



Harry Ahlgren

Testiskriptien parsiminen Pythonilla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

27.2.2023

Tiivistelmä

Tekijä:	Harry Ahlgren
Otsikko:	Testiskriptien parsiminen Pythonilla
Sivumäärä:	20 sivua
Aika:	27.2.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Automaatiotekniikka
Ohjaajat:	Lehtori Matti Välikylä

Insinööritöön tarkoituksena oli luoda parseri XML-muotoon tallennetuille testiskripteille ja testimetodeille. Parseri keräsi kuvaukset, suorituserityksen ja olennaisista kentistä tiedot Java-koodiksi.

Työ tehtiin Unicus Oy:n alaisuudessa konsulttina asiakasyrityksessä. Osaprojektina tehtiin Finastra Fusion Markets Summit -ohjelmiston regressiotestiautomaatio käyttäen Silk Test -testausjärjestelmää. Tehtävänä oli siirtää kaikki tarpeellinen tieto olemassaolevista testeistä uusiin testeihin viimeistelyä varten.

Suunnitteluvaiheessa tarkasteltiin, mitkä elementit olivat tärkeitä ja mitä tuli parsia. Samalla jaoteltiin parsittavat tehtävät testiskripteihin ja testimethodeihin. Parsimen käyttöä kokeiltiin useasti ennen varsinaista ajoa, jolloin saatiin tieto, oliko riittävästi elementtejä parsittu.

Työn tuotoksena olivat parserien lisäksi valmiit Silk Test -testausohjelmiston skriptit ja testimetodit Java-koodina. Niitä käyttämällä pystyttiin hahmottamaan ja viimeistelemään toimiva testiskriptikokoelma.

Avainsanat: Python, Java, Silk Test, parseri

Abstract

Author:	Harry Ahlgren
Title:	Test Script Parser Using Python
Number of Pages:	20 pages
Date:	27 February 2023
Degree:	Bachelor of Engineering
Degree Programme:	Degree Programme in Electrical and Automation Engineering
Professional Major:	Automation Engineering
Supervisors:	Matti Välikylä, Senior Lecturer

The purpose of this Bachelor's Thesis project was to create a parser for parsing relevant code from test scripts and methods written in XML to Java. The parser collected the description, execution order and data relevant to testing into Java-code.

The work was carried out for a client company as a consultant for Unicus Oy. The work was part of a project to create a regression test batch using Silk Test. The aim of this part of the project was to move all essential data from the existing tests to facilitate the transition to Silk Test.

In the planning phase the importance of elements was evaluated as to which parts of the source should be parsed. The parsing tasks were divided between test scripts and test methods. The parser was tested multiple times before it was ready to gather information concerning whether the parsing was enough by itself.

The final resulting product was, in addition to the parsers, the Java-code for Silk Test for test scripts and methods. From the result it was possible to quickly evaluate how to proceed in finalizing the working test script batch.

Keywords: Python, Java, Silk Test, Parser

Sisällys

1	Johdanto	1
2	Tavoitteet ja toteutus	3
2.1	Regressiotestit	3
2.2	Metodit ja testiskriptit	4
2.3	Kehitysympäristö	4
2.4	Java	4
2.5	Python	5
2.6	Silk Test	5
2.7	JUnit	7
2.8	Confluence	7
2.9	Jira	8
3	Projektin kulku	8
3.1	Kartoitus	8
3.2	Valmisteluvaihe	9
3.3	Ohjelmointi	10
3.4	Parserien koeajo	10
3.5	Parserien käyttö	11
4	Suunnitelmat ja toteutumat	11
4.1	Testiskriptiparsin	11
4.2	Parsitut testiskriptit	13
4.3	Testimetodiparsin	14
4.4	Parsitut metodit	15
4.5	Ajettavat testiskriptit	16
4.6	Dokumentaatio	17
5	Työskentelyn arviointi	18
5.1	Ohjelmointiosuus	18
5.2	Projektiviestintä	18
5.3	Tilaajan tekemä arvio	19
6	Yhteenveto	19
	Lähteet	21

Lyhenteet

GUI: *Graphical User Interface*. Graafinen käyttöliittymä.

JVM: *Java Virtual Machine*. Spesifikaatio ja virtuaalinen ympäristö, joka suunniteltiin tulkitsemaan sille tulkattua tavukoodia laitteistojen omaksi tavukoodiksi. Sen laaja leviäminen on mahdollistanut Javan suosion ja myöhemmin myös kielten, jotka voidaan kääntää JVM:lle.

XML: *Extensible Markup Language*. Dokumenttien tiedostomuotojen standardi.

1 Johdanto

Tämä projekti oli toteutettu 2021–2022, jolloin työskentelin Unicus Oy:ssä automaatiotestauskonsulttina eräässä pohjoismaisessa pankissa. Tehtäväni oli luoda regressiotestejä varten testiskriptikokoelma. Tässä opinnäytetyössä käsitellään pääasiallisesti osaprojektia, jossa tein kaksi erillistä parseria, jotka parsivat testiskriptejä ja testimetodeja.

