Tampereen ammattikorkeakoulu

Ohjelmien automaattinen testaus

Toni Pakarinen

OPINNÄYTETYÖ
Lokakuu 2020

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

PAKARINEN, TONI:
Ohjelmien automaattinen testaus

Opinnäytetyö 56 sivua
Lokakuu 2020

Ohjelmiston testaus on yksi merkittävimmistä työvaiheista kehitettäessä uusia ohjelmia tietokoneille ja mobiililaitteille. Manuaalisesti tehtynä se on aikaa vievää ja kallista. Ohjelmistojen laajentuessa ja monimutkaistuessa on automaattisen testauksen merkitys kasvanut valtavasti. Se ei sido työntekijöitä, joten he voivat tehdä muita töitä. Lisäksi se nopeuttaa testaamista, mikä säästää aikaa ja rahaa. Niinpä hyvä automaattinen testausohjelma tuottaa merkittävän hyödyn kehittäjälleen.

Testityön päämääränä on kehittää ohjelmia, jotka toimivat mahdollisimman ongelmattomasti halutussa käyttöympäristössä. Ohjelmistotestausta tehdään siten koko sen kehityskaaren ajan. Mitä aikaisemmassa ohjelmiston kehitysvaiheessa virhe tai puute huomataan, sitä helpompi se on korjata ja samalla kustannukset pysyvät pieninä.

Automaattisen testauksen uudesta ohjelmaversiosta suorittaa tietokone automaattisesti. Tämä ei kuitenkaan poista manuaalisen testauksen tarvetta vaan täydentää sitä. Lisäksi automaattinen testaus vaatii jatkuvaa kehitystä ja ylläpitoa. Eri käyttötarkoituksiin soveltuvia automaattisia testiohjelmia on lukuisia. Oman haasteensa automaattisen testaukseen tuo oikean testiohjelman valinta ja testausautomaation opettelu.

Opinnäytetyössä käydään läpi merkittävimmät automaattisen testauksen ilmaiset tai lähes ilmaiset ohjelmistot. Näitä tarkastellaan niiden taustojen ja ominaisuuksien näkökulmasta. Testiohjelmistojen toimintaa havainnollistetaan muutamien yksinkertaisten testiesimerkkien avulla. Työssä tutustutaan seuraaviin automaattisen testauksen ohjelmistoihin: Postman, Apache JMeter, Selenium, Appium, Robot Framework ja Cypress. Lisäksi työssä tutustutaan muutamiin yleisiin automaatiotestauksen apuohjelmiin, työkaluihin ja palveluihin.

Asiasanat: automaattinen testaus, testaus, testausmenetelmät, testausohjelmat, testausapuohjelmat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

PAKARINEN TONI:
Software automating testing

Bachelor's thesis 56 pages
October 2020

Software testing is one of the most significant steps in developing new software for computers and mobile devices. When done manually, it is time consuming and expensive. As software expands and becomes more complex, the importance of automated testing has grown tremendously. It does not bind employees, so they can do other jobs. In addition, it speeds up testing, which saves time and money. Thus, a good automated testing program provides significant benefits to its developer.

The aim of the test work is to develop programs that work as seamlessly as possible in the desired operating environment. Software testing is thus performed throughout its development cycle. The earlier an error or deficiency is detected in the software development phase, the easier it is to fix and at the same time the cost remains low.

Automatic testing of the new software version is performed by the computer automatically. However, this does not eliminate the need for manual testing but complements it. In addition, automated testing requires continuous development and maintenance. There are numerous automated test programs suitable for different uses. Choosing the right test program and learning test automation brings its own challenges to automated testing.

The thesis reviews the most significant free or almost free software for automatic testing. These are viewed in terms of their backgrounds and characteristics. The operation of the test software is illustrated by a few simple test examples. The following automatic testing software is introduced: Postman, Apache JMeter, Selenium, Appium, Robot Framework and Cypress. In addition, the work introduces a few common automation testing utilities, tools, and services.

Key words: automated testing, testing, testing methods, testing programs, testing utilities

SISÄLLYS

LYHENTEET JA TERMIT

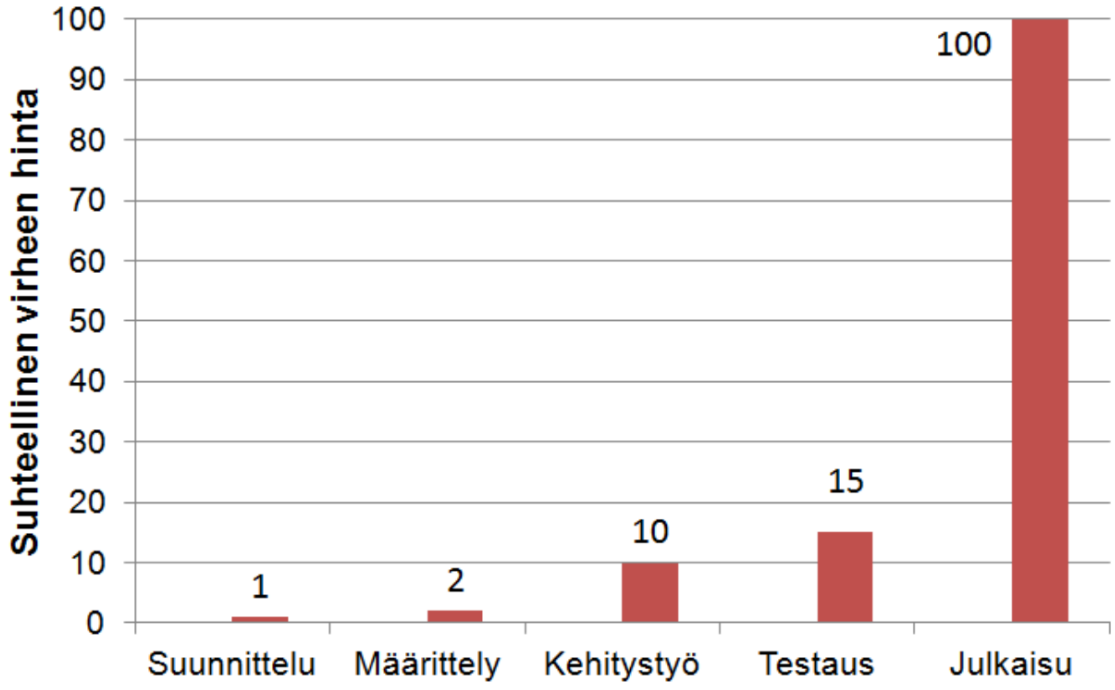
API	Application programming interface. Ohjelmointirajapinta, joka mahdollistaa eri ohjelmisto-osien kommunikoinnin
Java	Ohjelmointikieli ja ohjelmistoalusta
JavaScript	Web-ohjelmointikieli. Lyhenne JS
Python	Ohjelmointikieli
PostgreSQL	Tietokanta
macOS	Applen luoma käyttöjärjestelmä. Aiempia nimiä on OS X ja Mac OS X
GUI	Graphical user interface. Graafinen käyttöliittymä
C#	Ohjelmointikieli
PHP	Web-ohjelmointikieli
Perl	Ohjelmointikieli
Ruby	Ohjelmointikieli
.Net	Ohjelmistokomponenttikirjasto
Android	Linux-käyttöjärjestelmä-pohjainen mobiilikäyttöjärjestelmä
IOS	Applen mobiilikäyttöjärjestelmä
W3C	World Wide Web Consortium. Yritysten ja yhteisöjen yhteenliittymä, joka ylläpitää, sekä kehittää internetin standardeja ja suosituksia
NodeJS	Avoimen lähdekoodin alusta JavaScript koodin suorittamiseen palvelimilla
DevOps	Toimintamalli palveluiden tuotantoon
XML	Extensible Markup Language. Rakenteellinen kuvauskieli tietomassoille
Docker	Virtuaalikoneiden hallinta ohjelmisto
Kontti	Container. Dockerin yksi virtuaalikone
JSON	JavaScript Object Notation. Avoimen standardin tiedostomuoto tiedon tallennukseen ja välitykseen. Tiedostopääte json

JWT	JSON Web Token. Avoimen standardin menetelmä vaatimusten edustamiseksi turvallisesti kahden osapuolen välillä
React	JavaScript kirjasto
NextJS	React kirjastoon perustuva JavaScript kirjasto
GraphQL	Facebookin luoma API-kyselykieli
Material-UI	Reactille tehty GUI-kirjasto
HTML	Hypertext Markup Language. Perus Web-sivujen kirjoituskieli
HTMLUnit	Selain, jolla ei ole käyttöliittymää. Voidaan simuloida eri selaimia esim. Chrome ja Firefox
Xpath	XML Path Language. Web-sivustojen elementtien tai osien polun osoittaminen

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on luoda katsaus tietokone- ja mobiiliohjelmistojen automaattiseen testaukseen. Lähinnä on tarkoitus keskittyä ilmaisiin automaattisen testauksen ohjelmistoihin.

Saadaksemme toivotun kaltaiset ominaisuudet omaava ohjelmisto on testaus ensiarvoisen tärkeä ohjelmistoprojektin vaihe koko ohjelmiston kehitystyön ajan aina alkumetreiltä tuotteen luovutushetkeen. Se on aikaa vievin ja siten myös kallein työ. Näin ajatellen hyvä automaattinen testausohjelma tuottaa merkittävän taloudellisen hyödyn ohjelmiston kehittäjälle (kuvio 1). Tänä päivänä automaattinen testaus onkin kovan kehitystyön kohteena yllä mainittujen etujen vuoksi.



KUVIO 1. Testauksen suhteellinen kustannuskäyrä. Suunnittelu vaiheessa löydetty virhe maksaa kymmenesosan kehitystyön aikana löydetystä virheestä ja sadasosan julkaisun jälkeen löydetystä virheestä. (Kasurinen 2017, 13.)

Opinnäytetyössä tutustutaan useimmin käytettyjen automaattisten testausohjelmien ominaisuuksiin, pyritään arvioimaan edut ja mahdolliset puutteet ja selvittämään, mihin automaattista testausta voidaan käyttää ohjelmistoja kehitettäessä.

2 TESTAAMISESTA YLEISESTI

Testaamista tehdään ohjelmistotuotteen luonnin ja valmiin tuotteen parissa, ennen sen luovuttamista asiakkaiden käyttöön. Testauksella varmistetaan, että tuote toimii oikein ja on vaatimusmäärittelyjen mukainen eli ”varmistetaan että tehdään oikeaa tuotetta ja että tuote on tehty oikein.” (Kasurinen 2017, 8). Tämä kappale sisältää testaukseen yleisesti liittyvää termistöä ja niiden selityksiä.

2.1 Testaaminen ohjelmiston kehitysvaiheessa

Tuotetta testataan tuotteen luonnin aikana niin, että se vastaa tuotteelle suunniteltua arkkitehtuuria ja vaatimusmäärittelyä. Näin tuotteesta saadaan pahimmat virheet ja puutteet korjattua mahdollisimman aikaisessa vaiheessa. Seuraavaksi tarkastellaan tuotteen luonnin aikana käytettäviä eri testausmenetelmiä ja testauksen työvaiheita. (Kasurinen 2017, 37.)

2.1.1 Staattinen ja dynaaminen testaus

Staattinen testaus (static testing) tarkoittaa ohjelmiston testaamista niin, että ohjelmistoa ei varsinaisesti käytetä. Ohjelmistoa tutkitaan esimerkiksi koodianalysaattoreilla, koodiarvioilla ja arkkitehtuurisuunnittelun näkökulmasta. Staattisella testauksella pyritään tekemään perustason tarkistuksia, ennen kuin aloitetaan ohjelmiston tarkempi testaaminen. (Kasurinen 2017, 37.)

Dynaaminen testaus (dynamic testing) on staattisen testauksen vastakohta. Sillä tarkoitetaan ohjelmiston testaamista niin, että tuotetta käytetään. Ohjelmistolle annetaan esimerkki syötteitä ja katsotaan, miten tuote niihin reagoi. Useammat testausmenetelmät kuuluvat dynaamiseen testaukseen. (Kasurinen 2017, 37.)

2.1.2 Yksikkötestaus

Yksikkötestaus (unit testing) tarkoittaa ohjelmiston yhden moduulin, funktion tai olion toiminnan tarkastelua välittömästi toteutuksen yhteydessä. Yksikkötestaus on kaikista yleisin ja tavallisin testausmuoto, sitä käytetään lähes kaikkialla ohjelmistoalalla. (Kasurinen 2017, 37-39.)

2.1.3 Integraatiotestaus

Integraatiotestaus (integration testing) tarkoittaa ohjelmiston eri osien sovittamista yhteen niin, että ne toimivat yhdessä oikein. Usein joudutaan luomaan tynkäosia integraation testaamiseen. Tämä testausvaihe on yleensä yksikkötestaamisen jälkeen seuraava vaihe. Osien integrointijärjestys toteutetaan yleensä kolmella eri tavalla; alhaalta ylöspäin, ylhäältä alaspäin ja voileipätestauksella. (Kasurinen 2017, 39-40.)

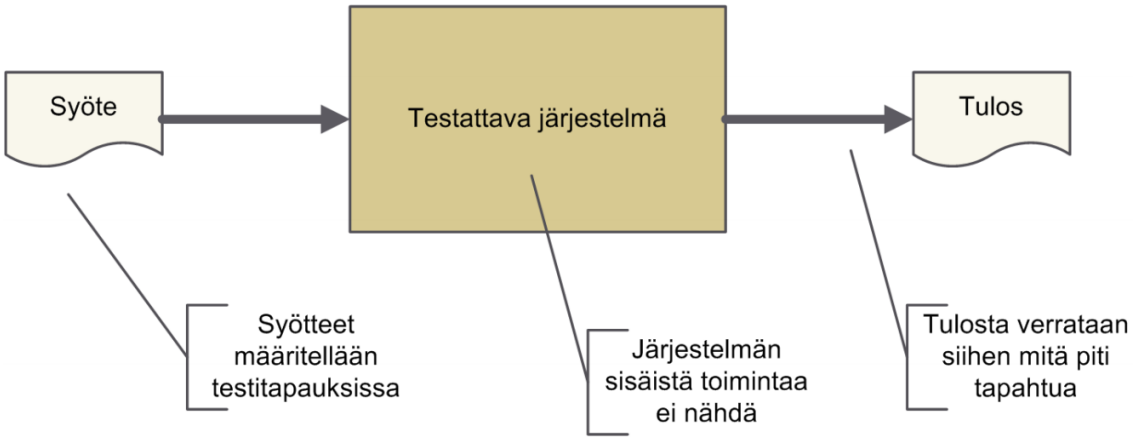
Alhaalta ylöspäin (bottom up testing) tarkoittaa, että ensin järjestelmään integroidaan matalimman tason komponentit (esimerkiksi verkkoyhteydet ja tietokannat), jotka kommunikoivat laitteiston tai järjestelmän kanssa. Sen jälkeen järjestelmään tuodaan seuraavat osat, jne. kunnes kaikki on testattu. **Ylhäältä alaspäin (top down testing**) on sama ajatus kuin alhaalta ylöspäin menetelmässä, mutta testaus aloitetaan järjestelmähierarkian huipulta. Testauksen edetessä siirrytään alemmas järjestelmähierarkiassa, kunnes saavutetaan rautataso. Järjestelmä sisältää kaikki komponentit ja osat heti testauksen alussa. Tämä testausmuoto antaa paremman yleiskuvan järjestelmästä. Se voi myös tuoda esiin toimintojen puutteita. **Voileipätestaus (sandwich testing)** on kahden edellisen testauksen yhdistelmä, jossa integraatiota lähestytään molemmista suunnista samanaikaisesti. Tässä testauksessa yleensä tarvitaan vähiten tynkäosia integraation testaukseen. (Kasurinen 2017, 39-40.)

2.1.4 Järjestelmätestaus

Järjestelmätestaus (system testing) tarkoittaa koko ohjelmiston testaamista yhtenä kokonaisuutena. Tämä testausvaihe tulee integraatiotestauksen jälkeen, kun kaikki osat on liitetty toisiinsa ja kaikki tynkäosat on poistettu. Järjestelmätestaus on yleisnimike kokonaisen ohjelmistojärjestelmän testaukselle. Järjestelmätestauksessa varmistetaan, että ohjelmistojärjestelmä on vaatimusmäärittelyiden mukainen. (Kasurinen 2017, 40-41.)

