/Documents/Builds/
kimmokissa/thesisdemoproject:windows-beta"
            }
        }
```

Kuvio 21. Julkaisu Jenkinsfilessä

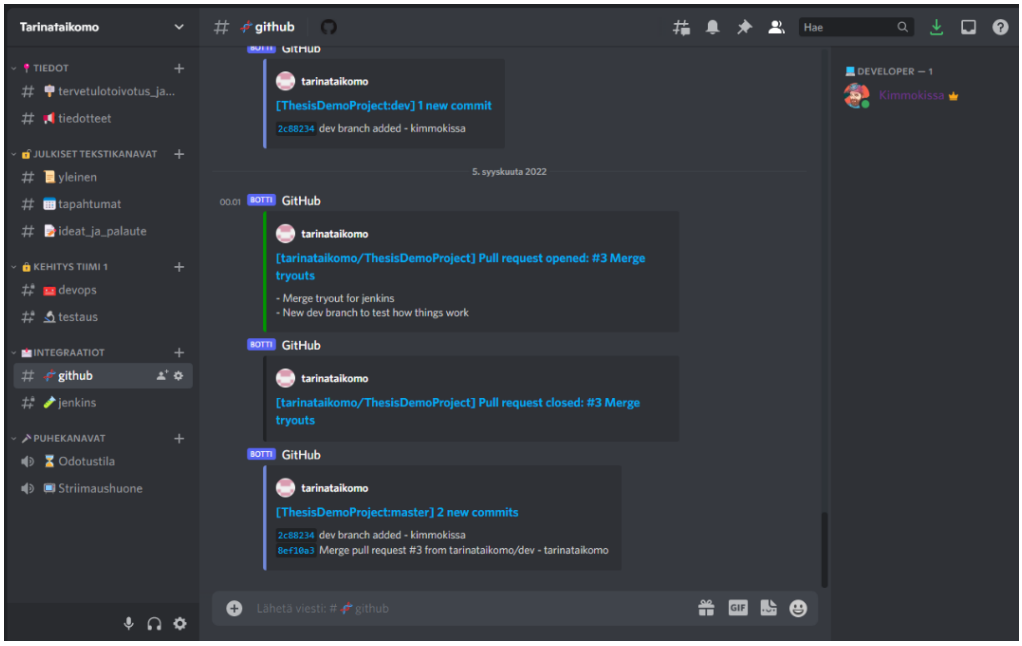


Kuvio 22. Demopeli onnistuneesti ladattavissa sivustolta

4.2.5 ChatOps ja jatkuva seuranta

Discord

Koska ChatOpsin on tarkoitus tuoda ihmiset ja prosessit yhteen läpinäkyvään työnkulkuun, on tärkeä valita siihen parhaat sovellukset, jolla saadaan kriteerit täytettyä. Päädyin ratkaisuun, jossa keskiössä on Discord. Discord täyttää käytännössä kaikki yleiset ChatOps-työkalun määritelmät (Communication & ChatOps Tools 2022).

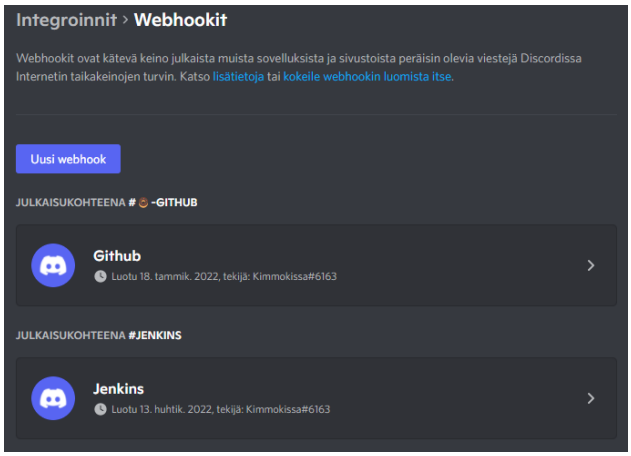


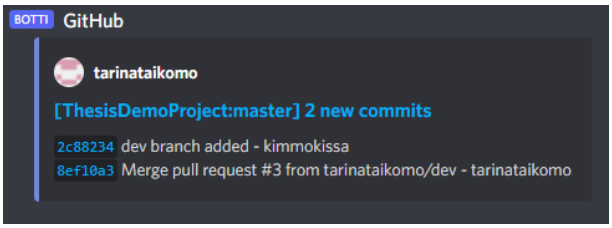
Kuvio 23. Discord ChatOps käytössä

Koitin myös muita ChatOps-työkaluja: Mattermost ja Rocket.Chat, sain asennettua kummatkin sovellukset ongelmitta, mutta en saanut kummassakaan integraatioita Jenkinsiin ja Githubiin toimimaan omassa toimiympäristössäni. Näissä sovelluksissa oli myös rajoitettu määrä integraatioita.

Osana jatkuvaa seurantaa on tärkeä saada palautetta mahdollisimman monesta elinkaaren vaiheesta, joten on loogista, että ohjaamme Discordiin esimeriksi Jenkins-automaatioputken tuloksia sekä GitHub-pakettivarastoon tulleita muutoksia tai kommentteja.

Jotta tämä kaikki olisi mahdollista, yhdistetään Discord Jenkinsiin ja GitHubiin webhookien avulla. Ensin luodaan Discordissa webhook-integraatiot. Kun Discordissa luodaan webhook, generoituu yksityinen web-osoite, millä yhdistetään webhook kohteeseen. Kuviossa 24 näkyvät projektille luodut webhookit. Jotta Jenkins osaisi ilmoittaa automaatioputken statuksen Discordiin, voidaan käyttää liitännäistä Discord Notifier. Jenkinsfilessä tätä liitännäistä voidaan kutsua järkevästi post osiossa, sillä on tärkeää että viesti lähtee vaikka tuotantoputki epäonnistuisi (ks. kuvio 26). Tässä kohtaa tulee käyttää Discordin puolelta saatua webhook-linkkiä.





