jamk.fi

AutoCAD-liitännäissovelluksen automaatiotestaus

Janne Möttölä

Opinnäytetyö
Maaliskuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Möttölä, Janne	Opinnäytetyö, AMK	Maaliskuu 2018
	Sivumäärä 77	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
AutoCAD-liitännäissovelluksen automaatiotestaus

Tutkinto-ohjelma
Ohjelmistotekniikan tutkinto-ohjelma

Työn ohjaaja(t)
Esa Salmikangas

Toimeksiantaja(t)
Inmics Software Engineering Oy

Tiivistelmä

Inmics Software Engineering Oy tarvitsi automatisoidun testausjärjestelmän AutoCAD-liitännäissovelluksensa kehitystyön parantamiseksi. Tätä varten tuli tehdä tutkimus AutoCAD-liitännäissovelluksen automaatiotestaamisesta, jonka tuloksiin perustuen tuli rakentaa prototyyppi kokoonpanosta. Kokoonpanon oli kyettävä kokoamaan, yksikkötestaamaan ja integraatiotestaamaan liitännäissovellus. Lisäksi sen piti toimia omalla testipalvelimellaan ja kyetä skaalautumaan tarpeen mukaan. Yksikkötestien ajamisen mahdollisuus tuli todentaa, ja testien lisäämiselle tuli rakentaa valmis pohja. Integraatiotestien testauskohteiksi määriteltiin liitännäissovelluksen tuottama graafinen esitys sekä XML-tuloste. Ennen prototyypin rakentamista tuli myös selvittää, mikä CI/CD-työkalu olisi tehtävään sopivin. Lisätoiveena oli, että automaatio kykenisi jatkuvaan julkaisuun.

Tuloksien mukaan AutoCAD-liitännäissovelluksen testaus voitiin toteuttaa yksikkötestaustasolla poikkeamatta yleisestä testauskäytännöstä, mutta integraatiotestien toteuttamiseen tarvittiin rakenne, jolla testit voitiin ladata AutoCADin säikeeseen. Gallio tarjosi, suurelta osin, toiminnallisuuden testien lataamiseen AutoCADin säikeeseen, jonka perusteella voitiin toteuttaa prototyyppi. AutoCADin piirustusten kääntelyä varten oli kuitenkin tehtävä muutoksia Gallion lähdekoodiin. CI/CD-työkaluja vertailtiin, ja tuotettiin kriteerien mukainen prototyyppi, joka kykeni vaadittuun toiminnallisuuteen. Tulos oli modulaarinen ja helppo laajentaa. Prototyyppikokoonpanon visuaalisten testien tuloksissa ilmeni satunnaista epäkonsistenttisuutta, jonka syytä ei saatu varmuudella yksilöityä, mutta syyksi epäiltiin palvelimen muistiresurssien puutetta. Muuten järjestelmä toimi virheettömästi, ja kokoonpano kykeni pienellä integraatiotyöllä myös jatkuvaan julkaisuun.

Avainsanat (asiasanat)
AutoCAD, automaatio, automaatiotestaus, testaus, liitännäinen, MbUnit, Gallio, TestApi, XMLUnit

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Möttölä, Janne	Bachelor's thesis	March 2018
		Language of publication: Finnish
	Number of pages: 77	Permission for web publication: x

Title of publication

Automatic testing of an AutoCAD plugin

Degree programme

Software Engineering

Supervisor(s)

Salmikangas, Esa

Assigned by

Inmics Software Engineering Inc.

Abstract

Inmics Software Engineering Inc. needed an automated testing system, to improve their AutoCAD-plugin's development process. To satisfy the need, a research of AutoCAD-plugin automation testing was to be made, and a prototype was to be built based on the results. The prototype was to be able to build, unit test and integration test the plugin. Additionally, it had to function on its own host, and be able to scale according to need. The possibility of running unit tests was to be verified, and a base for inserting tests was to be built. The defined targets for integration tests were the generated graphical product, and the printed XML files of the plugin. CI/CD-tools were also to be compared, to find the most adequate one for the job, before building the prototype. An additional request was that the automation could manage continuous delivery.

According to the results, the automated testing of an AutoCAD-plugin could be done, at unit testing level, without deviation from the usual testing methodology, but the integration testing required functionality, that could load the tests into AutoCAD's thread. Gallio offered, for the most part, functionalities to load the tests into AutoCAD's thread, to which a prototype could be based on, but to turn AutoCAD drawing's view, some changes to Gallio's code had to be made. CI/CD tools were compared, and a prototype was built, with respect and according to the given criteria. The result was modular, and easy to expand. Some inconsistencies were encountered while running visual tests, and the cause could not be identified with certainty, but a lack of server memory resources was suspected. Otherwise the system worked flawlessly, and the setup could also manage continuous delivery with minor integration work.

Keywords/tags (subjects)

AutoCAD, automation, testing, plugin, MbUnit, Gallio, TestApi, XMLUnit

Miscellaneous

Sisältö

Kuviot

1 Johdanto automaatiotestaukseen

Ohjelmistokehitys on monimutkainen prosessi, jossa on paljon erilaisia tehtäviä ja osa-alueita, jotka ovat usein monimutkaisia. Työntekijöiden kyky yleisellä tasolla vastata tehtävien kompleksisuuteen on vaihtelevaa, mutta jokainen tekee virheitä joskus. Väsymys, stressi ja moni muu tekijä vaikuttaa tekijän kykyyn selvitä saamastaan tehtävästä virheettä. Virheiden tekemisestä seuraa korjaustöitä, joka näkyy kehitysprosessissa mm. resurssien kulumisena. Segue Technologiesin kotisivuilla tarkastellaan resurssien kulutusta ohjelmistoprojekteissa seuraavasti.

Ohjelmistoprojekteissa, varsinkin suurissa sellaisissa, kuluu paljon resursseja tuotteen testaamiseen ja sen toiminnan varmistamiseen. Nämä resurssit ovat aika, henkilöstö ja muut lisäkustannukset, joiden kulutus voi nopeasti kasvaa yllättävän suureksi suhteessa kulutuksen ennalta arvioituun määrään. Automaatiotestaus voidaan usein implementoida vähentämään näiden resurssien kulutusta. (How Important is Test Automation in a Software Project? 2013.)

Koska resurssien kulutus, suhteessa saatavilla oleviin resursseihin, on jokaiselle yritykselle elintärkeä asia, voidaan heti havaita automaatiotestauksen potentiaalinen arvo sovelluskehityksessä. Jos resurssien kulutusta voidaan vähentää, se näkyy jäljelle jäävän resurssimäärän suuruudessa, ja siten yrityksen tehokkuudessa. Sovelluksia on kuitenkin monenlaisia, ja riippuen sovelluksesta, sen automatisoitu testaaminen voi olla haasteellista. Testattavasta kohteesta on saatava ulos vertailukelpoista tietoa, jotta sitä voidaan testata.

Inmics Software Engineering Oy:llä oli tarve järjestelmälle, jolla voitaisiin testata AutoCAD-liitännäissovelluksen toimintaa automaattisesti. Tässä opinnäytetyössä pyrittiin tutkimaan, kuinka automaatiotestaus voitaisiin toteuttaa AutoCAD-liitännäissovellukselle, ja tekemään löydettyyn tietoon perustuva prototyyppi. Tulosten avulla haluttiin vähentää inhimillisten virheiden määrää kehitystyössä, nopeuttaa sovelluskehitysprosessia ja laskea kulutettujen resurssien määrää.

2 Tutkimus ja aineisto

2.1 Työn luonne ja menetelmät

Pohjimmiltaan kyseessä oli kehittämistyö, jossa automaatiorakenne piti hahmotella ja toteuttaa. Hahmotteluosuudessa pyrittiin hankkimaan tietoa ja näkemystä, kuinka AutoCAD-liitännäissovelluksen automatisoitu testaamisrakenne voitaisiin toteuttaa. Tietoa haettiin verkosta, hyödyntämällä Googlea, sekä keskustelemalla alan ammattilaisten kanssa. Selvitystyössä kerättyä tietoa pyrittiin implementoimaan käytännössä, jotta voitaisiin varmentaa tiedon validiteetti ja selvittää, onko tieto hyödynnettävissä lopullisessa kokoonpanossa. Testi-implementaatio toimi siten tiedon analysointimenetelmän osana. Kaikkea tietoa ei kuitenkaan todennettu omalla implementaatiolla, sillä työstä olisi tullut kohtuuttoman laaja. Tietoja, joiden validiteettia ei todennettu koeimplementaatiolla, analysoitiin noudattamalla tervettä lähdekriittisyyttä. Kun vaaditut tiedot prototyypin toteuttamiseen oli kerätty, niiden perusteella räätälöitiin tilaajan kriteerien mukainen automaatiorakenne.

2.2 Käsitteiden määrittely

Liitännäissovellus (plugin) on toiseen sovellukseen kiinnitettävä moduuli, jolla voidaan tuoda uusia ominaisuuksia toisen sovelluksen käyttöön. Esimerkiksi selaimeen voi joutua asentamaan liitännäisen, jotta videon toistaminen toimii selaimessa. (Plugin 2017.)

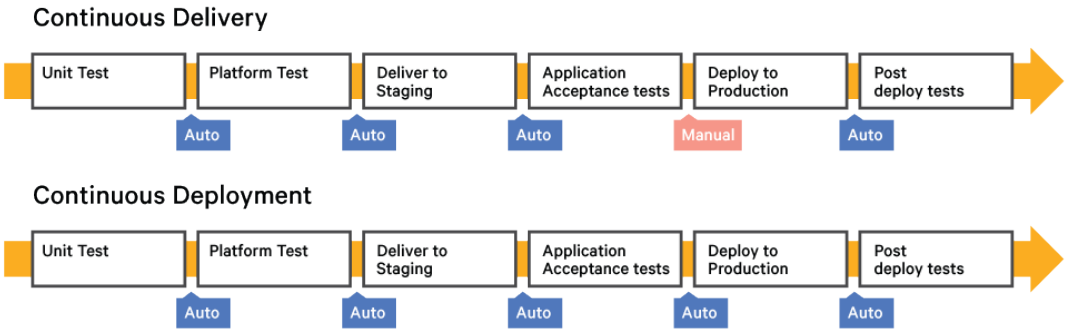
Automaatiotestaus on prosessi, jossa automatisoidut työkalut ajavat testejä kohteelle. Testit toistavat ennalta määriteltyjä testitapauksia, joiden tuloksia vertaillaan odotettuihin tuloksiin. Jos odotetut tulokset ja todelliset tulokset vastaavat toisiaan, ohjelma käyttäytyy kuten sen pitää, ja näiden testien osalta ohjelma on todennäköisesti virheetön. Jos odotetuissa ja saaduissa tuloksissa on eroavaisuuksia, jossain on virhe. (Automated vs. Manual Testing: The Pros and Cons of Each n.d.)

Jatkuva integraatio (continuous integration, CI) on käytäntö, jossa kehittäjät yhdistävät tuotoksiaan tiheään tahtiin, ja jossa tehdyt muutokset käännetään,

testataan ja tulokset raportoidaan heti integraation yhteydessä. Käytännön tavoitteena on tuottaa jatkuvaa ja nopeaa palautetta, jotta virhetilanteet saadaan nopeasti havaittua ja korjattua niiden syntymän jälkeen. (Najjar n.d.)

Jatkuva toimitus (continuous delivery, CD) on joukko menetelmiä, joilla varmistetaan, että jokainen muutos tuotteessa voidaan siirtää osaksi tuotantokoodia nopeasti ja turvallisesti varmistamalla niiden toimivuus automaatiotesteillä. Kun jokainen muutos saapuu julkaisuvalmiuteen testausautomaation läpikäyneenä, voidaan luottaa siihen, että tuote on julkaisukelpoinen ja voidaan laittaa tuotantoon napin painalluksella. (Caum 2013.) Mitään testejä ei voida pitää täydellisenä takeena tuotteen virheettömyydelle, mutta ne antavat silti tietyn asteista luotettavuutta.

Jatkuva julkaisu (continuous deployment, CD) on seuraava askel jatkuvasta toimituksesta (ks. kuvio 1), jossa myös julkaisu tuotantoon on automatisoitu muiden toimitusputken alueiden lisäksi (Caum 2013).



Kuvio 1. Jatkuva toimitus ja jatkuva julkaisu

Putki (pipeline) on automaatiorakenne, joka toteuttaa jatkuvaa julkaisua tai jotain sen osaa. Ei ole olemassa standardisoitua putkea, mutta tyypillisesti putket kattavat kolme tasoa: kääntämisautomaation, testausautomaation ja julkaisuautomaation. (Phillips 2014.)

Yksikkötestaus on testausmalli, missä testataan funktioita eristettynä muusta ohjelman kontekstista. Yksikkötestissä on aina staattinen, eli muuttumaton, alusta. (McFarlin 2012.)

Integraatiotestaus on testausmalli, missä testataan kohteen osien toimintaa yhdessä (Integration Testing n.d.).

Sovelluskehys, tai sovelluskehikko, on kokoelma toimintoja, joka voi toimia sovelluksen pohjana. Niiden käyttämisen tarkoitus on nopeuttaa sovelluksen kehittämistä, tarjoamalla valmista toiminnallisuutta. (Cristof 2017)

Testitapaus on yksittäinen toteutettava testi, johon kuuluu kokoelma ehtoja, joiden perusteella testaaja määrittelee testin onnistumisen tai epäonnistumisen (Bartlett 2018).

Artefakti on mikä tahansa jäänne, tai tuote, joka luodaan määritellyn prosessin aikana. JetBrainsin confluencessa määritellään käännösartefakti sekä sen käyttö vapaasti käännettynä seuraavasti. Käännösartefaktit ovat käännöksissä tuotettuja tiedostoja, joita tallennetaan palvelimelle. Tyypillisesti nämä sisältävät raportteja, lokeja, WAR-tiedostoja, jakelupaketteja tai muita vastaavia käännöksen tuotoksia. Kun käännös saadaan päätökseen, konfiguraatiossa määritelty käännöksen sisältö kerätään talteen artefakteina. (Alexandrova & Megorskaya 2015.)

Testifikstuuri on testitapauksen ympäristön, eli kontekstin, rakentava testauskehikon luokka (Japikse, 2008). Jotkut testitapaukset voivat vaatia ympärilleen tietynlaisen kontekstin toteutuakseen. Testifikstuurin tehtävä on luoda se konteksti.

Versionhallinta on konsepti, jossa käsitellään kohteen muutosten jäljittämistä ja hallintaa. Versionhallintajärjestelmä pitää kirjaa kohteeseen tehdyistä muutoksista. (What is version control n.d.)

Kommitti (commit) on koodiin tehtyjen muutosten kokoelma, tai muutospaketti. Sen sijaan että jokainen yksittäinen muutos tuottaisi oman merkintänsä versionhallintajärjestelmässä, muutoksia seurataan kommiteilla. (Nagele n.d.)

Git on yleisimmin käytetty versionhallintajärjestelmä, kaikista versionhallintajärjestelmistä (Ngan n.d.).

Subversion on avoimeen lähdekoodiin perustuva ilmainen versionhallintajärjestelmä (Collins-Sussma & Fitzpatrick & Pilato n.d.).

.NET on ilmainen, usealle käyttöjärjestelmälle kohdennettu kehitysalusta. .NET tukee useita ohjelmointikieliä, editoreita ja kirjastoja erilaisten sovellusten rakentamiseen. (What is .NET? n.d.)

API, lyhenne sanoista Application Programming Interface, on ohjelmistorajapinta, joiden avulla ohjelmistot voivat kommunikoida keskenään. Vertauskuvana APIn toiminnan havainnollistamiselle voidaan käyttää MuleSoftin tarjoamaa esimerkkiä ravintolasta, jossa asiakas tilaa ruokaa. Ravintola on järjestelmä, joka ottaa tilauksia vastaan tarjoilijoidensa kautta. Tarjoilija, eli rajapinta, odottaa tietynlaista tilausta, eli viestiä. Kun asiakas antaa tilauksen, tarjoilija vie sen syvemmälle järjestelmään, jossa reagoidaan tilaukseen ennalta määrätyllä tavalla. Tässä esimerkissä viesti kuljetetaan keittiölle, jossa tuotetaan asiakkaan tilaama annos. (What is an API? - - n.d.) API toimii siis rajapintana viestien vastaanottamiselle järjestelmässä.

Virtualisointi tarkoittaa, Technopedian mukaan, virtuaalisten resurssien luontia. Resurssit voivat olla esimerkiksi palvelimia tai käyttöjärjestelmiä. Virtualisoinnin päämääränä on hallinnoida työtaakkoja, tekemällä perinteisestä IT-ympäristöstä paremmin skaalautuvan. Virtualisointi on ollut IT-maailman osana jo pitkään. (Virtualization, n.d.)

DevOps, lyhennelmä sanoista Development ja Operations, on kokoelma käytäntöjä. Näillä käytännöillä automatisoidaan prosessit sovelluskehityksen ja IT-tiimien välillä, jotta ohjelmistoja voidaan julkaista nopeammin ja luotettavammin. DevOpsin konsepti on luotu rakentamaan yhteystyökulttuuri eri tiimien kesken, jotka ovat aikaisemin toimineet omissa yksiköissään. Sen hyötyjä ovat korkeampi luotettavuus, nopeammat julkaisut ja vahvempi ongelmanratkaisukyky organisaatiossa. (DevOps: Breaking the Development-Operations barrier n.d.)

HTML on kirjainlyhenne sanoista HyperText Markup Language. Se on kieli, joka luotiin mahdollistamaan verkkosivujen luonti. HTML koostuu tekstitiedostoon kirjoitetuista tageista, joita voi lukea selaimella, kuten Internet Explorerilla. Selain lukee HTML-tiedoston, ja muuntaa tekstin tagien määrittämään muotoon. (Shannon 2012.)

XML on lyhennelmä sanoista Extensible Markup Language. Toisin kuin HTML, XML on metakieli, jolla voidaan suunnitella muita markup-kieliä ja formaatteja. (What is XML? n.d.)

2.3 Toimeksianto

Toimeksiannon ydinajatus oli tutkimus testausautomaation toteuttamisesta Inmics Software Engineering Oy:n kehittämään AutoCAD-liitännäissovellukseen, tukemaan sen kehittämistä ja laatua. Tutkimuksessa tuli selvittää, kuinka voitaisiin toteuttaa testausautomaatio, joka toimisi omalla testipalvelimellaan, noudattaisi jatkuvan integraation periaatetta, kykenisi suorittamaan integraatio- ja yksikkötestejä, ja kykenisi skaalautumaan itsenäisesti tarpeen mukaan.

Integraatiotestit ja yksikkötestit tuli erotella toisistaan, ja yksikkötestit tuli ajaa ennen raskaampia integraatiotestejä. Jos yksikkötesteissä ilmenisi virhe, niin testiajo voitaisiin pysäyttää jo siinä, jolloin säästettäisiin raskaampiin integraatiotesteihin kulunut aika. Näin kehittäjät pääsisivät nopeammin tutkimaan ja korjaamaan virheitä. Integraatiotestien testauskohteiksi määriteltiin liitännäissovelluksen tuottama grafiikka, sekä XML-tuloste. Yksikkötestejä ei tarvinnut erikseen toteuttaa millekään kohteelle, mutta piti toteuttaa alusta, johon niitä voitaisiin lisätä.

Lisäksi määriteltiin, että erinäisiä CI/CD-työkaluja tuli tutkia, ja selvittää niistä soveltuvin vaihtoehto lopullisen automaatiorakenteen alustaksi. Työn tilaaja listasi viisi eri CI/CD-työkalua, joiden soveltuvuutta testirakenteen pyörittämiseen tuli vertailla. Jokaisesta tuli tehdä pieni kokeellinen toteutus, joiden perusteella lopullinen valinta voitaisiin tehdä. Listatut työkalut olivat Jenkins, TeamCity, GoCD, Team Foundation Server ja GitLab. Tavoitetilana oli kokoonpano, jossa uusi putken ajo käynnistettäisiin aina, kun versionhallintaan saapuisi uusi kommitti. Koodi tuli kyetä kokoamaan, testaamaan ja julkaisemaan työkalun avulla. Selvityksien pohjalta toimeksiantaja valitsisi sopivan CI/CD-työkalun, josta tulisi toteuttaa prototyyppikokoonpano, jonka versionhallinnasta vastaisi Git. Toissijaisena toiveena oli jatkuvan julkaisun toteuttaminen, jossa asiakkaalle saapuisi, napin painalluksella, kokeiltavaksi testausautomaation läpäissyt tuote.

2.4 Tutkimuksen rajaus

Keskeisenä elementtinä tutkimuksessa oli AutoDeskin tuottama sovellus, AutoCAD, ja Inmics Software Engineeringin Oy:n siihen tuottama liitännäissovellus. Kaikki, mikä ei koskenut näiden sovellusten automatisoitua testaamista, rajattiin pois.

Tutkimuksessa pidettiin keskeisenä uusien mahdollisuuksien kartoittamista sekä toteuttamista. Siten tutkimus ei keskittynyt aktiivisesti tarkastelemaan asioita tietoturvan kannalta, mutta tietoturvaa ei myöskään rajattu pois prototyyppitoteutuksesta.

3 Käytetyt teknologiat

3.1 Ohjelmointikielet

Työssä käytettiin useampia eri ohjelmointikieliä haluttujen tulosten saavuttamiseksi. Eniten käytettynä kielenä oli C#, mutta keskeisessä roolissa olivat myös erinäiset skriptikielet, kuten Windowsin komentolinja, powershell ja node.js. Muitakin kieliä olisi voitu käyttää, mutta valitut kielet vaikuttivat tehtävään sopivilta, ottaen huomioon kehitettävän kohteen ja sen kehitysympäristön.

3.2 Kirjastot ja sovelluskehykset

Kirjastoja ja sovelluskehyksiä tarvittiin tutkimuksen toteuttamiseen. Käytettyjä kirjastoja olivat MbUnit, TestApi ja XMLUnit ja Microsoft.Cci. Toteutuksessa käytettiin lisäksi itse rakennettuja kirjastoja, joita käsitellään tulososiossa.

MbUnit on generatiivinen testauskehikko, minkä päämääränä on tarjota käyttäjilleen korkean tason testifikstuureja, sekä poistaa kehittäjien tarve muokata testausrakenteen ydintä uusia testifikstuureja luodessa (de Halleux 2004). Vaikka de Halleuxin esittämä tieto on vanhaa, se osoittautui yhä tarkaksi ja käyttökelpoiseksi.

TestApi on yleishyödyllinen kirjasto ja API-kokoelma, joka sisältää erinomaisia toiminnallisuuksia testaukseen ja yleistyökalujen luomiseen. Se on kohdennettu .NET- ja Win32-pohjaisille sovelluksille. (TestApi - a library of Test APIs 2011.)

XMLUnit on Javalle ja .NETille kohdennettu kirjasto, mikä tarjoaa työkaluja xml-tiedostojen varmentamiseen ja vertailuun. Keskeisin ominaisuus siinä on eromoottori, millä käyttäjä saa täyden kontrollin kohteiden vertailuun, mitkä ovat hänelle tärkeitä. (XMLUnit n.d.)

Microsoft Cci on kirjastokokoelma ja API. Sen avulla voidaan tehokkaasti analysoida, tai muokata, .NET-kokoonpanoja, moduuleja ja pdb-tiedostoja. (Common Compiler Infrastructure - - n.d.)

3.3 Rajapinnat ja ohjelmat

Tutkimuksessa käytettiin useita ohjelmia ja rajapintoja, joiden avulla tuotettiin tehtävänannon mukainen rakenne. Käytössä olivat AutoCAD, Gallio, MSIL Disassembler, Visual Studio, NuGet ja VirtualBox.

AutoCAD on graafinen mallinnussovellus, jota käytetään työkaluna rakennusten pohjapiirustusten piirtämiseen sekä työstökappaleiden mallintamiseen teollisuudessa (What is AutoCAD? n.d.).