2.1.5 Mustalaatikkotestaus

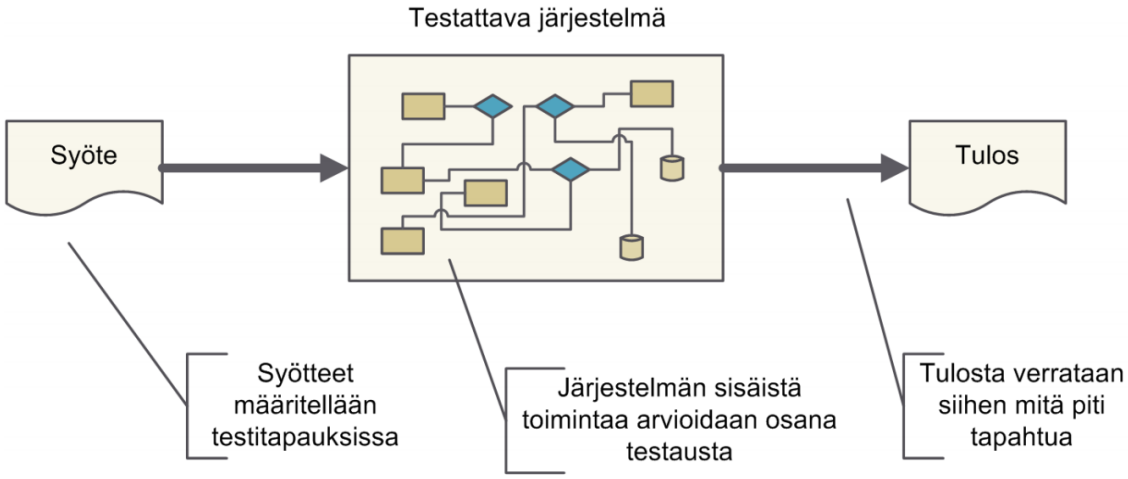
Mustalaatikkotestaus (black box testing)(kuvio 2) on perinteisin ja sen yksiselitteisyyden vuoksi usein automatisoiduin testimenetelmä. Se tarkoittaa järjestelmän testaamista siten, että testaaja ei tiedä mitä järjestelmän sisällä tapahtuu. Testaaja antaa järjestelmälle testisyötteitä, jotka on useimmiten määritelty testausdokumentaatiossa testitapauksen kohdalla. Järjestelmän antamaa tulostetta sitten verrataan siihen, mitä piti tapahtua ja tapahtuiko se oikein. Tästä johtuen mustalaatikkotestaus soveltuu hyvin automaattiseen testaamiseen. Mustalaatikkotestausta voidaan käyttää kaikissa testauksen työvaiheissa sen jälkeen, kun on olemassa käynnistyvä versio. (Kasurinen 2017, 41.)



KUVIO 2. Musta laatikko testaus (Kasurinen 2017, 41.)

2.1.6 Lasilaatikkotestaus

Lasilaatikkotestaus (white box testing, glass box testing, clear box testing)(kuvio 3) tarkoittaa järjestelmän testaamista siten, että testaaja tietää mitä järjestelmän sisällä tapahtuu ja pystyy tarkastelemaan järjestelmän sisäistä toimintaa testauksen aikana. Lasilaatikkotestaus on testaajan näkökulmasta huomattavasti vaativampi, koska testaajan täytyy tuntea järjestelmä kokonaan ja pystyä lukemaan järjestelmän lähdekoodia. Lasilaatikkotestaus on eräällä tavalla mustalaatikkotestauksen vastakohta. (Kasurinen 2017, 42.)



KUVIO 3. Lasilaatikko testaus (Kasurinen 2017, 42.)

2.1.7 Harmaalaatikkotestaus

Harmaalaatikkotestaus (grey box testing) tarkoittaa järjestelmän testaamista siten, että osa järjestelmän sisäisestä toiminnasta on tiedossa ja osa ei. Harmaalaatikkotestaus on yhdistelmä musta- ja lasilaatikkotestauksesta, jossa pyritään ottamaan kummankin testausmenetelmän parhaat ominaisuudet käyttöön. Harmaalaatikkotestausta käytetään, kun musta- tai lasilaatikkotestausta ei voida soveltaa. (Kasurinen 2017, 42-43.)

2.1.8 Regressiotestaus

Regressiotestaus (regression testing) on yleistermi ohjelmistojärjestelmän uudelleentestaamiselle. Kun järjestelmän osaa muutetaan tai on saavutettu osatavoite (milestone), järjestelmä testataan uudestaan. Näin varmistetaan, että se toimii oikein muutoksen jälkeen. Regressiotestaamista käytetään paljon automaattisessa testauksessa, josta myöhemmin lisää. (Kasurinen 2017, 43-44.)

2.2 Valmiin ohjelmiston testaus

Tiettyjä testauksia ei voi tehdä keskeneräisille tuotteille. Alla on käyty läpi testauksia, jotka vaativat lähes valmiin tai valmiin tuotteen, joka on olemassa ja toiminnassa, mutta sitä ei ole vielä julkaistu tai luovutettu asiakkaalle. (Kasurinen 2017, 45.)

2.2.1 Hyväksymistestaus

Hyväksymistestaus (acceptance testing) on ohjelmistotestauksen viimeinen vaihe. Sen tavoitteena on varmistaa, että ohjelmisto on riittävän korkealaatuinen ja täyttää sille vaatimusmäärittelydokumentaatiossa määritetyt vaatimukset. Yleensä tämä testaus suoritetaan kohdeympäristössä. Hyväksymistestaus on yleinen termi ohjelmistojärjestelmän tarkistukseen, jossa asiakas hyväksyy ohjelmiston valmiiksi. Tämä testausvaihe suoritetaan yleensä lakiteknillisistä syistä. (Kasurinen 2017, 46.)

2.2.2 Alfa- ja beta-testaus

Alfa-testaus on periaatteessa järjestelmätestauksen seuraava vaihe, jossa testataan, miten järjestelmä toimii kokonaisuutena. Alfa-testauksessa testataan, että järjestelmä on riittävän vakaa ja toimiva ennen beta-testausta. Alfa-testauksessa järjestelmän kaikki tärkeimmät ominaisuudet on toteutettu, pois lukien viimeistely ja mahdolliset ohjelmistovirheet. (Kasurinen 2017, 47.)

Beta-testaus on periaatteessa sama kuin hyväksymistestaus. Beta-testit suoritetaan käytännön olosuhteissa. Tässä testausvaiheessa voidaan vielä tehdä muutoksia järjestelmään. Beta-testauksessa oleva tuote (järjestelmä) ei ole vielä virallisesti julkaistu. (Kasurinen 2017, 47.)

2.2.3 Käytettävyystestaus

Käytettävyystestaus on nimensä mukaisesti järjestelmän käytettävyystestaamista, jossa painopiste on käyttöliittymän toimivuudessa ja intuitiivisuudessa. Käyttöliittymätestauksessa testataan, että käyttöliittymä toimii oikein ja suunnitellulla tavalla. Siinä myös tarkastetaan ja testataan, että käyttöliittymä on suunniteltu oikein. (Kasurinen 2017, 45.)

2.2.4 Kuormitustestaus

Kuormitustestaus (load testing) tarkoittaa ohjelmistojärjestelmän kykyä toimia tietyn käyttäjämäärän kanssa oikein. Sillä on kolme päätehtävää; löytää järjestelmän pullonkaulat, selvittää järjestelmän maksimikapasiteetti ja testata miten järjestelmä ja laitteisto selviytyy normaaleista olosuhteista. Kuormitus testauksen pidemmälle viety versio on rasitustestaus (stress testing), jossa testataan järjestelmän maksimaalinen toimintakyky, jolloin järjestelmä vielä toimii normaalisti. Kuormitus- ja rasitustestaus on käytännössä aina automaattista testaamista, joskin joskus avoimella alfa- tai beta-testauksella voidaan suorittaa kyseinen testaus. (Kasurinen 2017, 45-46.)

2.2.5 Suorituskykytestaus

Suorituskykytestaus (performance testing) tarkoittaa kevennettyä rasitustestausta. Testauksen tavoite on osoittaa, että ohjelmistojärjestelmä suoriutuu sille annetuista tehtävistä vaatimusmääritysten mukaisesti. (Kasurinen 2017, 46.)

2.2.6 Savutestaus

Savutestaus (smoke test) tarkoittaa ohjelmiston perusasioiden toimivuuden testausta. Savutestaus on leikkimielinen termi tällaiselle testaamiselle. Savutestillä varmistetaan ohjelmiston toimivuuden minimitaso. (Kasurinen 2017, 46.)

2.2.7 Tutkiva testaus

Tutkiva testaaminen (explorative testing) tarkoittaa testaamista, joka ei perustu tarkan mallin mukaiseen testaamiseen. Tutkiva testaaminen yrittää löytää mahdolliset vikatilat. Tässä testausmuodossa pyritään hyödyntämään testaajien tietotaitoa ja ymmärrystä, mistä viat tai virheet syntyvät ja löytyvät. Tämä testausmuoto on välillä hankala dokumentoida ja jäljitellä. (Kasurinen 2017, 47-48.)

Esimerkiksi: Ohjelmiston muistin tarve kasvaa koko toiminnan ajan vastoin oletuksia. Mikä ohjelmiston osa tai osat aiheuttavat tämän ongelman? Missä on ohjelmistovirhe tai virheet ja miten ongelman voi korjata? Tällaisten ongelmien selvittämiseen tutkiva testaaminen soveltuu hyvin.

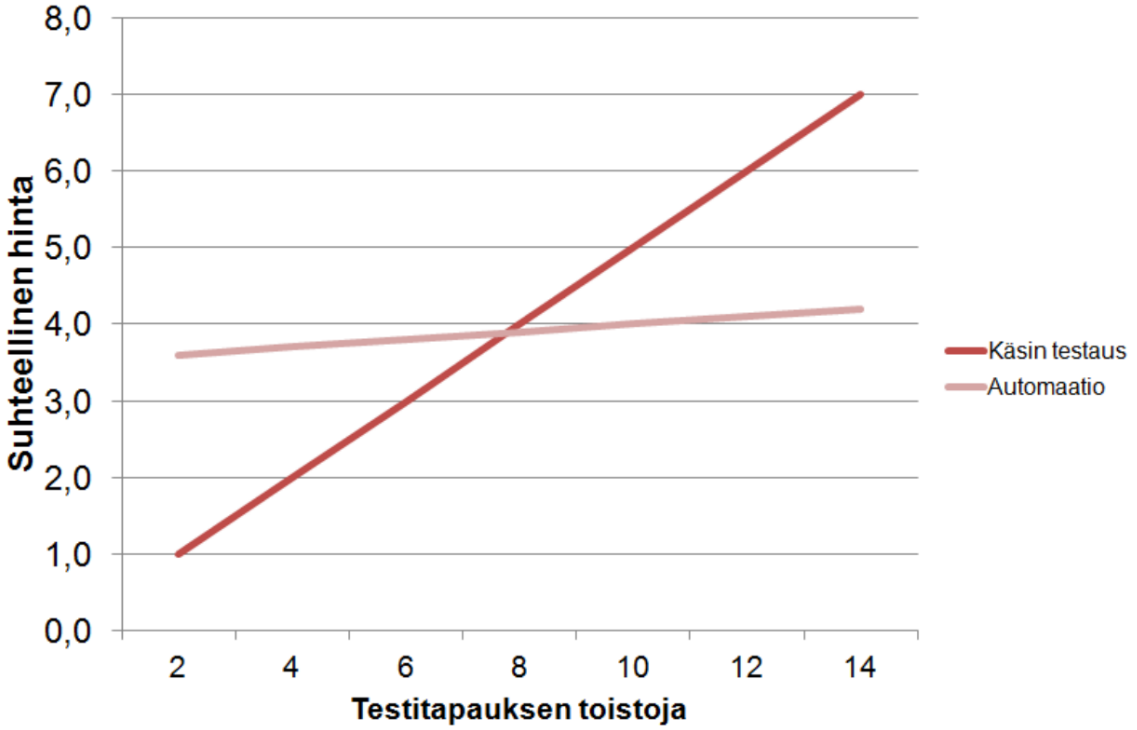
3 AUTOMAATTINEN TESTAUS

Automaattinen testaus tarkoittaa testausta, jonka tietokone suorittaa automaattisesti, kun uusi versio ohjelmistosta ilmestyy. Tietokone tallentaa testien lopuksi raportin, miten uusi versio toimii. Automaattisen testauksen yleisimpiä käyttökohteita on moduulien rajapinnat, verkkoprotokollan tarkistukset ja käyttöliittymätarkistukset. (Kasurinen 2017, 49.)

Automaattista testausta pidetään monesti manuaalisen testauksen korvaajana, mutta näin ei ole. Automaattinen testaus täydentää manuaalista testaamista. Tämän lisäksi automaatiotestaus vaatii jatkuvaa kehitystä, ylläpitoa ja resursseja. Automaattista testaamista käytetään yleensä regressiotestauksessa tai usein toistuvissa testeissä. Perimmäisenä ajatuksena automaatiotestauksessa on vapauttaa testaajat muihin tehtäviin. (Kasurinen 2017, 49.)

3.1 Käyttöönotto ja ongelmat

Vaikka regressiotestauksen automatisointi vähentääkin merkittävästi käsin tehtävän testauksen määrää, se ei sitä täysin poista. Kuvasta (kuvio 4) voidaan päätellä, että testauksen toistojen määrä on ratkaisevassa asemassa säästöjä haettaessa. (Kasurinen 2017, 50-51.)



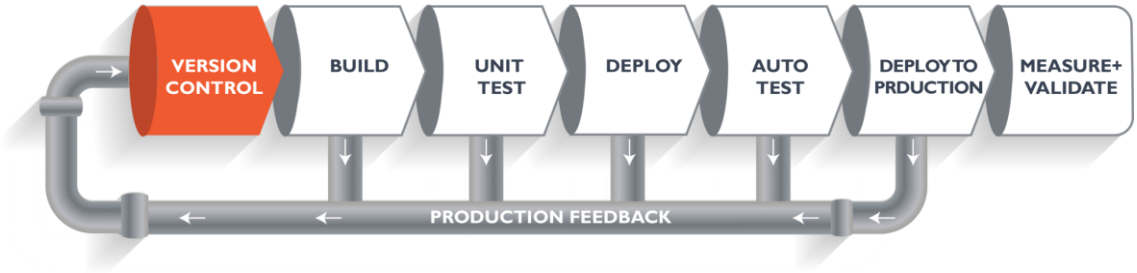
KUVIO 4. Automatisoitujen testitapausten kustannus vs. käsin testaukseen (Kasurinen 2013, 50.)

Automaattisen testauksen ongelmana on usein käyttötarkoituksen hämärtyminen, jolloin automaatiota yritetään soveltaa toimintoihin, joihin se ei sovellu. Haasteellista on myös kustannusten oikea budjetointi. Lisäksi oikeiden automaatiotyökalujen valinta, testausautomaation opettelu ja toteuttaminen jäävät usein liian vähälle huomiolle. (Kasurinen 2017, 50-51.)

3.2 Putki

Putki (pipeline) tarkoittaa automatisoitua prosessia, jossa suoritetaan jatkuva integrointi (CI) ja jatkuva käyttöönotto (CD). CI/CD on nykyaikaisen DevOps ympäristön selkäranka. Periaatteessa putken suorittamaa prosessia voidaan ajatella samanlaiseksi kuin ohjelmistokehityksen elinkaari. Kun (kuvio 5) versiohallintaan (versio control) saapuu uusi versio ohjelmistosta, putki aloittaa automaattisesti sen käsittelyn. Ensimmäisenä vaiheena on rakentaminen (build), jossa ohjelmisto käännetään ajettavaan muotoon. Tämän jälkeen suoritetaan yksikkötestit (unit test) ohjelmistoon. Kolmantena vaiheena on uuden version käyttöönotto (deploy)

testipalvelimelle. Seuraavaksi suoritetaan automaattiset testit (auto test) koko ohjelmistoon, johon kuuluu mm. integraatio-, käyttöliittymä- ja yhteensopivuustestit. Jos testit menevät läpi, suoritetaan putken viimeinen vaihe, eli käyttöönotto (deploy to production) tuotantopalvelimelle. Jokaisessa edellä mainitussa vaiheessa putken suoritus keskeytyy, jos jokin menee vikaan ja virheraportti lähetetään automaattisesti kehittäjille. Virheiden korjaamisen jälkeen putki ajetaan alusta asti uudelleen. Kun koodi on tuotannossa, tapahtuu lopullinen ohjelmiston mittaus ja validointi (measure and validate). (Tuli 2018.)

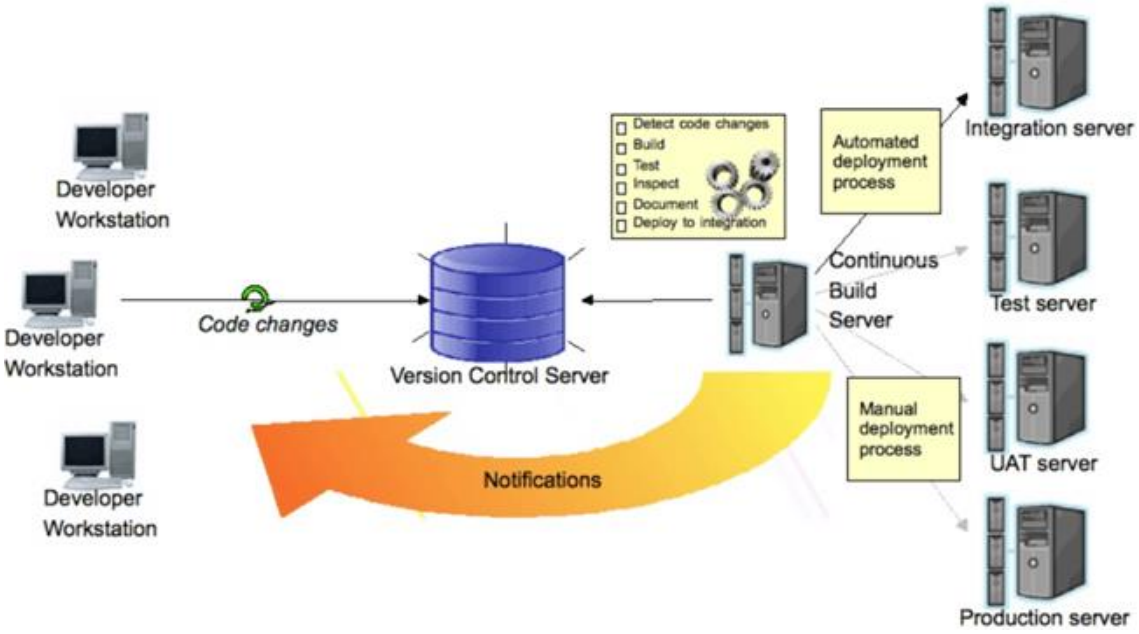


KUVIO 5. Yksinkertainen CI/CD putki (Tuli 2018.)