Kuvio 25. Onnistunut ilmoitus GitHubissa tapahtuneessa muutoksessa

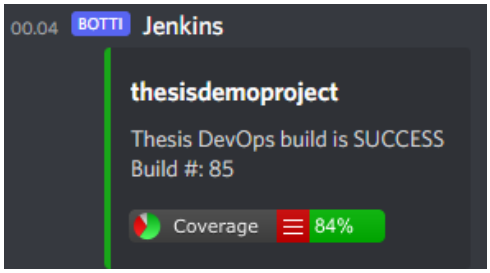
Kuvio 24. Projektin Discord webhookit

```
post {
    always{
        // Ilmoitetaan discordiin onko buildi onnistunut vai ei.
        discordSend (
            description: "Thesis DevOps build is ${currentBuild.currentResult}\nBuild #: ${env.BUILD_NUMBER}\n",
            result: currentBuild.currentResult,
            image: "http://88.113.245.162/buildreports/CodeCoverage/Report/badge_linecoverage.png",
            title: JOB_NAME,
            webhookURL: DISCORD_WEBHOOK
            )
    }
}
```

Kuvio 26. DiscordNotifications Jenkinsfilessä

Kun Jenkins on ajanut lävitse automaatioputken, lähettää se onnistuneesti tiedon, tässä tapauksessa onnistunut, Discord-kanavalle. Lisäsin myös lähetettävään viestiin koodikattavuuden prosenttitason (ks. kuvio 27). Useampien testauksien jälkeen huomasin, että koodikattavuuden badge

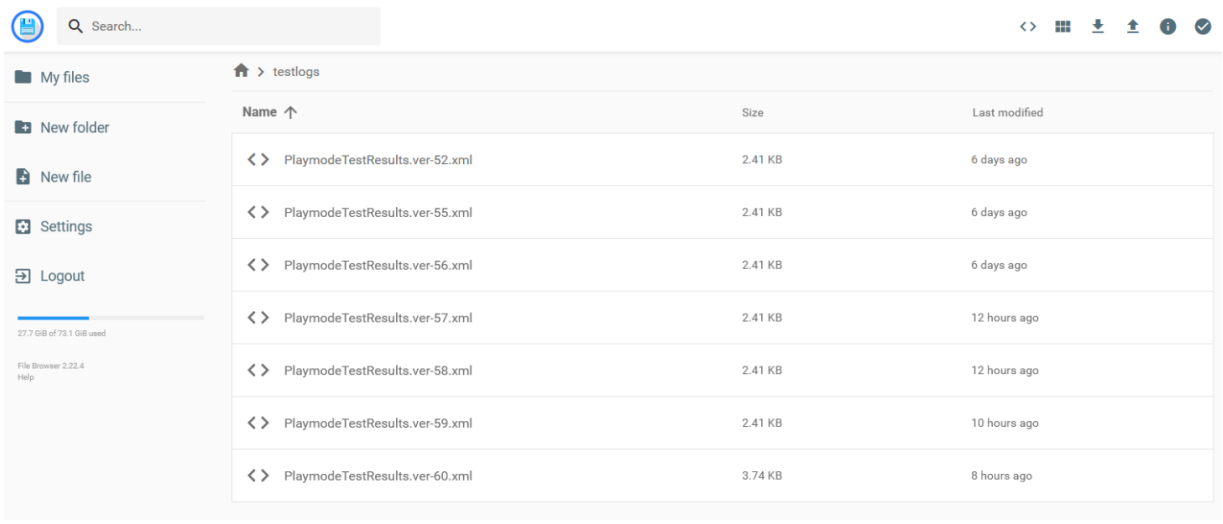
ei päivity esikatselussa aina oikeaksi, syynä tähän on Discordin webhookien upotusviestien välimuisti, jota päivitetään pidemmällä aikavälillä.



Kuvio 27. Ilmoitus onnistuneesta automaatioputken suorituksesta

File Browser

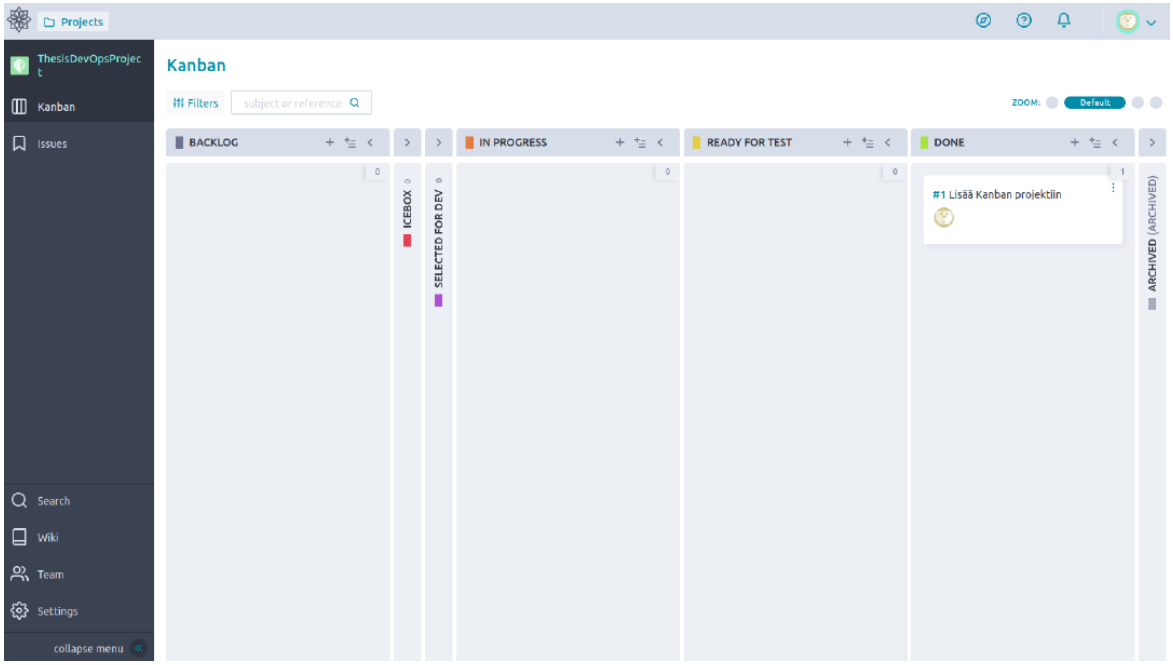
Kehittäjille luotiin myös intrassa toimiva File browser -sivusto, mihin Unity tallentaa lokitiedostoja Unityn testi- ja paketointivaiheista. Sivustoa voi myös hyödyntää kaikkeen muuhunkin sisäiseen tiedostojen jakamiseen. Tällä hetkellä tallennetaan yksikkötestien tulokset .xml muodossa sekä jo aiemmin mainittujen testi- ja paketointivaiheiden lokitiedostot tekstitiedostoina. Kuviossa 28 esimerkkinäkymä yksikkötestit buildiversion mukaan.



Kuvio 28. Testitulokset listattuna

Taiga

Valitsin Taigan projektinhallintajärjestelmäksi, koska siinä on GitHub -integraatio ja sovelluksen ulkoasu oli sopivan selkeä sekä erittäin muokattavissa oleva. Taiga tukee täysin scrum-viitekehystä sekä ongelmien (engl. issues) ja bugien seurantaa. Valitsin Taigasta version, mikä asennetaan docker-konttina. Taiga asennettiin ajamalla komento ”docker-compose up -d” kansiossa, jossa on docker-compose.yml -tiedosto.