Gallio on avoimeen lähdekoodiin perustuva testiautomaatioalusta .NET-testauskehikoille. Se kehitettiin toimimaan yhteisenä alustana, jotta jokaiselle testikehikolle ei tarvitsisi erikseen luoda rajapintoja sekä työkaluja, ja käyttää aikaa niiden rakentamiseen. Testauskehikoilla on kuitenkin sama pääpiirteinen toiminnallisuus keskenään. (Tenhundfeld 2008.)

MSIL Disassembler (ildasm.exe) on Microsoftin tarjoama .NET kokoonpanojen purkaja, joka kääntää konekielen takaisin luettavaan muotoon. Ei ole taattua, että MSIL Disassembler kykenee purkamaan käännetyn koodin täydellisesti takaisin alkuperäiseen luettavaan muotoonsa. (Maksimovic 2012.)

Visual Studio on integroitu kehitysympäristö, jolla voi kirjoittaa koodia tarkasti, ja tehokkaasti. Visual Studio tarjoaa toiminnallisuuden mm. koodin refaktorointiin, debuggaamiseen, testaamiseen ja moneen muuhun kehittäjälle keskeiseen asiaan. (Visual Studio IDE n.d.)

NuGet on pakettimanageri .NETille. NuGet työkalut tarjoavat mahdollisuuden jakaa paketteja yleiseen käyttöön, ja keinon hakea jaossa olevia paketteja. (What Is Nuget? n.d.) NuGet on siis alusta käännetyn koodin jakamiselle.

VirtualBox on virtualisointituote yritys- ja kotikäyttöön. Tuotteen kotisivuilla väitetään, että se on ainoa ammattilaisratkaisu, joka on vapaasti saatavilla avoimeen lähdekoodiin perustuvana ohjelmistona, joka on GNU General Public Version 2 -

lisenssin alla. VirtualBox toimii Windowsilla, Linuxilla, Macintoshilla ja Solariksella. (Welcome to VirtualBox.org! n.d.) Väitteet eivät sisällä päiväystä, joten tiedot voivat olla osittain vanhentuneita. VirtualBoxia ajettiin kuitenkin onnistuneesti Windows-alustalla, jossa virtualisoitiin Ubuntu 16.04 ja Windows 10 käyttöjärjestelmät.

4 Testauskokoonpanon hahmottelu

4.1 Projektin kääntäminen

Projektia, jota varten automaatiokokoonpanoa rakennettiin, kehitettiin Visual Studiolla, joten oli järkevä tarkastella saman kehitysympäristön käännöstyökaluja projektin kääntämiseen. Microsoftin verkkosivuilla tarjotaan ohjeita MSBuildin käyttämiseen komentolinjalta (MSBuild Command-Line Reference n.d.). Esimerkkitapauksessa MSBuildia voitiin komentaa seuraavasti (ks. kuvio 2).

```
MSBuild.exe MyProject.proj /t:rebuild
```

Kuvio 2: MSBuildin esimerkkikomento

Komennolle saattoi antaa vipuja, joilla pystyi määrittelemään aikaisempien käännösten hävittämisen ennen seuraavaa käännöstä, jotta voitiin olla varmoja käännösten olevan tuoreita (MSBuild Command-Line Reference n.d.). Kokeiluissa projekti onnistuttiin kääntämään komentolinjalta MSBuildin avulla, mutta piti vielä varmistaa, että ennen käännöstä palautettaisiin tarvittavat riippuvaisuudet projektiin, jotka eivät kulkeneet versionhallinnan mukana.

Microsoft tarjoaa, .NET ohjelmistojen paketinhallintaa varten, oman sovelluksensa - NuGetin. Se määrittelee, kuinka paketit luodaan, kuinka niitä käytetään ja tarjoaa mahdollisuuden palauttaa projektin käyttämät paketit MSBuild-työkaluketjussa, ennen kääntämistä. (Brockschmidt & Myers 2018.) NuGet saatiin Microsoftin sivuilta exe-tiedostona, ja sitä saattoi käyttää komentolinjalta. Sille tarvitsi vain määritellä komento ja kohdeprojekti (ks. kuvio 3).

```
C:\Nuget\nuget.exe restore Project.sln
```

Kuvio 3: NuGetin käyttö

4.2 Visuaalinen tulosten todennus testeissä

Liitännäissovelluksen integraatiotestikohteeksi määritellyn tuotoksen, eli grafiikan, testaamista lähestyttiin visuaalisilla testeillä. Visuaaliset testit tarjosivat keinon todentaa tuotokset tarkasti, ja ne kykenivät tuottamaan kehittäjälle selkeitä kuvia vertailukohteiksi testiajosta, joiden perusteella oli helppoa nähdä mikä meni vikaan. Jos ongelmaa olisi lähestytty numeropohjaisesti, tekemällä arvovertailuja koordinaattien ja objektien värien perusteella, olisi saattanut syntyä tilanteita, joista olisi vaikea tulkita mikä testiajossa meni pieleen.

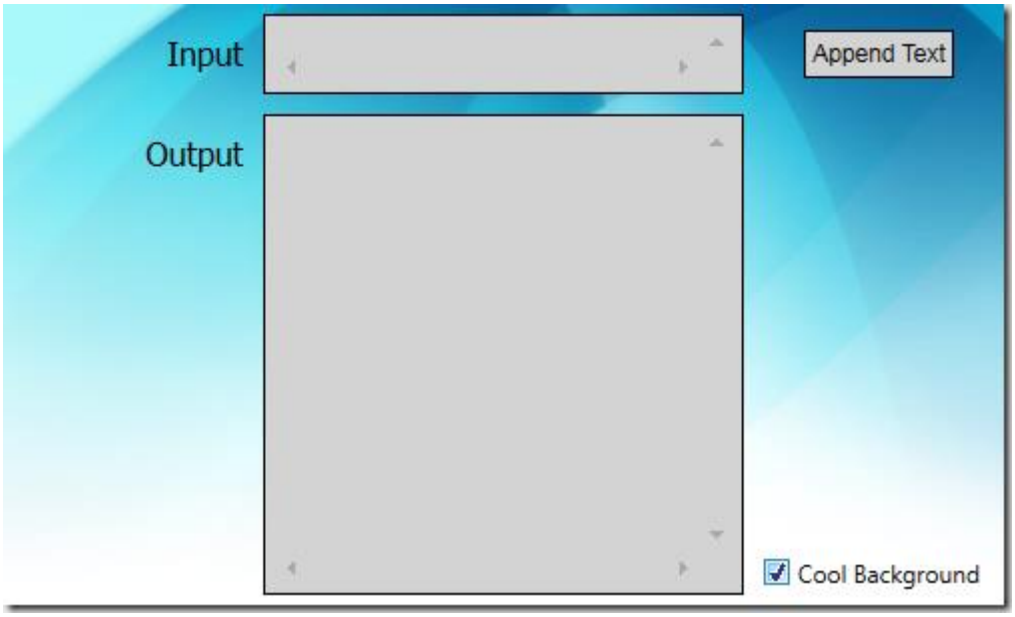
Bay (2016) kirjoittaa, että visuaaliset testit ajavat ennalta määrätyt toiminnot määrättyjen asetusten ollessa päällä ja ottavat kuvakaappauksen piirretystä näkymästä. Testi vertaa kuvakaappausta aikaisemmin validoituun vertailukuvaan ja eroavaisuuksien perusteella joko hyväksyy tai hylkää tuloksen. (Bay 2016.)

Manolov (2009) puolestaan toteaa, että visuaalisten tulosten vertailuun perustuvaa automaatiotestausta tulisi välttää, jos se on vain mahdollista. Jos sitä kuitenkin päädytään käyttämään, sen kanssa on oltava varovainen useasta eri syystä. Ohjelmien käyttöliittymillä on taipumus muuttua kehitysprosessin aikana, jolloin kehittäjä saattaa joutua päivittämään useita automaation ajamia testejä päivittäisellä tasolla. Lisäksi näytönohjainten tuottamassa kuvassa on pieniä eroavaisuuksia eri käyttöjärjestelmillä ja .NET-versioilla, minkä seurauksena testit mitkä onnistuivat yhdessä järjestelmässä voivat epäonnistua toisessa. (Manolov 2009.)

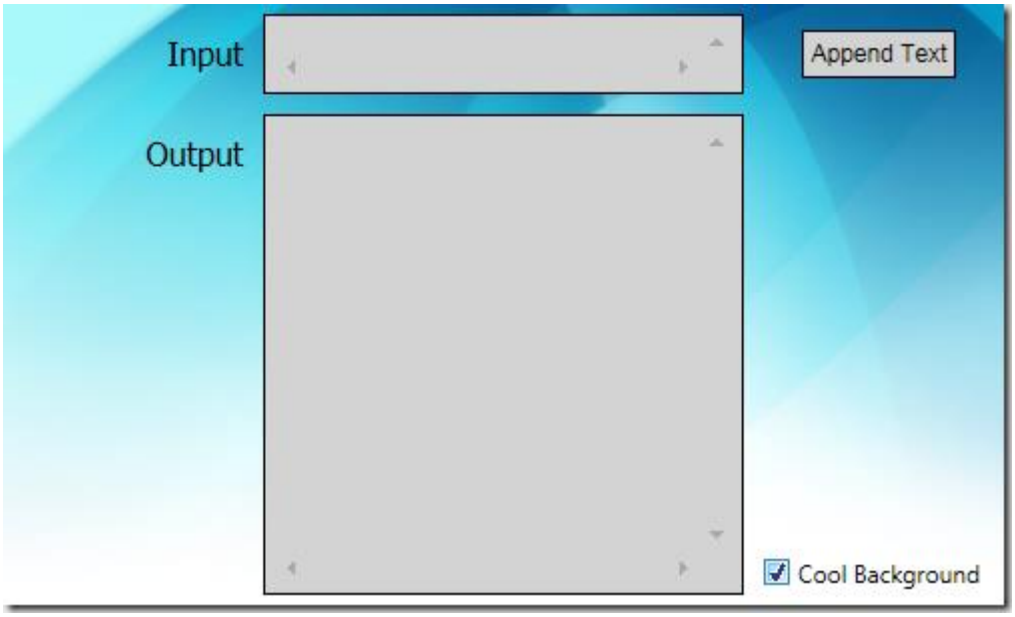
Myös Bay (2016) huomauttaa, että visuaaliset automaatiotestit ovat hauraita. Eri alustat, laitemallit ja näytönohjaimet voivat tuottaa erilaisia tuloksia. Jotta saadaan konsistentteja tuloksia, testit on suoritettava testifarmilla, jonka kokoonpanon muuttumattomuudesta voidaan olla varmoja. (Bay 2016.)

TestApin toiminnallisuus kykenee hoitamaan kuvien kaappaamisen, niiden vertailun ja tallentamisen. Odotetusta tuloskuvasta ja testissä tuotetusta kuvasta voidaan tehdä risteytyskuva, jossa kaikki kuvien identtiset pikselit kumoavat toisensa muodostaen mustia pikseleitä risteytyskuvaan. Tuloksena on siis sitä mustempi kuva, mitä lähempänä kuvat ovat toisiaan. (Manolov 2009.) Kun verrataan odotettua kuvaa

(ks. kuvio 4) ja testissä kaapattua kuvaa (ks. kuvio 5), on käytännössä mahdotonta huomata niiden eroja toisistaan ihmissilmällä.

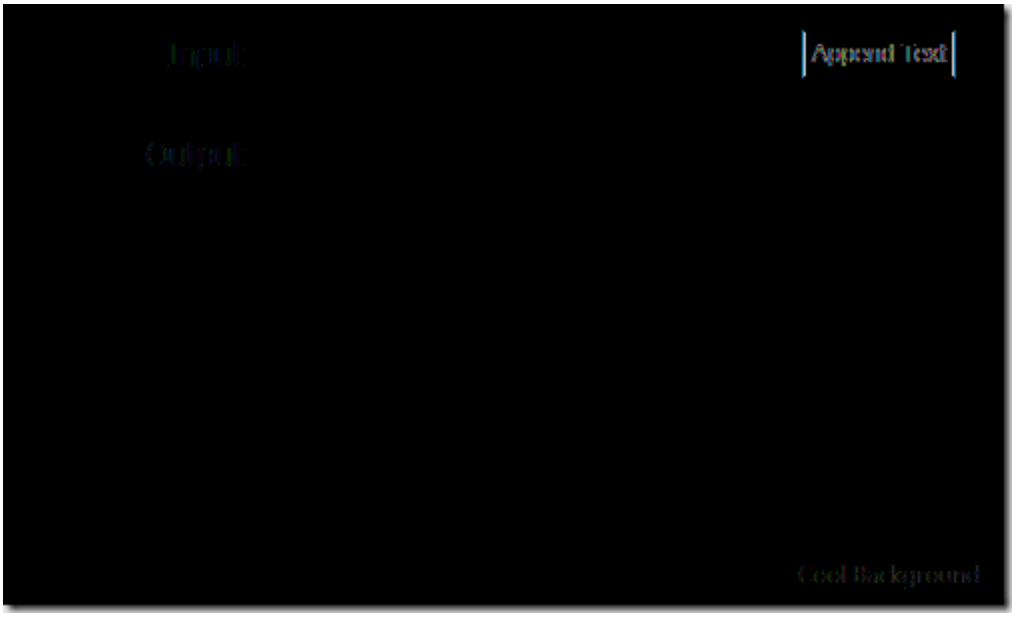


Kuvio 4. Odotettu kuva



Kuvio 5. Testissä kaapattu kuva

Kuvilla on kuitenkin selkeitä eroavaisuuksia, kun katsotaan risteytyskuvaa (ks. kuvio 6). Identtiset pikselit ovat täysin mustia, ja toisistaan poikkeavat pikselit vaalenevat erojen suuruuden mukaan. Eroja ilmenee eritoten tekstin kohdalla. Muunnetussa risteyskuvassa täysin mustat pikselit on muunnettu vaaleanpunaisiksi (ks. kuvio 7), jotta kaikki erot näkyvät selkeästi.



Kuvio 6. Risteytyskuva

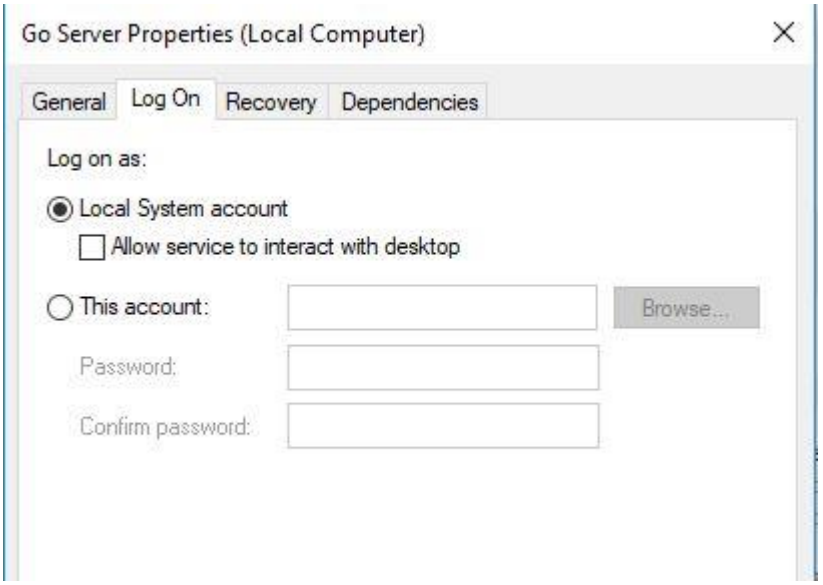
Kuvio 7. Muunnettu risteytyskuva

TestApissa voidaan käyttää toleranssikarttoja (ks. kuvio 8), joilla voidaan rajata ongelma-alueet noudattamaan erilaisia toleransseja. Lisäksi voidaan antaa toleransseja, kuinka monta pikseliä otettu kuva saa korkeintaan poiketa odotetusta kuvasta ja kuinka suuria sävyeroja poikkeavissa pikseleissä saa olla (Manolov 2009).

Kuvio 8. Toleranssikartta

Kun toiminnallisuutta testattiin, kävi ilmi, että mikäli haluaa ottaa kuvia automatisoidusti Windows-työpöydästä tai, siinä näkyvästä sovelluksesta, automaatio ei voi pyöriä palveluna – ellei testissä käsitelty sovellus tarjoa jotain erityistä toiminnallisuutta sen toteuttamiseen vaatimatta oikeuksia näytönohjaimeen.

Keith Dahlby (2011) huomauttaa artikkelissaan, että ongelma voidaan kiertää ajamalla toiminnallisuus käyttäjän alla palvelun sijaan, mutta ei pidä siitä, koska se on vaikeampaa ylläpitää. Hän ei kuitenkaan kerro, miksi ylläpito on vaikeampaa. Toiseksi vaihtoehdoksi hän tarjoaa palvelun ajamista lokaalilla järjestelmätilillä, jolle annetaan erikseen oikeudet vuorovaikuttaa työpöydän kanssa. (Dahlby 2011.) Windows 10 -järjestelmässä oikeuksien konfigurointi löytyi palvelun ominaisuuksista (ks. kuvio 9).



Kuvio 9: Palvelun oikeudet työpöytään

Alustavissa testeissä oikeuksien lisäämisestä huolimatta syntyi virhetilanne. Vaihtamalla palvelun käyttäjätilin alle virhe saatiin kierrettyä, ja kuvakaappaukset taltioitua.

AutoCAD tarjoaa natiivisti toiminnallisuuden kuvien ottamiseen piirustuksesta (DocumentExtension.CapturePreviewImage Method n.d.). Tätä toiminnallisuutta käyttämällä voitiin, ilman erillisiä oikeuksia työpöytään tai näytönohjaimeen, generoida kuva piirustuksen sisällöstä. Erillistä oikeuksien konfiguraatioita palvelulle ei siis ollut välttämätöntä toteuttaa, ellei tarvittu kuvakaappauksia käyttöliittymästä.

4.3 Testien ajaminen AutoCAD-liitännäissovellukselle

Iyer (2009) kirjoittaa omassa artikkelissaan, että AutoCADin testit on ajettava samassa säikeessä AutoCADin kanssa (Iyer 2009). Havainnot omissa kokeiluissa olivat linjassa Iyerin kirjoitusten kanssa integraatiotestien osalta, sillä AutoCAD-kirjastoja ei voitu hyödyntää AutoCADin säikeen ulkopuolella. Tämä ei ollut keskeistä yksikkötesteille, jotka ajettiin irrallaan kokonaisuudesta. Kuitenkin integraatiotestejä varten täytyi ratkaista, kuinka testit voitaisiin ladata AutoCADin säikeeseen.

Ratkaisu ongelmaan löytyi Galliosta, josta Iyer (2009) jatkaa seuraavasti. Gallio tarjoaa MbUnitille mahdollisuuden ajaa testit samassa säikeessä AutoCADin kanssa lataamalla välikappaleen AutoCADiin, jolla se lataa ja käynnistää testit samaan

säikeeseen. Galliolle ei tarvitse merkitä AutoCADin sijaintia, sillä se löytää sen rekisteriavaimella itse. (Iyer 2009.)

Gallion lähdekoodin perusteella oli nähtävissä, että Gallio lataa AutoCADiin oman liitännäissovelluksensa, joka sisältää komennon testien kiinnittämiselle. Kokeiluissa todettiin, että komento ajettiin AutoCADin käynnistyttyä, ja testikehikon ajo käynnistettiin dokumentin kontekstissa (Gallio/mbunit-v3 n.d.).

Koska MSTest oli ennestään tuttu testikehikko, sitä kokeiltiin MbUnitin sijaan, jotta minimoitaisiin opeteltavien moduulien määrä testirakennetta luodessa. Valitettavasti testien ajaminen MSTestillä osoittautui ongelmalliseksi, sillä Galliolla ei kyennyt paikantamaan MSTestiä testikoneelta. Ongelmaa yritettiin korjata muokkaamalla .NET-kehikon konfiguraatioita, Gallion konfiguraatioita, ympäristömuuttujia ja rekisteriavaimia, mutta tuloksetta. Kun keinot oli käytetty loppuun, eikä tuloksia syntynyt, siirryttiin MbUnittiin. Alustavat koetestit luotiin MbUnitilla, käännettiin ja ajettiin onnistuneesti Galliolla. Testit käynnistettiin antamalla Gallion komentorivityökalulle komento (ks. kuvio 10).

```
SET GALLIO_RUNTIME_PATH=C:\Program Files\Gallio\bin

"C:\Program Files\Gallio\bin\Gallio.Echo.exe"
"UnitTestProject\bin\release\UnitTestProject.dll" /r:AutoCAD /rt:html
```

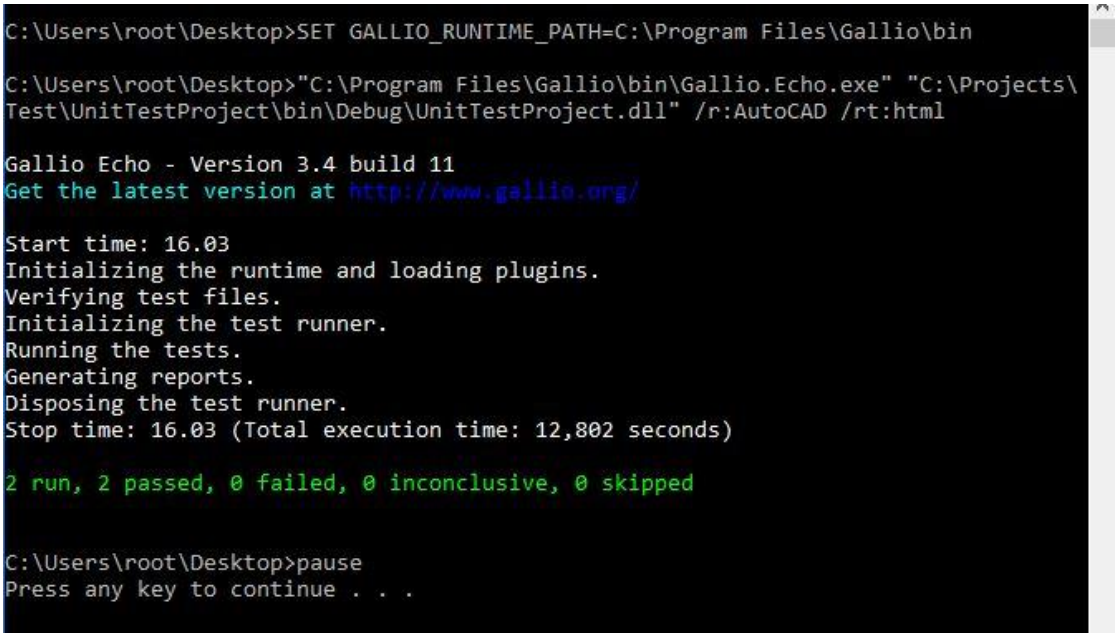
Kuvio 10. Gallion ajokomennot

Ensimmäisessä komennossa määriteltiin Gallion ajonaikainen sijainti. Tämä oli välttämätöntä testien ajolle, sillä Gallio ei kyennyt käynnistämään testejä AutoCADissa ilman sijainnin määrittelyä. Toisessa komennossa käynnistettiin itse Gallio ja määriteltiin testien sijainti, runnerin tyyppi ja tulosraporttien muoto. Raporttien muoto ei ole pakollinen, kuten voidaan nähdä geneerisestä Gallion käynnistyskomennosta (ks. kuvio 11).

```
Gallio.Echo.exe [TestLibrary] [TestRunnerType] [Additional options]
```

Kuvio 11. Gallion geneerinen käynnistyskomento

Komentojen ajon yhteydessä Gallio käynnisti AutoCADin, ajoi MbUnit-testit ja sammutti lopuksi AutoCADin (ks. kuvio 12). Testeissä käsiteltiin AutoCADin aktiivisena olevaa dokumenttia, ja todennettiin mahdollisuus käsitellä AutoCADin sisäistä dataa integraatiotestien aikana.