Putken CI/CD:n suorittamiseen on useita työkaluja esim. Jenkins ja Gitlab. Putken suoritukseen voidaan myös integroida Docker helpottamaan ja nopeuttamaan putken toimintaa. (Tuli 2018.)

3.3 Jatkuva integraatio

Jatkuva integrointi (continuous integration, CI) on ohjelmistonkehitysmenetelmä, jossa ohjelmoijat lisäävät tuotoksensa vähintään kerran päivässä koodivarastoon (repository tai version control server). Aina kun tuotos lisätään varastoon, ajetaan testit automaattisesti läpi koodivaraston, yhteensopivuuden ja koodivirheiden etsimiseksi. Joten jokaisen muutoksen jälkeen ohjelmisto rakennetaan ja testataan automaattisesti ohjelmistovirheiden löytämiseksi. Jatkuva integraatio suoritetaan yleensä automatisoidussa putkessa (kuvio 6), josta testeistä läpimennyt ohjelmisto lähetetään testiserverin käyttöön. Testin epäonnistuessa, putki ilmoittaa siitä ohjelmiston kehittäjille. (Guru99 n.d.c.)



KUVIO 6. Jatkuva integrointi (Guru99 n.d.c.)

Jatkuva integrointi otettiin käyttöön yli 20 vuotta sitten välttämään integraatiokaaosta. Se tapahtuu yleensä, jos projektin osien integrointi jätetään aivan projektin loppuun. (Guru99 n.d.c.)

Jatkuva integrointi vaatii omat ajopalvelimet ja suoritusympäristöt. Lisäksi se vaatii oman osaamisen ylläpitoa ja jatkuvaa kehitystä. Näin ollen toimiakseen kunnolla resurssointi pitää olla riittävä. Resursoinnista tinkiminen johtaa putken suorituksen odotusaikojen pitenemiseen. (Guru99 n.d.c.)

Jatkuvan integroinnin työkalut ja palvelut on käsitelty luvussa 5 syvällisemmin. Lähes kaikki automaattisen testauksen ohjelmistot käyttävät jonkinlaista putkea avukseen.

4 AUTOMAATTISEN TESTAUKSEN OHJELMISTOJA

Tässä luvussa käydään läpi muutamia yleisempiä ilmaisia tai lähes ilmaisia automaatiotestauksen ohjelmistoja. Niiden taustoja ja ominaisuuksia, sekä hyviä ja huonoja puolia tarkastellaan käyttäjän näkökulmasta.

Kaikissa automaatiotesti esimerkeissä käytetään lainausjärjestelmä Web-sivustoa, jossa on myös oma takapää API. Takapää API on tehty NodeJS:än päälle käyttäen GraphQL-kirjastoa. Etupää on tehty NextJS-kirjastoa käyttäen, joka perustuu React-kirjastoon. Etupään GUI-kirjastona on Material-UI. Etu- ja takapään käyttäjän tunnistautumisessa on käytössä JWT-tunnistautuminen.

4.1 Postman-testausohjelmisto

Postman on testaustyökalu, joka skaalautuu hyvin API:en eli integraatiotestauksiin. Se on alun perin ollut API:en testaustyökalu, joka on laajentunut ajan saatossa muihinkin osa-alueisiin. Postmanin loi ja perusti Abhinav Asthanan vuonna 2012. Se syntyi toisen projektin sivuprojektina testauksen ja kehittämisen yksinkertaistamiseen ja nopeuttamiseen. Työkalun kysyntä kasvoi julkaisun jälkeen hyvin nopeasti. Abhinav Asthana joutui palkkaamaan apuun kaksi entistä kollegaansa Ankit Sobti ja Abhijit Kane. Kolmikko perusti Postman-osakeyhtiön ja on sen johdossa edelleen. (Guru99 n.d.b.; Chatterjee 2018; Postman n.d.a.)

Postmanin vahvuuksia on saavutettavuus, avoin lähdekoodi, erilaisten kokoelmien käyttö ja jakaminen. Lisäksi ohjelmisto pitää sisällään erilaisten testiympäristöjen luonnin ja lukuisten välitarkistuspisteiden olemassaolon. Virheiden etsintä sisäänrakennetun konsolin avulla, jatkuva integraatio ja automaatiotestaus kuuluvat myös Postmanin vahvuuksiin. Saavutettavuudella tarkoitetaan, että Postmania voi käyttää kuka tahansa, missä ja milloin vain omalla koneella olevalla ohjelmistolla tai kirjautumalla omalle Postman-tillillensä internetissä. Postmanin testit, testikokoelmat ja ympäristöt tallentuvat pilveen. Postmanissä voi luoda useita erilaisia testikokoelmia ja testiympäristöjä. Testikokoelmissa voi olla useita eri alikansioita ja testejä. Tällä tavoin testien järjestäminen on helpompaa.

Testiympäristöt voivat jakaa testikokoelmansa keskenään ja näin voidaan vähentää turhaa testien toistamista. Tällä testien parametrisoinnilla saadaan tehtyä uudelleen ketterämpiä testejä. Postmanissä on testikokoelmien ja ympäristöjen tuominen ja vieminen sisäänrakennettu. Suorat linkit kokoelmiin ovat myös mahdollisia, jos ne on tallennettu pilveen. Postman-testit palauttavat tulokset JSON-muodossa, jota on helppo lukea ja käsitellä tarpeen vaatiessa. Postmanista löytyy myös graafinen jatkuva integraatiotestaus ja automaattinen testaus, mikä tunnetaan nimellä collection runner. Sen suoritusta voi seurata Postmanin ohjelman ikkunassa graafisesti. (Guru99 n.d.b.; Chatterjee 2018.)

Postmanin heikkouksia on joidenkin mielestä sisään rakennettu tuki putkimaiselle komentorivipohjaiselle CI/CD-suoritukselle. Kyseinen toiminto on aikaansaatu Newman lisäosalla. Newmanista kerrotaan lisää omassa alaluvussaan. Heikkouksiin voidaan lisätä myös maksullisuus pilvipalveluissa. Kolmantena heikkoutena on graafisten käyttöliittyminen testaaminen, johon Postman ei varsinaisesti sovellu. (Guru99 n.d.b.; Postman n.d.c.; Chatterjee 2018.)

4.1.1 Postman-työpöytäohjelmisto

Postman-työpöytäohjelmisto on saatavilla Windowsille, macOSlle ja Linuxille. Olemassa on myös kehitysversio edellä mainituille käyttöjärjestelmille, jos haluaa käyttöönsä viimeisimmät ominaisuudet. MacOSlle on saatavilla vain 64-bittinen versio. Windowsille ja Linuxille on myös saatavilla 32-bittinen versio 64-bittisen lisäksi. Työpöytäversion rajoitukset koskevat pilvipalveluita riippuen mikä versio pilvipalveluista on käytössä. (Postman n.d.c.; Postman n.d.b.)

4.1.2 Postman API ja Web-palvelut

Postman API:ssa on ilmainen perusversio. Siinä on suhteellisen rajattu käyttöoikeus ja käyttöaika kuukaudessa. Maksullisuuden takana on useita lisäominaisuuksia, esim. käyttäjien hallinta, lisäkirjautumisen mahdollisuudet, kiinteä IP-osoite ja auditointirekisterit (Audit logs). Maksullisia paketteja on Team ($12 kuukaudessa), Business ($24 kuukaudessa) ja Enterprise (yritys kohtainen hinta).

Hinnan noustessa lisääntyy käyttäjäien määrä, ominaisuudet ja palvelupyyntöjen maksimi. Näiden lisäksi on lisämaksua vastaan kertaluontoisia paketteja määräajaksi. Kertaluontoiset paketit vaativat minimissään Team-paketin. (Guru99 n.d.b.; Postman n.d.c.; Chatterjee 2018.)

Postman julkaisi vuonna 2020 kesällä Postman-Web avoimen beta-version, joka on postmanin työpöytäversio selaimelle. Selainversiota voi käyttää kaikilla nettiyhteyden omaavilla koneilla. Postman-Webin käyttö vaatii ilmaisen rekisteröitymisen. (Abhinav 2020.)

Postman-Web käyttää toimiakseen Postman-agenttia. Se on mikrosovellus, joka asennetaan selaimeen lisäosaksi. Se toimii paikallisesti koneella ja välittää API pyynnöt taustalla. Postman-agentilla voidaan kiertää selaimen asettamat toiminnalliset rajoitukset. Postman-Web käyttää Postman API:a, joten Postman API:n ilmaisversion rajaukset tulevat vastaan aika nopeasti. Postman blogin mukaan Postman-Web helpottaa työntekijöiden yhteistyötä yrityksissä ja testien jakamisessa muille. (Lane 2020; Abhinav 2020.)

4.1.3 Newman-lisäosa

Newman on Postmanin lisäosa, joka on tehty CI/CD-putken komentoriviversion ajoon. Newmanin tehtävä on ajaa valmiiksi tehty Postman-kokoelma komentorivillä ja luoda suorituksesta raportti komentoriville, pilvipalveluun tai/ja tiedostoon. Tällöin testit voidaan ajaa useimmilla CI/CD-palveluilla. Newman on ilmainen. Se on tehty NodeJS:n päälle JavaScriptillä kirjoitettuna. (Chatterjee 2018; GitHub postmanlabs/newman 2020.)

4.1.4 Automaatiotestiesimerkki Newmaniä käyttäen

Automaatiotestin luominen Postmanilla ja ajaminen CI-putkessa Newman-lisäosaa käyttäen. Postmanillä luodaan kokoelma, joka sisältää erilaisia testejä eri

API-osoiteisiin. Valmis kokoelma viedään JSON-muotoisena tallennettuun tiedostoon. Se sisältää kaiken tarvittavan, mitä komentorivillä testien ajamiseen tarvitaan. (Mangan 2020.)

Alla olevassa kuvassa (kuva 1) on lainausjärjestelmän tämän hetkellisen käyttäjän tietojen tarkistustestit. Ensimmäinen testi tarkistaa onnistuiko API-pyyntö ja toinen testi, kuinka kauan pyyntöön meni. Tämän jälkeen parsitaan runkokentän tiedot JSON-muotoon. Lopuksi tarkistetaan tietokenttien sisältötyyppi. Jos tarkastettavat kentät on tyhjiä kyseinen testi epäonnistuu.

```
1  // Tuliko onnistuneen pyynnön koodi (200) testi?
2  pm.test("Status code is 200", () => {
3      pm.response.to.have.status(200);
4  });
5
6  // Saantiinko palaute alle 200ms testi?
7  pm.test("Response time is less than 200ms", () => {
8      pm.expect(pm.response.responseTime).to.be.below(200);
9  });
10
11 // Parsitaan vastauksen body kentän sisältö
12 var data = JSON.parse(responseBody);
13
14 // Testataan tuliko data ja onko data oikeassa muodossa?
15 pm.test("Current user information, is ok?", () => {
16     pm.expect(data.data.currentUser.id).to.be.a("string");
17     pm.expect(data.data.currentUser.isActive).to.be.true;
18     pm.expect(data.data.currentUser.firstName).to.be.a("string");
19     pm.expect(data.data.currentUser.lastName).to.be.a("string");
20 })
```

KUVA 1. Tämän hetkellisen käyttäjän tietojen testauskoodi

JSON-testitiedoston ajaminen komentorivillä tai putkessa Newmaniä apuna käyttäen on helppoa valmiilla testitiedostolla. Suoritus tapahtuu yhdellä käskyllä, jossa Newmanille annetaan ajettavaksi JSON-testitiedosto. Alla olevassa kuvassa (kuva 2) on onnistunut ajo Newmanillä. Kuvassa näkyy suoritetut testit, onnistumiset, suoritus ajat ja tietojen siirtomäärä.

```
→ Login test
  POST http://localhost:3050/ [200 OK, 409B, 181ms]
  ✓  Status code is 200
  ✓  Response time is less than 200ms
  ✓  Succesfully login, have JWT and is it right format?

→ Current user
  POST http://localhost:3050/ [200 OK, 301B, 17ms]
  ✓  Status code is 200
  ✓  Response time is less than 200ms
  ✓  Current user information, is ok?

→ Allusers list
  POST http://localhost:3050/ [200 OK, 15.41KB, 31ms]
  ✓  Status code is 200
  ✓  Response time is less than 200ms
  ✓  All users information, is ok?
```

	executed	failed
iterations	1	0
requests	3	0
test-scripts	3	0
prerequest-scripts	0	0
assertions	9	0
total run duration: 369ms		
total data received: 15.55KB (approx)		
average response time: 76ms [min: 17ms, max: 181ms, s.d.: 74ms]		

KUVA 2. CI-putkessa Newmanillä ajetut testitulokset

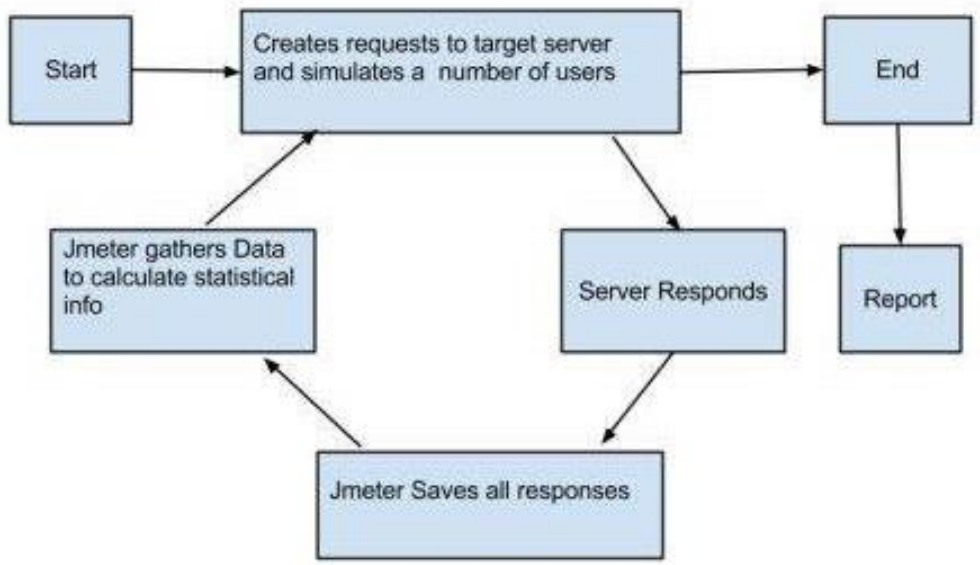
Usein CI-putkiajossa käytetään Newmanissä lisäreportteria, joka lähettää tulokset palvelimelle tai sivustolle. Raportoitavat tulokset muokataan ennen lähetystä esimerkiksi JSON- tai XML-muotoon. Muokkaus onnistuu Newmanin lisäparametrejä käyttäen. (GitHub postmanlabs/newman 2020.)

4.2 Apache JMeter-testausohjelmisto

Apache JMeter on avoimen lähdekoodin verkkosovellusten testausohjelmisto, joka on suunniteltu tekemään suorituskyky-, kuormitus- ja stressitestejä järjestelmille. Sillä voi suorittaa lukuisia muitakin testejä eri protokollille ja tekniikoille. Siinä on hyvä tuki regressiotestaukselle. JMeter on Javalla-koodattu Java-työpöytäsovellus, josta löytyy graafinen käyttöliittymä. Se toimii Windows-, Linux- ja macOS-käyttöjärjestelmillä. (Guru99 n.d.e.; Tutorialspoint n.d.a.; Performancelab n.d.)

Stefano Mazzocchi (Apache Software Foundation säätiö) loi JMeterin. Hän tarvitsi testausohjelmaa Apache JServin suorituskyvyn testaamiseen (Apache Jserv kutsutaan nykyään Apache Tomcat). Versio 1.0 julkaistiin joulukuussa vuonna 1998. Myöhemmin Apache kehitti JMeteriin paremman käyttöliittymän ja uusia ominaisuuksia. Viimeisin versio 5.3 on julkaistu toukokuussa vuonna 2020. (Tutorialspoint n.d.a.; Apache JMeter n.d.b.; Apache JMeter n.d.a.)

JMeterin toiminta perustuu käyttäjäryhmän simulaatioon. Alla oleva kuvio (kuvio 7) näyttää toimintakaavion simulaatiosta. Simuloitu käyttäjäryhmä lähettää pyyntöjä kohdepalvelimelle. Palvelin reagoi ja vastaa pyyntöihin, JMeter tallentaa kaikki vastaukset ja suorituksiin kuluneet ajat. Tämän jälkeen JMeter laskee ja luo staattisen informaation testien suoriutumisesta kohteeseen. Tätä sykliä voidaan jatkaa testaajan haluaman määräajan. Kun testaus lopetetaan, JMeter tulostaa raportin testien suoriutumisesta taulukoiden ja kuvioiden avulla. (Tutorialspoint n.d.a.)