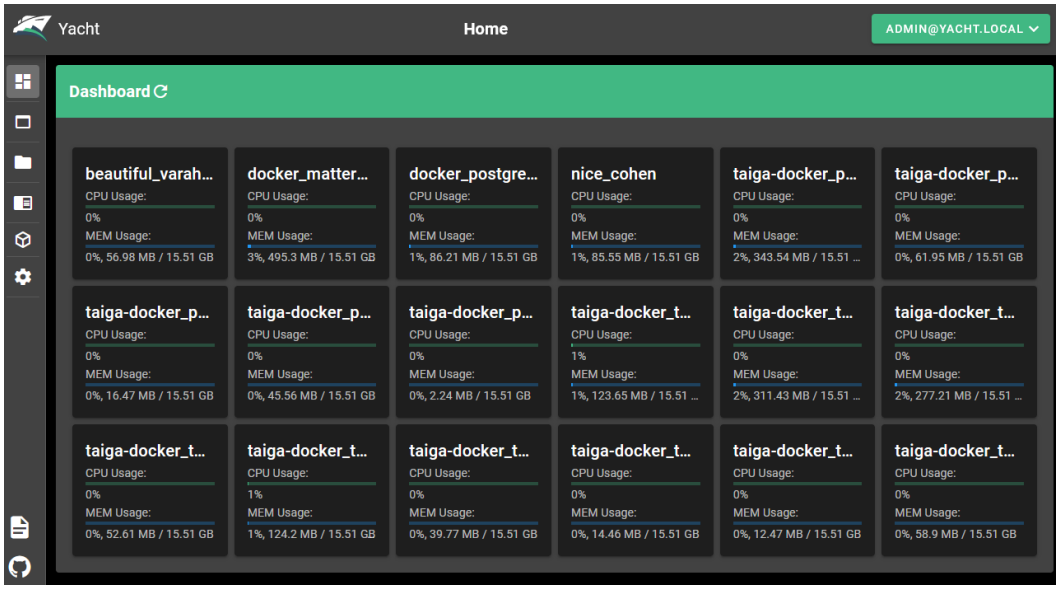


Kuvio 29. Taigan Kanban-taulu

4.2.6 DevOps-elinkaaren seuranta

Yacht

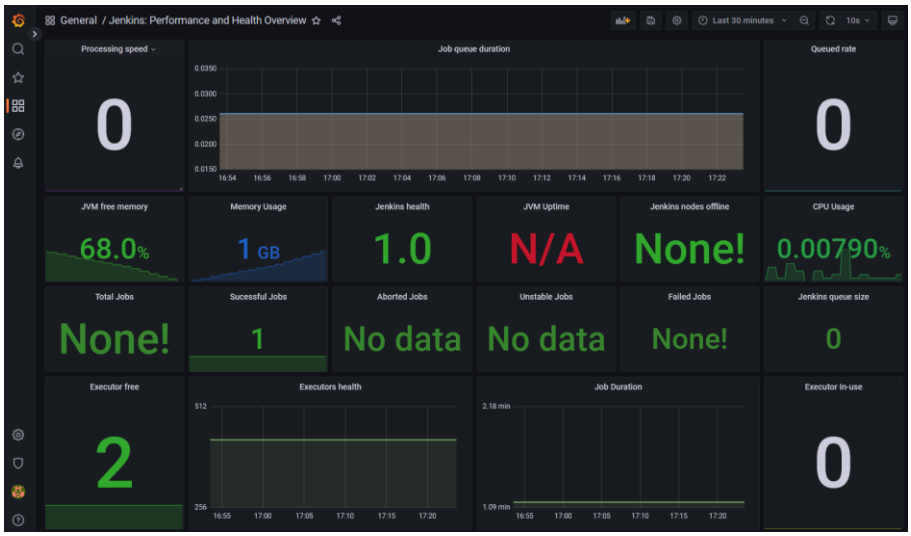
Koska päädyin käyttämään ja kokeilemaan docker-konteista ajettavia sovelluksia, on tarpeellista lisätä työkalupakkiin sovellus, jolla voi hallita docker-kontteja. Yacht näyttää näppärästi yhdellä ruudulla kaikki ajossa olevat kontit ja niitä voidaan tarkastella laajemmin. Yachtilla on mahdollista luoda docker-compose.yml -tiedostoja, joilla saadaan pystytettyä uusia kontteja. Tutkin myös Portainer community editionin käyttämistä, mutta päätin valita Yachtin konttien hallintaan lähinnä ulkonäöllisistä syistä, sillä en kokenut että kumpikaan oli toistaan parempi.



Kuvio 30. Konttien seurantaa Yachtissa

Grafana

DevOps-elinkaaresta vastaavan henkilön - ja miksi ei muutkin, on hyvä nähdä visuaalisesti esitettynä, miten Jenkins suoriutuu ja missä kunnossa ylipäätänsä palvelin on. Grafanassa voi esittää useita erilaisia infograafinäkymiä, esimerkiksi tässä käyttötarkoituksessa, Prometheuksen avulla Jenkins palvelun statistiikkaa.

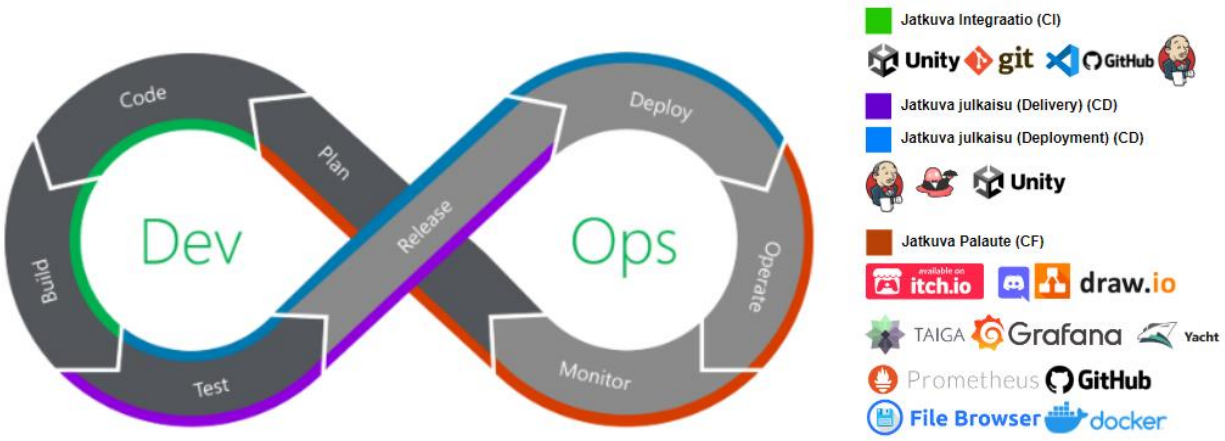


Kuvio 31. Jenkinsistä saatua statistiikkaa Grafanassa

5 Tulokset ja johtopäätökset

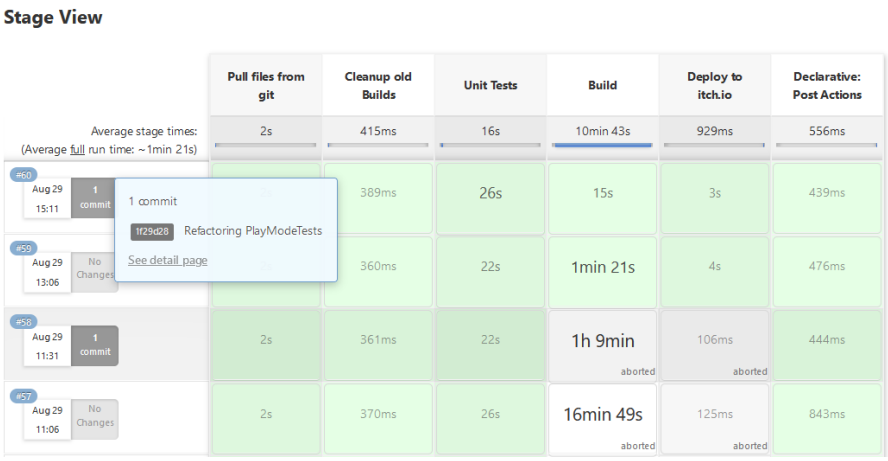
5.1 Tulokset

Lopputuloksena on toimiva DevOps-elinkaari, joka on muokattavissa, helposti ylläpidettävä sekä täyttää kaikki annetut kriteerit. Itse elinkaarta suunnitellessa voi huomata, että työkalupakin sovelluksista suuri osa keskittyy esimerkiksi tiedonkeruuseen ja sen esittämiseen, eikä sinäänsä palvele itse automaatioputkea (ks. kuvio 32). Tämä on hyvä muistutus siitä, että esimerkiksi isommissa organisaatioissa DevOpsia suunnittelee ja ylläpitää yleensä eri henkilö/tiimi, kuin kehittäjät tai tuoteomistajat.