Kuvio 12. Gallion testiajo

Toisin kuin integraatioesteissä, yksikkötestien tapauksessa voitiin jättää runnertyyppi kokonaan määrittelemättä, jolloin Gallio valitsi käyttöönsä automaattisesti IsolatedProcess-runnerin.

Integraatiotestien tapauksessa kappaleita piti saada käänneltyä AutoCADissa, ja niistä piti saada kuvakaappauksia, mistä aiheutui vaikeuksia. AutoCADin näkyvin keino tiedostojen avaamiseen osoittautui asynkroniseksi. Se palautti dokumenttiobjektin käyttöön heti, mutta sen asettaminen aktiiviseksi ohjelmassa tuli viiveellä. AutoCADissa saattoi olla useita dokumentteja auki, joista jokainen oli omalla välilehdellään, mutta vain yksi saattoi olla kerrallaan aktiivisena.

Jos dokumentti ei ollut aktiivisena, kun sen näkymää käsiteltiin, se aiheutti AutoCADissa poikkeuksen. Tämä tilanne realisoitui testeissä aina, sillä testit etenivät liian nopeasti suhteessa tiedoston avaamisnopeuteen. Tyypillisessä tilanteessa tämä olisi helppo kiertää asettamalla testejä ajava säie odottamaan tiedoston avaamisen ajaksi, mutta tässä tilanteessa toimenpide pysäytti myös tiedoston avaamisen, sillä molemmat tapahtumat olivat samassa säikeessä. Asynkroninen tiedostojen avaaminen ei siis ollut vaihtoehto, jos samassa AutoCAD-instanssissa haluttiin avata useita dokumentteja.

Tämä ongelma voitiin, yhtenä vaihtoehtona, kiertää käynnistämällä oma AutoCAD-instanssi jokaista kuvaa kohden erikseen. AutoCAD otti käynnistyksen yhteydessä

kuvatiedoston polun vastaan, jolloin tiedosto asetettiin automaattisesti aktiiviseksi heti ohjelman käynnistyttyä (Startup switches for AutoCAD n.d.). Gallio osasi kiinnittyä jo olemassa olevaan AutoCAD-instanssiin, joten kuvia voitiin käydä läpi komentolinjan silmukassa, jossa jokaiselle kuvalle käynnistettäisiin oma AutoCAD, johon Gallio kiinnitettäisiin.

AutoCADilla on kuitenkin olemassa myös komento synkroniseen tiedostojen avaamiseen, mutta komennon edellytyksenä on sen suorittaminen koko ohjelman kontekstissa dokumentin kontekstin sijaan (AcApDocManager::appContextOpenDocument n.d.). AutoCAD-komentoja voidaan ajaa ohjelman kontekstissa, asettamalla komentoon CommandFlag.Session -parametri (Autodesk.AutoCAD.Runtime.CommandFlags Enumeration n.d.).

Koska Gallio on avoimeen lähdekoodiin perustuva projekti, siitä voitiin tehdä pienillä muutoksilla näiden tietojen perusteella oma käännös, jossa testikehikko käynnistettiin koko ohjelman kontekstissa dokumentin kontekstin sijaan (ks. kuvio 13). Näin onnistuttiin toteuttamaan integraatiotestit yhdessä AutoCAD-instanssissa.

```
/// <summary>
/// Creates an <c>AcadTestDriver</c> instance and registers it as a service.
/// </summary>
/// <remarks>
/// <para>
/// This command blocks on the calling thread until <c>Shutdown</c> is called.
/// </para>
/// </remarks>
[CommandMethod("CREATEENDPOINTANDWAIT", CommandFlags.Session)]
0 references
public static void CreateEndPointAndWait()
{
    string ipcPortName;
    if (!GetIpcPortName(out ipcPortName))
        return;

    Guid linkId;
    if (!GetLinkId(out linkId))
        return;

    ShimWithLoader.Run(ipcPortName, linkId);
}
```

Kuvio 13: Muutettu Gallion koodi

Kun käännös oli tehty ja integraatiotestit oli onnistuneesti ajettu, ryhdyttiin toteuttamaan yksikkötestejä. Valitettavasti oman Gallio-käännöksen toteuttamisessa

oli tapahtunut regressiota – käännös oli rikkonut muut runnerit, mukaan lukien IsolatedProcess-runnerin, jolla yksikkötestit oli tarkoitus ajaa. Runnerin korjaaminen oli keskeistä, jotta yksikkötestit saatiin eriytettyä integraatiotesteistä.

Gallion virheviestien perusteella kävi ilmi, että pdb-tiedostojen metadatan lukemiseen oli tullut ongelma. Gallion käyttämä Microsoft.Cci -kirjasto, jolla käsiteltiin pdb-tiedostoja, heitti virheilmoituksen, josta ilmeni, että kirjastossa ei ollut riittävää toiminnallisuutta tiedostojen metadatan käsittelyyn.

Nimimerkki Remco (2016) kirjoittaa aiheesta seuraavasti NCrunchin foorumeilla. Gallio referoi vanhaa Microsoft.Cci kokoonpanoa, joka ei osaa käsitellä uudempaa pdb-tiedostoformaattia. Gallion kehitysaikana Visual Studio 2015, eikä Roslyn compiler, olleet vielä olemassa, ja kun Roslyn compiler esiteltiin, Microsoft teki pienen muutoksen pdb-tiedostoformaattiin sisällyttääkseen siihen metadataa debuggauksesta. Tämä käytännössä rikkoi kaikki aikaisemmat Microsoft.Cci versiot. (Remco, 2016)

Kun Galliosta tehtiin käännös uudemmalla Visual Studiolla, käännöksessä tuotetut pdb-tiedostot olivat myös uudempaa formaattia, eikä Gallion mukana tullut Microsoft.Cci kirjasto kyennyt käsittelemään niitä. Microsoft.Cci:n uudemmassa versiossa oli otettu huomioon muuttunut tiedostotyyppi (Microsoft/cci n.d.). Ongelma saatiin korjattua integroimalla uudempi versio Gallioon. Näillä muutoksilla Gallio kykeni ajamaan integraatiotestit ja yksikkötestit liitännäissovellukselle erikseen, ja toivottu toiminnallisuus kyettiin niiden osalta toteuttamaan.

4.4 CI/CD-työkaluvaihtoehdot

4.4.1 Jenkins

Najjar kertoo Jenkinsin olevan ilmainen, Java-pohjainen, työkalupaketti jatkuvan integraation toteuttamiseen. Sitä käytetään yleisesti Java-pohjaisissa projekteissa, mutta se soveltuu erinomaisesti myös .NET-pohjaisiin projekteihin, sillä Jenkins kykenee ajamaan MSBuild-skriptoja ja toimii useiden .NET-versionhallintajärjestelmien kanssa. Jenkins omaa Najjarin mukaan myös erittäin aktiivisen liitännäiskehittäjäkunnan. (Najjar n.d.)

Edurekan verkkosivuilla kirjoitetaan, että Jenkins perustuu avoimeen lähdekoodiin. Sitä käytetään kääntämään ja testaamaan ohjelmistoja haluttuina ajanhetkinä, ja se mahdollistaa jatkuvan integraation lisäksi jatkuvan toimituksen, suurella määrällä testaus- ja julkaisuteknologioita. (What is Jenkins? 2016.)

Jenkinsin omilla verkkosivuilla puolestaan kerrotaan seuraavaa. Pipeline-liitännäisen avulla käyttäjät voivat implementoida projektin koko putken tiedostoon, sisältäen kääntämisen, testaamisen ja julkaisun, jolloin automaatioputkea kohdellaan osana koodia versionhallinnassa. Lisäksi Jenkinsin verkkosivuilla tarjotaan demoa jatkuvasta toimittamisesta Jenkinsiä käyttäen. (Pipeline as Code with Jenkins n.d.)

Jenkins wikisivuilla tiedotetaan, että Jenkins kykenee toimimaan tehtävien koordinoijana sekä tekijänä, mutta vaihtoehtoisesti se voi delegoida työtaakkaa muille koneille, jolloin sen tarvitsee keskittyä vain koordinoimaan tehtäviä. Hyödyntämällä ”master/agent” moodia, jossa Jenkins-palvelin toimii koordinaattorina, ja hyödyntää agenttikoneita resursseina kääntämis- ja testikäytössä, voidaan vähentää koordinoivan palvelimen työtaakkaa huomattavasti ja siten hallinnoida useita projekteja samaan aikaan. (Kawaguchi & Soref 2018.)

Ejaz (2014) listaa melko kattavasti Googlen foorumeilla, Jenkinsin ja GoCDn välisistä eroavaisuuksista käymässään keskustelussa, Jenkinsin hyviä ja huonoja puolia. Hyviin puoliin hän luettelee liitännäisillä laajennettavuuden, integroinnin versionhallintaan, ensiluokkaisen ja laajan tuen useille työkaluille. Lopuksi hän lisää, että Jenkins on hyvin tunnettu ja laajan yhteisön tukema. Erityisinä etuina GoCDhen verrattuna Ejaz (2014) mainitsee joustavan asennuksen sekä aloittamisen helppouden master-only -moodin takia. Huonoista puolista hänellä oli myös mainittavaa. Jenkinsin mallinnuskyky ja visualisointi on heikkoa verrattuna GoCDhen, ja Jenkinsissä ei ole Fan-in tukea, jolloin Jenkins yrittää kääntää myös yhteensopimattomat kokonaisuudet tuhlaten aikaa. Ejaz (2014) huomauttaa, ettei hän ole Jenkins-ammattilainen, joten hänelle ei ole tuttua, kuinka hyvin laajennuksilla saadaan paikattua Jenkinsin vajavaisuuksia. (Ejaz 2014.) Ejazin listaamat väitteet perustuvat melko vanhaan tietoon, ottaen huomioon digitaalisen maailman kehitysnopeuden. Jenkinsin sekä GoCDn kehittäjillä on ollut siten aikaa tehdä kehitystyötä puutteiden saralla. Tästä syystä tiedot saattavat olla puutteellisia, tai epätarkkoja.

Jenkinsin kotisivujen mukaan Jenkins tarjoaa APIn, jonka kautta voidaan tehdä kyselyitä Jenkinsin tilasta, käynnistää putkia tai luoda uusia tehtäviä (Remote access API, n.d.). NCloudin mukaan Jenkins tukee myös agenttien kontittamista (Considering TeamCity for Continuous delivery? Here is what you need to know. n.d.).

Jenkinsin koeajo

Kun Jenkinsiä asennettiin käyttövalmiuteen vertailutarkoituksessa, asentaminen oli melko suoraviivaista työkalun itsensä osalta. Putken pystyttäminen oli hankalampaa kuin ohjelmiston asennus.

Kokoonpanossa oli kaksi virtuaalikonetta. Ensimmäinen piti sisällään Jenkinsin, sekä tarvittavat kääntö- sekä testausohjelmistot. Toinen kone toimi ulkopuolisena laukaisimena putken käynnistämiseen. Viestin lähettäminen ulkoverkossa toimivasta GitHubin versionhallintapalvelimesta yrityksen sisäverkossa toimivalle Jenkinsille testausvaiheessa nähtiin ylimääräisenä hankaluutena, joka päätettiin ohittaa luomalla virtuaalikone putken laukaisua varten erikseen. Näin saatiin todennettua putken käynnistyminen ulkopuolisesta lähteestä saapuvilla webhookilla.

Toimivan CI/CD-putken tekemiseen tarvitsi hakea joitain liitännäisiä. GitHub-liitännäinen tarvittiin versionhallintaliitoksen luomiseen, ja multibranch pipeline -liitännäinen tarvittiin varsinaisen putken luomiseen, jossa oli vaiheet käännökselle, testaamiselle ja julkaisulle. Liitännäisiä oli saatavilla valtavasti, eikä niitä ollut suurelta osin dokumentoitu kovinkaan laajasti, lukuun ottamatta mainintoja käyttötarkoituksista.

Jenkins konfiguroitiin hakemaan kohteen koodi GitHubin repositoriosta, GitHub-liitännäisen avulla, aina webhook-ilmoituksen saapuessa. Jenkins kykeni käynnistämään käännöksen, testit ja julkaisun. Multibranch pipeline -liitännäisen mukana tuli tehtävien visualisoitu esitys Jenkinsin käyttöliittymässä, josta selvisi niiden tila riittävällä tarkkuudella. Työtaakan jakamista ei testattu master-agent -moodilla, vaan kokeessa käytettiin master-only moodia.

Jatkuvan julkaisun putkikonfiguraatio voitiin sisällyttää versionhallinnassa kulkevaan tekstitiedostoon, jolloin koko projekti voitiin testausautomaatiota myöten keskittää yhteen paikkaan. Tiedostoon määriteltiin putken vaiheet, ja siihen saattoi halutessaan asettaa muuttujia ajoa varten.

Ongelmien ilmetessä syyt sai kaivettua lokeista, mitkä kertoivat kattavasti putken toiminnasta. Loki ei ollut kuitenkaan valitettavasti reaaliaikainen. Testien osalta tarkemmat tiedot piti kaivaa testiraporteista. Raportit voitiin lisätä artefakteina testiajon tuloksiin, muokkaamalla versionhallinnassa mukana kulkevaa putken määritelmää.

Virallinen dokumentaatio on melko kattava, ja sitä kautta sai suuren osan putkea koskevista ongelmista ratkottua. Paljon aikaa kului silti StackOverflowta selatessa. Epämieluisaksi toimintamalliksi Jenkinsin käytössä muodostui liitännäisten välttämättömyys, sillä Jenkins ei itsenäisesti kyennyt toteuttamaan edes yksinkertaista webhookkia. Liitännäiset saattoivat olla selkeästi dokumentoituna, mutta suuressa osassa dokumentointi oli vajavaista, ja käyttö siksi hankalaa. Mikäli Jenkinsillä toteutetaan suurempaa kokonaisuutta, niin tuloksena saattaa olla paketti, mikä pyörii lähinnä firman sisäisen hiljaisen tiedon voimalla, ellei sitä dokumentoida erikseen sisäisesti.

Tämän alustavan kokemuksen perusteella voitiin sanoa, että Jenkins kykeni hoitamaan siltä vaaditut toiminnot. Jenkinsistä kuitenkin paistoi läpi sen ikä, heti alusta saakka, ja siitä jäi kuva vanhaksi käyneenä rakennelmana, jonka vajavaisuuksia on koitettu paikata lisäilemällä ominaisuuksia jälkikäteen. Ikä toi kipujen lisäksi kuitenkin mukanaan myös viisautta, mikä näkyi Jenkinsin toiminnallisuuden joustavuudessa.

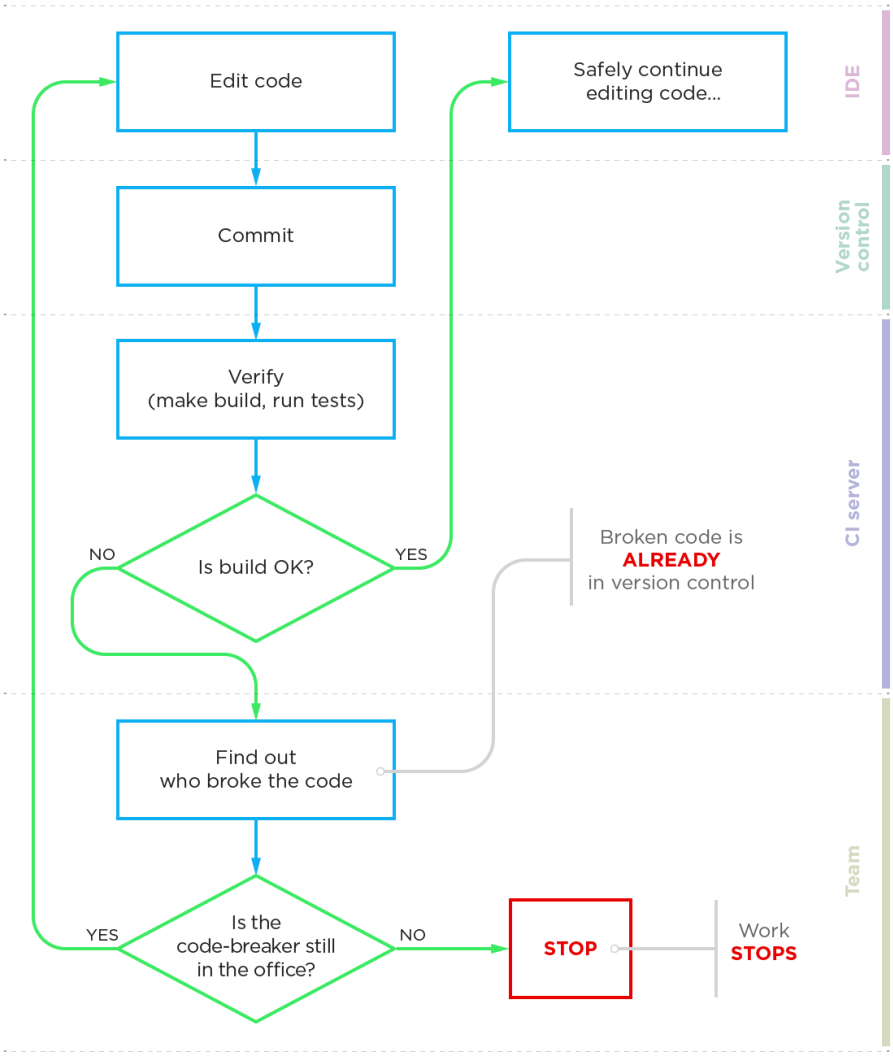
4.4.2 TeamCity

JetBrainsin omilta sivuilta käy ilmi, että TeamCity on kaupallinen tuote. Sillä on kuitenkin olemassa ilmaiseditio, jossa kääntökonfiguraatiot on rajoitettu kahteenkymmeneen, ja kääntöagentit kolmeen. (Buy TeamCity n.d.)

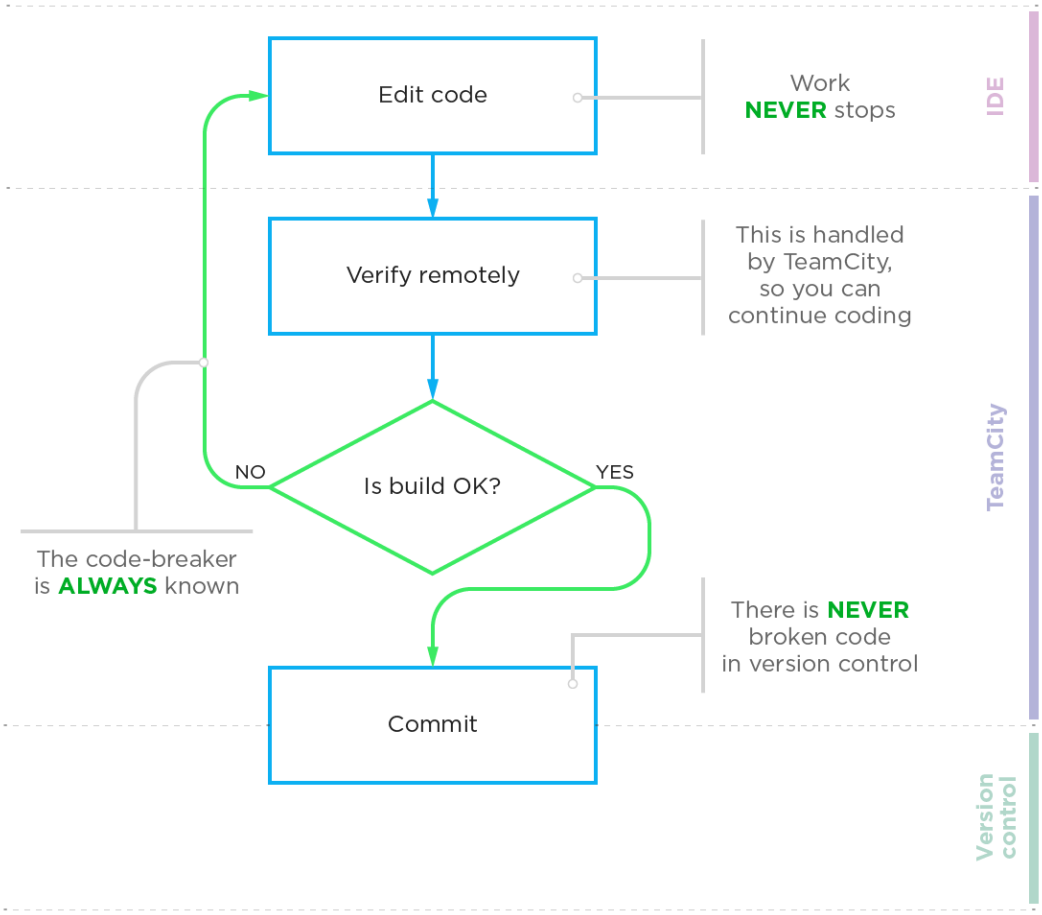
UpGuardin verkkosivuilla kirjoitetaan, että myös TeamCity on Java-pohjainen jatkuvan integraation toteuttamistyökalupaketti, joka markkinoi itseään helppokäyttöisenä jopa niille, joille jatkuva integraatio on uutta. Se toimii Linux- sekä Windows-pohjaisissa käyttöjärjestelmissä, ja integroituu Visual Studion sekä Eclipsen kanssa. (TeamCity vs Jenkins for Continuous Integration 2017.)

NCloudsin artikkelissa mainitaan, että DevOps insinöörien ja ylläpitäjien olisi viisasta testata TeamCityä kartoittaessaan vaihtoehtoja. Artikkelissa lisätään, että etummaisena TeamCityn viehättävissä ominaisuuksissa on sen ylivertainen integraatio versionhallinnan kanssa. Se antaa mm. käyttäjän määritellä yksityiskohtaisen yhteyden versionhallintajärjestelmään, rajata mitä versiohaaroja järjestelmän tulee seurata ja milloin käännös laukaistaan. Erityisenä vahvuutena TeamCity kykenee testaamaan kommitin toimivuuden, ennen kuin se viedään versionhallintaan. (Considering TeamCity for Continuous delivery? Here is what you need to know 2017.)

JetBrainsin tarjoaman kaavioparin perusteella on helppo nähdä ennalta testatun kommitin potentiaalinen arvo kehitystiimin sisällä (Pre-Tested Commit - - n.d.). Ensimmäisessä kaaviossa nähdään, kuinka versionhallintaan päätynyt virheellinen koodi saattaa hidastaa, tai jopa pysäyttää, muiden tiimiläisten työn (ks. kuvio 14). Toisessa kaaviossa on nähtävissä, kuinka työntekoa halvaannuttavia virhetilanteita ei pääse syntymään tiimin sisälle, kun kommitin koodi on ennalta testattu (ks. kuvio 15).



Kuvio 14: Ennalta testaamattoman kommitin toimintakaavio



Kuvio 15: Ennalta testatun kommitin toimintakaavio

NCloud kirjoittaa lisäksi seuraavasti. TeamCity ei tue sellaisenaan kontittamista, mutta liitännäisten avulla nekin voidaan ottaa käyttöön. Käyttöliittymästä on tehty intuitiivinen käyttää, ja dokumentaatio on hyvin ylläpidetty sekä ajan tasalla. Asennus on helppoa ja kustomoitavaa, ja lokaalien tiedostojen sijaan käyttäjällä on optio tallentaa haluttuun tietokantaan. (Considering TeamCity for Continuous delivery? Here is what you need to know 2017.)

Samassa artikkelissa viitataan TeamCityn ottaneen käyttöönsä toimintamallin, jossa myös osaa CI/CD-putken elementeistä hallinnoidaan, ja konfiguroidaan, versionhallinnan lävitse kulkevalla koodilla. Tällöin myös putken konfiguraatio on alisteinen samoille käytännöille kuin itse ohjelmistokoodi. Lisäksi nClouds huomauttaa, että TeamCityn putkessa kulkevien artefaktien manageroinnin ja tarkkailu on tehty helpoksi, ja TeamCity tukee integraatiota pilvipalveluihin AWS:n, Azuren ja VMWaren osalta. (Considering TeamCity for Continuous delivery? Here is what you need to know 2017.)