KUVIO 7. JMeter toiminta (Tutorialspoint n.d.a.)

JMeterin hyviä puolia ovat; helppo asennus, kolmannen osapuolen laajennusten saatavuus, tuki useille työkaluille, iso tukiyhteisö, ilmaisuus ja avoin lähdekoodi. Sillä voi testata natiivisovelluksia ja Web-sivuja. Siinä on hyvä CI-tuki ja raportointimahdollisuus useassa yleisessä muodossa esim. JSON, XML, HTML. Hyvänä ominaisuutena on myös täysi monisäikeinen ajo ja rajaton kuormituksen tuotantokapasiteetti. (Performancelab n.d.; Tutorialspoint n.d.a.)

Puutteena JMeterissä mainittakoon tuen puute JavaScriptille ja AJAX-pyynnöille. Muistinkäyttö kasvaa helposti suureksi GUI-tilassa. Tietyn rajan jälkeen suuri muistin käyttö ja suuri käyttäjä määrä aiheuttaa virheitä. Myös vaikeudet testata monimutkaisia JavaScript- tai dynaamisia sovelluksia aiheuttavat ongelmia. Lisäksi maksullisissa kilpailevissa ohjelmissa on enemmän ominaisuuksia. (Performancelab n.d.)

JMeterissä on hyvä tuki komentoriville, joten CI-ajo onnistuu loistavasti. JMeter:in voi integroida useimpiin CI-palveluihin esim. Jenkinsiin. Siinä on useita kolmannen osapuolen laajennoksia CI-ajoa helpottamaan ja raportointiin. Eräs hyvä laajennus on JMeter Control Center (jltc), joka löytyy Githubista MIT-lisenssillä. Se on verkkosovellus ja hallintapaneeli CI-ajoon JMeterille. Siitä löytyy myös

Docker-versio. JMeter Control Center on ohjelmoitu Pythonilla. Se käyttää tietokantana PostgreSQL:lia. (Sudonull n.d.; Performancelab n.d.; GitHub innogames/JMeter-Control-Center 2019.)

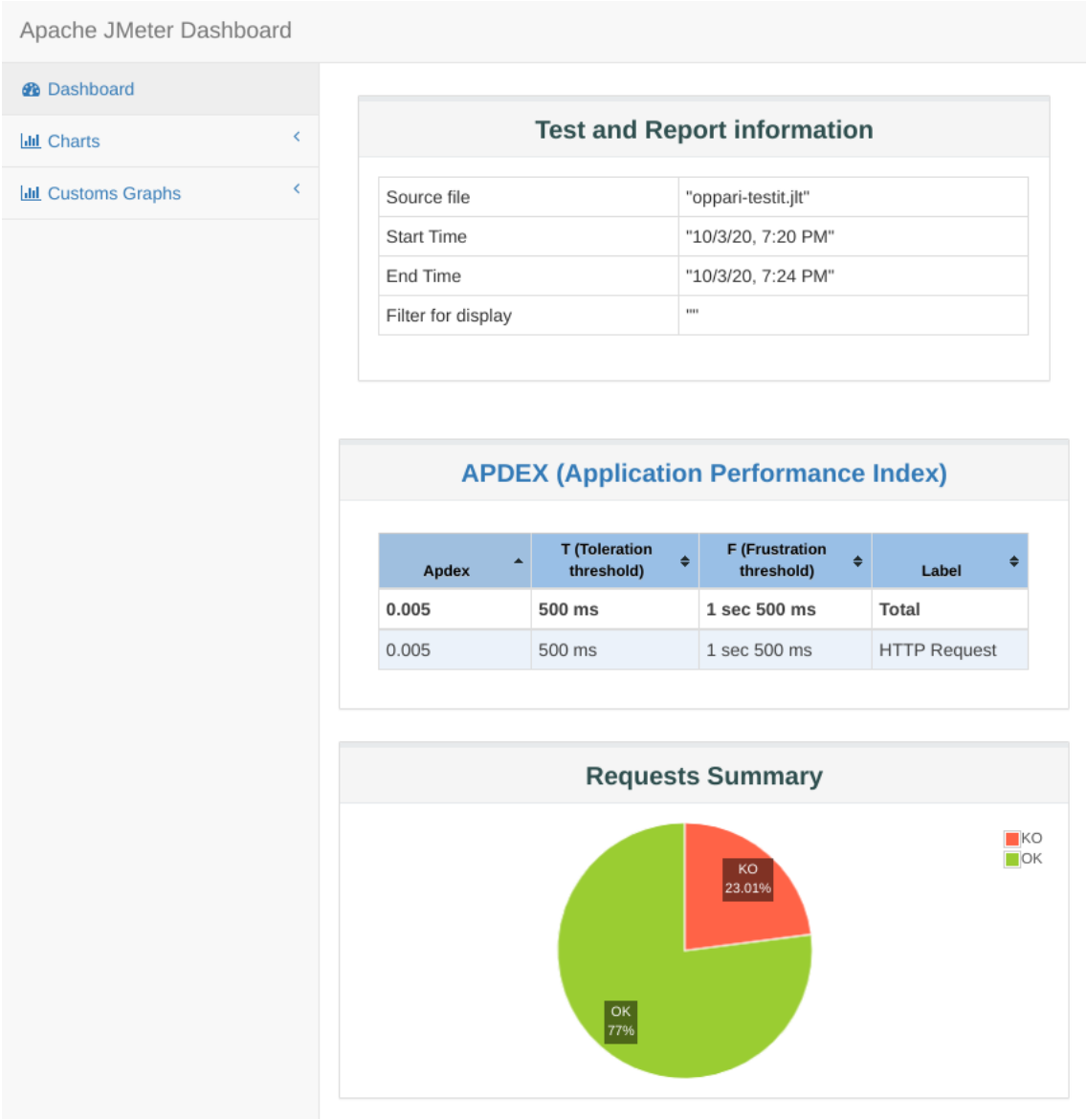
4.2.1 Automaatiotestiesimerkki JMeterillä

Automaatiotestin luominen Apache JMeterillä tapahtuu GUI:n kautta. Esimerkki testinä on suorituskyky testi etupäälle. Käyttäjiä simuloivia säikeitä (threads) testissä on 10000, ylös ajoaika säikeellä 10 sekuntia ja yksi ajo silmukka. Kun testi on valmis JMeter GUI:ssa, se tallennetaan jmx-päätteiseen tiedostoon. Tämän jälkeen valmis testitiedosto on helppo ajaa komentorivillä tai CI-putkessa. Alla olevassa kuvassa (kuva 3) on komentorivi suoritus testistä. (Guru99 n.d.a.)

```
summary +   1948 in 00:00:21 =   93.2/s Avg:  9027 Min:   285 Max: 17399 Err:    17 (0.87%) Active: 8053 Started: 10000 Finished: 1947
summary +   3500 in 00:00:30 =  116.7/s Avg: 29994 Min: 12576 Max: 48734 Err:    20 (0.57%) Active: 4553 Started: 10000 Finished: 5447
summary =   5448 in 00:00:51 =  107.0/s Avg: 22497 Min:   285 Max: 48734 Err:    37 (0.68%)
summary +   1898 in 00:00:30 =   63.3/s Avg: 60977 Min: 41308 Max: 71707 Err:     0 (0.00%) Active: 2655 Started: 10000 Finished: 7345
summary =   7346 in 00:01:21 =   90.8/s Avg: 32439 Min:   285 Max: 71707 Err:    37 (0.50%)
summary +   2654 in 00:00:57 =   46.9/s Avg: 118093 Min: 71162 Max: 131271 Err:  1957 (73.74%) Active: 0 Started: 10000 Finished: 10000
summary =  10000 in 00:02:18 =   72.7/s Avg: 55172 Min:   285 Max: 131271 Err:  1994 (19.94%)
Tidying up ...    @ Sat Oct 03 19:24:56 EEST 2020 (1601742296638)
... end of run
```

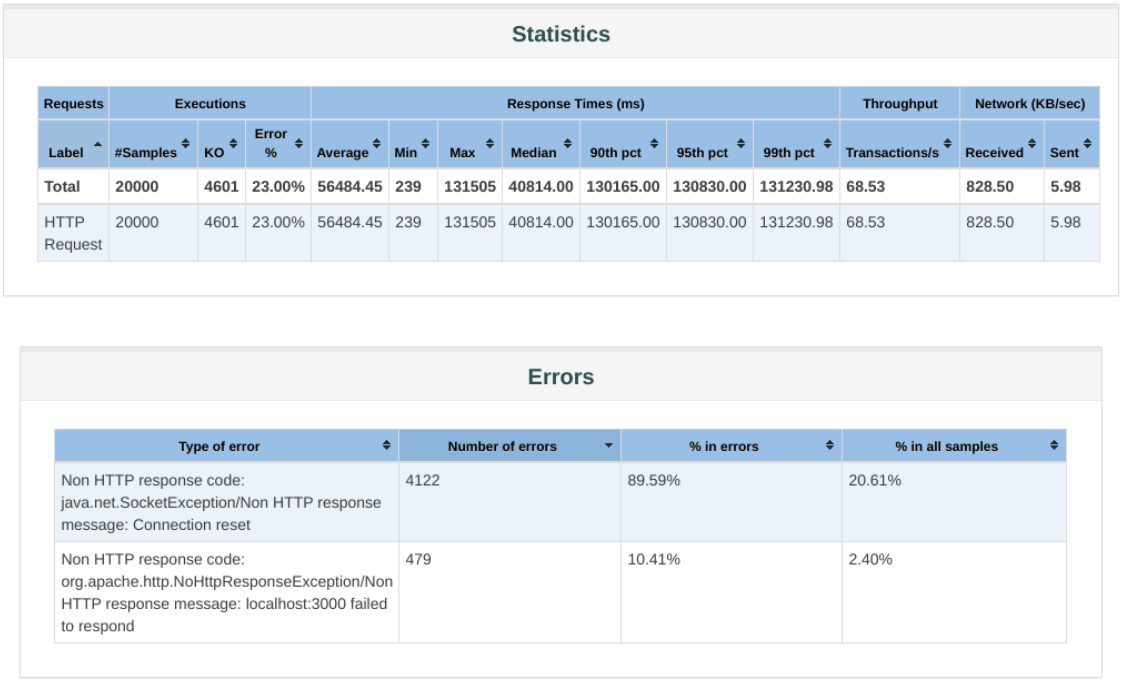
KUVA 3. Komentorivitestin suoritus JMeterillä

Testin suorituksen jälkeen JMeter tallentaa tulokset jlt-pääteiseen tiedostoon. JMeterillä voi luoda kyseisestä tiedostosta eri pääteisiä tiedostoja esimerkiksi XML- tai HTML-raportin. Alla olevassa kuvassa (kuva 4) on HTML-raportin etusivu. HTML-raportissa on tulokset taulukko- tai/ja kuvaajamuodossa, josta testaaja voi tehdä analyysin testien onnistumisesta. (Guru99 n.d.a.; Loisel 2018.)



KUVA 4. JMeterillä generoitu HTML-raportti testin tuloksista

Alla olevassa kuvassa (kuva 5) etusivun raportti jatkuu statistiikka- ja virheosiolla. Taulukossa suorituskyky (throughput) kertoo palvelimen pyyntöjen käsittelynopeuden. Se on tärkein parametri testeissä. Virheet (vika prosentti tai KO pyyntöjen määrä) kertoo, kuinka monta pyyntöä palvelimelta jäi käsittelemättä testin aikana. (Guru99 n.d.a.; Loisel 2018.)



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	20000	4601	23.00%	56484.45	239	131505	40814.00	130165.00	130830.00	131230.98	68.53	828.50	5.98
HTTP Request	20000	4601	23.00%	56484.45	239	131505	40814.00	130165.00	130830.00	131230.98	68.53	828.50	5.98

Errors

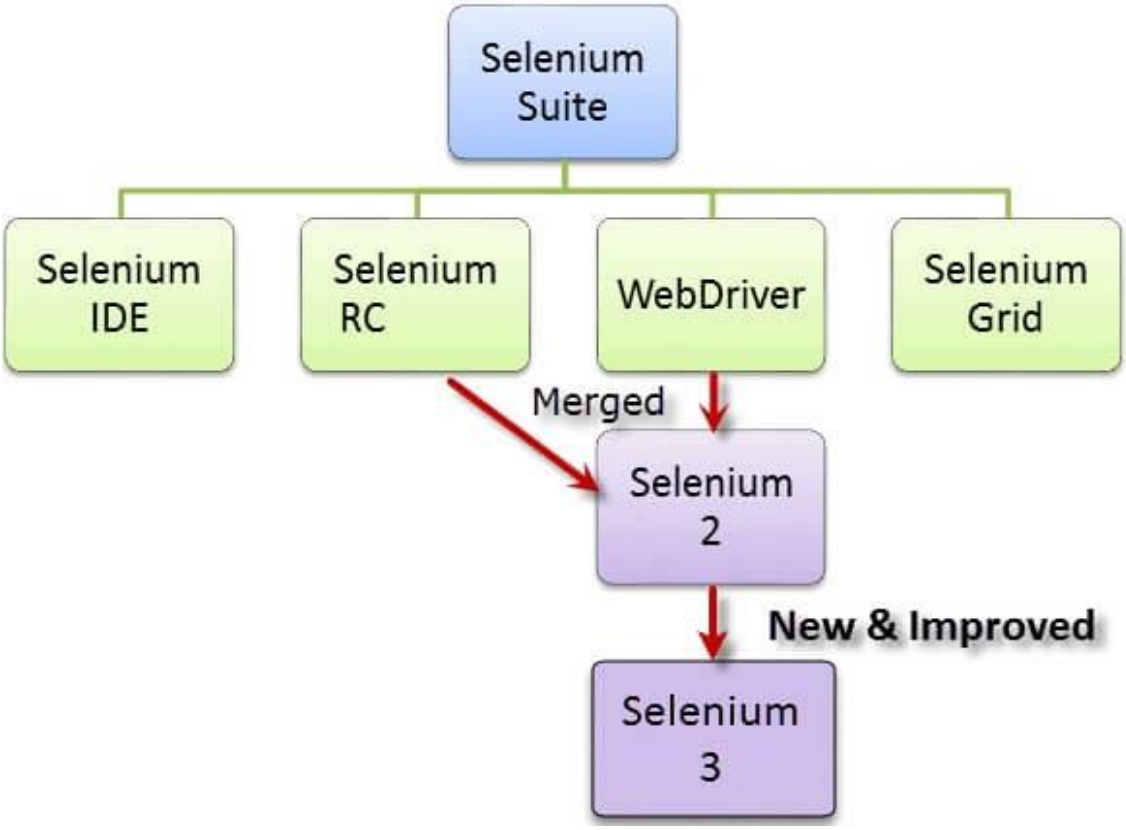
Type of error	Number of errors	% in errors	% in all samples
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset	4122	89.59%	20.61%
Non HTTP response code: org.apache.http.NoHttpResponseException/Non HTTP response message: localhost:3000 failed to respond	479	10.41%	2.40%

KUVA 5. JMeterillä generoitu HTML-raportti testin tuloksista, statistiikka ja löytyneet virheet

JMeterin testien teko ja analysointi vaatii testaajilta ja analyysin tekijöiltä perehtymistä. Tulosten käsittelyyn löytyy erilaisia kolmannen osapuolen maksullisia ja ilmaisia lisäosia. (Loisel 2018.)

4.3 Selenium-testausohjelmisto

Selenium on avoimen lähdekoodin ilmainen automatisoitu testausjärjestelmä. Selenium nimi tulee vitsistä. Selenium kehitystyön alussa sen pahin kilpaileva ohjelmisto oli Mercury Interactive. Seleeni on vastalääke elohopea (mercury) myrkytykselle. Seleniumia käytetään web-sovellusten testaukseen ja validointiin eri selaimilla ja alustoilla. Seleniumin ohjelmisto on paketti, mikä on suunniteltu vastaamaan eri testaustarpeita. Se sisältää alun perin 4 eri osaa (kuvio 8) Selenium IDE, Selenium RC, WebDriver ja Selenium Grid. Myöhemmin Selenium RC ja WebDriver yhdistyvät ja siitä syntyi Selenium 2. (Guru99 n.d.f.)



KUVIO 8. Selenium ohjelmisto paketti (Guru99 n.d.f.)

Seleniumin joka osalla on ollut oma luojansa. Seleniumin Coren kehitti Janson Huggins vuonna 2004. Se tarvitsee toimiakseen JavaScriptiä. Selenium Coressa testattavan kohteen ja testiohjelman piti olla samalla koneella eli samassa verkkotunnuksessa. Tämän ongelman korjaamiseksi Paul Hammant loi palvelimen, joka hämäsi selaimen uskomaan, että testattava kohde ja Selenium Core ovat samassa verkkotunnuksessa. Tätä järjestelmää kutsutaan nimellä Selenium Remote Control (Selenium RC) tai Selenium 1. Selenium RC oli ensimmäinen automatisoitu web-testaus työkalu. Testien luomiseen se tukee seuraavia ohjelmointikieliä; Java, C#, PHP, Python, Perl, Ruby. Selenium RC:ssä on monia hyviä ominaisuuksia. Se toimii useilla eri käyttöjärjestelmillä ja selaimilla. Lisäksi sillä voi luoda toistuvia ja ehdollisia testejä. Se sisältää kehittyneen ja valmiin API:n ja suorittaa testit nopeammin kuin Selenium IDE. Huonoja ominaisuuksiakin löytyy. Se vaatii asentamisen tietokoneelle ja Selenium RC palvelimen ajamista. Selenium RC:n käyttö vaatii ohjelmointiosaamista ja testien suoritus on hitaampaa kuin WebDriveriä käyttäen. Selenium RC:n API sisältää sekavia ja tarpeettomia

käskyjä. Selenium 1 ei tue mobiili käyttöjärjestelmiä kuten Android ja IOS. (Guru99 n.d.f.; Dua 2019a.)