Kuvio 32. DevOps-työkalupakki (Pennington 2019, muokattu)

Automaatioputken osuus toimii täyttäen kaikki kriteerit, eli GitHubin käytön ja julkaisun Itch.io-kauppapaikalle. Automaatioputki tunnistaa GitHubissa päähaaraan tehdyt muutokset, jonka jälkeen palvelin lataa ne ja ajaa yksikkötestit ja koodikattavuustestin, rakentaa paketin ja julkaisee sen Itch.io-sivustolle. Tulokset toimitetaan ChatOps-alustalle. Kuviossa 33 näkyy viimeisin ajo, mikä on onnistunut, sekä historia aikaisemmista ajoista.



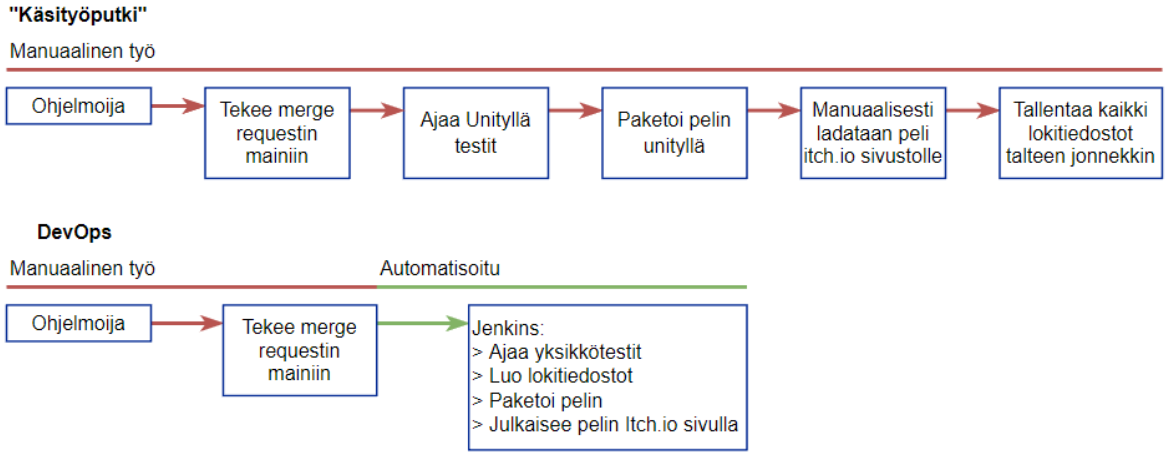
Kuvio 33. Valmis CI/CD-automaatioputki

5.2 Johtopäätökset

DevOps-elinkaarta suunnitellessa ja pystyttämisessä suurin osa ajasta meni ohjelmistojen toimivuuden testaamisen. Tämän jälkeen eniten aikaa meni ChatOpsin ja jatkuvan palautteen tutkimiseen ja testaamiseen. Alun vaikeuksien jälkeen Jenkins CI/CD-automaatioputki oli suhteellisen helppo pystyttää ja saada Unity luomaan paketin GitHubista ladatusta tietovarastosta ja Butlerin viemään se Itch.io-sivustolle.

Testauksen osalta päästiin asetettuun tavoitteeseen, mutta parantamisen varaa on huomattavasti. Luottaminen pelkästään yksikkötestaukseen ja koodikattavuuteen automaatiossa on hieman kyseenalaista.

Näiden havaintojen perusteella voisi tulla äkkiä johtopäätökseen, jossa aloittavalle indie-peliyritykselle ei ehkä ole järkeä miettiä täydellistä DevOps-elinkaarta, vaan ainakin alussa keskittyä enemmän CI/CD-automaatioputken käyttöön. Mutta, etujen havannointi on selkeää (ks. kuvio 34), jokainen vaihe tai minuutti pois käsintehtävästä työstä on kehitystä. DevOps mahdollistaa skaalautuvuuden, eli tulevaisuudessa jo yhden ihmisen lisääminen projektiin nopeuttaa toimenpiteitä, koska mahdollisimman monta vaihetta on automatisoitu. Itse DevOps-elinkaaren suunnittelu on aikaa vievää, mutta pystyttämisen jälkeen seuranta ja ylläpidollinen työ vie vain vähän aikaa.



Kuvio 34. Autiomaatioputken edut vs. käsityö, ohjelmoijan näkökulmasta

Yhtenä kriteereistä oli että DevOps-elinkaaren pitää olla skaalautuva ja helppo ylläpitää. Skaalautuvuus Jenkinsissä on ainakin helppoa, sillä se mahdollistaa uusien automaatioputkien luonnin helposti. Docker ja Yacht mahdollistavat uusien sovellusten lisäämisen ympäristöön helposti selaimen kautta. Jenkinsiin voi myös lisätä uusia agentteja, jos tarvitaan lisää tehoa esimerkiksi paketointiin. Nämä agentit voivat olla mitä tahansa koneita, mitkä voivat ajaa Javaa. Ylläpitoa helpottaviksi elementeiksi valitsin verkkoselaimella käytettävät graafiset käyttöliittymät, sillä niitä on paljon helpompi käyttää ja ymmärtää kuin ottamalla etäyhteys palvelimeen komentokehotteella.