JetBrainsin confluence-sivuilla on ohje TeamCityn REST APIn käyttöön. Ohjeiden perusteella API mahdollistaa tietyn asteisen käännösten hallinnoinnin ja niiden tilojen selvittämisen. (Alexandrova & Yargo 2018.) Ohjeet ovat tuoreita, joten ne ovat todennäköisesti ajan tasalla ja tarkkoja.

TeamCityn koeajo

TeamCityn käyttäminen oli helpompaa suhteessa Jenkinsiin. Käyttöliittymä oli selkeä, ja TeamCity toimi käännöksen ja testien osalta moitteetta oikein konfiguroituna. Gallion raportit sai helposti TeamCityn käyttöliittymän puolelle näkyviin artefakteina, kun ajon oli kertaalleen suorittanut ja kansiorakenne oli luotu.

Jeff Brown (2008) kirjoittaa blogissaan, että Gallio sisältää myös liitännäisen TeamCityyn, jolla se kykenee lähettämään TeamCitylle tietoja ajettavista testeistä. Blogissa mainitaan, kuinka Gallion versiossa 3.04 liitännäinen voidaan aktivoida vivulla, mutta kokeilujen perusteella versiossa 3.4 vipua ei ole enää olemassa (ks. kuvio 16). (Brown 2008.)

```
C:\Users\root\Desktop>SET GALLIO_RUNTIME_PATH=C:\Program Files\Gallio\bin

C:\Users\root\Desktop>"C:\Program Files\Gallio\bin\Gallio.Echo.exe" "C:\Projects\Test\
UnitTestProject\bin\Debug\UnitTestProject.dll" /r:AutoCAD /rt:html /e:TeamCityExtensio
n,Gallio.TeamCityIntegration
Error: Unrecognized argument '/e:TeamCityExtension,Gallio.TeamCityIntegration'.
```

Kuvio 16: Virheellinen vipu

Blogissa kuitenkin mainitaan, että Gallio on siirtymässä automaattiseen TeamCityn integraatioon (Brown 2008). Vaikutti siltä, että automaattinen integraatio oli jo toteutettu, mutta sitä ei oltu dokumentoitu blogiin.

Käännösagentit olivat pakollinen osa TeamCityn toimintaa. Agentit suorittivat putkissa määritellyt tehtävät, joita koordinoiva palvelu jakoi. Agentti oli asennettava, jotta kokonaisuus toimi. Vaikka TeamCity ei tukenutkaan master-only toimintoa, voitiin samalle palvelimelle asentaa myös työt tekevä agentti, jolloin tulos oli käytännössä sama. Halutessaan agentteja saattoi asentaa myös muille koneille, jolloin työtaakka jakautui.

TeamCitystä löytyi toiminnallisuus koodimuutosten jälkeiseen putken laukaisuun, mutta se ei toiminut webhookilla. Sen sijaan TeamCity kyseli

versionhallintajärjestelmältä jatkuvasti, onko kohderepositoriossa tapahtunut muutoksia ja toimi sen mukaisesti. Putkessa kyettiin kääntämään, testaamaan ja julkaisemaan kohteen koodi. Käyttöliittymästä kävi ilmi, mitkä putket olivat käynnissä ja mikä niiden tila oli. Lokeista sai putkien ajotiedot selville.

Kokeilujen perusteella todettiin, että TeamCity kykeni suorittamaan siltä vaaditun toiminnallisuuden tehtävänannon kontekstissa. Dokumentaatio oli melko selkeää ja ajan tasalla, kuten nCloudsissa kirjoitettiin. Suurin osa tarvittavasta toiminnallisuudesta oli integroituna valmiiksi työkaluun, eikä sitä tarvinnut korvata kolmannen osapuolen toteutuksilla, tai rakentaa itse. TeamCity vaikutti siten kilpailukykyiselle vaihtoehdolle.

4.4.3 GoCD

GoCD tarjoaa kotisivujensa mukaan rinnakkaista tai jonossa ajoa, helppoa riippuvuuksien hallintaa sekä tarpeen mukaan julkaisua. Sivuilla mainitaan, että GoCD perustuu avoimeen lähdekoodiin, ja tarjoaa paljon erilaisia ominaisuuksia, mutta sen todellinen voima on visualisaatio työnkulusta. Visualisaation avulla on helpompaa olla perillä siitä, mitä putkessa tapahtuu. (WHY GoCD? n.d.)

Bhatia (2017) kirjoittaa, että GoCD on ensiluokkainen konsepti jatkuvan toimituksen työkaluna ja tarjoaa intuitiivisen käyttöliittymän julkaisuputkien luomiseen Amazon Web Servicesissä. Hän ei kerro, kenelle tai kenen kanssa hän työskentelee, mutta lisää, että he testasivat GoCDtä validoidakseen ymmärrystään aiheesta. Testin jälkeen he siirtyivät käyttämään GoCDtä määrittelemään kaikkia julkaisu- ja toimitusputkiaan. (Bhatia 2017.)

Toisaalta Mo (2017) kirjoittaa tietoturvayhtiö SimpliSafella työskentelevän Arthur Burkartin kertoneen päinvastaisesti, että vaikka GoCD on erinomainen työkalu SimpliSafen tarpeisiin, se ei kuitenkaan tarjoa sen enempää jatkuvaa julkaisua kuin Jenkinskään. Mo (2017) attribuoi Burkartin sanomaksi myös, että GoCD tarjoaa erittäin vähän jatkuvan toimituksen osalta, on jäykkä, muutoslokit ovat epäselkeitä ja seuraavaan versioon päivittäminen on siksi epävarmalla pohjalla ja ongelmallista toteuttaa. (Mo 2017.) Mo (2017) ei tarjoa tiedonlähteestään muuta tietoa kuin nimen, joten on lukijan harkinnan alaista, kuinka luotettavana tätä lähdettä pitää.

Ejaz (2014) listaa melko kattavasti käymässään keskustelussa myös GoCDn hyviä ja huonoja puolia. Hyviin puoliin hän listaa automaation visualisaation arvovirtakartalla, hyvän mallinnuskyvyn kokonaiskuvasta, helpon rinnakkaisajettavuuden, jäljitettävyyden, tietoturvan, lupien hienosäädön käyttäjille, yhden klikin varmuuskopioinnin ja putkien eristämisen ryhmiin, agenttien eristämisen resursseihin ja molempien sitomisen yhteen ympäristöillä. Huonoihin puoliin Ejaz ei tarjonnut läheskään yhtä pitkää listaa. Hän mainitsee liitännäisten vähyyden, kustomoitavien sähköposti-ilmoitusten riittämättömän muokattavuuden sekä usean palvelimen tuen. Hän ei kuitenkaan mainitse, puuttuuko usean palvelimen tuki kokonaan vai onko se vain vajavaista. (Ejaz 2014.) Ejazin väitteet saattavat olla epätarkkoja, sillä niiden julkaisun jälkeen työkalun kehittäjillä on ollut vuosia aikaa korjata ongelmia.

GoCDn omissa käyttöohjeissa kerrotaan GoCDn käyttävän myös agentteja töiden jakamiseen. Ohjeissa selvitetään agenttien toimintaa seuraavasti. Tavallisen GoCD-agentin toimintamalliin kuuluu jatkuva töiden kysely koordinaattoripalvelimelta, johon palvelin joko vastaa tai ei, riippuen agentin soveltuvuudesta avoinna oleviin työtehtäviin. Soveltuvalle agentille annetaan työ, epäsopivalle ei anneta. GoCD tarjoaa myös elastisia agentteja, joita voidaan luoda lennosta vastaamaan töiden tarpeita. Elastisten agenttien hyödyntäminen vaatii lisämoduulin, jonka kanssa koordinaattori voi keskustella. Käyttäjän tulee itse suunnitella moduuli vastaamaan omia vaatimuksiaan, jotta saavutetaan tavoiteltu toiminnallisuus. Kun koordinaattori näkee työn jonossa, jonka määrittelyssä on elastisen agentin profiilin id, se ottaa yhteyttä profiilissa määriteltyyn moduuliin ja ilmoittaa olemassa olevasta tehtävästä. Tässä vaiheessa moduuli voi esimerkiksi luoda kontin sisälle tehtävään sopivan agentin, joka ottaa työn vastaan, tai jättää reagoimatta saatavilla olevaan tehtävään – käyttäjä määrittelee sen moduulinsa toiminnalla itse. (Elastic agents n.d.)

GoCDn kattavien API-ohjeiden perusteella voidaan päätellä, että GoCD tarjoaa myös APIn käyttäjälleen. Ohjeissa käsitellään keinoja, joilla saa selville esimerkiksi tietoja artefakteista, putkista, agenteista ja tehtävistä. Lisäksi API tarjoaa keinoja eri toiminnallisuuksien käynnistämiseen. (Introduction n.d.) Jos kokoonpanoon integroitujen moduulien täytyy pysyä ajan tasalla putkien toiminnasta, tai toimia

käskyttävänä osapuolena, API tarjoaa niille erinomaisen keinon roolin toteuttamiseen.

GoCDn koeajo

GoCDn asennus oli yksinkertainen ja nopea. Koordinoiva palvelu ja agentti asennettiin erillisistä, GoCDn sivuilta saatavista, asennuspaketeista, joilla saatiin eriteltyä koordinaattori ja työt tekevä kääntäjäpalvelin. Käyttö oli melko helppoa, kun luki dokumentaatiota. Paria ongelmaa lukuun ottamatta kokonaisuus tuntui käytännölliselle ja käyttöliittymä informatiiviselle. Ongelmiksi käytön aikana muodostui lähinnä polkujen asettaminen, sillä dokumentaatiossa sekä virheilmoituksessa jätettiin mainitsematta, että polun on oltava suhteellinen. Toiminnallisuus kommitin jälkeiseen putken laukaisuun löytyi, mutta se toimi webhookin sijaan jatkuvalla versionhallinnan pollaamisella, kuten TeamCityssä.

GoCD kykeni kääntämään, testaamaan ja julkaisemaan kohdeprojektin hyvin, kun putki saatiin määriteltyä. Putken ja tehtävien tilat näkyivät käyttöliittymässä visualisoituna. Tuotettuja testiraportteja varten oli olemassa oma systeemi, jossa ne voitiin näyttää työnäkymän yhteydessä GoCDn käyttöliittymässä. JUnit ja NUnit formaatin XML-tiedostoihin oli erillinen sisäänrakennettu tuki, mutta GoCD kykeni tarvittaessa näyttämään minkä vain artefaktin oikein konfiguroituna – näin testitulosraporttien tarkastelu toimi melko vaivattomasti, käyttäjän kannalta, suoraan käyttöliittymästä. Valitettavasti suoraa parsintatukea ei ollut MbUnitin tuottamalle XML-formaatille, joten sitä ei voitu hyödyntää ilman konversiota tuettuihin formaatteihin. HTML-raportti voitiin kuitenkin näyttää käyttöliittymässä.

Tarvittaessa kaikkeen keskeiseen tietoon pääsi käsiksi APIn kautta, joka mahdollisti tiedon kuljettamisen muihin integroituihin moduuleihin verkon kautta. Kokonaisuutena GoCD vaikutti erittäin pätevälle työkalulle kenen tahansa käyttäjän tarpeisiin, ja se kykeni vastaamaan sille asetettuihin haasteisiin tilatun työn kontekstissa.

4.4.4 Team Foundation Server

Team Foundation Server, lyhyesti TFS, on Microsoftin vastaus tiimitason sovelluskehityshaasteisiin. Team Foundation Serverin kotisivuilla kirjoitetaan

seuraavasti. TFS tarjoaa keskitetyn paikallisen versionhallintapalvelimen, Team Foundation Version Controlin, mikäli sellaista on tarvetta pystyttää, sekä integroituu Gitin kanssa. TFS tukee Javaa, ja sisältää setin työkaluja joilla voidaan integroida olemassa oleva IDE, kuten Visual Studio, Eclipse tai Android Studio suoraan versionhallintaan. Team Foundation Serveriin voi integroida myös omia tai kolmannen osapuolen työkaluja käyttäen REST API ja OAuth 2.0 standardeja. (Team Foundation Server n.d.)

Kotisivuilla mainitaan myös, että TFS tukee jatkuvan integraation ja jatkuvan julkaisun käytänteitä. Sillä voidaan julkaista suoraan Azureen, tai mille tahansa kohdealustalle. (Continuous Integration and Delivery n.d.)

Microsoftin ohjeista käy ilmi, että TFS tukeutuu myös käytäntöön, jossa työt tekevä palvelu on erillinen asennus koordinaattoripalvelusta. Ohjeissa mainitaan, että kääntämiseen ja julkaisuun agentin asentaminen on pakollinen toimenpide. (Build and Release Agents 2016.)

Team Foundation Server REST APIn käyttöohje vaikutti puolestaan erittäin laajalle. Siellä käsiteltiin vaikuttamista tileihin, profiileihin, käännöksiin, versionhallintaan, projekteihin ja käytännössä kaikkiin keskeisiin asioihin, mitä TFS tarjosi. API soveltui myös ohjeiden perusteella tiedonhakuun. (REST API Overview - - 2017.)

Kotisivuilta selviää, että TFS on kaupallinen tuote, joten ilmaiseksi sitä ei saa. Osa liitännäisistä, kuten testimanageri, paketinhallinta ja yksityiset putket vaativat lisämaksun. Microsoft tarjoaa mahdollisuuden hoitaa oston kuukausimaksuna tai kertasummana. (Team Foundation Server Pricing n.d.)

Team Foundation Serverin koeajo

Pintapuolinen toteutus Team Foundation Serveristä onnistui melko helposti. TF-palvelimen tarjoamaan paikalliseen versionhallintaan tehtiin repositorio, jonne lisättiin ajettava koodi. Samalle palvelimelle asennettiin agentti, jonka tehtävänä oli toteuttaa putkessa määritelty toiminnallisuus. Valitettavasti TFS ei tukenut natiivisti MbUnitin tuottaman raportin formaattia, joten siihen jouduttiin rakentamaan jäsentäjä, joka käänsi testeistä tuotetun XML-raportin tuettuun NUnit-formaattiin.

TFS tarjosi selkeän käyttöliittymän ja visualisoinnin putkien tilanteista. Putket olivat suhteellisen helppoja pystyttää, ja projekti kyettiin kääntämään, testaamaan ja julkaisemaan toivotusti. Putkien määrittelyssä oli paljon valmiita tehtävätemplaatteja, joita saattoi hyödyntää.

Kokonaisuutena TFS vaikutti erittäin hiotulle työkalulle, ja sitä oli helppo käyttää. TFS oli viimeisimpiä kandidaatteja CI/CD-putkivaihtoehdoista, mitä testattiin, joten oli mahdollista, että TFS oli erityisen helppo pystyttää aikaisempien kokeilujen tuoman harjoituksen vuoksi. Kokemuksen sulavuutta ei kuitenkaan voi missään nimessä attribuoida pelkästään aikaisempaan harjoitukseen, vaan TFS oli epäilyksettä erinomainen tuote. Koeajon perusteella voitiin todeta, että TFS kykeni toimittamaan kitkattomasti siltä vaaditut tehtävät.

4.4.5 GitLab

Viimeisimpänä vertailukohteena oli GitLab. GitLabin kotisivuilla kerrotaan sen olevan avoimeen lähdekoodiin perustuva tuote, joka pyrkii nitomaan koko DevOpsin toiminnan sisäänsä (About Us n.d.). Kotisivuilla lisätään, että GitLabin käyttöön ottaminen on ilmaista. Jos halutaan tukea tuotteen käyttämiseen, sitä on saatavilla maksua vastaan. (Below are the options to self-host GitLab n.d.)

Ramos (2016) kirjoittaa GitLabin toimivan alustana jatkuvalle integraatiolle, jatkuvalle julkaisulle ja jatkuvalle toimitukselle, joka tukee väitettä GitLabin pyrkimyksestä nitoa sisäänsä koko DevOpsin toiminta (Ramos 2016).

GitLab käyttää oman dokumentaationsa mukaan samanlaista palvelujärjestelyä, kuin GoCD, tehtäviä suorittavien instanssien osalta. Kääntöpalvelimelle asennetaan erillinen palvelu kyselemään ja suorittamaan tehtäviä. Instansseja kutsutaan GitLabissa agenttien sijaan runnereiksi. Runner suorittaa saamansa tehtävät, ja raportoi tuloksista GitLab-koordinaattorille. (GitLab Runner 2017.)

GitLabin verkkodokumentaatiossa oli ohjeet GitLab APIn käyttöön. Ohjeissa selitetään, kuinka APIn kautta voidaan komentaa GitLabia, tai kerätä siitä tietoa. API tarjoaa ohjeiden mukaan tietoa mm. putkista, tehtävistä, julkaisuista, ympäristöistä, nimiavaruuksista, käyttäjistä, runnereista, palveluista, asetuksista ja paljosta muusta. (GitLab API n.d.)

GitLabin koeajo

Vaikka GitLabia kokeiltiinkin viimeisimpänä, se ei todellakaan jäänyt vähäisimmäksi. GitLab oli todella helppo pystyttää ja se toimi moitteettomasti lupaamiensa toiminnallisuuksien puitteissa. Koordinaattori ja runner asennettiin erikseen, jolloin tehtävien työtaakka oli helppo jakaa usealle palvelimelle tarvittaessa.

GitLab tarjosi oman tiketöintijärjestelmän ja versionhallinnan käytettäväksi CI/CD-toiminnallisuuden lisäksi. Kohdeprojektin sai lisättyä GitLabin versionhallintaan, ja siitä voitiin laukaista julkaisuputki kommitin yhteydessä. Putkia kyettiin laukaisemaan monella muullakin keinolla, kuten APIn kautta, tai ajastettuna. Integraatiotyötä versionhallinnan ja CI/CD-toiminnallisuuden kanssa ei tarvinnut tehdä, sillä ne olivat valmiiksi integroituna.

Koeajon aikana kävi ilmi, ettei GitLabin käyttöliittymässä kyetty näyttämään testien tulosraportteja tai artefakteja. Tämä ei ollut välttämätöntä, sillä suurimman osan testejä koskevasta informaatiosta sai selville lukemalla lokia, mikäli testit vaan rakennettiin oikein. Lisäksi kaikki keskeiset artefaktit voitiin lähettää, tai hakea, esitettäväksi muualle – GitLabin API ja käyttöliittymä tarjosivat toiminnallisuuden artefaktien hakemiseen. Putkeen itseensä saatettiin lisäksi määritellä artefaktien lähettäminen toiselle esitysalustalle.

GitLabin omaan issue trackeriin oli avattu tiketti, jossa pyydettiin HTML-artefaktien esittämistoiminnallisuutta koordinaattorin käyttöliittymään, joten GitLab mahdollisesti tarjoaa kyseisen toiminnallisuuden tulevaisuudessa (Eastwood 2017).

Koeajon perusteella todettiin, että GitLab kykeni toimittamaan siltä vaaditun toiminnallisuuden. GitLab kykeni kääntämään, testaamaan ja julkaisemaan projektin vaatimusten puitteissa, ja sitä oli miellyttävä käyttää. Sen käyttöliittymässä oli selkeä visualisoitu esitys putkista, tehtävistä ja niiden tiloista. Dokumentaatio oli erittäin tarkkaa. Kokonaisuutena GitLab jätti mielikuvan itsestään pitkälle hiottuna, ja pätevänä kokonaisuutena.

4.5 Automatisoitu resurssien hallinta ja skaalautuminen

Järjestelmän oli kyettävä reagoimaan järkevästi hiljaiseloon, jotta se ei kuluttaisi tarpeettomasti resursseja. Osa CI/CD-työkaluvaihtoehdoista tarjosi natiivisti skaalausautomaation agenteilleen tai runnereilleen, joiden avulla olisi voitu joko kontittaa tai luoda lennosta tehtävän suorittajia, mutta AutoCADin asentaminen osana suorittajan luontioperaatiota oli liian raskas, että sitä olisi ollut järkevä toteuttaa.

Walmsleyn mukaan AutoCADin voi käynnistää konsolitilaan AutoCAD Core Consolen avulla, jossa käyttäjällä on mahdollisuus antaa komentoja ohjelmalle ilman visuaalista käyttöliittymää (Walmsley 2012). Jos konsoli voitaisiin asentaa nopeasti, ja siihen voitaisiin ladata liitännäissovellus testeineen, niin se voisi olla mahdollisesti järkevä tapa kontittaa testiajo. Valitettavasti AutoCAD Core Console ei kykene lataamaan liitännäisiä käyttöönsä, jotka referoivat muita kirjastoja kuin AcCoreMgd.dll ja AcDbMgd.dll (Goncalves 2015). Tämä teki Core Consolesta tarpeettoman tutkimuksen kontekstissa, ja sulki pois AutoCADin asentamisen osana testausautomaatiota. AutoCAD oli siis oltava valmiiksi asennettuna jollain palvelimella, jota käynnisteltäisiin tai sammuteltaisiin tarpeen mukaan. AutoCADin kontittamista ei lähdetty empiirisesti testaamaan.

Koska jokainen CI-työkaluvaihtoehto tarjosi APIn, jonka kautta pääsi selville tarjolla olevista työtehtävistä, oli mahdollista luoda palvelu, joka kyselisi jatkuvasti koordinaattorilta käännösten tiloja ja käynnistelisi, tai sammuttelisi, saamiensa tietojen perusteella virtuaalikoneita, joille olisi asennettuna agentti, Gallio ja AutoCAD. Agentti kyselisi aina käynnistyessään töitä koordinaattorilta, ja näin voitaisiin, melko yksinkertaisesti, järjestää koordinaattorille laskentatehoa tehtävien tekemiseen tarpeen vaatiessa, kuluttamatta tarpeettomasti resursseja.

Toinen mahdollisuus olisi kontrolloida fyysisiä palvelimia tarpeen mukaan, mutta työn tilaaja koki virtuaalikoneiden hallinnoinnin riittäväksi. Prototyypissä päätettiin siten käyttää fyysisten palvelinten sijaan virtuaalipalvelimia.

4.6 Jatkuva toimitus asiakkaalle

Tilaajalle riitti putken läpäisseiden kirjastojen lähettäminen Subversion-julkaisurepositorioon julkaisun toteuttamiseksi. Kaikki tutkittavana olevat työkalut tarjosivat mahdollisuuden jatkuvaan toimitukseen ja jatkuvaan julkaisuun, joten mikä tahansa työkaluista saattoi toimia alustana kokoonpanossa.

Valitettavasti Subversion ei tarjonnut sopivaa autentikointimenetelmää, jolla automaatio olisi voinut kirjautua tokenilla versionhallintaan, joten päätettiin, että automaatiolle tehtäisiin versionhallintaan oma käyttäjätili, jonka tunnukset säilytettäisiin kääntäjäpalvelimella. Toteutusta varten oli selvitettävä, oliko testeistä läpi mennyt versio poikkeava tuotannossa olevasta versiosta, jotta vältettäisiin tarpeettomia julkaisuja, ja siten tarpeettomia versionhallintamerkintöjä.

Trifonov (2012) kirjoittaa, että jokaisesta käännöksestä syntynyt dll-tiedosto on hieman erilainen, koska tiedostoon sisällytetään MVID (Module Version ID), eikä tiedostojen hash-arvoja voida siten järkevästi vertailla. Kirjastoja voidaan kuitenkin vertailla pelkän kirjoitetun koodin osalta, jos se käännetään konekielestä selkokielelle ja siitä parsitaan ongelmakohdat pois. (Trifonov 2012)

Trifonov (2012) jatkaa, että helpoin tapa purkaa kirjasto selkokielelle on käyttää MSIL Disassembleria, eli ildasm.exe tiedostoa. Disassembler jättää kuitenkin omia merkintöjään koodiin purkaessaan sitä, joten nekin on parsittava pois. Kohteesta on kaiken kaikkiaan poistettava MVID, image base ja päivämäärä. (Trifonov 2012)

Tiedostojen vertailun lisäksi Subversioniin, jolla julkaisurepositorio pyöri, oli kyettävä antamaan automaation mukana informatiivisia kommittiviestejä. CI/CD-työkalujen APIt tarjosivat laajasti tietoa niiden sisäisistä tapahtumista, joten kommittiviestien rakentaminen voitiin perustaa API-kyselyihin.