Patrick Lightbody kehitti Selenium Grid:in testien suoritusaikojen minimoimisen. Se pystyy ottamaan kuvankaappauksia kaikista tärkeistä vaiheista testin aikana ja lähettämään seleenikomentoja eri tietokoneille ja selaimille samanaikaisesti, sekä suorittamaan rinnakkaisajoa. Selenium Gridiä käytetään yhdessä Selenium RC:n kanssa. (Guru99 n.d.f.)

Vuonna 2006 japanialainen Shinya Kasatani loi Selenium IDE:n, joka on Firefox-laajennus. Sillä voi automatisoida suoritettavat testit tallennus- ja toisto-ominaisuuden avulla. Selenium IDE on yksinkertainen kehitysympäristöjärjestelmä. Sen voi asentaa Firefox selaimeen laajennuksena, kuten muutkin Firefox laajennukset. Yksinkertaisuuden vuoksi sitä tulisi käyttää vain prototyyppi työkaluna. Vaativammat testit joudutaan luomaan muilla Seleniumin työkaluilla. Selenium IDE toimii vain Firefox-selaimen kanssa. Se ei vaadi ohjelmointiosaamista testien tekemiseen, kuten muut Selenium työkalut. (Guru99 n.d.f.)

WebDriver osan Simon Stewart loi vuonna 2006. Se on käyttöympäristöjen välinen testausjärjestelmä, joka pystyy hallitsemaan selainta käyttöjärjestelmän tasolla. WebDriver on monessa suhteessa parempi kuin Selenium IDE tai RC. WebDriver ei käytä automaatiossa JavaScriptiä ja on ohjelmoitu Javalla. Se vaatii Javan toimiakseen. WebDriverin tuetut kielet on samat kuin Selenium RC:ssä. WebDriver ei tarvitse toimiakseen Selenium RC palvelinta. Testien suoritus aika on lyhyempi kuin Selenium RC:ssä tai IDE:ssä, kuitenkaan se ei sisällä ajoaikaisia ilmoituksia suorituksesta ja testien tuloksista. Se ei myöskään tue heti uusia selaimia, kuten RC. (Guru99 n.d.f.)

Vuonna 2008 Selenium kehittäjät päättivät yhdistää WebDriverin ja Selenium RC:n yhdeksi kokonaisuudeksi. Se julkaistiin 2011 ja sai nimen Selenium 2 (Selenium-WebDriver). Tästä seurasi, että Selenium RC:tä ylläpidetään ilman kehitystyötä. Selenium 2:n mukana tulee tuki mobiilikäyttöjärjestelmille Android ja IOS, sekä päättömälle HTMLUnit selaimelle. Selenium 2 tukee aiemmin mainittuja ohjelmointikieliä ja lisäksi esim. .NET:iä. (Guru99 n.d.f.; Selvarajah 2019; Dua 2019a.)

Selenium 3 julkaistiin syksyllä 2016 sisältäen suuria muutoksia aiempaan. Selenium Core poistettiin kokonaan ja korvattiin takaosaisella WebDriverillä. Selenium RC API:n ylläpito lopetettiin. Selenium 3:sta tuli W3C-standardi. Jokaisen selaimen julkaisija luo oman WebDriverin, mikä antaa paremman tuen selaimille. Mobiilipuolen kehitys siirtyi erilaisille kolmannen osapuolen testausjärjestelmille, jotka perustuvat Seleniumiin. (Selvarajah 2019; Dua 2019b.)

Selenium 4 on alfa-vaiheessa, joten tarkka julkaisupäivä ei ole tiedossa. Muutosta versioon 3 nähden on uusi W3C-protokola kommunikointiin. Muista muutoksista mainittakoon tuen poisto Opera- ja PhantomJS-selaimelta, W3C WebDriver Spec yhteensopivuus, vakaampi Selenium Grid Dockerille, useat API päivitykset ja standardoidut päivitetyt dokumentaatiot. (Selvarajah 2019.)

4.3.1 Esimerkki automaatiotestistä Seleniumilla

Automaatiotestin luomiseen Seleniumilla on tarjolla monta vaihtoehtoa. Eräs helpoimmista on seuraavanlainen. Testaaja asentaa selaimeen asennetavan Selenium IDE:n. Sen avulla testaaja luo testit graafisesti nauhoitustoiminnolla, joka generoi testikoodin. Kun testit ovat valmiit, testaaja tallentaa ne JSON-tiedostoon. Tiedosto ajetaan komentorivillä tai CI-putkessa Selenium SIDE Runner nimisellä Selenium IDE:n lisäosalla. Sille annetaan tallennettu tiedosto parametrinä ja tarvittavina argumentteina. (Newell 2020; Selenium IDE 2019.)

Alla olevassa kuvassa (kuva 6) on onnistunut ajo lainausjärjestelmän etupäähän. Ensimmäisessä testissä tapahtuu lainausjärjestelmään sisään- ja uloskirjautuminen. Toisessa testissä tapahtuu sisäänkirjautuminen ja käyminen kaikilla järjestelmän alasivuilla. Kuvassa näkyy myös testeihin kulunut aika.

```
info:    Running oppari-testit.side
 PASS  ./oppari-testit.test.js (43.51s)
  oppari-testit
    ✓ Login_+_logout_testi  (27220ms)
    ✓ Nakymat_testi (15155ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        43.665s
Ran all test suites.
```

KUVA 6. Selenium SIDE Runnerin onnistunut ajo lainausjärjestelmän etupäähän

Selenium SIDE Runner tukee myös Selenium Gridiä, sillä voi ajaa testit useille eri selaimelle. Myös lisäparametreilla voi luoda tulokset tiedostoon eri muodoissa. (Selenium IDE 2019.)

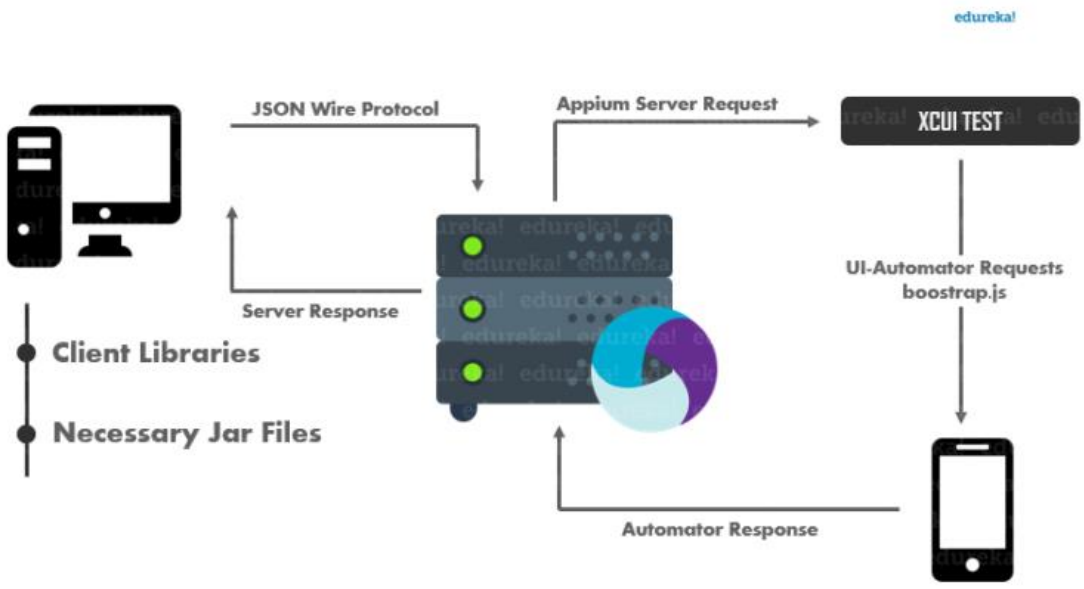
4.4 Appium-mobiilitestausohjelmisto

Appium on avoimen lähdekoodin automaatiotestauksen työkalu. Se on monialustainen ja sitä käytetään testien automatisointiin natiivi, hybridi- ja web-sovelluksissa. Appium on tehty Seleniumin päälle käyttäen Seleniumin kirjastoja ja osia. Appiumia kehittää ja ylläpitää JS-säätiö. Appium oli alun perin mobiili automaatiotestaus työkalu Android- ja IOS-sovelluksille. Sen viimeisimmissä päivityksissä laajentui testi mahdollisuus myös Windows ja macOS sovellutuksille. (Paul 2020b; Appium n.d.b.; Appium n.d.a.; Javatpoint n.d.)

Vuonna 2011 Dan Cuellar tarvitsisi automaatiotestisovelluksen IOS sovelluksille. Ongelma oli kuitenkin soveltumattomuus hänen tarpeelleen. Joten hän koodasi oman testiohjelman, joka pohjautui Applen toimittamaan UIAutomation-järjestelmään. Hän nimesi sen IOSAuto:ksi. Vuonna 2012 hän oli puhujana Selenium konferenssissa. Osana esitystään hän kertoi pintapuolisesti testiohjelmasta. Kovan kiinnostuksen vuoksi hän joutui vielä tarkentamaan testiohjelmansa sisältöä pikaesityksenä. Neljä kuukautta myöhemmin Janson (Selenium Coren luoja) otti yhteyttä Daniin, törmätessään IOS testiongelmiin ja muisti Danin esityksen asiasta. Danin koodi ei ollut julkista, joten Janson rohkaisi Dania julkaisemaan sen

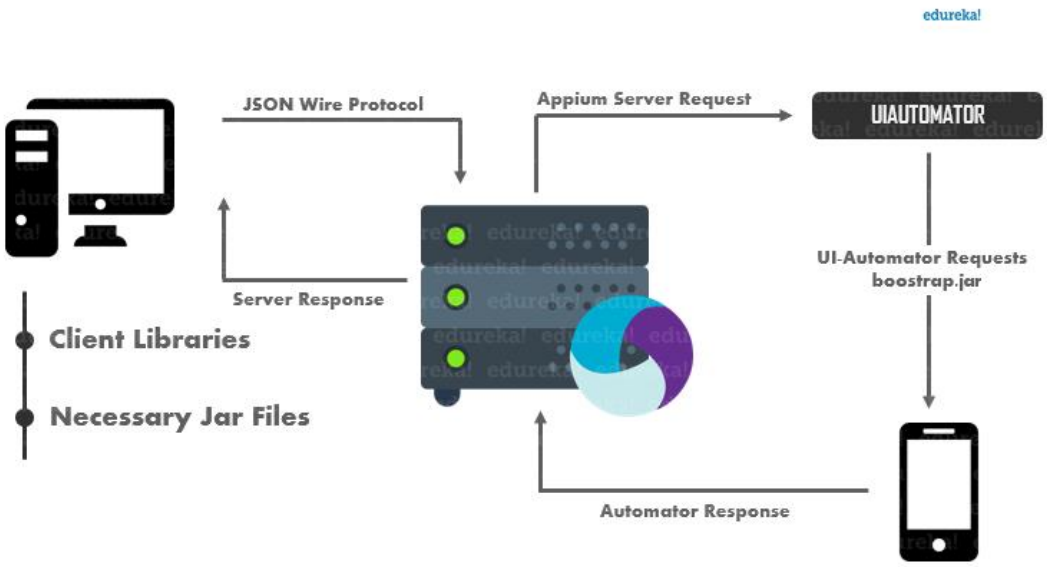
avoimena muiden käyttöön. Tämän jälkeen Dan julkaisi koodista eri koodikielisiä versioita. Janson lisäsi tuen Selenium WebDriverille. Vähän tämän jälkeen Dan ja Janson päättivät vaihtaa ohjelman nimen Appium:ksi (sovellusten selenium). Tammikuussa 2013 Sauce Labs otti Appiumin kehityksen hallintaansa, koodasi sen uudestaan NodeJS:sälle ja valitsi koodaus kieleksi JavaScript:in, koska se oli enemmän käytetty kieli kuin C# tai Python. Vuoden 2013 alussa Appium sai Android ja Selendroid tuen. Toukokuussa 2014 Appiumista julkaistiin 1.0 versio. Vuoden 2016 lopulla Sauce Labs lahjoitti Appiumin JS-säätiölle, joka jatkaa sen kehitystä edelleen. (Appium n.d.a.)

Appium IOS käyttää Applen XCUI Test API-sovellusta toimiakseen. XCUI Test on automaatiojärjestelmä, joka tulee Applen XCode-ohjelmiston mukana. Alla olevassa kuviossa (kuvio 9) on Appiumin IOS arkkitehtuurikaavio. Appium käyttäjä ottaa yhteyden Appium-palvelimelle ja kommunikoi sen kanssa JSON Wire-protokollan kautta. Appium-palvelin luo automaattisesti istunnon käyttäjälle ja tarkistaa käyttäjän haluamat ominaisuudet. Lisäksi se luo yhteyden tarvittaviin järjestelmiin esim. XCUI Test. XCUI Test kommunikoi bootstrap.js-kirjaston kanssa, joka on käynnissä joko oikeassa laiteessa tai emulaattorissa. Bootstrap.js-kirjasto suorittaa testit annetussa laiteessa ja raportoi onnistumisen takasin Appium-palvelimelle. (Paul 2020a.)



KUVIO 9. Appium IOS arkkitehtuurikaavio (Paul 2020a.)

Appium Android käyttää UIAutomator-järjestelmää toimiakseen. UIAutomator on Androidin rakentama ohjelma järjestelmäautomaatio tarkoituksiin. Alla olevassa kaaviossa (kuvio 10) on Appiumin Android arkkitehtuurikaavio. Appium-käyttäjä ottaa yhteyden Appium-palvelimelle ja kommunikoi sen kanssa JSON Wire-protokolla yhteyden kautta. Appium-palvelin luo automaattisesti istunnon käyttäjälle ja tarkistaa käyttäjän haluamat ominaisuudet. Lisäksi se luo yhteyden tarvittaviin järjestelmiin esim. UIAutomator. UIAutomator kommunikoi bootstrap.jar-kirjaston kanssa, joka on käynnissä joko oikeassa laiteessa tai emulaattorissa. Bootstrap.jar-kirjastolla on TCP-palvelimen rooli. Bootstrap.jar-kirjasto suorittaa testit annetussa laiteessa ja raportoi onnistumisen takasin Appium palvelimelle. (Paul 2020a.)



KUVIO 10. Appium Android arkkitehtuuri kaavio (Paul 2020a.)

Appiumin valmiit testit voidaan ajaa komentorivillä, mikä mahdollistaa testien suorittamisen eri CI/CD-palveluissa esim. Jenkins. CI/CD-putkessa testit ajetaan mobiilisimulaattorissa ja putki raportoi tulokset eteenpäin. (Singh 2018.)

4.5 Robot Framework-testausohjelmisto

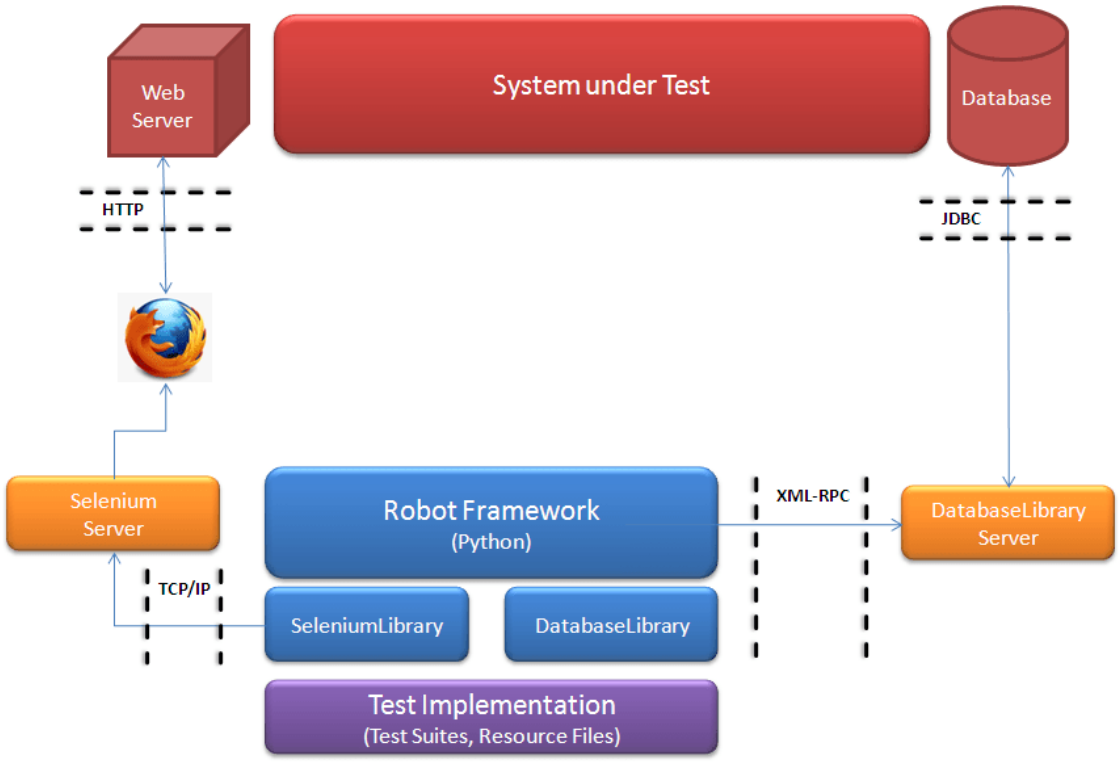
Robot Framework on Suomesta lähtöisin oleva avoimen lähdekoodin testiautomaatiojärjestelmä. Se on kehitetty hyväksymistestaukseen ja hyväksymistestilähtöiseen kehitykseen, sekä robottiprosessien automatisointiin (RPA). Koodaus on tehty Pythonilla. Robot Framework on julkaistu Apache lisenssi 2.0:lla. (Tutorialspoint n.d.b.; Robotframework n.d.a.)