Regressiotestien viitekehyksessä testiskriptillä tarkoitetaan skriptiä, joka tarkoittaa sen, että ohjelman haluttu toiminnallisuus on säilynyt läpi muutosten ja päivityksen. Tämä useimmiten tarkoittaa, että versiopäivityksen jälkeen ohjelmassa voi samoilla klikkauksilla ja näppäimistönpainalluksilla saada aikaiseksi saman lopputuloksen kuin ennen versiopäivitystä (1, s. 31).

Automaatiotestaukseen kuului tässä projektissa kokoelma skriptejä, jotka ajetaan jokaisen versiopäivityksen yhteydessä. Tekijä oli aiemmin automatisoinut Chrome-selaimelle tehtyjä skriptejä, joilla varmistettiin FACT-nimisen ohjelmiston ennallaan säilynyt toiminnallisuus. Tämä työ oli osa samaa kokonaisuutta, jonka avulla pankin on tarkoitus testata kaikki ohjelmistot rutiininomaisesti ennen kuin päivitykset päästetään tuotantoon.

Toisin kuin selaimelle kirjoitettuihin skripteihin, jotka yleisesti käyttävät webdriveit, tässä käytettiin Windowsin omaa elementtien tunnistamista. Windows-ohjelman säikeistä haettiin elementtejä, joihin simuloitiin tekstin syöttöä ja klikkauksia. Usein näiden toimintojen jälkeen haettiin jonkin elementin sisältö ja verrattiin sitä odotusarvoon.

Tämä opinnäytetyö tehtiin ohjelmistoalan konsulttiyrityksen Unicus Oy:n alaisuudessa. Tilaajana oli eräs pohjoismainen pankki, joka halusi käyttää testauksessa yrityksen automaatiotestauspalveluita nopeuttamaan, parantamaan ja varmistamaan ohjelmistopäivitysten testausprosessia.

Asiakas painotti työmme osalta käyttöliittymän toimivuuden testausta. Tässä opinnäytetyössä keskityn osaan työstä, jossa tein parsijan jo valmiina olleille regressiotesteille, jotka siirrettiin järjestelmästä toiseen.

Pohjana oli aikaisempien tekijöiden luoma testiskriptikokoelma, joka oli tallennettu XML-muotoon, ja vähäinen määrä muuta dokumentaatiota. Parsijan tehtävänä oli ottaa olennaiset tiedot testiskriptistä talteen, jolloin pystyimme käyttämään lähes suoraan tätä dataa vaikeaselkoisen raakatiedon lukemisen sijaan.

Projektin tavoite oli luoda parseri, joka kerää XML-muodossa olevasta datasta testiskriptiin tietoa. Parseri toimii siten, että se etsii tiettyjä osia XML:stä ja kääntää ne Java-koodiksi. Parseri oli toiminnallisuudeltaan jaettu kahteen osaan. Ensimmäinen osa keskittyi testiskripteihin ja toinen metodeihin.

Testit itsessään sisälsivät tietoa siitä, mitä arvoja haluttiin syöttää ohjelmalle. Esimerkiksi vaihtokaupan vastapuolen lyhenne ja kaupan sisältö. Metodit vastaavasti ottivat parametreiksi tietoa testiskripteiltä ja syöttivät ne GUI:lle käyttäen Windows-ohjelman elementtien tunnistetietoja, joita meidän testausohjelmistomme Silk Test osasi käyttää.

Kuvassa 1 näkyy ote parsimattomasta XML-tiedostosta. Kuvassa ainoa tärkeä talteenotettava tieto seuraa <ID>-kenttää: se kertoo elementin, jota testi käytti. Loput kentistä eivät ole parserin kannalta kiinnostavia, sillä kaikki muu sisältö on ohjeistusta, mitä ID:lle tehdään. Tässä tapauksessa testin tarkoitus on klikata nappia, jonka tunniste on "Back" indeksissä 10 Summitin työkalurivillä. ID on tunniste, jolla elementti löytyy työpöydältä.

```

<FindCriteria />
<MainGUICtrl>
  <ID>[Toolbar : Shared Tools] Tool : 219:Back - Index : 10 </ID>
  <RecGUICtrlId>64136</RecGUICtrlId>
  <RecStepId>5937</RecStepId>
  <ContentValue />
  <Name>Back</Name>
  <IsClickSupported>False</IsClickSupported>
  <TypeID>50000</TypeID>
  <IsEnabled>True</IsEnabled>
  <IsChecked>False</IsChecked>
  <IsExpanded>False</IsExpanded>
  <IsKeyboardFocusable>False</IsKeyboardFocusable>
  <IsPassword>False</IsPassword>
  <Provider>Standard</Provider>
  <Index>0</Index>
  <X>22.727272727272</X>
  <Y>63.636363636363</Y>
  <RecordedAction>None</RecordedAction>
  <XPath />
  <Attributes />
</MainGUICtrl>

```

Kuva 1. XML-muodossa oleva toiminto eräästä metodi-tiedostosta

2 Tavoitteet ja toteutus

Tämän projektin tavoitteena oli tuottaa Java-kielellä kirjoitettu testiskriptiko-koelma Finastra Fusion Summit -ohjelman regressiotestauksen automatisointia varten. Testausohjelmistona toimi Silk Test. Alunperin testejä oli yli sata kappaletta XML-muodossa. Metodeja oli myös parisen sataa kappaletta, jotka tuli kääntää Silk Testille.