Julkaisuvaihe voitiin siis suorittaa seuraavasti. Aina, kun putki suorittaisi testit onnistuneesti, se jäisi odottamaan jonkun käyttäjän manuaalista hyväksyntää julkaisuvaiheeseen. Kun käyttäjä aktivoisi julkaisun, niin automaatio kloonaisi julkaisurepositorion, tai päivittäisi olemassa olevan lokaalin repositorion vastaamaan julkaisurepositoriota, omilla tunnuksillaan. Kun repositorio olisi valmiina, siitä vertailtaisiin Trifonovin (2012) esittämällä metodilla tiedostoja ja ylikirjoitettaisiin

muuttuneet tiedostot testit läpäisseillä versioilla. Siinä tapauksessa, että tiedostoja olisi ylikirjoitettu, automaation tarvitsisi vain lähettää muutokset versionhallintaan uuden kommittiviestin kanssa, joka rakennettaisiin API-kutsuilla saaduista tiedoista, ja julkaisu olisi valmis.

5 Tulokset

5.1 Prototyyppiin valittu CI/CD-työkalu

Kun työkalut oli tutkittu riittävän läheisesti, työn tilaajalle toimitettiin tiedot työkalujen eroavaisuuksista. Jokainen vertailtu tuote kykeni tarjoamaan vaaditun toiminnallisuuden – siksi lopullinen päätös valinnasta tehtiin hinnan, dokumentaation ja helppokäyttöisyyden perusteella.

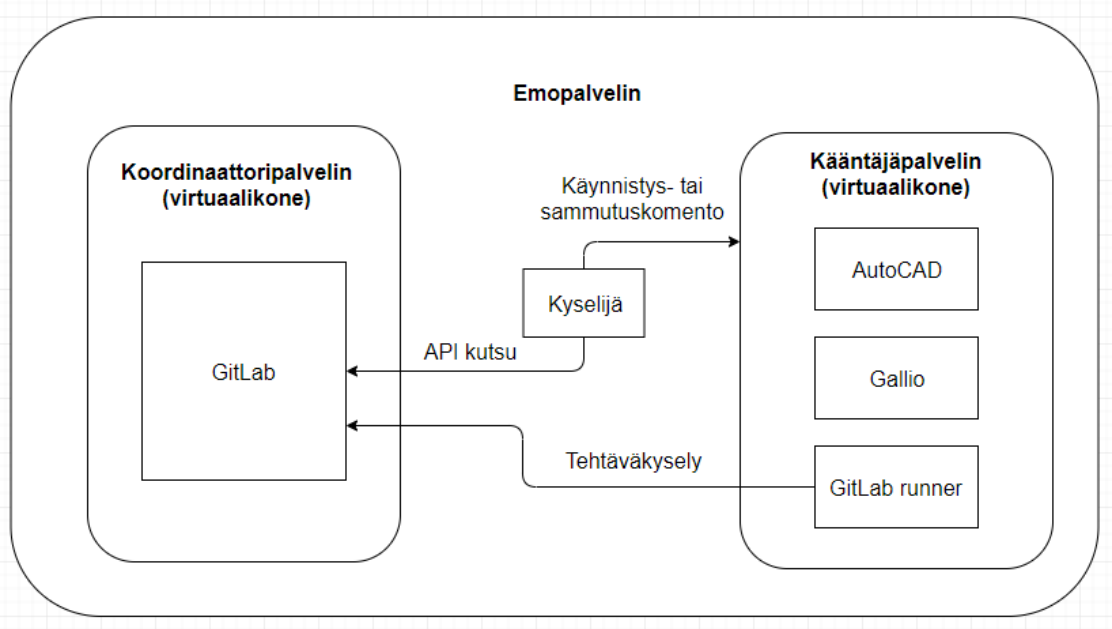
Sopivimmiksi vaihtoehdoiksi nousivat GitLab ja GoCD. Molemmat olivat ilmaisia työkaluja, jotka tarjosivat selkeän käyttöliittymän putkien ajamiselle ja tarkkailulle. Suurin ero työkalujen välillä oli pystyttämisen vaikeustaso, sekä artefaktien esittäminen. GitLab ei kyennyt esittämään artefakteja käyttöliittymässään, kun taas GoCDn pystyttäminen oli GitLabia hankalampaa.

Testausrakenteen sisällä luotuihin testiraportteihin pääsi käsiksi vain artefakteina kaikissa työkaluissa, joten artefaktien tarkastelu oli elintärkeä ominaisuus - siten GitLabin kannalta oli epäedullista, että sitä ei voitu vielä tehdä natiivisti käyttöliittymän kautta, vaan artefaktit piti hakea käyttäjän koneelle ennen tarkastelua. GoCDn käyttäminen paikoin puolestaan hankalaa, eikä sen dokumentaatiosta aina selvinnyt miksi jokin sen toiminnallisuus käyttäytyi oudosti. Valinta GitLabin ja GoCDn välillä ei ollut ilmiselvä. Lopulta GitLab valittiin prototyyppiin, ja siten lopulliseen tilaajalle käyttöönotettavaan kokoonpanoon, koska GitLab vaikutti erittäin pitkälle hiotulta kokonaisuudelta, ja sen ainoaan havaittuun vajavaisuuteen oli jo tiketti auki kehittäjillä.

5.2 Prototyyppi

5.2.1 Palvelimet

Prototyyppikokoonpanossa oli kolme palvelinta – kaksi Oracle VirtualBoxilla pyörivää virtuaalikonetta ja emokone, joka pyöritti virtuaalikoneita (ks. kuvio 17). Koordinaattoripalvelin toimi alustana GitLabille, kun taas kääntäjäpalvelin sisälsi varsinaiseen testaamisprosessiin tarvittavat ohjelmistot ja kirjastot.



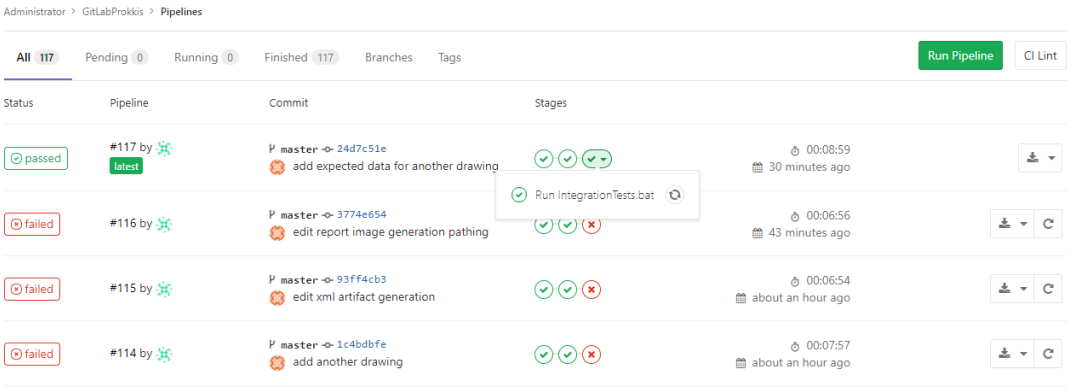
Kuvio 17: Automaation arkkitehtuuri

Emokoneella pyöri virtuaalikoneiden lisäksi kyselijäpalvelu, joka tarkkaili jatkuvasti putkien tiloja GitLabin tarjoaman APIn kautta, ja hallinnoi kääntäjäpalvelimen käynnissä oloa. Kun uusi putki käynnistyi, esimerkiksi versionhallintaan tulleen muutoksen myötä, kyselijä huomasi sen ja käynnisti kääntäjäpalvelimen. Heti käynnistyttyään virtuaalikoneella pyörivä GitLab runner -palvelu pyysi tehtävää GitLabilta, ja sai tehtäväkseen ajaa putken sisällön. Kun kyselijä ei saanut API-kutsussa tietoja yhdestäkään ajettavasta, tai ajossa olevasta, putkesta, se lähetti sammutuskomennon kääntäjäpalvelimelle.

5.2.2 GitLab ja GitLab runner

Prototyyppikokoonpanossa hyödynnettiin SSL-teknologiaa ja https-protokollaa GitLabin osalta. GitLabille tehtiin itse allekirjoitettu sertifikaatti, joka täytyi ottaa huomioon kaikkien palvelimen kanssa kommunikoivien osien integroinnissa.

Kokoonpanossa hyödynnettiin GitLabin tarjoamaa versionhallintaa. GitLab käynnisti putket toivotusti kommitin yhteydessä, ja näytti tilannekuvan putkista Pipelines-välilehdellä (ks. kuvio 18).

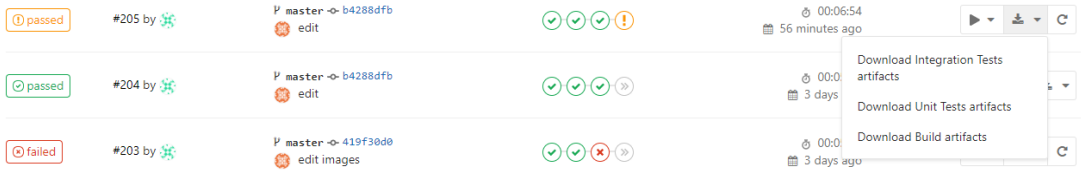


Kuvio 18: Esimerkki GitLabin putkinäkymästä

Kun GitLab runneria rekisteröitiin koordinaattoriin, käytettiin sertifikaattia, johon koordinaattorin käyttämä root-sertifikaatti luotti, jotta runner kykeni keskustelemaan GitLabin kanssa.

Runner kyseli jatkuvasti päällä ollessaan koordinaattorilta tehtäviä, otti niitä vastaan ja suoritti saamansa tehtävät. Tehtävien lopussa runner lähetti koordinaattorille tiedostoja artefakteina, kun niin oli määritelty. Muun muassa käännösvaiheen tuotokset lähetettiin koordinaattorille, josta ne haettiin uudelleen testivaiheisiin hyödynnettäväksi.

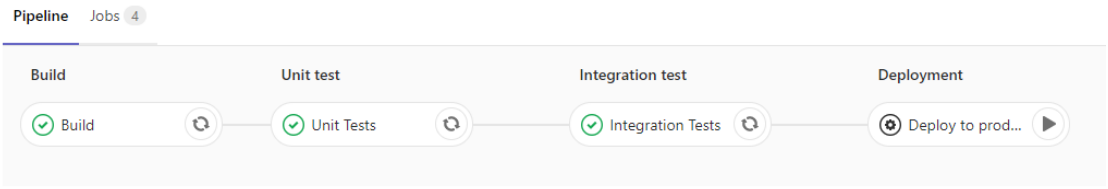
Käyttöliittymästä näki testiajojen tilat helposti, ja epäonnistuneiden testiajojen tapauksissa epäonnistumiseen johtaneet syyt. Putken loki toimi reaaliajassa, josta toimintaa saattoi seurata. Lisäksi käyttöliittymän kautta kykeni hakemaan eri vaiheiden artefaktit manuaalisesti (ks. kuvio 19).



Kuvio 19: Artefaktien haku

Jos testiajo epäonnistui, testien vertailumateriaali lähetettiin koordinaattorille raportin mukana artefakteina. Mikäli virhe tapahtui kuvien vertailussa, raportin mukaan sisällytettiin virheen aiheuttaneet kuvat, kun taas xml-vertailussa sinne sisällytettiin molemmat xml-tiedostot. Kaikissa tapauksissa syyt oli helppo kaivaa käyttöliittymän kautta esiin.

Ajettavassa putkessa oli neljä vaihetta: Build, Unit Tests, Integration Tests ja Deployment – jokaisessa vaiheessa oli yksi, vaiheen toiminnallisuuden toteuttava, tehtävä. Julkaisuvaihe jätettiin manuaaliseksi toimeksiantajan toiveiden mukaisesti, jolloin vaihe piti erikseen aktivoida toteutuakseen (ks. kuvio 20).



Kuvio 20: Projektin lopullinen putken muoto

Vaiheet, ja niiden sisäiset tehtävät, määriteltiin GitLabin versionhallinnassa kulkevalla gitlab-ci.yml tiedostolla. Tehtäville määriteltiin kohteet ja ehdot, joiden perusteella automaatio poimi tiedostoja suorittamisen jälkeen artefakteina, lähetettäväksi koordinaattorille. Generoidut artefaktit kulkeutuivat seuraavaan vaiheeseen käytettäväksi automaattisesti (ks. liite 1).

Käännösvaiheen artefakteiksi määriteltiin kaikkien projektien käännöskansiot, joita tarvittiin testeissä tai julkaisussa, kun taas testitehtävien artefakteiksi määriteltiin testitulosten kansiot. Putki-instanssin sisältöä käsitteleviä muuttujia annettiin API-kyselyä varten, julkaisuvaiheen powershell skriptille, muiden tietojen mukana.

5.2.3 Kyselijä

Nodella kirjoitettu kyselijä hallinnoi kääntöpalvelimen päällä oloa prototyypissä. Palvelulle annettiin sertifikaatti SSL-yhteydenottoon, ja GitLabin tuottama merkki (token), jotka valtuuttivat palvelun kyselemään GitLabin APIn kautta tietoja putkista. Palvelu ohjelmoitiin toistamaan kyselyjä loputtomasti callback-loopilla, jossa oli 20 sekuntia aikaa välissä.

Palvelun määrittelyssä listattiin kuunneltava portti ja näytettävä sivu siltä varalta, että palvelun tilaa haluttaisiin tarkastella verkon välityksellä (ks. liite 2). Tuotantoversioon sivulle voitaisiin lisätä esimerkiksi virheloki, nopeamman virhetarkastelun tuottamiseksi. Esimerkkitapauksessa, jossa huomiota vaativia putkia ei löytynyt, kyselijä varmisti, että virtuaalikone oli suljettuna (ks. kuvio 21).

```
Creating a request...
Response ok...
No pipelines active, shutting down running virtual machines

Done. Waiting for 20000ms until next call.
```

Kuvio 21: Kyselijä sammuttaa virtuaalikonetta

Virhetilanteet, jotka syntyivät, kun palvelija yritti käynnistää jo käynnistyvää kääntäjäpalvelinta, tai sammuttaa sammunutta kääntäjäpalvelinta, jätettiin VirtualBoxin huoleksi, eivätkä ne aiheuttaneet ongelmia automaatiolle. Tuotantoversiossa voitaisiin tarkkailla virtuaalikoneen tilaa ennen komentojen antamista, mikäli se koettaisiin tärkeäksi. Kun kyselijä havaitsi pending-tilassa olevan putken, se käynnisti kääntäjäpalvelimen (ks. kuvio 22).

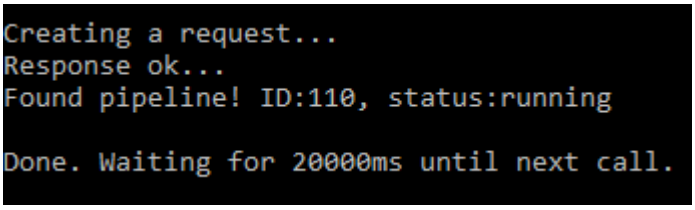
```
Creating a request...
Response ok...
Found pipeline! ID:110, status:pending
Pending pipeline found. Booting VM.

Done. Waiting for 20000ms until next call.
```

Kuvio 22: Kääntäjäpalvelimen käynnistäminen

Heti kun virtuaalipalvelin käynnistyi, sen sisäinen GitLab runner heräsi kyselemään koordinaattorilta töitä. Havaitut putket, jotka olivat running-tilassa, huomioitiin

myös, eikä kääntäjäpalvelinta suljettu, ennen kuin kaikkien putkien ajo oli saatettu loppuun (ks. kuvio 23).



Kuvio 23: Kyselijä havaitsee ajossa olevan putken

Kun saadussa vastauksessa ei ollut tietoja odottavista tai ajossa olevista putkista, palvelu sammutti taas kääntöpalvelimen. Kyselystä, vastauksen tulkinnasta ja toimenpiteistä oli vastuussa joukko funktioita, joihin toiminnallisuus oli pilkottu. Funktiot lisäsivät aktiivisesti konsolimerkintöjä, jotta kehitysvaiheessa oli helpompaa selvittää tilannekulkua (ks. liite 3).

5.2.4 Gallio

Gallio asennettiin kääntäjäkoneelle, jossa se ajoi yksikkötestejä ja integraatiotestejä liitännäissovellukselle muunnellun lähdekoodin voimin. GitLab runner käynnisti Gallion putken testausvaiheissa, ajamalla versionhallinnassa kuljetettuja batch-skriptoja. Yksikkötesteissä käytettiin IsolatedProcess-runneria ja integraatiotesteissä AutoCAD-runneria. Gallio tuotti html-muotoisen raportin testeistä, joka lähetettiin koordinaattorille artefaktina käyttäjän tarkasteltavaksi.

5.2.5 Käännös

Koodin kääntäminen toteutettiin ajamalla MSBuildia ja NuGetia batch-skriptillä. Skriptissä määriteltiin projektin sijainti, palautettiin projektille keskeiset paketit ja lopulta käynnistettiin käännös MSBuildilla (ks. kuvio 24).

```
SET HCAD=%cd%\branches\AutoCAD2018\HirsiCAD.sln
C:\Nuget\nuget.exe restore "%HCAD%"
"C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe" "%HCAD%" /t:Clean,Build /p:Configuration=%1 /p:Platform="Any CPU"
```

Kuvio 24: Build.bat

Käännöskomennossa määriteltiin, että projektin käännöskansio tuli siivota ennen käännöstä, ja uusi käännös kohdennettaisiin kaikille CPU-alustoille, parametrina tulevalla konfiguraatiolla. Käyttöön otetussa versiossa voisi olla järkevää tehdä

käännös release-konfiguraatiolla, sillä se on konfiguraatio, jossa tuote tyypillisesti päätyy asiakkaalle.

5.2.6 Yksikkötestit

Yksikkötestit käynnistettiin ajamalla batch-skripti (ks. kuvio 25). Skriptissä määriteltiin polkuja tiedostoille ja käynnistettiin Gallio.Echo.exe. Galliolle annettiin vipuina tuotetun raportin formaatti, nimi ja haluttu sijainti, josta GitLab runner saattoi poimia sen artefaktina ajon jälkeen.

```
:: set the current directory
SET DIR=%~dp0

:: set the dll which contains tests to run
SET TESTSDLL=branches\AutoCAD2018\HirsiCADUnitTests\bin\Debug\HirsiCADUnitTests.dll

:: set the relative report directory
SET REPORTDIR=%DIR%..\..\UnitReports

:: report name
SET REPORTNAME=UnitReport

:: set path to gallio
SET GALLIO_RUNTIME_PATH=C:\Program Files\Gallio\bin

:: run tests
"%GALLIO_RUNTIME_PATH%\Gallio.Echo.exe" "%TESTSDLL%" /report-type:html /report-directory:"%REPORTDIR%" /report-name-format:%REPORTNAME%
```

Kuvio 25: UnitTests.bat

Yksikkötestit ajettiin ensin, sillä jos virhe tapahtuisi jo yksikkötesteissä, säästettäisiin integraatiotesteihin kuluva aika. Yksikkötestit suoritettiin toisistaan erillisinä testeinä, eikä niille tarvinnut luoda rakennetta testitapausten generointiin. Yksikkötestifunktiot merkittiin Test-attribuutilla, jotta testauskehikko tunnisti ne testeiksi (ks. kuvio 26).

```
[Test]
public void DummySumTest()
{
  int a = 10;
  int b = 20;
  int sum = a + b;

  Assert.AreEqual<int>(sum, 30);
}
```

Kuvio 26: Prototyypin yksikkötesti

5.2.7 Integraatiotestit

Integraatiotestien käynnistäminen tapahtui batch-skriptillä, joka oli saman kaltainen kuin yksikkötesteissä käytetty ratkaisu. Kuten yksikkötestien käynnistysskriptissä,

IntegrationTests.bat sisälsi useita rivejä määrittelyjä, jonka päätteeksi käynnistettiin Gallio.Echo.exe. Tulosraportin tuottamiseen liittyvien määrittelyjen lisäksi Galliolle annettiin vipu, jossa nimettiin runnertyypiksi AutoCAD, jolloin testikirjasto ladattiin ajon yhteydessä AutoCADin säikeeseen (ks. kuvio 27).

```
:: set the current directory
SET DIR=%~dp0

:: set the dll which contains tests to run
SET TESTSDLL=branches\AutoCAD2018\HirsiCADIntegrationTests\bin\Debug\HirsiCADIntegrationTests.dll

:: set the relative report directory
SET REPORTDIR=%DIR%..\..\IntegrationReports

:: report name
SET REPORTNAME=IntegrationReport

:: set autocad path
SET ACADPATH=C:\Program Files\Autodesk\AutoCAD 2016\acad.exe

:: set path to gallio
SET GALLIO_RUNTIME_PATH=C:\Program Files\Gallio\bin

:: run tests
"%GALLIO_RUNTIME_PATH%\Gallio.Echo.exe" "%TESTSDLL%" /runner:AutoCAD /runner-property:AcadExePath="%ACADPATH%"^
 /report-type:html /report-directory:"%REPORTDIR%" /report-name-format:%REPORTNAME%
```

Kuvio 27: IntegrationTests.bat

Jotta liitännäissovelluksen toiminnallisuutta kyettiin integraatiotestaamaan, myös liitännäinen jouduttiin lataamaan AutoCAD-sovellukseen testien lisäksi. Lataaminen saatiin tehtyä merkkaamalla testiluokka TestFixture-attribuutilla, sekä lataamisen suorittava metodi FixtureSetup-attribuutilla. Näin määriteltiin metodi ajettavaksi, ennen testitapausten luomista ja ajamista, joka latasi liitännäissovelluksen AutoCADiin (ks. kuvio 28). Ladattavan tiedoston sijainti täytyi olla AutoCADin luotettujen polkujen listalla, jotta ladatessa ei syntynyt varmistusdialogia. Liitännäissovellus olisi voitu automatisoidusti ladata myös käyttämällä AutoCADin natiivia toiminnallisuutta, liitännäisten lataamiseen, AutoCADin käynnistymisen yhteydessä.

```
[FixtureSetUp]
0 references | 0 changes | 0 authors, 0 changes
public void LoadCADPlugin ()
{
  // must be in a path trusted by autocad
  Assembly.LoadFrom(string.Format("{0}\\CADPlugin.dll", Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)));
}
```

Kuvio 28. Liitännäissovelluksen lataaminen testiympäristöön

Koska testien lukumäärä piirustusta kohden saattoi olla teoriassa mikä tahansa äärellinen luku, ja piirustuksen avaamiseen kuluva aika mitattiin kokonaisissa sekunneissa, oli kehitettävä testausmalli, jossa piirustuksen avaamisen yhteydessä

sille ajettaisiin kaikki testit kerralla. Näin tiedostojen avaamiseen ja sulkemiseen käytetty aika minimoitaisiin.

MbUnitin TestSuite-luokka sisälsi toiminnallisuuden testitilanteen valmisteluun sisältämilleen testitapauksille. Perimällä TestSuite-luokka päästiin ylikirjoittamaan tämä toiminnallisuus avaamaan tiedosto ennen testien ajoa, ja sulkemaan se, kun testit oli ajettu (ks. liite 4). Luokka, jolle toiminnallisuus periytettiin, oli nimeltään CADTestSuite.

Luokalle annettiin kenttä, johon käyttäjä saattoi asettaa luonnin yhteydessä tiedostopolun piirustukseen. Ennen kun testitapauksia ryhdyttiin ajamaan, OnSetupSelf tarkisti, oliko luokan kentässä tallella tiedostopolku josta hakea piirustus, ja kutsui SetupDrawing-funktiota, mikäli oli. SetupDrawing avasi saamastaan polusta löytyvän piirustuksen (ks. liite 4).