Vuonna 2006 Pekka Klärck valmistui Helsingin Teknillisestä Yliopistosta. Hän teki diplomityön laajoista ohjelmistokehityksistä. Tämän työn pohjalta hän aloitti kehitystyön, jonka seurauksena valmistui Robot Framework automaatiotestausohjelmisto. Ohjelmiston kehitys aloitettiin alun perin Nokia Networks-yhtiössä. Robot Framework julkaistiin vapaana lähdekoodina kaikkien saataville vuonna 2008. Siitä lähtien Pekka Klärck on ollut kyseisen ohjelmiston pääkehittäjänä. Robot Frameworkia ylläpitää Robot Framework säätiö. Versio 2.0 julkaistiin 26 kesäkuuta 2008 ja 3.0 versio 31 joulukuuta 2015. (Klärck n.d.a.; Klärck n.d.c.; Klärck n.d.b.; Robotframework n.d.a.; Software Testing Help 2020; PyPI n.d.)

Robot Frameworkin etuja ovat avoin lähdekoodi ja mahdollisuus laajennukseen kolmannen osapuolen ulkoisilla kirjastoilla ja toiminnoilla. Se toimii avainsanojen avulla eikä vaadi monimutkaista koodia toimiakseen. Avainsanat ovat lyhyitä ymmärrettäviä sanoja. Käyttäjät voivat myös luoda omia avainsanoja. Robot Framework tukee tietojoukkojen testausta, mikä mahdollistaa datapohjaisen testaamisen. Haittana on tuen puute sisäkkäisille silmukoille, joten monimutkaisten testien teko on vaikeaa. (Software Testing Help 2020.)

Robot Frameworkissä on lukuisia vakiokirjastoja ja paljon ulkoisia kirjastoja. Seuraavaksi käydään läpi muutamia tärkeimpiä ulkoisia kirjastoja. SeleniumLibrary-kirjasto antaa tuen web-sivujen testaamiselle. AppiumLibrary-kirjasto antaa tuen Android- ja IOS-käyttöjärjestelmien ohjelmien testaamiselle. DataDriver Library-kirjasto antaa tuen tietokantojen käyttöön. RPA Framework-kirjasto sisältää kokoelman avoimen lähdekoodin kirjastoja ja työkaluja. (Robotframework n.d.b.)

Robot Framework testausarkkitehtuuri on alla olevassa kuviossa (kuvio 11). Selenium kirjasto vaatii aina käynnissä olevan Selenium-palvelimen. Palvelin ja tietokantapalvelin voivat olla käynnissä samalla koneella kuin Robot tai eri koneella. (Jaspers 2012.)



KUVIO 11. Robot Frameworkin testausarkkitehtuurikaavio. SeleniumLibrary-kirjastolla testataan Web-sivut ja DatabaseLibrary-kirjastolla tietokannat. SeleniumLibrary-kirjasto luo Selenium-palvelimen, jonka kautta Web-testi pyynnöt kulkevat selaimelle, jossa testit suoritetaan. DatabaseLibrary-kirjasto luo DatabaseLibrary-palvelimen, jossa tietokanta testit suoritetaan. DatabaseLibrary-palvelin tekee tarvittavat tietojen kyselyt tietokantoihin. Kaaviossa on myös merkitty kommunikointi tekniikka osien välillä. (Jaspers 2012.)

Robot Framework järjestelmän voi ajaa komentorivillä, joten se toimii CI-putkessa. Sen raporteissa on listattu onnistumiset, epäonnistumiset, testiin mennyt aika ja testien vaiheet. (Jaspers 2012; Tutorialspoint n.d.b.)

4.5.1 Robot Framework esimerkkitesti

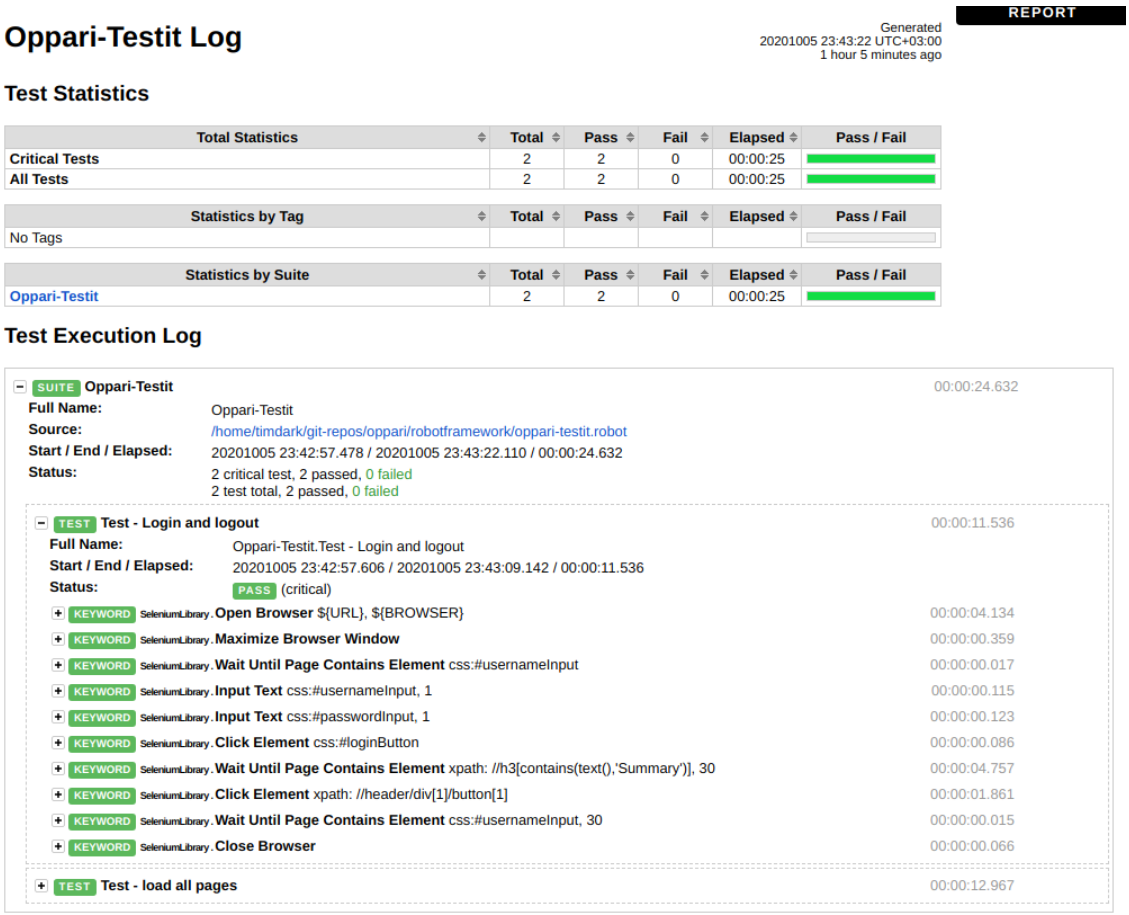
Automaatiotestien luominen Robot Frameworkilla on yksinkertainen prosessi. Käyttäjä luo robot-pääteisen tiedoston, johon kirjoitetaan testit Python-ohjelmointikielellä ja Robot Framework syntaksia käyttäen. Valmis tiedosto ajetaan komentorivillä tai CI-putkessa. (Waseem 2019.)

Alla olevassa kuvassa (kuva 7) näkyy onnistunut testiajo lainausjärjestelmä sivuston etupäähän. Testauksessa on käytetty Web-testaukseen SeleniumLibrary-kirjastoa.

```
==============================================================================
Oppari-Testit
==============================================================================
Test - Login and logout                                               | PASS |
------------------------------------------------------------------------------
Test - load all pages                                                 | PASS |
------------------------------------------------------------------------------
Oppari-Testit                                                         | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
==============================================================================
```

KUVA 7. Onnistunut komentorivi tai CI-putki ajo

Onnistunut ajo luo automaattisesti suorituskuvat, raportin, ja loki tiedostot, jotka on HTML-muodossa. Alla olevassa kuvassa (kuva 8) on testien loki tiedosto, josta näkyy testien onnistumiset ja mitä testeissä tehtiin. Kuvassa on ensimmäinen testin sisältö näkyvissä.



REPORT

Oppari-Testit Log

Generated
20201005 23:43:22 UTC+03:00
1 hour 5 minutes ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	2	2	0	00:00:25	
All Tests	2	2	0	00:00:25	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Oppari-Testit	2	2	0	00:00:25	

Test Execution Log

SUITE **Oppari-Testit** 00:00:24.632

Full Name: Oppari-Testit
Source: /home/timdark/git-repos/oppari/robotframework/oppari-testit.robot
Start / End / Elapsed: 20201005 23:42:57.478 / 20201005 23:43:22.110 / 00:00:24.632
Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed

TEST **Test - Login and logout** 00:00:11.536

Full Name: Oppari-Testit.Test - Login and logout
Start / End / Elapsed: 20201005 23:42:57.606 / 20201005 23:43:09.142 / 00:00:11.536
Status: PASS (critical)

KEYWORD SeleniumLibrary . **Open Browser** ${URL}, ${BROWSER}	00:00:04.134
KEYWORD SeleniumLibrary . **Maximize Browser Window**	00:00:00.359
KEYWORD SeleniumLibrary . **Wait Until Page Contains Element** css:#usernameInput	00:00:00.017
KEYWORD SeleniumLibrary . **Input Text** css:#usernameInput, 1	00:00:00.115
KEYWORD SeleniumLibrary . **Input Text** css:#passwordInput, 1	00:00:00.123
KEYWORD SeleniumLibrary . **Click Element** css:#loginButton	00:00:00.086
KEYWORD SeleniumLibrary . **Wait Until Page Contains Element** xpath: //h3[contains(text(),'Summary')], 30	00:00:04.757
KEYWORD SeleniumLibrary . **Click Element** xpath: //header/div[1]/button[1]	00:00:01.861
KEYWORD SeleniumLibrary . **Wait Until Page Contains Element** css:#usernameInput, 30	00:00:00.015
KEYWORD SeleniumLibrary . **Close Browser**	00:00:00.066

TEST **Test - load all pages** 00:00:12.967

KUVA 8. Robot Frameworkin automaattisesti luoma loki tiedosto

Ohjelmointi osaamista ei tarvita paljoa, kun kaikki testikomennot on selkokielisiä sanoja. Alla olevassa kuvassa (kuva 9) on aiemman suorituksen ensimmäisen testin koodi.

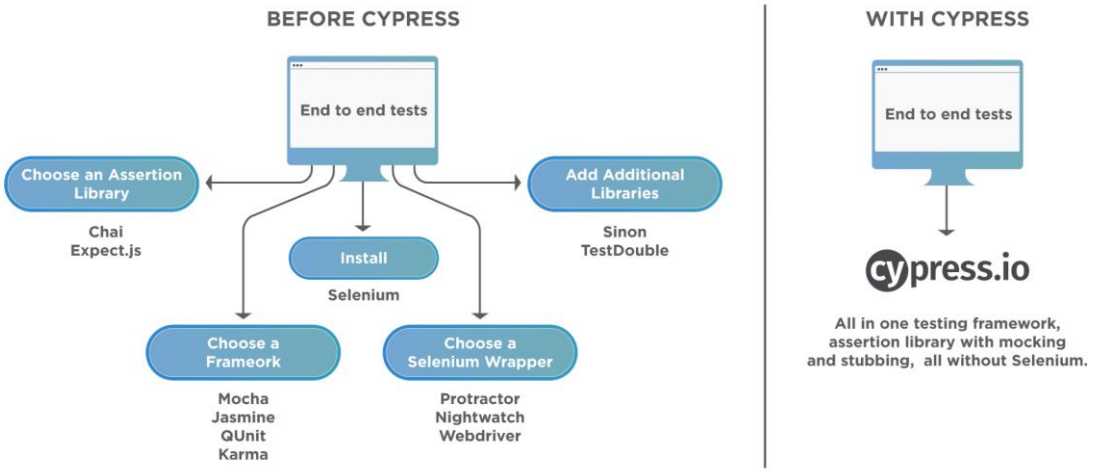
```
 9  Test - Login and logout
10      Open Browser                        http://localhost:3000                   chrome
11      Maximize Browser Window
12      Wait Until Page Contains Element    css:#usernameInput
13      Input Text                          css:#usernameInput                      1
14      Input Text                          css:#passwordInput                      1
15      Click Element                       css:#loginButton
16      Wait Until Page Contains Element    xpath: //h3[contains(text(),'Summary')]  30
17      Click Element                       xpath: //header/div[1]/button[1]
18      Wait Until Page Contains Element    css:#usernameInput                      30
19      Close Browser
```

KUVA 9. Ensimmäisen testin testikoodi. Testiohjelma aukaisee Chrome-selaimen ja menee paikalliselle Web-sivustolle, jossa on sisäänkirjautumisen ikkuna. Testiohjelma syöttää käyttäjä nimeksi ja salasanaksi numero yksi ja painaa sisäänkirjautumisen painiketta. Kun sisäänkirjautuminen on onnistunut ja etusivu ladattu, testiohjelma painaa uloskirjautumisen painiketta ja odottaa, että on uloskirjautunut Web-sivustolta.

Robot Frameworkilla on helppo tehdä testejä Web-sivustoihin. Se tukee suoraan hyvin Xpathiä elementtien polkujen osoittamiseen. Kuvassa (kuva 9) on osoitettu kaksi elementin polkua Xpathillä.

4.6 Cypress-testausohjelmisto

Cypress on avoimen lähdekoodin automaatiotestaustyökalu. Se on tehty uusimpien web-sivustojen testaukseen, erityisesti web-integraatio ja päästä päähän (end to end) UI-testaukseen. Cypress on julkaistu MIT-lisenssillä, kirjoitettu JavaScriptillä ja toimii NodeJS päällä. Testit sille kirjoitetaan JavaScriptillä. Cypress ei käytä Seleniumia. Cypressillä tehty automaatiotestausjärjestelmä (kuvio 12) on erilainen ja yksinkertaisempi kuin Seleniumilla. (Toolsqa n.d.; Khetarpal 2020b; Bharadwaj 2018.)

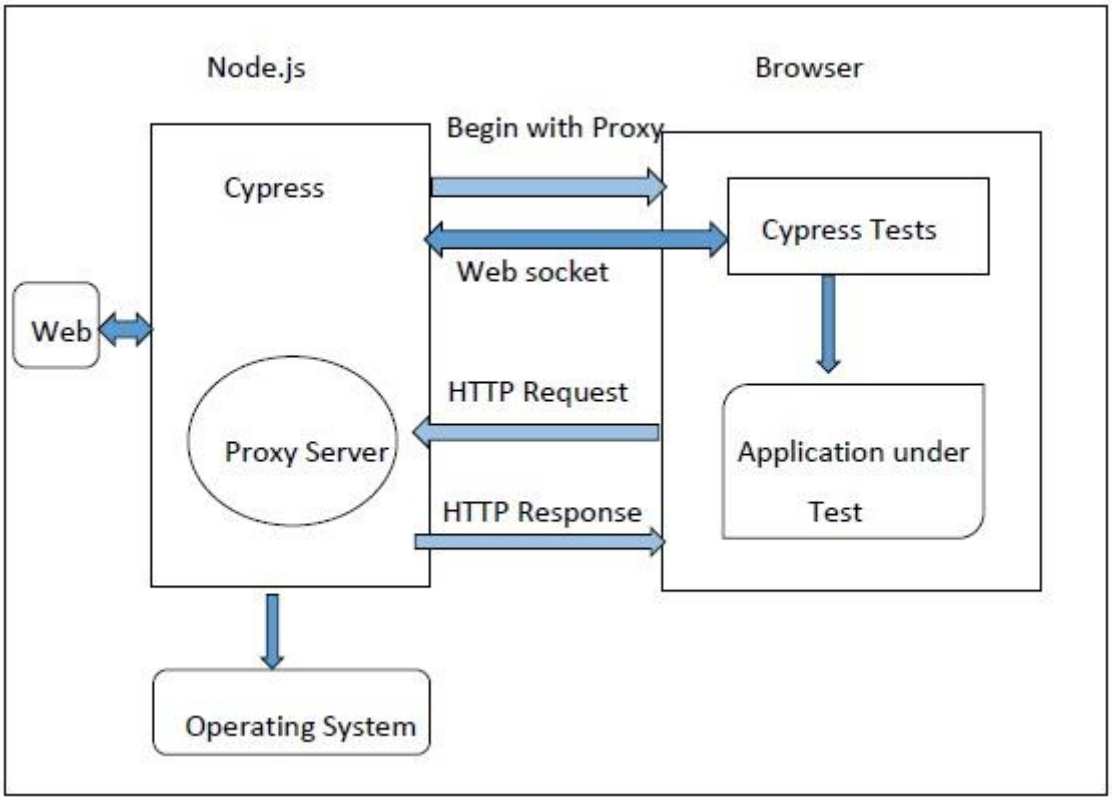


KUVIO 12. Testausautomaatio vertaus Seleniumilla ja Cypressillä (Khetarpal 2020b.)