2.1 Regressiotestit

Tavoiteltava regressiotesti on sellainen, että se testaa usein käytettyä toiminnallisuutta ja selvittää, onko testattavaan ohjelmistoon tehty muutos sellainen, että se rikkoo toiminnallisuutta (2, s. 65). Hyvä testi on myös helposti ymmärrettävä kuvailtuna ja ylläpidettävä.

Parsittujen testien arviointi kuului myös tavoitteisiin. Testit, joilla oli päällekkäinen kattavuus, pyrittiin tunnistamaan. Samoin heikosti ymmärrettävät testit eivät parsittunakaan olleet hyödyllisiä projektin kannalta, joten niiden tunnistaminen oli osa tehtävää.

Testiskripteistä pyrittiin myös karsimaan osia parsituista testiskripteistä, jotka eivät vastanneet asiakkaan vaatimusmäärittelyyn. Skripteissä ei asiakkaan toiveesta seurattu tarkkoja lukuja tai päivämääriä, vaan keskityttiin graafisen käytölliittymän toiminnallisuuden säilymiseen päivitysten yli.

2.2 Metodit ja testiskriptit

Metodit koostuivat erilaisista käskyistä, esimerkiksi oikealla napilla klikkauksesta kenttään, tekstin kirjoituksesta tai jonkin menun valitsemisesta. Niiden avulla navigoidaan testattavana olevassa ohjelmistossa.

Metodien ja testien osalta tavoite oli parsia ne kokonaisuudessaan XML-formaatista Java-koodiksi. Testit koostuivat pääosin kutsuista metodeihin. Ne molemmat tuli saattaa ohjelmoijalle ymmärrettävään muotoon, josta ne pystyisi helposti täydentämään toimivaksi Java-koodiksi.

2.3 Kehitysympäristö

Kehitys tapahtui Windows 10 -virtuaalikoneissa, joihin ei ollut annettu kuin peruskäyttäjän oikeudet. Kyberturvallisuuden takia virtuaalikoneet oli myös palomuurilla rajattu pois muista kuin tarpeellisista järjestelmistä.

Kirjoitin koodia osittain Pythonin omassa IDLE-kehitysympäristöissä virtuaalikoneen rajoitusten vuoksi. Versionhallinnoinnissa käytettiin Gitiä ja Bitbucketia. Sekä Pythonilla kirjoitetut parseriskriptit että tuotoksena olleet Java-tiedostot säilöttiin Bitbucketiin.

2.4 Java

Varsinaisten testiskriptien ohjelmointikieleksi valikoitui Java aikaisemman Silk Testillä toteutetun projektin pohjalta, joka tarvittiin Javalle saatavia PDF-kirjas-toja avuksi testaamaan dokumenttitulosteita.

Java on oliopohjainen ohjelmointikieli, jonka kehittivät Sun Microsystemsillä työskennelleet Bill Joy ja Ryan Gosling vuonna 1995. Myöhemmin, vuonna 2010, se päätyi Oraclen omistukseen. (3.)

Tässä projektissa käytössä oli pitkän tuen piirissä oleva Java 8. Java 8 on julkaistu vuonna 2014 ja on edelleen tuettu vuoteen 2030 asti. Erittäin pitkän tuensa vuoksi Java 8 on yleisesti käytössä.

2.5 Python

Python on korkeatasoinen ja oliopohjainen ohjelmointikieli, joka on erittäin helppolukuinen. Pythonissa käytetään sarkaimien sijaan sisennyksiä koodin jakamisessa lohkoihin.

Pythonin vahvuuksiin kuuluu sen helppolukuisuus ja sen kirjoittamisen nopeus. Yhdellä koodirivillä saadaan usein enemmän aikaiseksi kuin useilla riveillä Javaa tai useimpia muita korkeatasoisia ohjelmointikieliä. (4.)

Python valikoitui helppoutensa ja parsimiseen soveltuvuutensa vuoksi työkalukseni parsimiseen. Vaikuttavana tekijänä oli myös oma osaaminen. Käytössä oli Pythonin versio 3.8.