5.2.1 Jatkokehitys

Koska DevOpsille ei ole olemassa yhtä täydellistä ratkaisua, on vaikea sanoa, mikä olisi se paras jatkokehityksen kohde. Sanoisin silti, että tämän DevOps-elinkaaren kannalta olisi hyvä perehtyä testauksen laajuuteen. Olisi mielenkiintoista tutkia, miten esimerkiksi performance benchmarking -testaus sekä UI -automaatiotestaus lisättäisiin elinkaareen. Staattiset koodianalyysit kuten Code Quality ja Code Security olisi myös hyvä lisätä CI/CD-automaatioputkeen. Testaukseen tarvittaisiin myös tapa miten Testaajille generoidaan muutostiedot, jotta he osaavat testata oikeita asioita.

Lähtisin myös tutkimaan Jenkinsin multibranch-pipeline -ominaisuutta tarkemmin ja kokeilisin luoda useita erilaisia automaatioputkia erilaisiin käyttötarkoituksiin, kuten esimerkiksi vaikka konversiotyökalujen automatisointi pikselitaiteelle (engl. pixel art), automaattisen päivityslogin luontiin tai lokalisointiin.

Kesken tutkimusta päädyin käyttämään docker-kontteja Taigaan ja Yacht:iin. Huomatessani, miten helppo niitä on pyörittää, kehittäisin jatkossa DevOps-elinkaaren niin, että kontittaisin kaiken mahdollisen. Tämä helpottaisi DevOps-elinkaaren ylläpidossa, kun sovellukset eivät ole laitteistoriippuvaisia ja sen voisi pystyttää helpommin docker-compose.yml -tiedostolla.

5.3 Vastaukset tutkimuskysymyksiin

5.3.1 Millaisia DevOps-työvälineitä on saatavilla ilmaiseksi?

DevOpsiin sopivien ilmaisten työvälineiden kirjo on erittäin laaja, uusia työkaluja kehitellään jatkuvasti. Jokaiselle ilmaiselle työkalulle löytyy useita maksullisia vastaavaa toimintaa ajavia sovelluksia, sekä monilla maksullisilla sovelluksilla on saatavilla ilmainen versio, mikä on yleensä rajattu toiminnallisuuksiltaan. Taulukossa 2 on vertailuna listattu teknologia/työkalupakkiin valittuja ohjelmia ja niille vastaavia maksullisia ohjelmia, siitä näkökulmasta, mihin ne on otettu käyttöön.

Taulukko 2. Vertailua maksullisiin DevOps-työkaluihin.

Nimi	**1. Verrokki**		**2. Verrokki**	
GitHub (versionhallinta)	**BitBucket**		**GitLab**	
	- Ilmainen versio 5lle käyttäjälle - Cloud versiossa tuettuna CI/CD	- 3$ per käyttäjä per kk - Ilmainen versio todella rajattu	- Ilmainen versio 5lle käyttäjälle - CI/CD tuettuna	- 19.99$ dollaria per käyttäjä per kk
Jenkins (CI/CD)	**TeamCity**		**GitHub**	
	- professional tason lisenssiserveri ilmainen - Kolme ilmaista build agenttia	- Enterprice serveri maksaa 2000$ vuodessa - build agentit maksavat 299$ per agentti vuodessa (kolmen ylittävät)	- 2000 minuuttia ilmaista laskentaaikaa kuukaudessa	- 44$ per käyttäjä vuodessa - Maksullinen Team versio tarjoaa vain 3000min laskentaaikaa kuukaudessa - Monet toiminnallisuudet maksujen takana.
Taiga (Projektinhallinta)	**Jira**		**Zenhub**	
	- Ilmainen maks. 10 käyttäjälle	- Ilmainen vain hyvin rajattu ja vain pilviversio - 7.5$ per käyttäjä per kk	- Ilmainen julkisille projekteille - Integroituu githubiin	- 8.33$ per käyttäjä per kk

Discord (ChatOps)	Slack		Microsoft Teams	
	- Periaatteessa ilmainen käyttö	- Ilmainen versio sisältää vain 90 päivän keskusteluhistorian - 6,75€ per käyttäjä per kk (Pro)	- Ilmaisversion toiminta hyvin rajattua - Microsoft 365 Business Basic sisältää hyvin paljon microsoftin työkaluja ja integraatioita	- 3,4€+alv per käyttäjä per kk (Essentials) - 5,1€+alv per käyttäjä per kk (365 Business Basic)

Yleinen trendi siis on, että kaikista maksullisista sovelluksista saa ns. pienen tiimin version rajatuin ominaisuuksin, joilla koitetaan houkutella käyttäjä maksulliseksi asiakkaaksi. Jos haluaa täysin vapaat kädet, pitää pysyä täysin avoimen lähdekoodin tuotteissa ja jos taas kokee että pärjää rajatuin ominaisuuksin, voi monia maksullisien ohjelmien ilmaisversioita käyttää hyvin.

5.3.2 Millainen DevOps-elinkaari soveltuu parhaiten tarpeisiini?

Ei ole olemassa parasta elinkaarta. Tulevalle indie-peliyritykselleni sopii yksinkertaisesti sellainen elinkaari, mikä toteuttaa ainakin CI/CD-automaatioputken, projektinhallintaa sekä ChatOpsia. Lopulliseen DevOps-elinkaareen on vaikea ottaa kantaa, koska se riippuu täysin muovautuvasta yrityskulttuurista ja tehtävästä projektista, eli DevOps on itsessään iteroituva prosessi.

6 Pohdinta

6.1 Tutkimuksen luotettavuus

Koska ”toimeksiantajana” on tutkimuksen tekijä, on luotettavuutta tutkimuksen näkökulmasta hyvä kyseenalaistaa. Onko toimeksiannon kriteereitä helpompi muovata omaan tahtoon? Ehkä, mutta todellisuudessa kysymys on tarpeesta ja koska kriteerit on kuitenkin määritelty, voi niiden pohjalta tehdä vertailua lähtöasetelman ja tuloksen välillä. Saatuja tietoja, ohjeita sekä esimerkkejä voi kuitenkin jokainen lukija halutessaan käyttää. Kanasen (2015) ja Pernaan (2013) ohjeiden mukaan tehty kehittämistutkimus parantaa tutkimuksen luotettavuutta.

Luotetettavuutta olisi voinut parantaa, jos olisi tehnyt jokaisen ns. alasyklin välillä esimerkiksi havainnoinnit siitä, kuinka jokainen palanen nopeuttaisi työntekoa verrattuna lähtötilanteeseen ja

edelliseen alasykliin. Nyt voi tehdä vain vertailun lähtötilanteen ja ensimmäisen syklin välillä. Jokaisessa alasyklissä on kuitenkin pyritty noudattamaan Kanasen (2015, 41-42) kuvailemaa muutosyklin vaiheita: kokeilu/toteutus ja arviointi.