Cypress sai alkuunsa kesäkuussa 2014. Kehitys tapahtui suljetussa beta-versiossa ja julkinen beta 1.0 versio julkaistiin lokakuussa 2017. Vuonna 2018 syyskuussa Cypressitä julkaistiin ensimmäinen vakaa versio. Versio 5.0 julkaistiin elokuussa 2020 ja kehitys jatkuu edelleen. (Cypress n.d.a.; Mann 2017; Mann 2018.)

Alla olevasta arkkitehtuuri kuviosta (kuvio 13) selviää Cypressin toimintaperiaate. Se on selaimen sisällä ja taustalla toimii NodeJS-palvelin. NodeJS-palvelin ja

Cypress ovat jatkuvassa vuorovaikutuksessa toistensa kanssa. Ne säätävät ja suorittavat toimintoja toistensa tukemiseksi. Tämän kautta Cypressillä on pääsy web sovelluksen etu- ja takapäähän. (Bhattacharjee 2020.)



KUVIO 13. Cypress arkkitehtuuri. Cypress toimii ja ajaa testit selaimen sisällä. Sen taustalla toimii NodeJS-palvelin. Cypress selaimessa ja NodeJS-palvelin keskustelevat koko ajan toistensa kanssa. Tämän avulla Cypressillä on pääsy sovelluksen etu- ja takapäähän, sekä verkkokerroksen liikenteeseen. (Bhattacharjee 2020.)

Cypressin avulla on helppo testata moderneja Web-sovelluksia, jotka perustuvat JavaScriptiin. Sen etuna on myös riippumattomuus Seleniumista. Se toimii selaimen sisällä, joten testit ajetaan myös selaimen sisällä. Tällä tavoin lisätään hallintaa, luotettavuutta ja nopeutta testeissä. Cypressiä voi käyttää kehittäjät ja testaajat. Cypressin koodinmuokkaus on helppoa ja sitä voi ajaa CI-putkessa. (Bhattacharjee 2020.)

Cypressin haittoina ovat toimivuus vain JavaScriptillä, sekä tuen puute välilehdille tai ponnahdusikkunoille. Cypressillä voi ajaa vain yhtä selainta kerralla. Se ei sovellu mobiilisovellusten testaamiseen. Sen jokaisessa testissä pitää olla vain yksi

alkuperäinen internetosoite. Cypressin haitoiksi voidaan myös lisätä Cypress Dashboardin maksullisuus, ilmainen versio on hyvin rajattu. (Bhattacharjee 2020; Khetarpal 2020b; Docs Cypress n.d.; Cypress n.d.b.)

Alla olevassa taulukossa (taulukko 1) on Cypressiä ja Seleniumia verrattu toisiinsa. Siinä näkyy erot näiden automaatiotestausohjelmistojen välillä. Esiin tulee hyvin myös aiemmin mainitut Cypressin hyödyt ja haitat. Selenium on parempi etä- ja rinnakkaissuorituksessa, selaimen välilehtien vaihdossa ja laajempana yhteisönä. Toisaalta Cypress on parempi JavaScriptin suorituksessa, testien videotallennuksessa ja suoritusten nopeudessa. (Khetarpal 2020b.)

TAULUKKO 1. Cypressin ja Seleniumin vertailu - hyödyt ja haitat (Khetarpal 2020b.)

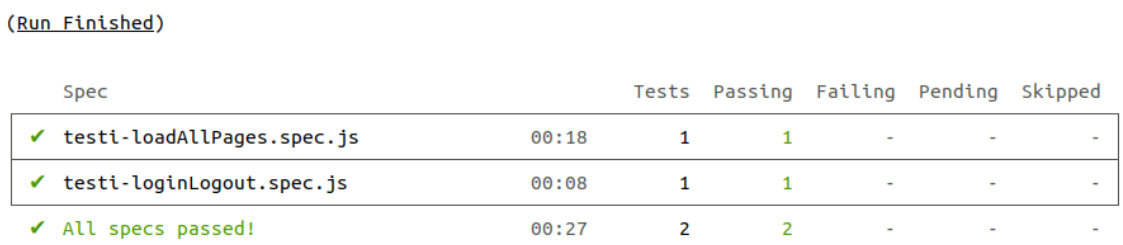
	Selenium	Cypress
Speed	😐	🙂
Wait for Element	😐	🙂 🙂
Remote Execution	🙂 🙂 🙂	☹
Parallel Execution	🙂	☹
Headless	🙂	🙂
Screenshot	😐	😐
Video	☹	🙂
Documentation	😐	🙂
Community	🙂 🙂 🙂	😐
Execute JS	🙂	🙂 🙂 🙂
Switch Tabs	🙂	☹
Several Browsers	🙂	🙂
Load Extensions	🙂	🙂
Manage Cookies	🙂	🙂

Cypress Dashboard-palvelu on valinnainen Web-pohjainen palvelu. Se on erittäin hyödyllinen CI-ajoissa, koska se tallentaa pilveen tulokset, kuvankaappaukset ja videot ajetuista testeistä. Se sisältää myös projektien ja käyttäjien hallinnan. Yhdellä luodulla Cypress Dashboard-tilillä voi hallita kaikkia projekteja. Ilmaisversio on sitä vastoin hyvin rajattu käyttäjien ja maksimi testitallennusten suhteen. (Khetarpal 2020a; Cypress n.d.b.)

4.6.1 Testiesimerkki Cypressillä

Automaatiotestien luominen Cypressillä on suoraviivainen toimenpide. Käyttäjä asentaa Cypressin ja ohjelmoi testitiedostot JavaScriptillä. Testit voi testata Cypressin GUI:lla graafisesti. Lopuksi testit voidaan ajaa komentorivillä tai CI-putkessa testit.

Alla olevassa kuvassa (kuva 10) on onnistunut komentorivi testiajo lainausjärjestelmän etupäähän. Kuvassa näkyy myös testien suoritusaika.



KUVA 10. Ajettujen testien tulos ja suoritusaika

Jokaisesta komentorivillä tai CI-putkessa ajetusta testistä tulee pienimuotoinen yhteenveto komentoriville (kuva 11). Myös nauhoitetun testivideon tiedostopolku. Testivideolla näkyy vaihevaiheelta koko testin kulku. Cypress luo videon automaattisesti.

KUVA 11. Ensimmäisen testin tulos ja testivideon tallennuspaikka

Alla olevassa kuvassa (kuva 12) on ensimmäisen testin JavaScript-koodi. Kuvassa (kuva 11) se on suoritettuna onnistuneesti lainausjärjestelmän etupäähän.

```
1   context("Testi login and logout", () => {
2     beforeEach(() => {
3       cy.visit("/");
4     });
5
6     it("Testi login and logout", () => {
7       cy.get("#usernameInput").type("1").should("have.value", "1");
8
9       cy.get("#passwordInput").type("1").should("have.value", "1");
10
11      cy.get("#loginButton").click();
12
13      cy.xpath("//h3[contains(text(),'Summary')]");
14
15      cy.wait(2000);
16
17      cy.xpath("//header/div[1]/button[1]").click();
18    });
19  });
```

KUVA 12. Ensimmäisen testin testikoodi. Testiohjelma aukaisee Chrome-selaimen ja menee paikalliselle Web-sivustolle, jossa on sisäänkirjautumisen ikkuna. Testiohjelma syöttää käyttäjä nimeksi ja salasanaksi numero yksi ja painaa sisäänkirjautumisen painiketta. Kun sisäänkirjautuminen on onnistunut ja etusivu ladattu, testiohjelma painaa uloskirjautumisen painiketta ja odottaa, että on uloskirjautunut Web-sivustolta.

Cypressin hyvä dokumentaatio auttaa testien teossa paljon. Xpath on saatavilla kolmannen osapuolen lisäosana Cypressiin. Kuvassa (kuva 12) on osoitettuna kaksi elementti polkua Xpathia apuna käyttäen.

5 AUTOMAATTISEN TESTAUKSEN TYÖKALUJA

Automaattisessa testauksessa käytetään monenlaisia aputyökaluja. Tässä luvussa tarkastellaan muutamia useimmin käytettyjä palveluita. Kaikista alla mainituista löytyy ilmainen versio ja joistakin maksullinen laajennettu versio.

5.1 Git-työkalu

Linus Torvalds ja Linux kehitysyhteisö kehitti oman versiohallintatyökalun vuonna 2005 nimeltään Git. Sen tavoitteita ovat nopeus, yksinkertainen suunnittelu ja vahva tuki epälineaariselle kehitykselle. Lisäksi Git on täysin tuettu, pystyy käsittelemään suuria projekteja ja on vapaata lähdekoodia. (Chacon & Straub 2020, 13.)

Git on versionhallintajärjestelmä, jonka tarkoituksena on auttaa kehittäjiä työskentelemään yhdessä suurten ohjelmistoprojektien kanssa. Git hallitsee tiedostojen kehitystä tiedostoversioarkistoiden avulla tarkasti ja hallitusti. (Bryan & Hester n.d.)

Git on periaatteessa komentorivityökalu, johon löytyy kolmannen osapuolen graafisia ulkoasuja. Sitä käyttävät useat palvelut internetissä, esim. GitHub, GitLab ja BitBucket. Joistakin näistä versiohallinta- ja arkistointipalveluista lisää seuraavissa luvuissa. (Chacon & Straub 2020, 17, 142; Bryan & Hester n.d.)

5.2 GitHub-palvelu

GitHub on pilvessä oleva versionhallinta ja arkiston isännöintipalvelu. Se on avointa lähdekoodia. GitHub hallinnoi kaikilla ohjelmointi kielellä kirjoitettuja lähdekoodiprojekteja. Se seuraa jokaisen projektin muutoksia ja pitää kirjaa niistä. GitHub käyttää Git-version hallintatyökalua. GitHub syntyi vuonna 2008 toisen projektin sivutuotteena. Seuraavana vuonna GitHubin kysyntä lähti kasvamaan

vauhdilla. Vuonna 2018 Microsoft osti GitHubin. Nykyään sillä on yli 40 miljoonaa kehittäjää. (Bradford 2019; Lardinois 2020; Shah n.d.)

GitHubissa käyttäjille perusominaisuudet ovat ilmaisia. Se tarkoittaa rajattomia yksityisiä ja julkisia projekteja, joissa voi olla rajaton määrä kehittäjiä. Se sisältää 500MB:in pakettivaraston projekteja varten. Perusominaisuuksiin kuuluu myös ilmainen pääsy GitHub Actions palveluun, joka on GitHubin CI/CD- ja automaatiopalvelu. Palveluun saa ilmaista putken suoritusaikaa 2000 minuuttia kuukaudessa. Putken suoritusaikaa kuukaudessa voi ostaa lisää. Perusominaisuuksien ohella GitHubista löytyy maksullisia lisäpalveluita kuukausihintaan yksityisille ja yrityksille. Perusominaisuudet ovat myös kaupallisille yrityksille ilmaisia. (Lardinois 2020.)

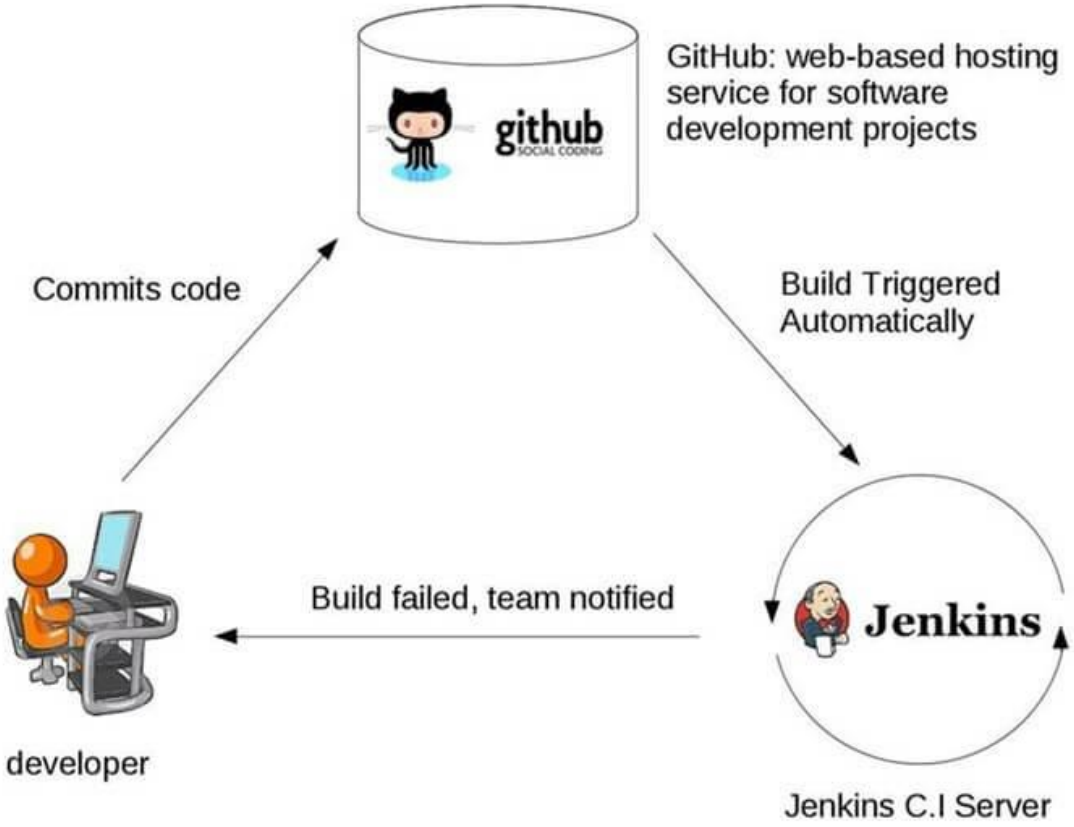
5.3 Jenkins-ohjelmisto

Jenkins on avoimen lähdekoodin jatkuvan integroinnin työkalu ja palvelin. Se on kirjoitettu Java-ohjelmointikielellä. Jenkins oli aluksi nimeltään Hudson, jonka perusti vuonna 2004 SUN Microsystemissä toiminut Java kehittäjä Kohsuke Kawaguchi. Vuonna 2011 SUN microsystem omistavalla Oraclella ja Hudsonin avoimen lähdekoodin yhteisöllä oli riita Hudsonista, niinpä avoimen lähdekoodin yhteisö teki oman haaransa Hudsonista ja nimesi sen Jenkinsiksi. Tämän jälkeen Hunson ja Jenkins jatkoivat toimintaa itsenäisesti omina palveluinaan. Ajan myötä Jenkinsistä tuli suosittu. Hudsonin ylläpito on lopetettu. Jenkinsin suosio perustuu siihen, että kehittäjät ovat luoneet sen kehittäjille. (Guru99 n.d.d.)

Jenkinsiä kehittää nykyään hyvin avoin yhteisö toimijoita. Siitä julkaistaan joka kolmen kuukauden välein uusi vakaa versio. Jenkins tukee laajennusten kautta satoja uusia ominaisuuksia. Jenkins tukee myös pilvipohjaisia arkkitehtuureja. (Guru99 n.d.d.)

Usein Jenkins integroidaan toimimaan automaattisesti jonkin versiohallintapalvelimen kanssa, esim. GitHub. Alla olevassa kuvassa (kuvio 14) on Jenkins integraatio GitHubin kanssa ja ohjelmiston kehittäjän rooli systeemissä. Ohjelmiston kehittäjä lähettää uuden version ohjelmasta GitHubiin. Jenkins tunnistaa, että

uusi versio on saapunut ja alkaa ajamaan sille määriteltyjä tehtäviä esim. rakentaminen, testaaminen, jne. Jos kaikki menee läpi onnistuneesti, uusi versio ohjelmasta lähetetään testi palvelimelle. Jos jokin menee pieleen, siitä informoidaan ohjelmiston kehittäjiä. (Guru99 n.d.d.)



KUVIO 14. Ohjelmoija, GitHub ja Jenkins synergia (Guru99 n.d.d.)

Jenkinsin ongelmana on vanhentunut ja vaikeakäyttöinen käyttöliittymä. Lisäksi käyttö vaatii oman palvelimen. Jenkinsin ylläpito ja käyttö on työlästä, se vaatii perehtymistä, esim. palvelimen ylläpitoon, valvontaan, ohjelmointiin ja asetusten määrityksiin usealla eri tavalla. Jatkuva integrointi rikkoutuu helposti jo pientenkin asetusmuutosten vuoksi. (Guru99 n.d.d.)