2.6 Silk Test

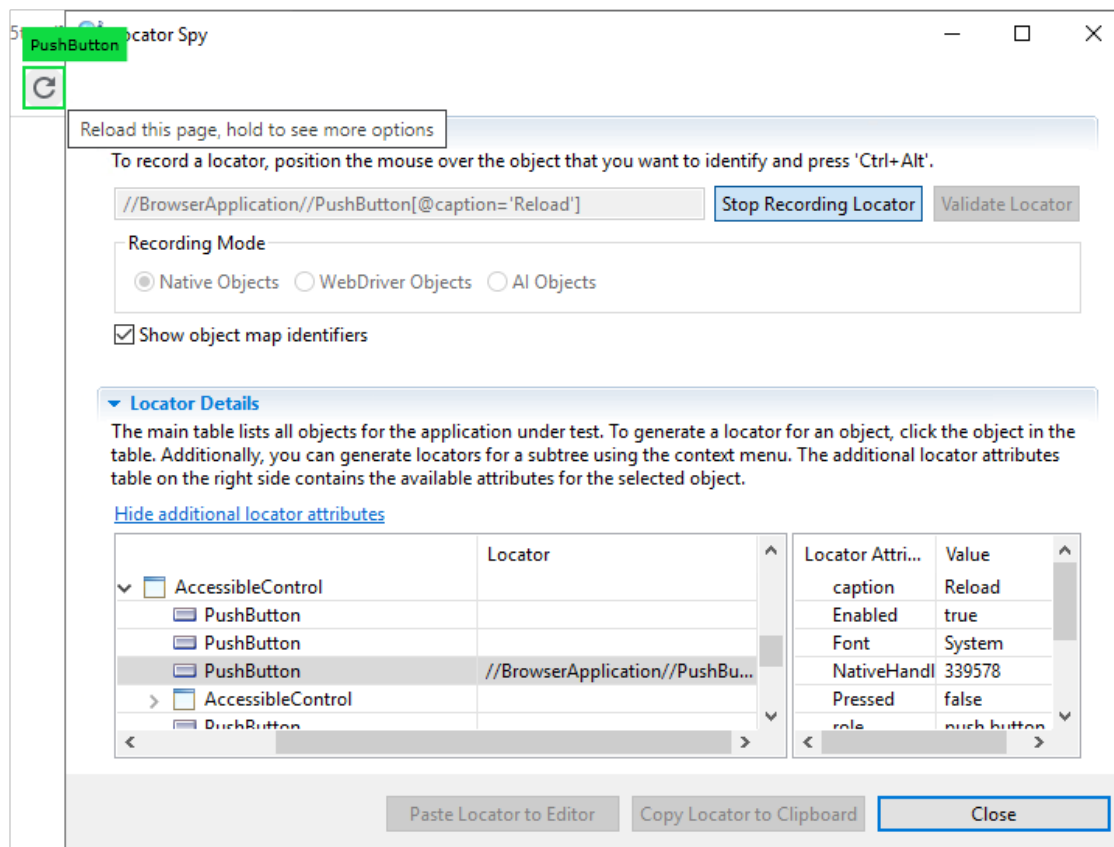
Toteutuksessa oli asiakkaan pyynnöstä käytetty GUI:n testaamiseen hyvin soveltuvaa Silk Test -testiohjelmistoa. Tilaajan yhteyshenkilöllä oli kokemusta saman ohjelmiston käytöstä IBM:llä Summitin testauksessa, ja se vaikutti myös tavoitteiden hahmottamiseen.

Silk Test on testiohjelmisto, jossa on tuki muun muassa Javalle, C#:lle ja Pythonille. Tuki on kuitenkin paras Java-ympäristölle. Silk Testin kehittäjä Microfocus oli brändännyt Javaa varten Eclipse-kehitysympäristön nimellä Silk4J, ja tämä oli käytössä, kun viimeistelimme parsittua Java-koodia. (5.)

Tekijä oli myös aiemmin käyttänyt Silk Testiä tilaajalla aikaisemmassa projektissa, jossa automatisoitiin FACT-nimistä selainpohjaista ohjelmistoa.

Tämän osaprojektin kannalta tärkeitä ominaisuuksia Silk Testissä oli nauhoitus-toiminnot Recorder ja Locator Spy, joiden avulla pystyi selvittämään Windows-ohjelmista elementtien tunnisteet. Näillä elementtien tunnisteilla Silk Test kykeni löytämään esimerkiksi ikkunan, kentän tai napin, johon kohdistaa jokin toiminto.

Locator Spy löytää Windowsin ja selaimen automaatioon tarpeelliset elementit kun hiiri viedään niiden päälle. Kuvassa 2 nähdään esimerkki reload-napin löytämisestä selaimessa. Löydetty elementti näkyy vihreänä ja elementin polku näkyy tekstikentässä.



Kuva 2. Locator Spy tunnistaa elementtejä selaimesta ja Windows-ohjelmista

Elementin polku määrittyy siten, että polun juuri on ohjelmisto tai selain. Siitä eteenpäin navigoidaan elementtien ominaisuuksien perusteella. // tarkoittaa, että etsitään mistä tahansa alipolusta elementti ja / tarkoittaa suoraan seuraavaa polkuosoitetta.

Locator Spyllä pystytään myös varmistamaan, onko jokin elementti oikeasti löydettävissä jossakin näkymässä kirjoittamalla tekstikenttään lokaattori ja painamalla Validate-nappia.

2.7 JUnit

Skriptien suorituksessa Silk Test käytti Javassa yleisesti käytettyä JUnit 4 -kirjastoa. JUnit on avoimen lähdekoodin järjestelmä yksikkötestaukselle JVM-ympäristössä. JUnitia käytetään Javassa annotaatioin, ja kun testiskriptin metodin edelle kirjoitetaan "@Test"-annotaatio, on metodi silloin ajettava testiskripti (6).

JUnit-testin voi käynnistää komentoriviltä tai Silk Testin käyttämästä Silk4J/Eclipsestä. Silk Testin oman Open Agent -ajurin käynnissäolo vaaditaan Silk Testillä kirjoitetulta skriptiltä.