Tutkimuksessa ei ollut kolmansia osapuolia ja mikään tieto ei ollut salaista. Henkilötietoja ei käsitelty, eikä tutkimuslupaa tarvittu. Plagioinnilta vältytään käyttämällä lähteitä ja viittauksia asianmukaisesti. (Jyväskylän ammattikorkeakoulun eettiset periaatteet 2018.)

6.2 Lähteiden luotettavuus ja eettisyys

Pysymällä käytännössä lähes täysin artikkeli- ja e-kirja -tietolähteissä, mahdollistan mahdollisimman tuoreen tiedon käyttämisen. Pyrin pysymään pääasiallisesti alle viisi vuotta vanhoissa tietolähteissä. Koitin myös varmistaa, että kirjoittajalla olisi ainakin jotain kokemusta kirjoittamastaan. Lähteet ovat alkuperäislähteitä, eikä käytössä ole toissijaisia lähteitä. Lähteet olivat pääsääntöisesti kansainvälisiä, koska suomalaisia lähteitä ei löytynyt tarpeeksi.

Lähteiden eettisyyttä silmällä pitäen, koitin suosia julkisia lähteitä, välttäen maksumuurin takana olevaa lähdeaineistoa. Pyrin suosimaan sovellusten kehittäjien virallisia sivustoja ja ohjeita sovellusten käyttöönotossa, sillä heidän dokumenttinsa antavat varmasti luotettavimman lähteen sovellusten käyttöön. Näitä lähteitä ei ole käytetty päättelemään tuotteiden soveltuvuutta elinkaareen, sillä heidän tietonsa siitä ovat jääviä.

Tutkimusta tehdessäni tuli vastaani paljon mielestäni mielenkiintoisia artikkeleita ja videoita tutkimusaiheesta tai sen läheltä, mitkä eivät soveltuneet lähteiksi sellaisenaan. Tein näistä liitteeksi koosteen ”mielenkiintoisia artikkeleita”, joita vahvasti suosittelen lukemaan jos DevOps ja pelinkehitys kiinnostavat.

Lähteet

Abildskov, J. 2021. Mitä on DevOps? Blogipostaus Eficoden verkkosivuilla. Viitattu 29.1.2022. https://www.eficode.com/fi/blog/mita-on-devops.

About Itch.io. 2022. Itch.io:n verkkosivu. Viitattu 8.3.2022. https://itch.io/docs/general/about.

Alexander, T. 2018. Developing a DevOps Testing Strategy: Benefits, Best Practices & Tools. Viitattu 8.3.2022. https://smartbear.com/blog/devops-testing-strategy-best-practices-tools/.

Buchanan, I. N.d. History of DevOps. Atlassian verkkosivu. Viitattu 29.1.2022. https://www.atlassian.com/devops/what-is-devops/history-of-devops.

Communication & ChatOps Tools. 2022. Plutora verkkosivu. Viitattu 5.9.2022. https://www.plutora.com/ci-cd-tools/communication-chatops-tools.

Coster, S. 2020. Stress-Free Game Development: Powering Up Your Studio With DevOps. Virtuaaliesitys GDC:n Youtube kanavalla. Viitattu 19.4.2022. https://www.youtube.com/watch?v=t9HRzE7_2Xc.

DevOps: Breaking the development-operations barrier. 2022. Atlassian verkkosivu. Viitattu 8.3.2022. https://www.atlassian.com/devops.

Forsell, J. 2020. DevOps-Toimintamalli. Opinnäytetyö, Tampereen ammattikorkeakoulu. Viitattu 8.3.2022. https://urn.fi/URN:NBN:fi:amk-2020120726476.

Fowler, M. 2006. Continous Integration. Artikkeli martinfowler verkkosivulla. Viitattu 29.1.2022. https://www.martinfowler.com/articles/continuousIntegration.html.

Hall, T. N.d. DevOps Best Practices. Blogiteksti Atlassian verkkosivulla. Viitattu 26.3.2022. https://www.atlassian.com/devops/what-is-devops/devops-culture.

Hamilton, T. 2022a. What is Test Driven Development (TDD)? Blogiteksti Guru99 verkkosivuilla. Viitattu 29.1.2022. https://www.guru99.com/test-driven-development.html.

Hamilton, T. 2022b. What is WHITE Box Testing? Techniques, Example & Types. Blogiteksti Guru99 verkkosivulla. viitattu 10.4.2022. https://www.guru99.com/white-box-testing.html.

Jenkins User Handbook. 2022. Jenkins sovelluksen Linux asennuksen dokumentaatio Jenkins verkkosivulla. Viitattu 29.4.2022. https://www.jenkins.io/doc/book/installing/linux/.

Jyväskylän ammattikorkeakoulun eettiset periaatteet. 2018. pdf-tiedosto JAMKin intrassa. Viitattu 13.9.2022. https://www.jamk.fi/fi/file/eettiset-periaatteet.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas: miten kirjoitan kehittämistutkimuksen vaihe vaiheelta. Jyväskylä: Jyväskylän ammattikorkeakoulu. E-kirja. Viitattu 29.8.2022. https://janet.finna.fi, Booky.

Kim, G., Humble, J., Debois, P., Willis, J., Allspaw, J. 2016. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press. E-kirja. Viitattu 27.3.2022. https://janet.finna.fi, Skillsoft Books ITPro.

Koutifaris, A. 2018. Test Driven Development – What it Is, and What it Is Not. Artikkeli sivustolla freeCodeCamp. Viitattu 4.4.2022. https://www.freecodecamp.org/news/test-driven-development-what-it-is-and-what-it-is-not-41fa6bca02a2.

Kralj, K. 2022. What Do You Need to Know About AAA in C# Unit Testing. Blogipostaus methodpoet verkkosivustolla. Viitattu 10.4.2022. https://methodpoet.com/aaa-in-unit-testing/.

Lynch, M. 2018. Why Good Developers Write Bad Unit Tests. Blogipostaus mtlynch verkkosivustolla. Viitattu 10.4.2022. https://mtlynch.io/good-developers-bad-tests/.

Pennington, J. 2019. The Eight Phases of a DevOps Pipeline. Artikkeli medium.com verkkosivulla. Viitattu 31.8.2022. https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba.

Pittet, S. N.d. What is Continuous Deployment? Ohje Atlassian verkkosivu. Viitattu 8.3.2022. https://www.atlassian.com/continuous-delivery/continuous-deployment.

Pernaa, J. 2013. Kehittämistutkimus tutkimusmenetelmänä. Helsingin yliopiston tuhat-artikkelit. Viitattu 5.9.2022. https://helda.helsinki.fi/handle/10138/317958.

Raza, M. 2021. Explained: Monitoring & Telemetry in DevOps. Blogiteksti bmc verkkosivustolla. Julkaistu 14.9.2021. Viitattu 28.3.2022. https://www.bmc.com/blogs/devops-monitoring-telemetry/.