5.4 GitLab-ohjelmisto

GitLab on valmis kehitysalusta ja sen ydin on avointa lähdekoodia. GitLab on yhdistelmä GitHubia ja Jenkinsiä, johon on lisätty useita projektien hallintaan liittyviä työkaluja. GitLab vaatii oman palvelimen, kuten Jenkins. GitLabin koodi varasto ja versiohallinta on sen omalla ajopalvelimella, eikä pilven reunalla, kuten GitHubissa. Tämä on joillekin yrityksille hyvin tärkeä asia. (GitLab n.d.e.)

GitLabiä käytetään maailmalla yli 100000 organisaatiossa. GitLab perustettiin 2011 Ukrainassa, perustajina olivat (nykyinen) GitLabin CTO Dmitriy ’DZ’ Zaporozhets ja työntekijä Valery Sizov. Ohjelmointi kielenä on Ruby. Vuonna 2012 Sid Sijbrandij liittyi kehitystiimiin ja toimii nykyään GitLabin CEO:na. Sidin ansiosta GitLab tuli maailman tietoisuuteen ja vuonna 2013 julkaistiin GitLab EE:sta (Enterprice Edition) versio, jossa on suurten yritysten vaatimat ominaisuudet. GitLab EE on GitLabin maksullinen versio. Vuonna 2014 GitLabistä tuli virallisesti osakeyhtiö. Yritys ja sen alusta jatkoivat kasvuaan. Tänä päivänä se on maailman suurin etäyritys, jonka arvo on 2.75 miljardia dollaria. (GitLab n.d.a.; GitLab n.d.c.; GitLab n.d.d.; GitLab n.d.b)

6 POHDINTA

Opinnäytetyön tarkoituksena oli luoda katsaus ohjelmistojen automaattiseen testaukseen. Mukana oli sekä tietokone- että mobiilisovellutuksien ilmaiset tai lähes ilmaiset testausohjelmat. Saadaksemme oikein toimiva ja vaatimusmäärittelyn mukainen ohjelmisto on testaaminen ensiarvoisen tärkeää koko ohjelmistokehityksen ajan. Se on aikaa vievin ja kallein työ. Automaattisella testaamisella voidaan vähentää merkittävästi työntekijöiden regressiotestien työtaakkaa. Automaattinen testaus ei kuitenkaan korvaa manuaalista testausta vaan täydentää sitä. Se vaatii jatkuvaa ylläpitoa, kehitystä ja resursseja.

Automaattiseen testaukseen soveltuu parhaiten regressiotestaus ja usein toistuvat testaukset. Staattinen-, dynaaminen-, yksikkö-, järjestelmä-, integraatio-, alpha-, beta-, käytettävyys-, kuormitus-, rasitus-, suorituskyky-, savu- ja mustalaatikkotestaukseen voidaan käyttää automaattista regressiotestausta. Sitä vastoin siihen eivät sovellu lasilaatikko-, harmaalaatikko- ja hyväksymistestaus, eikä tutkivatestaus.

Postman on jokaisen ohjelmoijan yleistyökalu, mutta harva tietää, että se soveltuu myös automaatio testaukseen. Se on yksi parhaista integraatio ja API:en testaustyökaluista.

Apache JMeter on vanha, mutta toimiva rasitus-, kuormitus- ja suorituskykytestausohjelmisto. Kilpailijoita on lukuisia, useat ovat kuitenkin maksullisia tavalla tai toisella. Toimivuutensa ja ikänsä ansiosta Jmeteriä käytetään usein vanhoissa ohjelmistotaloissa uusien tulokkaiden sijasta.

Työssä käsitellään automaattiseen Web-testaukseen Selenium-, Robot Framework- ja Cypress-testausjärjestelmiä. Selenium on vanhin ja eniten maailmalla käytetty. Cypress on uusin tulokas. Seleniumilla voi testata lähes kaikkia ohjelmistojärjestelmiä. Cypressin etuna on JavaScript ohjelmointikieli ja haittana skaalautumattomuus muuhun kuin Web-testaukseen. Taulukossa (taulukko 1) tuo hyvin esiin Seleniumin ja Cypressin erot Web-testauksessa. Jos ohjelmointi osaaminen on rajallista, on hyvä valinta Robot Framework.

Moderneissa Web-sivustojen testauksissa Xpathiä joudutaan käyttämään usein, koska ne on optimoituja ja tietokoneen kääntämiä ohjelmointikoodista. Tästä johtuen osien ja elementtien nimet ovat hyvin monimutkaisia ja usein niiden id-nimet puuttuvat kokonaan.

Mobiili puolen automaatiotestauksesta käsitellään Appium ja Robot Framework testiohjelmia. Appium on puhdas mobiilitestausjärjestelmä, kun taas Robot Framework vaatii lisäosia. Mobiiliautomaatiotestaus on helpompaa Robot Frameworkilla etenkin, jos ohjelmointiosaaminen on puutteellista.

Automaatiotestauksessa käytetään usein eri työkaluja, jotka tavalla tai toisella liittyvät putkeen ja CI:hin. Jenkins on vanha, hyvin toimiva ilmainen putki. Github sai hiljattain oman vastineensa putkelle kilpailemaan Jenkinsin kanssa. Gitlabissa on myös oma putkensa. Maksullisissa koodivarastoissa on yleensä omat putkitoteutuksensa. Koodivarastot toimivat usein Gittiä käyttäen.

Automaattisten ohjelmistotestausmenetelmien runsaslukuisuus ja ominaisuuksien kirjavuus vaatii käyttäjiltä erityistä paneutumista asiaan. Sitä ei helpota jatkuva testiohjelmien uusiutuminen. Niinpä ohjelmistojen kehittäjien koulutuksessa tulisi kiinnittää entistä enemmän huomiota automaattisen testauksen käyttömahdollisuuksien tarkasteluun.

Opinnäytetyön edetessä tuli selväksi kuinka laaja-alaisia ja keskenään sidonnaisia eri testiohjelmat ovat. Periaatteessa jo yhden automaattisen testausohjelman perusteellinen läpikäyminen olisi yhden opinnäytetyön aihe. Näin ollen jokaisen yksittäisen ohjelman tarkka läpikäyminen oli mahdottomuus tämän työn puitteissa.

LÄHTEET

Abhinav, A. 2020. Announcing Postman for the Web, Now in Open Beta. Julkaistu 16.6.2020. Luettu 30.7.2020. https://blog.postman.com/announcing-postman-for-the-web-now-in-open-beta/?utm_source=marketo&utm_medium=email&utm_campaign=july-newsletter&mkt_tok=eyJpIjoiTkdJeE4yTmlPV05tTm1VdyIsInQiOiJcL25tdktYckpCTmhtK1B5ZUFhWUFrTUZPa1JTQWEydVJUc0pPVHAzWnhvXC9WWUE3WENHRkpuaTJaVXR6TGN0OFwvOTR6RVN2WURKZngzV21tajdYRGhUazg5d2RHenNJclwvNFVHRFJkN3JjXC9LXC95RzNwWTZTdU0wVXRJdjdsbHdWUSJ9

Apache Jmeter. n.d.a. Binaries. Luettu 14.9.2020. https://archive.apache.org/dist/jmeter/binaries/

Apache Jmeter. n.d.b. History of Previous Changes. Luettu 14.9.2020. https://jmeter.apache.org/changes_history.html

Appium. n.d.a. The History of Appium. Luettu 11.8.2020. http://appium.io/history.html?lang=en

Appium. n.d.b. The Mac Driver For OS X. Luettu 11.8.2020. http://appium.io/docs/en/drivers/mac/

Bharadwaj, S. 2018. Why you should switch to Cypress for modern web testing?. Julkaistu 10.6.2018. Luettu 29.8.2020. https://medium.com/@shivambharadwaj/https-medium-com-shivambharadwaj-why-you-should-switch-to-cypress-for-modern-web-testing-5d3739a19e6

Bhattacharjee, D. 2020. Cypress Architecture (Test Automation). Julkaistu 5.8.2020. Luettu 30.8.2020. https://www.tutorialspoint.com/cypress-architecture-test-automation

Bradford, L. 2019. What Is GitHub, and Why Should I Use It?. Päivitetty 30.5.2019. Luettu 21.7.2020. https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946

Bryan, J. & Hester, J. n.d. Happy Git and GitHub for the 52ser – Chapter 1 Why Git? Why GitHub?. Luettu 9.7.2020. https://happygitwithr.com/big-picture.html

Chacon, S. & Straub, B. 2020. Pro Git. E-book. Apress.

Chatterjee, U. 2018. Postman Tutorial for Automation : All you need to know. Julkaistu 25.6.2018. Luettu 14.7.2020. https://www.cuelogic.com/blog/postman-tutorial-for-automation

Cypress. n.d.a. Changelog. Luettu 4.9.2020. https://docs.cypress.io/guides/references/changelog.html#5-1-0

Cypress. n.d.b. Cypress Dashboard Pricing Plans. Luettu 30.8.2020. https://www.cypress.io/pricing/

Docs Cypress. n.d. Permanent trade-offs. Luettu 30.8.2020. https://docs.cypress.io/guides/references/trade-offs.html#Permanent-trade-offs-1

Dua, A. 2019a. What is difference between selenium 1 and selenium 2?. Julkaistu 5.8.2019. Luettu 10.8.2020. https://www.tutorialspoint.com/what-is-difference-between-selenium-1-and-selenium-2

Dua, A. 2019b. What is difference between selenium 2 and selenium 3?. Julkaistu 29.7.2019. Luettu 10.8.2020. https://www.tutorialspoint.com/what-is-difference-between-selenium-2-and-selenium-3

GitHub innogames/Jmeter-Control-Center. 2019. Description. Päivitetty 25.3.2019. Luettu 14.9.2020. https://github.com/innogames/JMeter-Control-Center

GitHub postmanlabs/newman. 2020. Newman the cli companion for postman. Päivitetty 13.7.2020. Luettu 14.7.2020. https://github.com/postmanlabs/newman

GitLab. n.d.a. About Us. Luettu 7.6.2020. https://about.gitlab.com/company/

GitLab. n.d.b. GitLab CEO. Luettu 7.6.2020. https://about.gitlab.com/handbook/ceo/

GitLab. n.d.c. History of GitLab. Luettu 7.6.2020. https://about.gitlab.com/company/history/

GitLab. n.d.d. Meet our team. Luettu 7.6.2020. https://about.gitlab.com/company/team/#close-modal

GitLab. n.d.e. What is GitLab?. Luettu 7.6.2020. https://about.gitlab.com/what-is-gitlab/

Guru99. n.d.a. How to Use JMeter for Performance & Load Testing. Luettu 3.10.2020. https://www.guru99.com/jmeter-performance-testing.html

Guru99. n.d.b. Postman Tutorial for Beginners with API Testing Example. Luettu 14.7.2020. https://www.guru99.com/postman-tutorial.html

Guru99. n.d.c. What is continuous integration?. Luettu 6.6.2020. https://www.guru99.com/continuous-integration.html

Guru99. n.d.d. What is Jenkins? Continuous integration (CI) Tool. Luettu 6.6.2020. https://www.guru99.com/jenkin-continuous-integration.html

Guru99. n.d.e. What is Jmeter? Introduction & Uses. Luettu 14.9.2020. https://www.guru99.com/introduction-to-jmeter.html

Guru99. n.d.f. What is Selenium? Introduction to Selenium Automation Testing. Luettu 31.7.2020. https://www.guru99.com/introduction-to-selenium.html

Jaspers, T. 2012. Robot Framework Tutorial – A Complete example. Julkaistu 27.4.2012. Luettu 19.8.2020. https://blog.codecentric.de/en/2012/04/robot-framework-tutorial-a-complete-example/

Javatpoint. n.d. Appium Tutorial. Luettu 11.8.2020. https://www.javatpoint.com/appium

Kasurinen, J., P. 2017. Ohjelmistotestauksen käsikirja. E-book. Docendo.

Khetarpal, A. 2020a. Cypress Dashboard Service. Julkaistu 11.7.2020. Luettu 4.9.2020. https://www.toolsqa.com/cypress/cypress-dashboard-service/

Khetarpal, A. 2020b. What is Cypress: Introduction and Architecture. Julkaistu 4.4.2020 Luettu 29.8.2020. https://www.toolsqa.com/cypress/what-is-cypress/

Klärck, P. n.d.a. Experience. Luettu 19.8.2020. http://eliga.fi/experience.html

Klärck, P. n.d.b. Welcome!. Luettu 19.8.2020. http://eliga.fi/index.html

Klärck, P. n.d.c. Writings. Luettu 19.8.2020. http://eliga.fi/writings.html

Lane, K. 2020. Introducing the Postman Agent: Send API Requests from Your Browser without Limits. Julkaistu 16.6.2020. Luettu 30.7.2020. https://blog.postman.com/introducing-the-postman-agent-send-api-requests-from-your-browser-without-limits/

Lardinois, F. 2020. Github is now free for all teams. Julkaistu 14.4.2020. Luettu 21.7.2020. https://techcrunch.com/2020/04/14/github-is-now-free-for-all-teams/

Loisel, J. 2018. JMETER RESULT ANALYSIS: THE ULTIMATE GUIDE. Julkaistu 25.4.2018. Luettu 3.10.2020. https://octoperf.com/blog/2017/10/19/how-to-analyze-jmeter-results/

Mangan, S. 2020. A Beginner’s Guide to Automated API Testing (Postman/Newman). Julkaistu 4.2.2020. Luettu 3.10.2020. https://blog.scottlogic.com/2020/02/04/GraduateGuideToAPITesting.html

Mann, B. 2017. Cypress is now public beta. Julkaistu 10.10.2017. Luettu 4.9.2020. https://www.cypress.io/blog/2017/10/10/cypress-is-now-public-beta/

Mann, B. 2018. We’re officially out of beta!. Julkaistu 20.9.2018. Luettu 4.9.2020. https://www.cypress.io/blog/2018/09/20/cypress-is-officially-out-of-beta/

Newell, N. 2020. How To Install Selenium Tools on Ubuntu 18.04. Julkaistu 10.1.2020. Luettu 5.10.2020. https://www.liquidweb.com/kb/how-to-install-selenium-tools-on-ubuntu-18-04/

Paul. 2020a. A Deconstruction of the Appium Architecture. Päivitetty 13.7.2020. Luettu 17.8.2020. https://www.edureka.co/blog/appium-architecture/

Paul. 2020b. What is Appium & How it Works? | Beginners Guide To Appium. Päivitetty 13.7.2020. Luettu 11.8.2020. https://www.edureka.co/blog/what-is-appium/

PerformanceLab. n.d. Jmeter vs Loadrunner: What Is Better for Performance Testing. Luettu 14.9.2020. https://performancelabus.com/jmeter-loadrunner-for-performance-testing/

Postman. n.d.a. About Postman. Luettu 14.7.2020. https://www.postman.com/about-postman/

Postman. n.d.b. Downloads Postman for Windows. Luettu 14.7.2020. https://www.postman.com/downloads/

Postman. n.d.c. Plans and Pricing. Luettu 14.7.2020. https://www.postman.com/pricing/

PyPI. n.d. Release history. Päivitetty 4.5.2020. Luettu 19.8.2020. https://pypi.org/project/robotframework/#history

Robotframework. n.d.a. Introduction. Luettu 19.8.2020. https://robotframework.org/

Robotframework. n.d.b. Libraries. Luettu 16.8.2020. https://robotframework.org/#libraries

Selenium IDE. 2019. Command-line Runner. Päivitetty 29.7.2019. Luettu 5.10.2020. https://www.selenium.dev/selenium-ide/docs/en/introduction/command-line-runner

Selvarajah, V. 2019. Selenium 4 (alpha) is Released: What's new?. Julkaistu 5.5.2019. Luettu 10.8.2020. https://medium.com/@4400vijay/selenium-4-alpha-is-released-whats-new-7fce5e8cd926

Shah, H. n.d. How GitHub Democratized Coding, Build a $2 Billion Business, and Found a New Home at Microsoft. Luettu 21.7.2020. https://usefyi.com/github-history/

Singh, H. 2018. CI/CD for mobile apps using Appium tool. Julkaistu 12.12.2018. Luettu 18.8.2020. https://www.devinitiate.com/ci-cd-for-mobile-apps-using-appium-tool/

Software Testing Help. 2020. Robot Framework Tutorial – Features and Software Installation. Päivitetty 2.8.2020. Luettu 19.8.2020. https://www.softwaretestinghelp.com/robot-framework-tutorial/

Sudonull. n.d. Load testing, history of process automation. Luettu 14.9.2020. https://sudonull.com/post/65120-Load-testing-history-of-process-automation

Toolsqa. n.d. Cypress Tutorial. Luettu 29.8.2020. https://www.toolsqa.com/cypress-tutorial/

Tuli, S. 2018. Learn How to Set Up a CI/CD Pipeline From Scratch. Julkaistu 10.9.2018. Luettu 16.6.2020. https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Tutorialspoint. n.d.a. jMeter – Overview. Luettu 14.9.2020. https://www.tutorialspoint.com/jmeter/jmeter_overview.htm

Tutorialspoint. n.d.b. Robot Framework Tutorial. Luettu 19.8.2020. https://www.tutorialspoint.com/robot_framework/index.htm

Waseem, M. 2019. Know all About Robot Framework With Python. Päivitetty 27.11.2019. Luettu 8.10.2020. https://www.edureka.co/blog/robot-framework-tutorial/