Suorituksesta tallentuu logi. Logi sisältää Open Agentin ollessa käynnissä kuvaakaappauksen jokaisesta toimenpiteestä, jonka testi on tehnyt. Logista löytyy kuvakaappaus ennen ja jälkeen toimenpiteen. Mikäli JUnit-testi käynnistetään käyttäen Silk Testin omaa Silk Central -ajoympäristöä ja sen execution serveriä, on mahdollista ottaa myös videokaappaus suoritetuista testiskripteistä. (5.)

2.8 Confluence

Confluence on yleisesti käytössä oleva organisaatiowikiohjelmisto, jota käytetään verkkoselaimella. Sillä voidaan luoda wikisivustoja organisaation sisäiseen käyttöön. (7.)

Confluencessa tavoite oli tallettaa arkkitehtuuriin opastusta, artikkeleita testimenetelmistä, metodologiasta ja selitystä testiskriptikokoelman käytöstä ja tietoa siitä, miten ohjelmistoja käytetään.

Keskeisimpiä artikkeleita Confluencessa on kokemuspohjalta luomani vianetsintään keskittyvä artikkeli ja opastus siitä, kuinka infrapuoelta voi tehokkaasti pyytää avustusta eräisiin toistuviin käyttäjästä riippumattomiin ongelmatilanteisiin.

2.9 Jira

Jira on ohjelmistoyritys Atlassianin tehtävienhallintaohjelmisto. Sen avulla tehtäviä voidaan luoda ja osoittaa itselle tai muille käyttäjille (8). Se toimii myös osaltaan dokumentaationa. Tavoitteena oli dokumentoida kaikki olennaiset asiat helposti löydettäväksi Jiran kautta.

Tässä projektissa Jirassa oli jokaiselle skriptille oma tehtävänsä. Esimerkiksi SFS-2 viittasi swap-kaupan luomista testaavaan testiskriptiin. Myös parsimille oli luotu omat tehtävänsä ja kuvauksensa Jirassa.

3 Projektin kulku

3.1 Kartoitus

Ensimmäinen vaihe oli selvittää XML-tiedostoista, mitä testauksen kannalta olennaisia tietoja ne sisälsivät. Tätä varten piti perehtyä sekä Finastra Fusion Markets Summit -ohjelmaan että XML-tiedostojen sisältöön.

Summitin osalta kartoitus sisälsi peruskäytön opetteluja. Tässä käsiteltiin asioita, kuten mistä napista pääsee navigoimaan ”Money Markets” -näkymään ja mikä nappi käynnistää jonkin toimenpiteen.

Samanaikaisesti käytiin läpi parsittavia testimetodeja, katsottiin, mitä niiden oli tarkoitus tehdä, ja millaisiin Windows-ohjelman elementteihin ne olivat vuorovai-
kutuksessa.

Yleiskuvan saanti XML-muodossa olevista testiskripteistä näytti, että tiedos-
tossa testiskriptit ja testimetodit eivät olleet suoritusjärjestyksessä. Jokaisessa
vaiheessa oli kuitenkin tieto indeksistä, jonka avulla pystyi selvittämään, missä
järjestyksessä parsitun skriptin olisi parsimisen jäljiltä oltava.

Tilaajan kanssa keskustelin siitä, mitä ominaisuuksia testiskripteissä tulisi olla.
Sitä varten perehdyin XML-tiedostoihin ja tilaajan näkemykseen hyvästä tes-
tiskriptistä.

Edellisellä toimittajalla, joka oli XML-muotoiset testiskriptit ja testimetodit toimit-
tanut, oli näkemys, joka poikkesi tilaajan näkemyksestä. Useimmat skripteistä
tekivät numeerisia vertailuja. Tilaajan mielestä oli kuitenkin olennaista keskittyä
GUI-testaukseen. Tilaajalla oli erikseen testejä, jotka tutkivat arvojen oikeelli-
suutta, joten sen lisäksi, että numeeristen arvojen vertailu oli haastavaa, oli se
myös pitkälti turhaa työtä.

3.2 Valmisteluvaihe

Valmisteluvaiheessa kerättiin XML-tiedostot kahteen eri kansioon niiden tyyppin
mukaan, testiskriptit ja testimetodit omaansa. Samalla tehtiin havaintoja siitä,
miten parsittu koodi tulisi jaotella Java-luokiksi. Päädyttiin ratkaisuun, jossa jo-
kainen testiskripti olisi oma luokkansa metodeineen, mutta testimetodit tulisi ja-
kaa luokkiin kokonaisuuden perusteella. Esimerkiksi Swap-kaupoista tuli jokai-
selle Swap-kaupan tyyppille oma Java-luokkansa, mutta kaikki Swap-metodit ha-
luttiin yhteen luokkaan.

Tässä vaiheessa käytiin lävitse vaatimuksia toisen automaatiotestaajan kanssa,
jotta varmistuttiin siitä, että kartoitusvaiheen ajatukset olivat säilyneet eheinä ja
käyttökelpoisina.

3.3 Ohjelmointi

Ohjelmointi tapahtui tämän insinööriyön aiheena olevan osaprojektin sisällä Pythonia käyttäen, mutta tuotos oli Java-koodia. Aluksi Pythonissa luotiin pohja menetelmälle, joka kävi läpi työhakemiston jokaisen .xml-päätteisen tiedoston. Tämä läpikäyminen sijoitettiin päämetodiin ja sillä kutsuttiin menetelmää, joka oli varsinainen parsija. Tässä käytettiin tiedoston nimeä parametrina.