Schmitt, J. 2022. Technical debt: how to measure and manage it with DevOps. Blogiteksti circleci verkkosivulla. Viitattu 5.9.2022. https://circleci.com/blog/manage-and-measure-technical-debt/.

Todorov, G. 2022. 40+ DevOps Statistics You Should Know in 2022. Artikkeli strongdm sivustolla. Viitattu 19.4.2022. https://www.strongdm.com/blog/devops-statistics.

Tyagi, D. 2021. Crunch Culture: The Ugly Truth of Video Game Industry. Artikkeli medium verkkosivustolla. Viitattu 19.4.2022. https://medium.com/@deepanshut041/crunch-culture-the-ugly-truth-of-video-game-industry-65d8e9d90cdd.

Wickramasinghe, S. 2021. Testing in DevOps: Concepts, Best Practices & More. Blogiteksti bmc verkkosivustolla. Viitattu 8.3.2022 https://www.bmc.com/blogs/devops-testing/.

What is DevOps? 2022. Amazon AWS verkkosivu. Viitattu 29.1.2022. https://aws.amazon.com/devops/what-is-devops/.

Unit test basics. 2022. Artikkeli Microsoftin Visual Studio dokumentaatio sivustolla. Viitattu 10.4.2022. https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2022.

Unit test best practices. 2021. Artikkeli Microsoftin DotNet dokumentaatio sivustolla. Viitattu 10.4.2022. https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices.

Unity Code Coverage. 2022. Unityn dokumentaatio Code Coverage lisäosasta. Viitattu 5.9.2022. https://docs.unity3d.com/Packages/com.unity.testtools.codecoverage@1.2/manual/index.html.

Unity Documentation. 2022. Unityn oma dokumentaatio sivusto, valittu sovellusversio 2021.3:n sivusto. Viitattu 31.8.2022. https://docs.unity3d.com/Manual/.

Liitteet

Liite 1. Jenkinsfile-tiedosto

```
pipeline {
  agent any

  environment {
    UNITY_VERSION = "2021.2.7f1"
    PROJECT_NAME = "ThesisDevOpsProject"

    DISCORD_WEBHOOK = credentials('DISCORD_WEBHOOK_KEY')
    GIT_LINK = credentials('GIT_GHP_AUTH')
    UNITY_USERNAME = credentials('UNITY_ID')
    UNITY_PASSWORD = credentials('UNITY_PASSWD')
    UNITY_SERIAL = credentials('UNITY_PRO_KEY')
    UNITY_PATH = "/media/kimmokissa/unity/Editors/${UNITY_VERSION}/Editor/Unity"

    UNITY_TEST_LOG_PATH = "/var/www/tarinataikomo/buildreports/testlogs"
    BUILD_TARGET = "/media/kimmokissa/Documents/Builds/${PROJECT_NAME}/${PROJECT_NAME}-ver.${env.BUILD_NUMBER}.exe"
    UNITY_PROJECT_PATH = "/media/kimmokissa/unity/Projects/"
    BUILD_LOG_PATH = "/var/www/tarinataikomo/buildreports/buildlogs"
    BUILD_VER = "${env.BUILD_NUMBER}"
  }

  stages {
    stage("Pull files from git"){
      steps{
        // Lataa tiedostot GitHubista ja siirrä ne parempaan kansioon.
        // Samalla varmistetaan että jenkins käyttäjällä on aina oikeus kaikkiin tiedostoihin.
        git GIT_LINK
        sh 'sudo cp -R  /var/lib/jenkins/workspace/thesisdemoproject ${UNITY_PROJECT_PATH}'
        sh "sudo chmod 777 /media/kimmokissa/unity/Projects/"
        sh "sudo chown -R jenkins:jenkins /media/kimmokissa/unity/*"
      }
    }
    stage("Cleanup old Builds"){
      steps{
        // Poistetaan vanha buildi sisältö
        sh "sudo rm -rf /media/kimmokissa/Documents/Builds/*"
      }
    }
    stage("Code Coverage"){
      // Code coverage
      steps {
        sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -serial ${UNITY_SERIAL} -batchmode -projectPath ${UNITY_PROJECT_PATH}thesisdemoproject -
debugCodeOptimization -enableCodeCoverage -coverageResultsPath /var/www/tarinataikomo/buildreports/CodeCoverage/ -coverageOptions "generateHtmlReport;generateBadgeReport;assem-
blyFilters:+my.assembly.*" -nographics -quit'
        sh 'sudo cp -R /media/kimmokissa/unity/Projects/thesisdemoproject/CodeCoverage/ /var/www/tarinataikomo/buildreports/'
        sh 'sudo chmod 777 /var/www/tarinataikomo/buildreports/CodeCoverage/'
      }
    }
    stage("Unit Tests"){
      // Ensin tehdään playmode testit.
      steps {
        sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -serial ${UNITY_SERIAL} -batchmode -projectPath ${UNITY_PROJECT_PATH}thesisdemoproject -
runTests -testResults ${UNITY_TEST_LOG_PATH}/PlaymodeTestResults.ver-${BUILD_VER}.xml -logFile ${BUILD_LOG_PATH}/TestLog-ver-${BUILD_VER}.txt -testPlatform PlayMode -nographics'
      }
    }
    stage("Build"){
      // Onnistuneiden testien jälkeen vasta buildataan.
      steps{
        sh '${UNITY_PATH} -username ${UNITY_USERNAME} -password ${UNITY_PASSWORD} -serial ${UNITY_SERIAL} -batchmode -nographics -buildTarget Win64 -buildWindows64Player
${BUILD_TARGET}  -projectpath ${UNITY_PROJECT_PATH}thesisdemoproject -logFile ${BUILD_LOG_PATH}/BuildLog-ver-${BUILD_VER}.txt -quit'
      }
    }
    stage("Deploy to itch.io"){
      steps{
        // Lähettää onnistuneen buildin Itch.ioon.
        sh "/etc/butler/butler push /media/kimmokissa/Documents/Builds/ kimmokissa/thesisdemoproject:windows-beta"
      }
    }
  }
  post {
    always{
      // Ilmoitetaan discordiin onko buildi onnistunut vai ei.