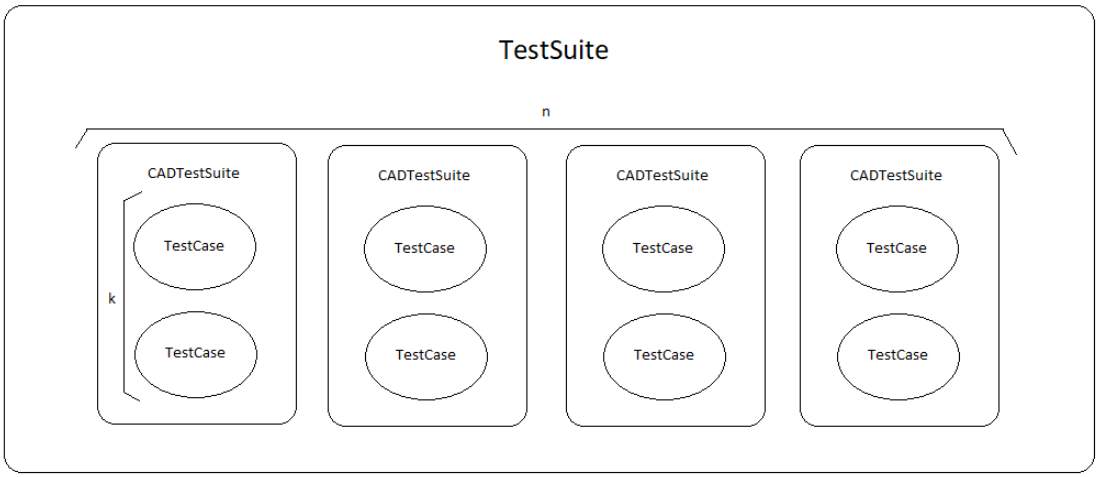
Testitapausten luontiin toteutettiin dynaaminen malli, joka pystyi käymään läpi n-määrän tiedostoja ja luomaan jokaiselle samat testitapauksensa (ks. kuvio 29). Metodi merkittiin attribuutilla DynamicTestFactory. Jokaista AutoCAD-piirustusta kohden tehtiin kaksi testitapausta – toinen kuvakaappauksille, toinen XML-tulosteelle.

```
[DynamicTestFactory]
0 references | 0 changes | 0 authors, 0 changes
public IEnumerable<Test> CreateDocumentTestCases ()
{
  TestSuite testSuite = new TestSuite("Tests");

  string assemblyFolder = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
  string[] directories = Directory.GetDirectories(string.Format("{0}\\{1}", assemblyFolder, CADIntegrationTests.TestTargetsFolder));
  foreach (string testDirectory in directories)
  {
    string[] drawings = Directory.GetFiles(testDirectory, "*.dwg");
    if (drawings.Length == 1)
    {
      CADTestSuite testCaseWrapper = new CADTestSuite(testDirectory.Split('\\').Last());
      testCaseWrapper.DrawingToLoad = String.Format(drawings.FirstOrDefault());
      testCaseWrapper.Children.Add(this.CreateSnapshotsTestCase(testDirectory));
      testCaseWrapper.Children.Add(this.CreateXmlTestCase(testDirectory));
      testSuite.Children.Add(testCaseWrapper);
    }
  }
  return new List<Test>() { testSuite };
}
```

Kuvio 29. Dynaaminen testitapausten luonti

Dynaamisten testitapausten luonnissa syntyvässä oliorakenteessa oli TestSuite, joka sisälsi n-määrän CADTestSuiteja, missä n = piirustusten määrä, missä jokainen CADTestSuite sisälsi k-määrän testitapauksia, missä k = uniikkien testitapausten määrä (ks. kuvio 30). Näin jokaiselle kuvalle voitiin tehdä haluttu määrä testejä avaamisen ja sulkemisen välissä.



Kuvio 30. Dynaamisten testien oliorakenne

5.2.8 Kuvien vertailu

Kuvavertailussa kohdetta käänneltiin eri asentoihin ja siitä otettiin kuvakaappauksia, joita vertailtiin jo olemassa oleviin, aikaisemmin validoituihin, kuvakaappauksiin. Testitapauksen itsensä luonti tapahtui erillisessä funktiossa, joka palautti testitapauksen (ks. kuvio 31).

```
private TestCase CreateSnapshotsTestCase (string testDirectory)
{
  return new TestCase("Compare snapshots", () =>
  {
    // this should be safe, since the drawing is set up before the test runs
    Document doc = Application.DocumentManager.MdiActiveDocument;
    List<string> failures = new List<string>();

    try
    {
      this.RunHigh3DSnapshotTest(doc, testDirectory, failures);
    }
    catch (Exception exc)
    {
      failures.Add(exc.Message);
    }

    bool result = true;
    string message = string.Empty;

    if (failures.Count > 0)
    {
      result = false;
      foreach (string failure in failures)
        message += String.Format("{0}\n", failure);
    }

    Assert.IsTrue(result, message);
  });
}
```

Kuvio 31. Kuvankaappaustestitapauksen luonti

Testitapauksen konstruktoriin annettiin testitapauksen nimi ja anonyymi funktio, minkä tuli sisältää assertio, eli vertailu. Vertailussa voitiin verrata mitä tahansa dataa, mutta tässä tapauksessa tutkittiin totuusarvoa, jonka tila määräytyi funktioiden mukana kuljetetun failures-listan sisällön mukaan.

Lista oli olemassa mm. virhetilanteita varten, joita rakenteessa saattoi syntyä. Testien aikana heitetyt käsittelemättömät virheet kaatoivat koko testiajon, jolloin osa testeistä jäi suorittamatta, joten ne oli napattava ja käsiteltävä sopivalla tavalla. Listaa kuljetettiin testimetodien mukana ja napattujen virheiden viestit taltioitiin listaan, kuljetettavaksi takaisin varsinaisen vertailun tekevälle funktiolle.

Virhetilanteiden lisäksi listassa kulkeutuivat viestit epätoivotuista testituloksista, kuten kuvien eroavaisuuksista, sekä huomautukset raporttiin lisätyistä artefakteista. Jos listassa oli sisältöä testiajon lopussa, assertio merkitsi testin epäonnistuneeksi ja palautti listassa olevista viesteistä rakennetun yhdistelmäviestin testikehikolle, jotta tiedot epäonnistumisen syystä ja ongelmakohteesta päätyivät lokeihin ja raportteihin.

Anonyymin funktion alussa haettiin aktiivisena oleva dokumenttiobjekti AutoCADilta talteen, jotta piirustukseen voitaisiin vaikuttaa testiajon aikana, ja yritettiin asettaa visuaalinen tyyli kuvalle. Mikäli tyylin asettaminen epäonnistui jostain syystä, kuvavertailu jätettiin tekemättä ja testi merkittiin epäonnistuneeksi. Toiminnallisuus visuaalisen tyylin muuttamiseksi hyödynsi tekijänoikeuksien alaista materiaalia, joten se jätettiin pois julkaistusta toiminnallisuudesta. Ramamoorthy käsittelee kuvatyylin muuttamista omassa blogissaan, mikäli se on lukijalle keskeinen aihe (Ramamoorthy, 2012).

Dokumenttiobjekti annettiin RunSnapshotTests-metodille sen kutsun yhteydessä, muiden parametrien mukana. RunSnapshotTests-metodin tehtävä oli selvittää kuvassa olevien entiteettien mittasuhteet, ja kutsua SnapshotTest-metodia silmukassa jokaista ViewDirection-enumia kohden (ks. liite 5). Kuvakaappauksia otettiin siis yksi jokaisesta näkökulmasta, jonka ViewDirection-enum määritteli (ks. kuvio 32).

```
public enum ViewDirection
{
    Top,
    Bottom,
    Front,
    Back,
    Left,
    Right,
    SeIso,
    SwIso,
    NeIso,
    NwIso
}
```

Kuvio 32. ViewDirection-enum

Mittasuhteiden määrittelyssä käytettiin myös tekijänoikeuksien alaista materiaalia, joten sekin jätettiin pois työn julkisesta materiaalista. AutoDeskin omilla verkkosivuilla käsitellään model spacen sisäisten entiteettien mittasuhteiden selvittämistä, jossa aiheesta voi lukea lisää (Display Drawing Extents and Limits - - 2015).

SnapshotTest-metodin tehtävä oli ajaa funktiot, mitkä muuttivat kuvan näkymää ennen kuvakaappauksen ottamista ja ottivat varsinaiset kuvakaappaukset. Funktiossa haettiin parametrina annettua enumia vastaava vektori, käännettiin kuva vektoria vastaavaan kulmaan, otettiin kuvakaappaus ja lopulta tehtiin kuvakaappausten vertailu. (ks. kuvio 33).

```
private bool SnapshotTest (string testDirectory, Document doc, Extents3d ext, ViewDirection vDir, ref string errorMessage)
{
    Vector3d vector = this.GetViewVector(vDir);
    this.SetView(vector, doc, ext);
    Snapshot snapshot = this.TakeSnapshot(doc);
    return this.CompareSnapshots(testDirectory, snapshot, vDir.ToString(), ref errorMessage);
}
```

Kuvio 33. SnapshotTest-funktio

Enumia vastaava kuvakulma haettiin funktiolla GetViewVector (ks. liite 6). Varsinainen kuvan kääntäminen tehtiin metodissa SetView (ks. kuvio 34). Metodi otti parametreina vastaan katselusuunnan määrittävän vektorin, dokumenttiobjektin, minkä näkymä tuli muuttaa, sekä kuvassa olevien entiteettien mittasuhteet. SetView sovitti kuvan niin, että kaikki entiteetit mahtuivat kuvaan katselusuunnan ollessa parametrissä määritelty vektori.

```
private void SetView (Vector3d viewDir, Document doc, Extents3d extents)
{
  Database db = doc.Database;
  Editor editor = doc.Editor;

  // get the current view
  ViewTableRecord view = editor.GetCurrentView();

  // Translate WCS coordinates to DCS
  Matrix3d viewTransform = Matrix3d.PlaneToWorld(viewDir).PreMultiplyBy(Matrix3d.Displacement(view.Target - Point3d.Origin))
                                                         .PreMultiplyBy(Matrix3d.Rotation(-view.ViewTwist, view.ViewDirection, view.Target))
                                                         .Inverse();

  extents.TransformBy(viewTransform);
  view.ViewDirection = viewDir;
  view.Width = (extents.MaxPoint.X - extents.MinPoint.X) * 1.2;
  view.Height = (extents.MaxPoint.Y - extents.MinPoint.Y) * 1.2;
  view.CenterPoint = new Point2d((extents.MinPoint.X + extents.MaxPoint.X) / 2.0, (extents.MinPoint.Y + extents.MaxPoint.Y) / 2.0);

  // set the view
  editor.SetCurrentView(view);
}
```

Kuvio 34. Kuvakulman asettaminen

TakeSnapShot-metodi kaappasi kuvan vertailua varten. Metodi otti vastaan parametrina dokumenttiobjektin, josta kuva tuli kaapata (ks. kuvio 35). Kuva kaapattiin dokumenttiobjektin CapturePreviewImage-metodilla, jolle annettiin parametreina kuvan leveys ja korkeus, jotka oli ennalta määritelty. Mitä suuremmat dimensiot kuvakaappaukseen määritteli, sitä kauemmin tarkistuksissa kesti, sillä vertailut tehtiin pikseli pikseliltä.

```
private Snapshot TakeSnapshot (Document doc)
{
  // Increasing the resolution will raise test durations, as the comparison is done pixel by pixel.
  Bitmap bitmap = doc.CapturePreviewImage(CADIntegrationTests.ImageWidth, CADIntegrationTests.ImageHeight);
  return Snapshot.FromBitmap(bitmap);
}
```

Kuvio 35. Kuvan kaappaus.

Jos kuva haluttaisiin ottaa käyttöliittymästä, niin olisi käytettävä jotain muuta keinoa, sillä CapturePreviewImage-metodi ei palauttanut kuin kuvan dokumentin sisäisistä entiteeteistä. Käyttöliittymäkuvakaappaukset voitaisiin ottaa käyttämällä TestApin tarjoamaa toiminnallisuutta, kuten Snapshot.FromRectangle funktiota (Manolov 2009). Kuvien vertailutehtävän hoiti CompareSnapshots-metodi (ks. kuvio 36).

```
private bool CompareSnapshots (string testDirectory, Snapshot actual, string tag, ref string errorMessage)
{
  if (actual == null) throw new NullReferenceException("The actual image was null in CompareSnapshots()");
  bool result = false;
  Snapshot expected = null, difference = null;
  string expectedFilePath = testDirectory + String.Format("\\{0}{1}.{2}",
                            CADIntegrationTests.Expected, tag, CADIntegrationTests.ImageFormat.ToString());
  string drawingSpecificReportFolder = string.Format("{0}\\{1}", this.reportFolderPath, new DirectoryInfo(testDirectory).Name);
  if (File.Exists(expectedFilePath)){
    expected = Snapshot.FromFile(expectedFilePath);
    if (expected != null){
      if (expected.Width.Equals(actual.Width) && expected.Height.Equals(actual.Height))
      {
        difference = actual.CompareTo(expected);
        if (difference != null)
        {
          SnapshotVerifier verifier = new SnapshotColorVerifier(Color.Black, new ColorDifference());
          if (verifier.Verify(difference) == VerificationResult.Pass) result = true;
          else
          {
            if (!Directory.Exists(drawingSpecificReportFolder))
              Directory.CreateDirectory(drawingSpecificReportFolder);
            this.SaveImages(actual, expected, difference, drawingSpecificReportFolder, tag);
            errorMessage = string.Format("{0} mismatch! Available images will be uploaded as artifacts.", tag);
          }
        } else errorMessage = string.Format("Snapshot comparison returned a null difference snapshot.");
      }
      else // image sizes do not match
      {
        if (!Directory.Exists(drawingSpecificReportFolder))
          Directory.CreateDirectory(drawingSpecificReportFolder);
        this.SaveImages(actual, expected, difference, drawingSpecificReportFolder, tag);
        errorMessage = "Snapshots must have identical width and height to be comparable.";
      }
    } else errorMessage = string.Format("{0} was found in the folder, but SnapShot.FromFile(expected) returned null." +
                                        "See Snapshot.FromFile(Snapshot) method at TestAPI library.", tag);
  }
  else // file not found
  {
    if (!Directory.Exists(drawingSpecificReportFolder))
      Directory.CreateDirectory(drawingSpecificReportFolder);
    this.SaveImages(actual, expected, difference, drawingSpecificReportFolder, tag);
    errorMessage = string.Format("{0} missing! The actual taken snapshot will be uploaded as an artifact.", expectedFilePath);
  }
  return result;
}
```

Kuvio 36. Kuvakaappausten vertailu

Mikäli kuvat poikkesivat toisistaan tai ilmeni virhe, luotiin raporttikansioon alikansio, jonne tallennettiin saatavilla olevat kuvat, jotta ne kulkeutuisivat raportin mukana käyttäjälle. Kuvat tallennettiin funktiolla SaveImages (ks. kuvio 37).

```
private void SaveImages (Snapshot actual, Snapshot expected, Snapshot difference, string drawingSpecificReportFolder, string tag)
{
  if (actual != null)
    actual.ToFile(string.Format("{0}\\{1}{2}.{3}", drawingSpecificReportFolder, CADIntegrationTests.Actual, tag,
                                CADIntegrationTests.ImageFormat.ToString()), CADIntegrationTests.ImageFormat);
  if (expected != null)
    expected.ToFile(string.Format("{0}\\{1}{2}.{3}", drawingSpecificReportFolder, CADIntegrationTests.Expected, tag,
                                  CADIntegrationTests.ImageFormat.ToString()), CADIntegrationTests.ImageFormat);
  if (difference != null)
    difference.ToFile(string.Format("{0}\\{1}{2}.{3}", drawingSpecificReportFolder, CADIntegrationTests.Difference, tag,
                                    CADIntegrationTests.ImageFormat.ToString()), CADIntegrationTests.ImageFormat);
}
```

Kuvio 37. Kuvien tallentaminen raportin liitteeksi

Lisäksi eroavaisuuksien, tai virheiden, tapahtuessa sijoitettiin referenssinä funktioon tulleeseen stringiin viesti, ja muutettiin tulosmuuttuja epätodeksi. Näin viesti tallennettaisiin virhelistaan, jonka perusteella testitulos pääteltiin. Viestiin sisällytettiin tiedot ongelmasta, jotta kehittäjän olisi helppo päästä ongelmaan

käsiksi. Lopuksi CompareSnapshots palautti tulosmuuttujan kutsujalle vertailutuloksesta.

Otetuissa kuvissa ilmeni satunnaisesti epäkonsistentteja tuloksia, vaikka kokoonpano, koodi ja AutoCADin piirustukset pysyivät samoina. Työn aikana ei selvinnyt varmuudella, mistä oire johtui. Poikkeuksia ilmeni vain isometrisissä kuvissa, kappaleiden liitoskohdissa, eikä poikkeavia pikseleitä ollut useita – vain kourallinen per kuva. Pikselit olivat myös aina samoissa liitoskohdissa. Testikehikko totesi eroavien pikseleiden perusteella testin epäonnistuneeksi kuvien osalta, kun niitä ilmeni, ja liitti raporttiin poikkeavuuksiin liittyvät kuvat.

5.2.9 Xml-tiedostojen vertailu

Xml-tiedostojen vertailussa hyödynnettiin samaa mallia testitapauksissa, kuin kuvavertailussa (ks. kuvio 38). Xml-vertailussa ei kuitenkaan tarvinnut erillistä iterointia, vaan se piti toteuttaa vain kerran per kuva. Dokumenttiobjekti haettiin ja annettiin RunXmlTestCase-metodille failures-listan ja testikansion kanssa. Failures-listan perusteella tarkistettiin, onko testi syytä määrittää epäonnistuneeksi.

```
private TestCase CreateXmlTestCase (string testDirectory)
{
  return new TestCase("Compare xml outputs", () =>
  {
    string message = string.Empty;
    bool result = true;

    Document doc = Application.DocumentManager.MdiActiveDocument;
    List<string> failures = new List<string>();

    try
    {
      this.RunXmlTestCase(doc, testDirectory, failures);
    }
    catch (Exception exc)
    {
      failures.Add(exc.Message);
    }

    if (failures.Count > 0)
    {
      result = false;
      foreach (string failure in failures)
        message += String.Format("\n{0}", failure);
    }

    Assert.IsTrue(result, message);
  });
}
```

Kuvio 38. Xml-testitapauksen luonti

Xml-tulosteen käsitellään tekijänoikeuksien alaista tietoa, joten sen tuottamista tai parsimista ei voida käsitellä opinnäytetyön julkisena osana. Tulosteen vertailussa käytetään XMLUnitin toiminnallisuutta, DiffBuilderia (ks. kuvio 39).

```
private IEnumerable<Difference> GetXmlDifferences (string testDirectory)
{
  return DiffBuilder.Compare(Input.FromFile(testDirectory + "\\" + CADIntegrationTests.Expected + ".xml"))
                    .WithTest(Input.FromFile(testDirectory + "\\" + CADIntegrationTests.Actual + ".xml"))
                    .WithNodeFilter((XmlNode x) => { return !x.Name.Equals("Time") && !x.Name.Equals("Date"); })
                    .Build().Differences;
}
```

Kuvio 39. Xml-tiedostojen eroavaisuuksien haku

DiffBuilderille määritellään vertailtavat tiedostot, suodattimet, ja kutsutaan sen Build-funktiota. Kutsun jälkeen DiffBuilderin Differences-propertystä pääsee käsiksi löydettyihin eroavaisuuksiin enumeraationa.

Poikkeuksien löytyessä tulosteen tuotettu versio, sekä odotettu versio, tallennettiin raporttikansioon. Lokeihin ja raporttiin kuljetettiin aiheeseen sopivat virheviestit, jotka osoittivat kehittäjälle missä virhe oli.

5.2.10 Julkaisu

Prototyyppiä ei kiinnitetty suoraan julkaisurepositorioon, mutta siihen kehitettiin kaikki tarvittava toiminnallisuus – lukuun ottamatta subversionin käsittelyä. Subversionin käsittely jätettiin tilaajan hoidettavaksi, josta sovittiin erikseen tilaajan kanssa. Käsittelyä varten jätettiin kuitenkin selkeät merkinnät skriptiin, jotta integroiminen olisi mahdollisimman kivutonta.

Automaatio käynnisti julkaisuvaiheessa powershell-skriptin, jolla suoritettiin tiedostojen vertailu, sekä rakennettiin uusi kommittiviesti julkaisutehtävän tietojen pohjalta. Skripti tarvitsi parametreina tietoja projektista, tehtävästä, sertifikaatista, käännöskonfiguraatiosta ja vertailukansioista, joiden perusteella toiminnallisuus kyettiin toteuttamaan (ks. liite 7). Skripti latasi tarpeelliset kirjastot käyttöönsä ja kutsui niiden toiminnallisuuksia toteuttaakseen tehtävänsä. Ladatuista kirjastoista ajettiin kahta luokkaa – FileUpdateria ja Requesteria.

FileUpdater

FileUpdater-luokka, jonka vastuulla oli selvittää poikkeavuudet tiedostoissa, purki kohdekirjastot, parsi niistä käännöskohtaiset tiedot pois ja suoritti vertailut hash-

arvoilla. Luokalle annettiin attribuutti, joka määritteli sen näkyväksi Com-rajapinnassa, jotta sitä voitiin kutsua komentolinjalta (ks. kuvio 40).

```
[ComVisible(true)]
public static class FileUpdater
```

Kuvio 40: FileUpdater-luokka

Luokalle jätettiin vain yksi funktio julkiseksi, jotta sen käyttö olisi mahdollisimman selkeää. Julkinen funktio etsi sille parametreina annettujen kansioiden sisältä kaikki DLL-tiedostot, ja ryhtyi suorittamaan vertailuja nimen perusteella (ks. kuvio 41).

```
public static bool Run (string testBinariesFolder, string deploymentBinariesFolder)
{
  List<string> testBinaries = new List<string>(Directory.EnumerateFiles(testBinariesFolder, "*.dll"));
  List<string> deploymentBinaries = new List<string>(Directory.EnumerateFiles(deploymentBinariesFolder, "*.dll"));
  bool changesMade = false;

  foreach (string testFilePath in testBinaries)
    foreach (string deployFilePath in deploymentBinaries)
      if (Path.GetFileName(testFilePath).Equals(Path.GetFileName(deployFilePath)))
        if (FileUpdater.GetAssemblyFileHash(testFilePath) != FileUpdater.GetAssemblyFileHash(deployFilePath))
        {
          File.Delete(deployFilePath);
          File.Copy(testFilePath, deployFilePath);
          changesMade = true;
        }

  return changesMade;
}
```

Kuvio 41: Tiedostojen vertailu ja ylikirjoittaminen

Täsmäävyyksien löytyessä funktio suoritti purkamisen, parsimisen ja hash-vertailun, kutsumalla luokan muita funktioita. Mikäli kaksi saman nimistä tiedostoa löytyi, joilla oli eri sisältö, julkaisukansion tiedosto ylikirjoitettiin testikansion tiedostolla.

Kirjastojen ja purkaminen toteutettiin käyttämällä ildasm.exe -tiedostoa, ja hash-arvot tuotettiin käyttämällä MD5-algoritmia (ks. liite 8). Ongelmarivien poistossa käytettiin regexiä (ks. liite 9).