Jokainen .xml-tiedosto käsiteltiin merkkijonoksi ja tälle merkkijonolle tehtiin eri toimenpiteitä halutun datan saamiseksi. Ensimmäisenä haettiin merkkijonoksi menetelmän nimi ja kuvaus. Testimetodin ollessa kyseessä tämä nimi katkaistiin osaksi, joka kertoi mihin luokkaan testimetodi sijoitettaisiin, ja menetelmä tallennettiin kyseiseen luokkaan koko metodin nimellä.

Javan docstringiin tallennettiin löydetyn <Description>- ja </Description>-kenttien väli, jotta oli helppo seurata, mitä minkäkin metodin ja skriptin oli tarkoitus tehdä alkuperäisessä testissä. Muu sisältö lähdetiedoista käytiin lävitse hakien ensisijaisesti toimintoja ja niihin liitettyjä elementtejä siirrettäväksi Java-koodiksi.

3.4 Parserien koeajo

Kun parserit olivat valmiit, niillä parsittiin kokeeksi muutamia XML-tiedostoja. Ensimmäisenä huomattiin, että lähdetiedoissa oli erikoismerkkejä, jotka aiheuttivat parsimille ongelmia. Myös osa tiedostoista ei sisältänyt haettuja osasia, ja parsin kaatui niihin.

Näiden havaintojen jälkeen käsiteltiin parsimissa Regexiä avuksi käyttäen kaikki erikoismerkit pois ja parseriin laitettiin ehtolause, joka ohitti tietoa sisältämättömät tiedostot. Erikoismerkeillä ei ollut mitään merkitystä varsinaisten testien toiminnan kannalta, joten niiden poisto sisällytettiin jokaisen tiedoston läpikäymiseen. Samalla HTML-koodit, kuten ", muutettiin sitaateiksi tai vastaaviksi.

3.5 Parserien käyttö

Parserit tulivat tekijän ja toisen automaatiotestaajan käyttöön. Aluksi ajo suoritettiin kaikille testiskripteille ja testimetodeille, mutta parseria päivitettiin useasti uusien tilanteiden varalle, kun huomattiin uutta tietoa, jota XML-tiedostolta saatiin. Näin ollen parseria ajettiin pienemmille osille testiskriptejä kerralla, ja kun nämä skriptit oli viimeistelty Java-puolella otettiin uusi sarja skriptejä käsiteltäväksi.

Testiskriptejä parsiessa otettiin aluksi myös osan XML-tiedostosta testiskriptin kuvaukseksi, jotta pystyttiin varmistumaan siitä, että testeissä on oikea sisältö. Parsimien kehittyessä ja luottamuksen kasvaessa parsintaan jätettiin tämä osa pois. Samoin myöhemmin jätettiin kuvauksesta osa siirtämättä Java-koodiin. Sen sijaan talletettiin tämän kuvauksen osat Jira-tehtäviin, jotka oli liitetty testiskriptiin.

Testimetodien osalta koko kokoelma suoritettiin kokonaisuudelle XML-tiedostoja, jotka sisälsivät saman alkuliitteen. Esimerkiksi Bond-alkuiset testimetodit parsittiin yhdellä kertaa.

4 Suunnitelmat ja toteutumat

Testiskriptien osalta suunnitelmana oli saada kaikki testit parsittua. Parsitut testit tuli myös täydentää toiminnallisuudeltaan toistokelpoisiksi automaatiotesteiksi. Asiakkaalle toimitettiin projektin lopuksi dokumentaatio, toimivat parsimet sekä parsitut ja viimeistellyt testit.

4.1 Testiskriptiparsin

Parsimen tarkoitus on käydä läpi kaikki XML-tiedostot hakemistosta, jossa se ajetaan. Ennen parsimen ajoa, joko siirrytään hakemistoon jossa parsiminen tapahtuu tai siirretään pienempi erä parsittavia skriptejä ajoa varten kansioon. Eri-tyisesti koevaiheessa käytiin läpi pienempiä eriä parsittavia skriptejä.

Esimerkkikoodissa 1 nähdään, kuinka parsin käy läpi jokaisen käynnistyspaketin tiedoston, joka päättyy .xml-päätteeseen. Jokaisen tiedoston kohdalla metodi kutsuu menetelmää parsetest tiedoston sijainnilla, laskurin arvolla ja XML-tiedostojen määrällä.

```
def main():
    print("Summit XML test script parser v" + str(version) + "\n")

    cwd = os.getcwd() # get the current working directory
    xmlfiles = [entry.path.lower() for entry in os.scandir(cwd) if
entry.is_file() and entry.path.lower().endswith(".xml")]

    print(len(xmlfiles), "files found in " + os.getcwd())

    count = 0

    for filename in xmlfiles:
        count += 1
        parsetest(filename, count, len(xmlfiles))

    print("Parsed", count, ".xml files to java")

if __name__ == "__main__":
    main()
```

Esimerkkikoodi 1. Parsimen päämetodi.