      discordSend (
        description: "Thesis DevOps build is ${currentBuild.currentResult}\nBuild #: ${env.BUILD_NUMBER}\n",
        result: currentBuild.currentResult,
        image: "http://88.113.245.162/buildreports/CodeCoverage/Report/badge_linecoverage.png",
        title: JOB_NAME,
        webhookURL: DISCORD_WEBHOOK
        )
    }
  }
}
```

Liite 2. MainMenuPlayModeTests.cs-tiedosto

```
using System.Collections;
using NUnit.Framework;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.TestTools;
using UnityEngine.UI;

public class MainMenuPlayModeTests
{
   bool clicked;

   [SetUp]
   public void SetUp()
   {
        SceneManager.LoadScene("SampleScene");
   }

   [UnityTest]
   public IEnumerator TestStartButtonExists()
   {
        //Arrange
        var name = "start";
        // Act
        var startButton = GameObject.Find(name);
        // Assert
        Assert.NotNull(startButton);
        yield return new WaitForSeconds(2);
   }

   [UnityTest]
   public IEnumerator ClickStartMenuBtnExpectTrue()
   {
        //Arrange
        var name = "start";
        var startButton = GameObject.Find(name);
        // Act
        var setupButton = startButton.GetComponent<Button>();
        setupButton.onClick.AddListener(Clicked);
        setupButton.onClick.Invoke();
        // Assert
        Assert.IsTrue(clicked);

        yield return new WaitForSeconds(2);
   }

   private void Clicked()
   {
       clicked = true;
   }
}
```

Liite 3. Mielenkiintoista luettavaa

DevOps ja pelikehitys

DevOps in Game Development
https://medium.com/another-angle/devops-in-game-development-c89a9bf21787

Unity Build Automation
https://smashriot.com/unity-build-automation/

Tekninen velka

Tekninen velka – tunnusta, tunnista ja rajoita
https://www.sytyke.org/tapetilla/tekninen-velka-tunnusta-tunnista-ja-rajoita/

Tekninen velka ei näy kirjanpidossa. Pitäisikö näkyä? Vältä "korkoa korolle"-ilmiötä!
https://www.datapaaoma.fi/post/tekninen-velka-ei-n%C3%A4y-kirjanpidossa-pit%C3%A4isik%C3%B6-n%C3%A4ky%C3%A4-v%C3%A4lt%C3%A4-korkoa-korolle-ilmi%C3%B6t%C3%A4

Crunch pelialalla ja sen vaikutukset
https://www.theseus.fi/bitstream/handle/10024/352659/Kajavo_Jan.pdf

DevOps suunnittelu

The DevOps Mindset: A Step-by-Step Plan to Implement DevOps
https://hackernoon.com/the-devops-mindset-a-step-by-step-plan-to-implement-devops-s03p35rr

Tech Stack Intelligence (Esimerkkejä erilaisista teknologioista mitä yritykset käyttävät)
https://stackshare.io

DevOps vs muut

Is NoOps the End of DevOps? Deep Dives by Bunnyshell
https://www.bunnyshell.com/blog/is-noops-the-end-of-devops

What is NoOps? The quest for fully automated IT operations
https://www.cio.com/article/220351/what-is-noops-the-quest-for-fully-automated-it-operations.html

DevOps vs. Automation: What's The Difference?
https://www.bunnyshell.com/blog/devops-automation-difference

Liite 4. Aineistonhallintasuunnitelma

DevOps-elinkaari aloittavalle indie-peliyritykselle

A Data Management Plan created using dmptuuli

Creator: Kim Eriksson

Affiliation: Jyväskylän ammattikorkeakoulu

Template: Jyväskylän ammattikorkeakoulun opinnäytetyön aineistonhallintasuunnitelma

Last modified: 6.9.2022

1. Aineiston yleiskuvaus

Kuvaile, millaiseen aineistoon opinnäytetyösi perustuu. Millaista aineistoa kerätään, tuotetaan tai käytetään uudelleen? Missä tiedostomuodossa aineisto on?

Opinnäytetyön aineisto kerätään etsimällä tyypillisen DevOps putken mukaisten vaiheisiin, sekä kriteereihin sopivat sovellukset, ja vertailemalla niitä keskenään. Vertailu sovellusten välillä tulee olemaan taulukkomuodossa. Esitetyt sovellukset tullaan kuvailemaan mitä ne tekevät ja miten ne sopivat suunniteltuun vaiheeseen.

Muistiinpanot tulevat olemaan .docx ja kuvat .png muodossa.

Miten aineiston yhtenäisyys ja laatu varmistetaan?

Aineiston laatu varmistetaan tutkimalla lähteitä ja miten relevanttia tietoa se tarjoaa ongelmaan sekä aineiston julkaisu- ja/tai päivityspäivä vaikuttavat.

2. Eettiset periaatteet ja lainsäädäntö

Mitä juridisia seikkoja liittyy aineistonhallintaan (esim. tietosuojalaki ja muu aineiston käsittelyyn liittyvä lainsäädäntö)?

Mitään henkilökohtaista dataa ei kerätä, joten GDPR:ää ei tarvitse miettiä. Muuten noudatetaan JAMK:in ohjeistuksia.

Miten hallinnoit käyttämäsi, tuottamasi ja jakamasi aineiston oikeuksia? Onko aineisto salassa pidettävää?

Opinnäytetyön prosessin ulkopuolisilla ei ole oikeutta nähdä aineistoa. Pidän oikeuden tuottamaani mahdolliseen testiprototyyppiin millä kokeillaan DevOps putkea.

3. Dokumentointi ja metatiedot: aineiston keruun, sisällön ja käsittelyn dokumentointi

Miten dokumentoit aineistosi, jotta se on löydettävissä, saavutettavissa, yhteen toimiva ja uudelleen käytettävissä sekä itseäsi että muita varten?

En dokumentoi mitä opinnäytetyöaineistoa on käytössä, mutta kerron siitä opparissa. Aineisto tallennetaan OneDriveen ja GitHubiin. Opinnäytetyö julkaistaan theseus.fi sivustolle.

4. Tallentaminen ja varmuuskopiointi opinnäytetyön tekemisen aikana

Minne aineistosi tallennetaan ja miten se varmuuskopioidaan?

Aineistoa säilytetään OneDrivessä ja tutkimuksessa käytettävää demoprojektin lähdekoodia säilytetään Githubissa

Kuka valvoo pääsyä aineistoon, ja miten suojattua pääsyä aineistoon valvotaan?

Aineistoa ei jaeta muille.

5. Aineiston tallentaminen, avaaminen ja arkistointi opinnäytetyön valmistuttua

Mikä osa aineistosta voidaan asettaa avoimesti saataville tai julkaista? Missä ja milloin aineisto tai siihen liittyvät metatiedot asetetaan saataville?

Tutkimusaineiston osista ei tule mitään avoimesti saataville jälkikäteen, vaan kaikki hävitetään. Itselleni jää vain tietotaito toteuttaa soveltuva putki, kun sen aika on.

Mihin tutkimusaineisto arkistoidaan ja kuinka pitkäksi ajaksi?

Aineisto hävitetään, kun opinnäytetyö on valmis ja arvosteltu.

6. Aineistonhallinnan vastuut ja resurssit

Kuka vastaa aineistonhallinnasta eri vaiheissa? Tarvitaanko erillisiä resursseja?

Hallinnoin itse kaikkea aineistoa.