Requester

Requester-luokka keskusteli saamiensa tietojen perusteella GitLab APIn kanssa https-protokollaa käyttäen, ja sen tehtävänä oli rakentaa kommittiviesti julkaisua varten, putken laukaisseen kommitti tiedoista. Viestiin haluttiin saada mukaan alkuperäisen kommitin lähettäjä, alkuperäinen kommittiviesti ja käyttäjä, joka aktivoi julkaisun manuaalisen vaiheen. Luokasta tehtiin COM-rajapinnassa näkyvä samalla keinolla, kuin FileUpdaterista. Vain yksi julkinen funktio jätettiin näkyviin, käyttämisen selkeyden parantamiseksi (ks. kuvio 42).

```
public static string CreateDeploymentMessage (string gitlabUrl, string projectId, string jobId, string certificatePath)
{
  HttpWebRequest request = Requester.CreateRequest(gitlabUrl, projectId, jobId, certificatePath);
  HttpWebResponse response = (HttpWebResponse)request.GetResponse();
  return Requester.BuildMessageFromResponse(new StreamReader(response.GetResponseStream()).ReadToEnd());
}
```

Kuvio 42: Julkaisuviestin luonti

Funktio otti vastaan tietoja koordinaattorista, projektista, ajossa olevasta julkaisutehtävästä ja koordinaattorin kanssa kommunikointiin tarvittavasta sertifikaatista. Funktiossa kutsuttiin muita luokan yksityisiä funktioita, joilla rakennettiin kysely, purettiin vastaus ja rakennettiin siitä teksti (ks. liite 10). Teksti palautettiin funktiota kutsuvalle powershell-skriptille, jossa sitä saattoi käyttää julkaisussa.

6 Pohdinta

6.1 Prototyypin ongelmat

Otettujen kuvakaappausten epäkonsistenttisuus osoittaa Manolovin (2009) väitteen todeksi – visuaaliset testit ovat todella hauraita. Koneella, jolla kehitystyötä tehtiin, oli vain kahdeksan gigatavua muistia, joka ei riittänyt kahta virtuaalikonetta pyörittävälle kokoonpanolle optimaaliseen suorittamiseen. Kääntöpalvelimelle ei siten voitu allokoida optimaalista määrää muistia, jolla saattoi olla vaikutusta kuvien epäkonsistenttiin laatuun. Kun kääntäjäkoneelta karsittiin ylimääräiset ohjelmat pois, niin tuloksista tuli konsistentimpia, mutta ei vieläkään poikkeuksetta samoja. Käytös herättää syytä epäillä, että suuremmalla resurssimäärällä kääntäjäpalvelin voisi kyetä tuottamaan konsistentteja tuloksia.

Mikäli suuremmat resurssimäärät eivät korjaisi ongelmaa, voitaisiin ratkaisuna asettaa TestApissa käytettyjä toleranssikarttoja ongelmakohtien päälle. Vaihtoehtoisesti ongelma voitaisiin ratkaista jättämällä ongelmia tuottavat isometriset kuvakulmat pois testeistä – kohteista otettiin kuvat jokaisesta kohtisuorasta suunnasta isometristen kulmien lisäksi, joten testit kattaisivat silti kaikki oleelliset näkökulmat.

6.2 Vaatimusten toteutuminen

6.2.1 Työkalujen vertailu

Työssä vertailtiin CI/CD-työkaluja, joista tarjottiin perusteineen ehdotusta toimeksiantajalle valittavasta työkalusta. Valitun työkalun pääpiirteittäisestä soveltuvuudesta tehtävään saatiin varmuus työn aikana, sillä se tarjosi toivotun toiminnallisuuden alustaville tarpeille. Pidemmän käytön mukana tulevat ongelmat jäivät kuitenkin näkymättömiin, sillä työkalujen tutkimus oli niin laaja tehtävä, että sitä rajattiin yleisemmälle tasolle. Työkaluvertailussa tutkittiin viimeisiä vaihtoehtoja suppeammin, sillä loppua kohden alkoi olla selvillä keskeiset ominaisuudet, joita työkalulta vaadittiin. Kaikesta toiminnallisuudesta ei tehty todennusta, jota vertailtavista työkaluista viitattiin löytyvän, mutta vaatimusten puitteissa työkaluja vertailtiin silti riittävällä tasolla.

6.2.2 Jatkuva integraatio

Prototyyppikokoonpano noudatti jatkuvan integraation mallia – siinä koottiin, yksikkötestattiin ja integraatiotestattiin versionhallintaan saapunut koodi, heti sen saavuttua. Yksikkötestejä varten tuotettiin valmis alusta, johon kehittäjä saattoi kirjoittaa testejä haluamilleen kohteelle, ja integraatiotesteissä vertailtiin AutoCAD-sovelluksessa tuotettuja kuvia, sekä XML-tulosteita. Lisäksi yksikkötestit oli eroteltu integraatiotesteistä automaatiossa eri vaiheisiin. Voitiin siis todeta, että jatkuvan integraation osalta vaatimukset täyttyivät toivotulla tavalla.

6.2.3 Oma testipalvelin

Tuotetussa kokoonpanossa oli eroteltu koordinoiva testipalvelin kääntäjäpalvelimesta, ja jätetty mahdollisuus lisätä useampia kääntäjäpalvelimia koordinoitavaksi. Palvelimet keskustelivat toistensa kanssa verkon välityksellä, joka mahdollisti erottelun. Kokonaisuus oli modulaarinen, jonka ansiosta olisi helppo asentaa kaikki kokoonpanon osat omille palvelimilleen, mikäli työn tilaaja haluaisi ottaa kokoonpanon käyttöönsä.

6.2.4 Skaalautuvuus ja resurssien käyttö

Tuotettu prototyyppi kykeni testeissä skaalautumaan n-määrään kuvia ja testitapauksia, joten tulos skaalautui tarpeen mukaan testien osalta. Koontipalvelimella pyörivä kyselijä puolestaan piti huolta, ettei resursseja kulutettu tarpeettomasti palvelimella. Kokoonpanossa saattoi ajaa rinnakkain putkia, tarjoamalla lisää palvelimia GitLabin käyttöön – palvelimille täytyi asentaa vain samat työkalut ja rekisteröidä GitLab runner koordinaattorille.

6.2.5 Jatkuva julkaisu

Toimeksiantajan jo olemassa olevaa julkaisukäytäntöä ei rakennettu kokonaan uudelleen, mutta siihen tehtiin keskeiset lisätoiminnallisuudet prototyyppikokoonpanoon, jotta toiminnallisuus voitiin automatisoida haluttuun pisteeseen asti. Kokoonpano tarjosi mahdollisuuden lähettää kommitoitu koodi suoraan tuotantoympäristöön, napin painalluksella ja valmiiksi testattuna – toimeksiantajan toiveen mukaisesti. Toiminnallisuus vaati kuitenkin hieman integroimista toimeksiantajan puolesta.

6.3 Prototyypin luotettavuus

Oikein käytettynä kokoonpano toimi luotettavasti, lukuun ottamatta isometristen kuvakaappausten ongelmakohtia. Työn aikana ei selvinnyt varmaa syytä, miksi ongelmakohtia syntyi kuviin, mutta syyn epäiltiin olevan emokoneen muistin puute.

Kuten mikä tahansa työkalua, kokoonpanoa voitiin käyttää väärin, josta syntyisi ongelmia. Testifarmille oli keskeistä, että kääntöpalvelimet olivat identtisiä, jotta tuotetut kuvat olivat aina samanlaisia, joka teki osasta tuotoksen toiminnallisuutta potentiaalisesti haurasta.

Hans Passant (2010) väittää StackOverflowssa, ettei MSIL Decompiler pysty purkamaan luotettavasti lambdoja, anonyymejä funktioita tai koodia, joka käyttää async/await -avainsanoja (Passant 2016). Julkaisutoiminnallisuus voisi toimia virheellisesti, jos vertailtava koodi käsittelisi jotain edellä mainituista toiminnallisuuksista. Mikäli käytös tulkitaan kriittiseksi puutteeksi, on syytä harkita

vaihtoehtoista reittiä muuttuneiden kirjastojen yksilöintiin, tai julkaisujärjestelmän suurempaa refaktorointia.

6.4 Jatkokehittäminen

6.4.1 Lokaalin pilvipalvelun käyttöönotto

Koontipalvelu voitaisiin siirtää paikalliseen pilveen, jossa olisi enemmän laskentatehoa. Tällöin kääntäjäpalvelimen laskentateho ei olisi niin suuri haaste virtuaalikoneiden pyörittämiseen, ja voitaisiin paremmin toteuttaa integraatiotestien rinnakkain ajoa, jotka olivat AutoCADin takia melko raskaita.

6.4.2 Kuvakaappausten vakauttaminen

Kuvakaappausten ottamisessa voitaisiin selvittää, miksi otetut kuvat eivät olleet jokaisella kerralla samanlaiset, vaikka ne otettiin samalla alustalla, samalla koodipohjalla ja samoista AutoCAD-tiedostoista. Mikäli hajonta johtui koneen resurssien puutteesta, niin koontipalvelun siirtäminen tehokkaammalle alustalle toimisi siinä ratkaisuna.

6.4.3 Kuvatestitapausten eriyttäminen

Tuotetussa integraatiotestirakenteessa yhteen kuvatestitapaukseen asetettiin useita kuvakaappauksia ja vertailuja, jotka tyypillisesti jaettaisiin omiin testitapauksiinsa. Lopulliseen käyttöönottoversioon voisi harkita niiden erottelua toisistaan, jotta raporttiin syntyisi erillinen pass/fail -merkintä jokaisesta kuvakulmasta.

6.5 Tutkimuksen luotettavuus

Tutkimuksessa hankittu tieto perustuu suurelta osin ensi käden kokemukseen. Vaaditun toiminnallisuuden toteuttamista edeltävässä tutkimuksessa referoitiin paljon ulkoisia lähteitä, mutta lähteistä saadun, prototyypissä hyödynnetyn, tiedon todenperäisyys todennettiin työn aikana empiirisillä havainnoilla.

CI/CD-työkalujen ominaisuuksiin kohdistuneita väitteitä ei kuitenkaan todennettu kaikilta osin, sillä prototyypin toteuttamista jokaisen työkalun ympärille pidettiin liian

työläänä toteutettavaksi. Työkalujen kotisivuilta haettu tieto oli todennäköisesti paikkaansa pitävää tutkimuksen aikana, sillä työkalujen kehittäjät tuskin valehtelivat aiheesta. Työkalut kuitenkin muuttuvat jatkuvasti, jolloin niitä koskeva tieto vanhenee muutoksien osalta. Digitaalisessa maailmassa kehitys on lisäksi erityisen nopeaa, joten on järkevää pitää ensisijaisena työkaluja koskevan tiedon lähteenään työkalujen kehittäjiä, ajankohtaisen tiedon saamiseksi. Todennäköisesti työkaluista ei kuitenkaan poisteta ominaisuuksia tai toimintoja, joita tämä tutkimus käsittelee, mutta niitä saatetaan muuttaa, tai uusia lisätä.

Osa tiedosta, johon tutkimuksessa viitataan, on vanhaa, mutta kuitenkin validia. Galliota ei ole kehitetty vuosiin, joten on luonnollista, että sitä koskevat tiedotkin ovat vanhoja. Tiedot osoittautuivat tutkimuksen aikana paikkaansa pitäviksi. Sama koskee MbUnittia. Gallioon ei todennäköisesti ole tulossa muutoksia, jotka vaikuttaisivat tuloksissa esitetyn tiedon käyttökelpoisuuteen, kun huomioidaan lähivuosien kehitystyön poissaolo. Viimeisin muutos koodiin tehtiin vuonna 2013 (Gallio/mbunit-v3 n.d.).

Kaikki tutkimuksessa tehdyt tulkinnat ovat alisteisia inhimillisille virheille, eikä niitä voi missään tapauksessa pitää absoluuttisina. Tutkimus antaa kuitenkin vahvoja viitteitä, joiden perusteella voidaan tehdä päätöksiä suhteellisen hyvällä teoriapohjalla.

6.6 Tuloksen käyttökelpoisuus työn tilaajalle

Tehty työ vastasi keskeisimpiin haasteisiin, joita AutoCAD-liitännäissovelluksen testausrakenteen luominen piti sisällään. Tuloksen perusteella voidaan rakentaa käyttöversio testausautomaatiosta, suhteellisen pienellä työmäärällä, jonka avulla eliminoidaan inhimillisiä virheitä sovelluskehityksessä. Työ saavutti tavoitteensa, ja tuotti lisäarvoa toimeksiantajalle vaatimusten kontekstissa.

6.7 Työn arvo yleisellä tasolla

AutoCAD-liitännäissovelluksen testaamisesta oli melko vähän helposti löydettävää tietoa tarjolla. Mikäli joku pyrkii rakentamaan omaa testauskokoonpanoaan sovellukseen, joka perustuu AutoCAD-teknologiaan, tuotetusta työstä saattaa olla

suurtakin apua. Tiedon tarjonnan vähäisyyden voitaisiin kuitenkin tulkita johtuvan sen tarpeellisuuden vähäisyydestä, jolloin yleisellä tasolla työ ei todennäköisesti tarjoaisi kovinkaan monelle lisäarvoa. Kuitenkin työn tilauksen perusteella nähtiin, että oli olemassa instansseja, joille työ oli keskeinen – yleisesti tarjolla olevan tietomäärän perusteella, työ voi olla erittäin arvokas sitä tarvitsevalle.

CI/CD-työkalujen vertailun osalta työ ei ollut erityisen laaja, ja niitä koskeva tutkimustieto on altis muutoksille työkalujen omien kehitysprosessien myötä. Työssä keskeisin havainto kolmansille osapuolille on siksi, todennäköisesti, AutoCAD-liitännäissovelluksen integraatiotestien toteutusmahdollisuus Gallion avulla.

6.8 Yhteenveto

CD/CD-työkalujen tutkimisen perusteella mikä tahansa työkaluvaihtoehdoista olisi voinut toimia kokoonpanossa AutoCAD-liitännäissovelluksen jatkuvan integraation, ja jatkuvan julkaisun toteuttamisessa. Tehtävään valikoitu GitLab toimi prototyyppitasolla ongelmitta.

Tulosten perusteella nähtiin, että AutoCAD-liitännäissovellusta voidaan testata automatisoidusti. Yksikkötestien osalta toteutuksessa ei tarvinnut poiketa normaaleista käytänteistä, mutta integraatiotestien osalta oli pakko tehdä räätälöity toteutus, jotta testit saatiin ajettua AutoCADin säikeessä. Gallio toimi erinomaisesti, pienillä muokkauksilla, testausalustana integraatiotesteille sekä yksikkötesteille. Muokkauksia piti tehdä Gallion lähdekoodiin, jotta AutoCADissa auki olevan dokumentin kuvaa kyettiin kääntelemään. Gallion referoima Microsoft.Cci piti päivittää uusimpaan versioonsa, jotta lähdekoodin muokkauksessa rikkoutuneet Gallion runnerit saatiin korjattua. Tulos antoi myös selkeitä viitteitä siitä, että yleisellä tasolla visuaalisten testien käyttämistä kannattaa välttää, jotta vältetään hauraat ja virhealttiit testit.

Toimeksiannon mukainen kokoonpano toteutettiin. Se kykeni kääntämään, testaamaan ja julkaisemaan AutoCAD-liitännäissovelluksen. Integraatiotestit ja yksikkötestit olivat eroteltuna toisistaan, ja prototyyppi noudatti jatkuvan integraation sekä jatkuvan julkaisun käytänteitä. Käännös laukaistiin kommitin saapuessa versionhallintaan. Kokoonpano oli modulaarinen, pilkottu omille

palvelimilleen, helposti laajennettavissa ja kykeni itsenäiseen skaalautumiseen vaatimusten puitteissa.

Prototyypin lähdekoodit ja valmiit kokoonpanot jätettiin tilaajalle ohjeineen. Työn perusteella toimeksiantaja kykenee parantamaan sovelluskehitysprosessinsa luotettavuutta ja nopeutta automatisoimalla julkaisun. Automaatio vähentää inhimillisten virheiden määrää, ja vapauttaa kehittäjät keskittymään sovelluskoodiin paremmin. Työn tavoite täyttyi siten toivotulla tavalla.

Lähteet

About Us. N.d. GitLab. Verkkojulkaisu. Viitattu 21.1.2018. https://about.gitlab.com/about/

AcApDocManager::appContextOpenDocument. N.d. AutoDesk knowledge network. Verkkojulkaisu. Viitattu 5.11.2017. http://help.autodesk.com/view/OARX/2018/ENU/?guid=OREF-AcApDocManager__appContextOpenDocument_ACHAR__

Alexandrova, J. & Megorskaya, I. 2015. Build Artifact. Verkkojulkaisu. Viitattu 11.3.2018. https://confluence.jetbrains.com/display/TCD9/Build+Artifact

Alexandrova, J. & Yargo, Y. 2018. REST API. Verkkosivusto. Viitattu 19.2.2018. https://confluence.jetbrains.com/display/TCD10/REST+API

DevOps: Breaking the Development-Operations barrier. N.d. Atlassian. Verkkojulkaisu. Viitattu 11.3.2018. https://www.atlassian.com/devops

Autodesk.AutoCAD.Runtime.CommandFlags Enumeration. N.d. AutoDesk knowledge network. Verkkojulkaisu. Viitattu 5.11.2017. http://help.autodesk.com/view/OARX/2018/ENU/?guid=OREFNET-Autodesk_AutoCAD_Runtime_CommandFlags

Automated vs. Manual Testing: The Pros and Cons of Each. N.d. Base36. Verkkojulkaisu. Viitattu 10.9.2017. http://www.base36.com/2013/03/automated-vs-manual-testing-the-pros-and-cons-of-each/

Bartlett, J. 2016. What Is A Test Case In Software Testing? Verkkojulkaisu. Viitattu 2.3.2018. https://blog.testlodge.com/what-is-a-test-case-in-software-testing/

Bay, C. 2016. Graphics tests, the last line of automated testing. Verkkojulkaisu. Viitattu 10.9.2017. https://blogs.unity3d.com/2016/12/21/graphics-tests-the-last-line-of-automated-testing/

Below are the options to self-host GitLab. N.d. GitLab. Verkkojulkaisu. Viitattu 21.1.2018. https://about.gitlab.com/products/

Bhatia, H. 2017. Continuous Delivery With GoCD. Verkkojulkaisu. Viitattu 16.9.2017. http://site.clairvoyantsoft.com/continuous-delivery-gocd/

Brockschmidt, K. & Myers, A. 2018. An Introduction to NuGet. Verkkojulkaisu. Viitattu 28.2.2018. https://docs.microsoft.com/en-us/nuget/what-is-nuget

Brown, J. 2008. Announcing Gallio and MbUnit v3.0.4. Verkkojulkaisu. Viitattu 5.11.2017. http://blog.bits-in-motion.com/2008/10/announcing-gallio-and-mbunit-v304.html

Build and Release Agents. 2016. Microsoft. Verkkojulkaisu. Viitattu 11.3.2018. https://docs.microsoft.com/en-us/vsts/build-release/concepts/agents/agents

Buy TeamCity. N.d. JetBrains. Verkkosivusto. Viitattu 5.11.2017. https://www.jetbrains.com/teamcity/buy/#license-type=new-license

Caum, C. 2013. Continuous Delivery Vs. Continuous Deployment. Verkkojulkaisu. Viitattu 14.9.2017. https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff

Collins-Sussman, B. & Fitzpatrick, B. & Pilato, C. N.d. What Is Subversion? Verkkojulkaisu. Viitattu 2.3.2018. http://svnbook.red-bean.com/en/1.6/svn.intro.whatis.html

Common Compiler Infrastructure: Metadata API. N.d. Verkkojulkaisu. Viitattu 10.3.2018. https://archive.codeplex.com/?p=ccimetadata

Continuous Integration and Delivery. N.d. Microsoft. Verkkojulkaisu. Viitattu 13.11.2017. https://www.visualstudio.com/team-services/continuous-integration/

Considering TeamCity for Continuous delivery? Here is what you need to know. 2017. nClouds. Verkkojulkaisu. Viitattu 19.9.2017. https://www.nclouds.com/blog/continuous-delivery-using-teamcity/

Cristof, A. 2017. What exactly is a software framework? Verkkojulkaisu. Viitattu 2.3.2018. https://www.quora.com/What-exactly-is-a-software-framework

Dahlby, K. 2011. Allowing a Windows Service to Interact with Desktop without LocalSystem. Verkkojulkaisu. Viitattu 13.11.2017. https://lostechies.com/keithdahlby/2011/08/13/allowing-a-windows-service-to-interact-with-desktop-without-localsystem/

de Halleux, J. 2004. MbUnit: Generative Unit Test Framework. Verkkojulkaisu. Viitattu 15.11.2017. https://www.codeproject.com/Articles/6060/MbUnit-Generative-Unit-Test-Framework

Display Drawing Extents and Limits (.NET). 2015. AutoDesk. Verkkojulkaisu. Viitattu 3.3.2018. https://knowledge.autodesk.com/search-result/caas/CloudHelp/cloudhelp/2016/ENU/AutoCAD-NET/files/GUID-70B3140A-14F1-4F54-AF5F-8DEDB444C3BA-htm.html

DocumentExtension.CapturePreviewImage Method. N.d. AutoDesk knowledge network. Verkkojulkaisu. Viitattu 13.11.2017. http://help.autodesk.com/view/OARX/2018/ENU/?guid=OREFNET-Autodesk_AutoCAD_ApplicationServices_DocumentExtension_CapturePreviewImage_this_Document_uint_modoptIsLong_uint_modoptIsLong

Eastwood, E. 2017. Add online artifacts for HTML files. Verkkojulkaisu. Viitattu 4.2.2018. https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/14399

Ejaz, M. 2014. Re: How is GO different/better than Jenkins? Forumviesti. Viitattu 16.9.2017. https://groups.google.com/forum/#!topic/go-cd/SlfYlFjnA7o

Elastic agents. N.d. GoCD. Verkkojulkaisu. Viitattu 31.1.2018. https://plugin-api.gocd.org/current/elastic-agents/#elastic-agents

Gallio/mbunit-v3. N.d. GitHub. Verkkojulkaisu. Viitattu 5.11.2017. https://github.com/Gallio/mbunit-v3/blob/master/src/Extensions/AutoCAD/Gallio.AutoCAD.Plugin/Commands.cs

GitLab API. N.d. GitLab. Verkkojulkaisu. Viitattu 4.3.2018. https://docs.gitlab.com/ee/api/README.html

GitLab Runner. 2017. GitLab. Verkkojulkaisu. Viitattu 4.3.2018. https://docs.gitlab.com/runner/

Goncalves, A. 2015. StackOverflow. Forumviesti. Viitattu 19.2.2018. https://stackoverflow.com/questions/31262545/load-custom-net-dll-inside-accoreconsole-exe

How Important is Test Automation in a Software Project? 2013. Segue Technologies. Verkkojulkaisu. Viitattu 4.3.2018. https://www.seguetech.com/important-test-automation/

Integration Testing. N.d. Software Testing Fundamentals. Verkkojulkaisu. Viitattu 18.11.2017. http://softwaretestingfundamentals.com/integration-testing/

Introduction. N.d. Go. Verkkojulkaisu. Viitattu 4.3.2018. https://api.gocd.org/current/#introduction

Iyer, G. 2009. MbUnit + AutoCAD = Unit testing for CAD Plugins. Verkkojulkaisu. Viitattu 7.10.2017. http://ossandcad.blogspot.fi/2009/04/mbunit-autocad-unit-testing-for-cad.html

Japikse, P. 2008. Unit Testing with MbUnit (An Introduction). Verkkojulkaisu. Viitattu 4.3.2018. http://www.skimedic.com/blog/post/2008/05/31/Unit-Testing-with-MBUnit-(An-Introduction).aspx

Kawaguchi, K. & Soref, J. 2018. Distributed builds. Verkkojulkaisu. Viitattu 28.1.2018. https://wiki.jenkins.io/display/JENKINS/Distributed+builds

Manolov, I. 2009. Introduction to TestApi – Part 3: Visual Verification APIs. Verkkojulkaisu. Viitattu 10.9.2017. https://blogs.msdn.microsoft.com/ivo_manolov/2009/04/20/introduction-to-testapi-part-3-visual-verification-apis/

Maksimovic, Z. 2012. List of Microsoft.NET IL disassemblers. Verkkojulkaisu. Viitattu 11.3.2018. https://www.agile-code.com/blog/list-of-microsoft-net-il-disassemblers/