Esimerkkikoodissa 2 testimetodille haetaan nimi ja kuvaus XML-tiedostosta, jonka jälkeen nimen mukainen .java-tiedosto luodaan ja siihen luodaan myös Java-koodin normaalit alkutoimet templaattista. Templaattiin kuuluu esimerkiksi kirjastojen tuominen.

```
print("Parsing file : " + filename + " -> " + name + ".java [" +
str(count) + "/" + str(amount) + "]")
w = open("javas/" + name + ".java", "w", encoding = "utf")

w.write("/* RC test parsed with Summit test parser " +
str(version) + "\n")
w.write(desc + "*/\n")

w.write(template_class + name)
w.write(template_method)

w.write("public void verify"+ name + "() {\n") # method name
```

Esimerkkikoodi 2. Testimetodin nimi etsitään XML-tiedostosta.

Kommenttiin sijoitetaan XML-tiedostossa <Description>-rivillä ollut tieto, joka sisältää usein dokumentaatiota. Tyhjän <Description>-rivin tapauksessa uuteen kommenttiriviin tulee ainoastaan parserin versionumero ja tyhjä rivi.

Tämän jälkeen luodaan kirjastosta jokaisesta testin askeleesta avain-arvo-pari. XML:ssä askeleet eivät näy järjestyksessä, joten kun kirjaston avaimet otetaan talteen, ne järjestetään ennen kirjoittamista Java-koodiksi. Tähän tarkoitukseen soveltuvat hyvin Pythonin kirjastomenetelmät (9, s. 78).

Myös parametrit haetaan vastaavalla menettelyllä. Jokainen parametreistä esitellään Java-koodissa String-muuttujana ennen kuin kutsutaan metodia, jossa parametreja käytetään.

Odotusarvot ovat kuvauksia, joten odotusarvoja ei voida suoraan muuttaa koodiksi, vaan vertailu jää käsinkirjoitettavaksi. Siksi esimerkkikoodissa 3 liitetään jokainen rivi //-merkkeihin, mikä tarkoittaa Javassa kommenttiriviä. Kommenttirivit kirjoitetaan parsimisen jälkeen koodiksi kuvauksen mukaisesti.

```
if ("<ExpectedResults>" in stepdata):
    expected = stepdata.split("<ExpectedResults>")[1].split("</") [0]
    expected = "//" + "\n//".join(expected.split("\n"))
    w.write("\t\t// Expected results: \n")
    w.write(expected + "\n")
```

Esimerkkikoodi 3. Odotusarvojen parsiminen.

4.2 Parsitut testiskriptit

Javaksi parsituista testiskripteistä muodostui lähdekoodia, joka oli lähes, mutta ei täysin, ajokelpoista JUnit-testiä. Esimerkkikoodissa 4 näkyy ote parsitusta testistä. Siinä käynnistetään ensin Silk Test baseState-oliota käyttämällä ja kirjaututaan @Test-osiossa sisään testattavana olevaan ohjelmistoon.

```

@Before
Public void baseState() {
    BaseState baseState = new BaseState();
    baseState.execute(desktop);
}

@Test
public static void verifySwapTrade() {
    String user = "gauser";
    String password = "Z/q/pWiTree2dMQfjsn==";
    LaunchFT(desktop, user, password); // Launch Summit FT
}

```

Esimerkkikoodi 4. Ote Java-koodista parsinnan jälkeen.

@Before, @Test ja tästä esimerkkikoodista puuttuva @After ovat JUnit-kirjaston annotaatioita (6). @Before-osio ajetaan ennen jokaista testimetodia, @After ajetaan jokaisen testin jälkeen.

4.3 Testimetodiparsin

Metodiparsimen päämetodi oli hyvin samanlainen kuin testiskriptiparsin luvussa 4.1. esimerkkikoodissa 1 esitelty metodi. Metodeissa oli kuitenkin erilainen tallennustapa, sillä haluttiin koota kaikki samantyyppiset metodit tiedostoihin niiden luokan mukaisesti.

Swap-metodeille esimerkiksi luotiin vain yksi luokka, ja kaikki siihen luokkaan kuuluvat metodit tallennettiin peräkkäin tähän luokkaan. Esimerkkikoodissa 5 näkyy, miten jaottelu toimi.

```

savename = name.split("_")[0]
# take only the beginning, ex. BondDef from BondDef_Filln02
savename = re.sub("\d", "", savename) # remove all digits
savefile = subfolder_to_save + "/" + savename + ".java"
print("Parsing file : " + filename + " -> " + savefile)

```

Esimerkkikoodi 5. Testien jaottelu kokonaisuuksittain luokkiin.

Metodin askeleet piti myös järjestää askelnumerojärjestykseen. Se tapahtui leikkaamalla jokainen osio, jossa löytyy <Step>, ja järjestämällä heti <Index>-kohdalla seuraavalla numerolla. Esimerkkikoodissa 6 näkyy, miten askeleet laitettiin numerojärjestykseen metodiparserissa.

```

executions = data.split("<Step>")[1:]

for execution in executions:
    number = int(execution.split("</Index>")[0].replace("<Index>", ""))
    # to organize the steps in right order
    executiondata = execution.split("</Step>")[0] #.split("</Index>")[1]
    executiondata = "\n\t".join(executiondata.splitlines())
    steps[number] = executiondata

```

Esimerkkikoodi 6. Testiaskeleiden järjestäminen kirjastoon.