McFarlin, T. 2012. The Beginner's Guide to Unit Testing: What is Unit Testing? Verkkojulkaisu. Viitattu 18.11.2017. https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728

Microsoft/cci. N.d. GitHub. Verkkojulkaisu. Viitattu 15.2.2018. https://github.com/Microsoft/cci

Mo, C. 2017. Evaluating GoCD vs Spinnaker. Verkkojulkaisu. Viitattu 16.9.2017. http://blog.armory.io/evaluating-gocd-vs-spinnaker/

MSBuild Command-Line Reference. N.d. Microsoft. Verkkojulkaisu. Viitattu 26.2.2018. https://msdn.microsoft.com/en-us/library/ms164311.aspx

Nagele, C. N.d. An introduction to version control. Verkkojulkaisu. Viitattu 10.3.2018. http://guides.beanstalkapp.com/version-control/intro-to-version-control.html

Najjar, A. N.d. Jenkins vs. TeamCity. Verkkojulkaisu. Viitattu 14.9.2017. http://galilsoftware.com/jenkins-vs-teamcity/

Ngan, K. N.d. What is Git? Verkkojulkaisu. Viitattu 2.3.2018. https://www.visualstudio.com/learn/what-is-git/

Passant, H. 2016. C# Decompilers? Verkkojulkaisu. Viitattu 9.3.2018. https://stackoverflow.com/questions/4480091/c-sharp-decompilers

Phillips, A. 2014. Continuous Delivery Pipeline – What it is and Why it's so Important in Developing Software. Verkkojulkaisu. Viitattu 16.9.2017. https://devops.com/continuous-delivery-pipeline/

Pipeline as Code with Jenkins. N.d. Jenkins. Verkkojulkaisu. Viitattu 16.9.2017. https://jenkins.io/solutions/pipeline/

Plugin. 2017. Computer Hope. Verkkojulkaisu. Viitattu 16.3.2018. https://www.computerhope.com/jargon/p/plugin.htm

Pre-Tested Commit: No broken code in your version control. Ever. N.d. JetBrains. Verkkojulkaisu. Viitattu 19.9.2017. https://www.jetbrains.com/teamcity/features/delayed_commit.html

Ramamoorthy, B. 2012. Changing visual style. Verkkoblogi. Viitattu 3.3.2018. http://adndevblog.typepad.com/autocad/2012/03/changing-visual-style-using-autocad-net-api.html

Ramos, M. 2016. Continuous Integration, Delivery, and Deployment with GitLab. Verkkojulkaisu. Viitattu 26.2.2018. https://about.gitlab.com/2016/08/05/continuous-integration-delivery-and-deployment-with-gitlab/

Remco. 2016. Exception when building with NCrunch 2015 2.11.023. Forum viesti. Viitattu 17.2.2018. http://forum.ncrunch.net/yaf_postst1460_Exception-when-building-with-NCrunch-2015-2-11-023.aspx

Remote access API. N.d. Jenkins. Verkkosivusto. Viitattu 19.2.2018. https://wiki.jenkins.io/display/JENKINS/Remote+access+API

REST API Overview for Visual Studio Team Services and Team Foundation Server. 2017. Microsoft. Verkkojulkaisu. Viitattu 16.11.2017. https://www.visualstudio.com/en-us/docs/integrate/api/overview

Shannon, R. 2012. What is HTML? Verkkojulkaisu. Viitattu 11.3.2018. http://www.yourhtmlsource.com/starthere/whatishtml.html

Startup switches for AutoCAD. N.d. AutoDesk knowledge network. Verkkojulkaisu. Viitattu 5.11.2017. https://knowledge.autodesk.com/support/autocad/learn-explore/caas/sfdcarticles/sfdcarticles/Startup-switches-for-AutoCAD.html

Tenhundfeld, A. 2008. Gallio .NET Test Automation Platform. Verkkojulkaisu. Viitattu 7.10.2017. https://www.infoq.com/news/2008/07/Gallio

TeamCity vs Jenkins for Continuous Integration. 2017. UpGuard. Verkkojulkaisu. Viitattu 14.9.2017. https://www.upguard.com/articles/teamcity-vs.-jenkins-for-continuous-integration

Team Foundation Server. N.d. Microsoft. Verkkojulkaisu. Viitattu 13.11.2017. https://www.visualstudio.com/tfs/

Team Foundation Server Pricing. N.d. Microsoft. Verkkojulkaisu. Viitattu 13.11.2017. https://www.visualstudio.com/team-services/tfs-pricing/

Virtualization. N.d. Technopedia. Verkkojulkaisu. Viitattu 11.3.2018. https://www.techopedia.com/definition/719/virtualization

TestApi - a library of Test APIs. 2011. CodePlex Archive. Verkkojulkaisu. Viitattu 13.11.2017. http://testapi.codeplex.com/

Trifonov, V. 2012. Compare Two DLL Files Programmatically Using Hash. Code Project. Verkkojulkaisu. Viitattu 26.2.2018. https://www.codeproject.com/Articles/501631/Compare-two-DLL-files-programmatically-using-hash

Welcome to VirtualBox.org! N.d. VirtualBox. Verkkojulkaisu. Viitattu 4.3.2018. https://www.virtualbox.org/

Visual Studio IDE. N.d. Microsoft. Verkkojulkaisu. Viitattu 4.3.2018. https://www.visualstudio.com/vs/

Walmsley, K. 2012. The AutoCAD 2013 Core Console. Verkkoartikkeli. Viitattu 19.2.2018. http://through-the-interface.typepad.com/through_the_interface/2012/02/the-autocad-2013-core-console.html

What is an API? (Application Programming Interface). N.d. MuleSoft. Verkkojulkaisu. Viitattu 10.3.2018. https://www.mulesoft.com/resources/api/what-is-an-api

What is AutoCAD? N.d. Study.com. Verkkojulkaisu. Viitattu 9.9.2017. http://study.com/what_is_auto_cad.html

What is Jenkins? 2016. Edureka. Verkkojulkaisu. Viitattu 14.9.2017. https://www.edureka.co/blog/what-is-jenkins/

What is NuGet? N.d. NuGet. Verkkojulkaisu. Viitattu 4.3.2018. https://www.nuget.org/

What is version control? N.d. Atlassian Bitbucket. Verkkojulkaisu. Viitattu 10.3.2018. https://www.atlassian.com/git/tutorials/what-is-version-control

What is XML? N.d. The XML FAQ. Verkkojulkaisu. Viitattu 11.3.2018. http://xml.silmaril.ie/whatisxml.html

What is .NET? N.d. Microsoft. Verkkojulkaisu. Viitattu 10.3.2018. https://www.microsoft.com/net/learn/what-is-dotnet

WHY GoCD? N.d. GoCD. Verkkojulkaisu. Viitattu 19.9.2017. https://www.gocd.org/why-gocd/

XMLUnit. N.d. XMLUnit. Verkkojulkaisu. Viitattu 15.11.2017. http://www.xmlunit.org/

Liitteet

Liite 1. Gitlab-ci.yml

```
stages:
  - Build
  - Unit test
  - Integration test
  - Deployment

Build:
 stage: Build
 script:
  - Pipeline/Scripts/Build.bat %BUILD_CONFIGURATION%
 artifacts:
  name: "build_binaries"
  when: on_success
  expire_in: 1 day
  paths:
  - Project/bin
  - IntegrationTests/bin
  - UnitTests/bin
  - DeploymentUtilities/bin

Unit Tests:
 stage: Unit test
 script: Pipeline/Scripts/UnitTests.bat
 artifacts:
  when: always
  name: "unit_test_reports"
  paths:
  - UnitReports

Integration Tests:
 stage: Integration test
 script: Pipeline/Scripts/IntegrationTests.bat
 artifacts:
  when: always
  name: "integration_test_reports"
  paths:
  - IntegrationReports

Deploy to production:
  stage: Deployment
  when: manual
  environment:
    name: Production
  script: powershell.exe -ExecutionPolicy ByPass -File Pipeline\Scripts\Deploy.ps1 -serverUrl https://10.0.2.15 \
          -projectId %CI_PROJECT_ID% -jobId %CI_JOB_ID% -certificatePath C:\Opari\nodeserveri\certs\10.0.2.15.crt \
          -buildConfiguration %BUILD_CONFIGURATION% -testBinariesFolder Project\bin\ -deploymentBinariesFolder C:\Deployment
```

Liite 2. Kyselijäpalvelun alustaminen

```
const path = require("path"),
      express = require("express"),
      request = require('request') // for creating a https-request
      fs = require('fs'), // for reading certificates
      { exec } = require('child_process'),
      Pipeline = require('./pipeline.js'); // pipeline class

var DIST_DIR = path.join(__dirname, "dist"),
    PORT = 3000,
    app = express(),
    waitTime = 20000;

//Serving the files on the dist folder
app.use(express.static(DIST_DIR));

// Send index.html when clients navigate to this servers url, in case somebody wants to check in on the poller status
// Could be changed to show the latest log entries of the service, for easy log access.
app.get("*", function (req, res) {
  res.sendFile(path.join(__dirname, "index.html"));
});

app.listen(PORT);

// == GitLab api stuff below

const options =
{
    ca:     fs.readFileSync(path.join(__dirname, "certs/192.168.176.101.crt")) ,
    url:    'https://192.168.176.101/api/v4/projects/3/pipelines' ,
    headers:
    {
        'Private-token': "Sarh1vT34ByMZ4djzGR-" , // get this token from gitlab
        'User-agent': 'request'
    }
};

var pipelines = new Array();
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0"; // ONLY NEEDED FOR SELF SIGNED CERTIFICATES!
var VMName = "asd";

// this starts the api calls
CheckPipelines(WaitAndRecheck);
```

Liite 3. Kyselijäpalvelimen funktiot

```
function ParseJsonToArray(data)
{
    pipelines = new Array();
    var json = JSON.parse(data);

    for(var i = 0; i < json.length; i++)
    {
        if(json[i].status == 'running' || json[i].status == 'pending')
        {
            var pl = new Pipeline();
            pl.id = json[i].id;
            pl.status = json[i].status;
            pl.sha = json[i].sha;
            pl.ref = json[i].ref;
            console.log("Found pipeline! ID:" + pl.id + ", status:" + pl.status);
            pipelines.push(pl);
        }
    }
}

function RunCommand(command)
{
    exec(command, (err, stdout, stderr) => {
        if(err)
        {
            console.log(err);
            return;
        }

        console.log(stdout);
        console.log(stderr);
    });
}

function CheckPipelines(callback)
{
    console.log('\nCreating a request...');

    request(options, function (error, response, body)
    {
        if(error)
        {
            console.log('ERROR: ' + error);
        }
        else if(response.statusCode == 200)
        {
            console.log('Response ok...');
            ParseJsonToArray(body);

            if(pipelines.length > 0)
            {
                for(var i = 0; i < pipelines.length;i++)
                {
                    if(pipelines[i].status == 'pending')
                    {
                        console.log('Pending pipeline found. Booting VM.');
                        RunCommand('"C:\\Program Files\\Oracle\\VirtualBox\\VBoxManage.exe" startvm ' + VMName + ' --type headless');
                        break;
                    }
                }
            }
            else
            {
                console.log("No pipelines active, shutting down running virtual machines");
                RunCommand('"C:\\Program Files\\Oracle\\VirtualBox\\VBoxManage.exe" controlvm ' + VMName + ' acpipowerbutton');
            }
        }
        else
            console.log('Request failed. Response status code: ' + response.statusCode);

        console.log('\nDone. Waiting for ' + waitTime + "ms until next call.\n");

    });

    callback();
}

function WaitAndRecheck()
{
    setTimeout(function() { CheckPipelines(WaitAndRecheck) }, waitTime);
}
```

Liite 4. CADTestSuite-luokan toiminnallisuutta

```
#region    METHODS

/// <summary> Loads a drawing into AutoCAD synchronously from the given <paramref name="filepath"/> </summary>
/// <param name="filepath"> The absolute path to the AutoCAD drawing to be loaded. </param>
1 reference | prome, 5 days ago | 1 author, 1 change
public void SetupDrawing (string filepath)
{
  Document doc = null;
  DocumentCollection docManager = Autodesk.AutoCAD.ApplicationServices.Application.DocumentManager;

  if (docManager != null)
  {
    docManager.AppContextOpenDocument(filepath);
    doc = docManager.MdiActiveDocument;
    if (doc == null)
      doc = docManager.CurrentDocument;
  }
}

#endregion  METHODS

#region    OVERRIDES

/// <summary> Runs before each contained test case. </summary>
1 reference | prome, 5 days ago | 1 author, 1 change
protected override void OnSetupSelf ()
{
  if (this.drawingToLoad != null)
    this.SetupDrawing(drawingToLoad);
  base.OnSetupSelf();
}

/// <summary> Runs after each contained test case. </summary>
1 reference | prome, 5 days ago | 1 author, 1 change
protected override void OnTearDownSelf ()
{
  if (Application.DocumentManager.MdiActiveDocument != null)
    Application.DocumentManager.MdiActiveDocument.CloseAndDiscard();
  base.OnTearDownSelf();
}

#endregion  OVERRIDES
```

Liite 5. RunSnapshotTests-funktio

```
private void RunSnapshotTests (Document doc, string testDirectory, List<string> failures)
{
  try
  {
    Extents3d totalExtents = this.GetModelSpaceExtents(doc.Database);
    if (!totalExtents.MaxPoint.Equals(totalExtents.MinPoint))
    {
      Array directions = Enum.GetValues(typeof(ViewDirection));
      foreach (ViewDirection vDir in directions)
        try
        {
          string errorMessage = string.Empty;
          if (!this.SnapshotTest(testDirectory, doc, totalExtents, vDir, ref errorMessage))
            failures.Add(errorMessage);
        }
        catch (Exception loopException)
        {
          try
          {
            failures.Add(loopException.ToString());
          }
          catch
          { // Calling ToString on AutoCAD exceptions can cause problems
            failures.Add(loopException.Message);
          }
        }
    }
    else
      failures.Add("Zeroextents found, skipping test.");
  }
  catch (Exception exc)
  {
    try
    {
      failures.Add(String.Format("Exception! Directory: {0}, message: {1}", testDirectory, exc.ToString()));
    }
    catch
    { // Calling ToString on AutoCAD exceptions can cause problems
      failures.Add(String.Format("Exception! Directory: {0}, message: {1}", testDirectory, exc.Message));
    }
  }
}
```

Liite 6. GetViewVector-funktio

```
private Vector3d GetViewVector (ViewDirection vDir)
{
  Vector3d viewDir = new Vector3d();
  switch (vDir)
  {
    case ViewDirection.Top:
      viewDir = Vector3d.ZAxis; break;
    case ViewDirection.Bottom:
      viewDir = Vector3d.ZAxis.Negate(); break;
    case ViewDirection.Front:
      viewDir = Vector3d.YAxis.Negate(); break;
    case ViewDirection.Back:
      viewDir = Vector3d.YAxis; break;
    case ViewDirection.Left:
      viewDir = Vector3d.XAxis.Negate(); break;
    case ViewDirection.Right:
      viewDir = Vector3d.XAxis; break;
    case ViewDirection.SeIso:
      viewDir = new Vector3d(1.0, -1.0, 1.0); break;
    case ViewDirection.SwIso:
      viewDir = new Vector3d(-1.0, -1.0, 1.0); break;
    case ViewDirection.NeIso:
      viewDir = new Vector3d(1.0, 1.0, 1.0); break;
    case ViewDirection.NwIso:
      viewDir = new Vector3d(-1.0, 1.0, 1.0); break;
    default:
      throw new ArgumentException("Invalid ViewDirection: " + vDir);
  }

  return viewDir;
}
```

Liite 7. Deploy.ps1 skripti

```
Param([string]$serverUrl="", [string]$projectId="", [string]$jobId="", [string]$certificatePath="", [string]$buildConfiguration="", [string]$testBinariesFolder="", [string]$deploymentBinariesFolder="")

# Print variables for the GitLab logs
Write-Host "Parameter serverUrl: " $serverUrl;
Write-Host "Parameter projectId: " $projectId;
Write-Host "Parameter jobId: " $jobId;
Write-Host "Parameter certificatePath: " $certificatePath;
Write-Host "Parameter buildConfiguration: " $buildConfiguration;
Write-Host "Parameter testBinariesFolder: " $testBinariesFolder;
Write-Host "Parameter deploymentBinariesFolder: " $deploymentBinariesFolder;

# Define variables
$directoryPath = [environment]::CurrentDirectory;
$updaterUtility = "$($directoryPath)\DeploymentUtilities\bin\$($buildConfiguration)\DeploymentUtilities.dll";
$newtonsoft = "$($directoryPath)\DeploymentUtilities\bin\$($buildConfiguration)\NewtonSoft.Json.dll";

# Load assemblies
[System.Reflection.Assembly]::LoadFile($updaterUtility);
[System.Reflection.Assembly]::LoadFile($newtonsoft);

# Make sure the folder exists
If(!(test-path $deploymentBinariesFolder))
{
    Write-Host "Creating a folder for the new deployment clone...";
    New-Item -ItemType Directory -Force -Path $deploymentBinariesFolder

    # TODO SVN CHECKOUT TO $deploymentBinariesFolder
}
else
{
    # TODO SVN UPDATE AT $deploymentBinariesFolder
}

# Overwrites the dll files in the deployment folder, that are different from the files in the testFolder
Write-Host "Checking for differences in files...";
if([DeploymentUtilities.FileUpdater]::Run($testBinariesFolder, $deploymentBinariesFolder))
{
    Write-Host "Differences found, deploying!";

    # Build the commit message
    $commitMessage = [DeploymentUtilities.Requester]::CreateDeploymentMessage($serverUrl, $projectId, $jobId, $certificatePath);

    # TODO Commit/push using commitmessage
}
else
{
    throw "No differences found, no need to deploy.";
}
```

Liite 8. Hash-arvojen tuotto ja vertailu

```
/// <summary> Gets the assembly file hash. </summary>
/// <param name="path">Assembly file path.</param>
/// <returns>The hash of a disassembled assembly.</returns>
private static string GetAssemblyFileHash (string path)
{
  string tempFile = FileUpdater.GetDisassembledFile(path);
  return FileUpdater.CalculateHashFromStream(File.OpenRead(tempFile));
}

/// <summary> Calculates the hash from a stream. </summary>
/// <param name="stream"> The target stream to calculate hash from. </param>
/// <returns> The hash. </returns>
private static string CalculateHashFromStream (Stream stream)
{
  using (var readerSource = new System.IO.BufferedStream(stream, 1200000))
  using (var md51 = new System.Security.Cryptography.MD5CryptoServiceProvider())
  {
    md51.ComputeHash(readerSource);
    return Convert.ToBase64String(md51.Hash);
  }
}

/// <summary> Gets the disassembled version of an assembly file, with it's build-specific data removed. </summary>
/// <param name="assemblyFilePath"> The file path. </param>
/// <returns> Disassembled version of an assembly file. </returns>
private static string GetDisassembledFile (string assemblyFilePath)
{
  if (!File.Exists(assemblyFilePath))
    throw new InvalidOperationException(string.Format("The file {0} does not exist!", assemblyFilePath));

  string tempFileName = Path.GetTempFileName();
  ProcessStartInfo startInfo = new ProcessStartInfo(FileUpdater.ILDasmFileLocation, string.Format(FileUpdater.ildasmArguments, assemblyFilePath));
  startInfo.WindowStyle = ProcessWindowStyle.Hidden;
  startInfo.CreateNoWindow = true;
  startInfo.UseShellExecute = false;
  startInfo.RedirectStandardOutput = true;

  using (Process process = Process.Start(startInfo))
  {
    string output = process.StandardOutput.ReadToEnd();
    process.WaitForExit();

    if (process.ExitCode > 0)
      throw new InvalidOperationException(string.Format("Generating IL code for file {0} failed with exit code - {1}. Log: {2}",
                                                        assemblyFilePath, process.ExitCode, output));

    File.WriteAllText(tempFileName, output);
  }

  FileUpdater.RemoveUnnededRows(tempFileName);
  return tempFileName;
}
```

Liite 9. Ongelmarivien parsintatoiminnallisuus

```
/// <summary> Removes unwanted data from the file. </summary>
/// <param name="fileName"> The target file. </param>
private static void RemoveUnnededRows (string fileName)
{
  string fileContent = File.ReadAllText(fileName);

  //remove MVID
  fileContent = regexMVID.Replace(fileContent, string.Empty);
  //remove Image Base
  fileContent = regexImageBase.Replace(fileContent, string.Empty);
  //remove Time Stamp
  fileContent = regexTimeStamp.Replace(fileContent, string.Empty);

  File.WriteAllText(fileName, fileContent);
}

#endregion  STATIC METHODS

#region STATIC FIELDS

/// <summary> Specifies the regex for removing MVID data. </summary>
private static Regex regexMVID = new Regex("//\\s*MVID\\:\\s*\\{[a-zA-Z0-9\\-]+\\}", RegexOptions.Multiline | RegexOptions.Compiled);

/// <summary> Specifies the regex for removing image base data. </summary>
private static Regex regexImageBase = new Regex("//\\s*Image\\s+base\\:\\s0x[0-9A-Fa-f]*", RegexOptions.Multiline | RegexOptions.Compiled);

/// <summary> Specifies the regex for removing timestamp data. </summary>
private static Regex regexTimeStamp = new Regex("//\\s*Time-date\\s+stamp\\:\\s*0x[0-9A-Fa-f]*", RegexOptions.Multiline | RegexOptions.Compiled);

/// <summary> Specifies the file location for ildasm.exe. </summary>
private static string ILDasmFileLocation = Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), "ildasm.exe");

#endregion

#region     CONSTANTS

/// <summary> Specifies arguments for ildasm.exe. </summary>
private const string ildasmArguments = "/all /text \"{0}\"";

#endregion  CONSTANTS
```

Liite 10. Kyselyn rakentaminen ja vastauksen parsinta

```
/// <summary> Creates a request. </summary>
/// <param name="gitlabUrl">The url of the GitLab coordinator server. Protocol must be included. May not be <c>null</c>.</param>
/// <param name="projectId">The id of the target project in GitLab. May not be <c>null</c>.</param>
/// <param name="jobId">The id of the deployment job in GitLab. May not be <c>null</c>.</param>
/// <param name="certificatePath">The path to the certificate used to contact the GitLab api.</param>
/// <returns>A new request.</returns>
1 reference | prome, 6 hours ago | 1 author, 4 changes
private static HttpWebRequest CreateRequest (string gitlabUrl, string projectId, string jobId, string certificatePath)
{
  string fullAddress = string.Format("{0}/api/v4/projects/{1}/jobs/{2}", gitlabUrl, projectId, jobId);

  HttpWebRequest request = (HttpWebRequest)WebRequest.Create(fullAddress);
  ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;

  if (!string.IsNullOrEmpty(certificatePath))
  {
    X509Certificate cert = new X509Certificate(certificatePath);
    request.ClientCertificates.Add(cert);
  }

  WebHeaderCollection headers = new WebHeaderCollection();
  headers.Add("Private-token", "RrvDGuy8ZKBJJZsxwjYB"); // generate this in gitlab user settings
  request.Headers = headers;

  return request;
}

/// <summary> Parses the sought data from the response. </summary>
/// <param name="responseString">The response to parse.</param>
/// <returns>The parsed data.</returns>
1 reference | prome, 6 hours ago | 1 author, 1 change
private static string BuildMessageFromResponse (string responseString)
{
  try
  {
    Job job = JsonConvert.DeserializeObject<Job>(responseString);
    if (job != null)
      return string.Format("Deployer: {0}/{1}, Committer: {2}, Commit message: {3}", job.User.Name, job.User.UserName, job.Commit.AuthorName, job.Commit.Message);
    else
      throw new NullReferenceException("Job data was null at Requester.ParseResponse()!");
  }
  catch (Exception exc)
  {
    Exception newExc = new Exception("The response could not be parsed in Requester.ParseResponse()! Inner exception message:\n" + exc.ToString());
    throw newExc;
  }
}
```