Kun testiaskeleet oli tallennettu kirjastoon, käytettiin järjestämisalgoritmia niiden avaimille ja siten pystyttiin järjestämään askeleet oikeaan numerojärjestykseen. XML-tiedostossa askeleet eivät olleet järjestyksessä, niissä oli vain <Index>- ja </Index>-tunnisteiden välissä järjestyksnumero.

```

for key in keys:
    stepdata += "\nStep " + str(key) + ":\n"
    line = extractlines(steps[key])
    stepdata += line + "\n"
    if ("String" in line):
        parameterdata += ", " + ", ".join(" ".join(word for word in
row.split(" ") if not ":" in word) for row in line.split("\n") if
"String" in row)
# removes words with ":" in line and joins all lines with String
# in row

```

Esimerkkikoodi 7. Metodin parametrien kerääminen parametririville.

Parametrit kerättiin jokaisesta askeleesta metodikutsuun esimerkkikoodin 7 mukaisesti jokaisesta parametristä, joka oli määriteltä merkkijonoksi.

4.4 Parsitut metodit

Metodien osalta parsittu lopputulos vaati enemmän tekemistä kuin parsitut testiskriptit, koska XML-tietona olleet lokaattorit eivät olleet täysin yksi yhteen Silk Testin ja edellisen testiskriptiohjelmiston RightClickin välillä.

```

<DisplayName>Window(&apos;Finastra Fusion Markets Summit - Money
Market&apos;)</DisplayName>

```

Esimerkkikoodi 8. Ikkunan aktiiviseksi asettaminen XML-datassa.

Tapa, millä RightClick-ohjelmisto havaitsi Back-napin esimerkikoodissa 8, on hankalasti luettava. Esimerkkikoodissa 9 on parsittu versio.

```
desktop.<FormsWindow>find("/FormsWindow[@automationId='Finastra Fusion Markets Summit - Money Market*']")
```

Esimerkkikoodi 9. Summit-ikkuna Silk Test -koodin osana.

Esimerkkikoodissa 9 nähdään Silk Testin tapa valita Finastra Fusion Summit -ikkuna. Se ei kuitenkaan vielä tee mitään, vaan toiminto sille piti selvittää. Testimetodiparserille saatiin usein parhaimmillaan lopputulokseksi vain oikea automationId-kenttä ja päättelyn keinoin tuli keksiä, mitä elementille tehtiin.

Vaikka id tuli suoraan Windowsin tavasta havaita elementtejä, oli sekin useimmiten vain osittain oikein johtuen testiohjelmistojen eroavaisuuksista. Parsimen käyttö kuitenkin yksinkertaisti huomattavasti tätäkin prosessia. Parsimelta saatiin lähes käyttökelpoista koodia. Siihen tuli useimmiten vain lisätä toiminnallinen osa.

Esimerkkikoodi 9 tapauksessa kenttään voitiin lisätä käsky *.setActive();*-riville, jolloin koodipätkä täydentyi aktiivisen ikkunan valinnaksi.

```
datalines = {"value", "name", "defaultvalue", "parameters", "parameters", "displayname", "internalvarname"}
```

Esimerkkikoodi 10. Listaus datakentistä.

Talteenotettavia datakenttiä oli metodeissa seitsemän kappaletta. Esimerkkikoodissa 10 on esillä kentät, joista testimetodiparsimen kannalta oltiin kiinnostuneita.

4.5 Ajettavat testiskriptit

Ajettava testiskriptikokoelma oli koko projektin tavoite. Parseri ja parsitut testit olivat vain osa tätä kokonaisuutta. Kun testiskriptit ja testimetodit oli parsittu ja

manuaalisesti viimeistely, ne paketoitiin .JAR-tiedostoksi ja ladattiin Bitbuckettiin.

Paketoitujen testien ajo tapahtui Silk Central -nimisessä ohjelmistossa, jossa luotiin testisuunnitelma ja organisoitiin testiskriptikokoelma esisuoritettaviin ja normaaleihin testeihin.

Testien ajoa varten tarvittiin vielä käynnistysympäristö, joka koostui kolmesta Silk Test -palvelimesta, jotka ajoivat testejä ja keräsivät niiden ajoista tulokset. Virheiden tai epäonnistuneen ajon tuloksena Silk Test otti kuva- ja videokaappaukset, joiden avulla pystyi analysoimaan, mitä oli mennyt pieleen.

4.6 Dokumentaatio

Dokumentaatiossa oli useita kerroksia ja tallennussijainteja. Keskitetysti tietoa oli kerätty Jiraan ja Confluenceen. Testiskripteissä ja testimeteodeissa oli kommenttiosiossa paikallisiin toimintoihin liittyvää koodin dokumentaatiota.

Testeissä oli alkuvaiheessa usein toisinto samasta dokumentaatiosta, joka löytyi Jirasta, jotta tietoon pääsi nopeasti käsiksi. Parseri myös nouti XML:stä kuvaukset ja tallensi ne testiskripteihin.

Myöhemmässä vaiheessa projektia dokumentaatio siirtyi enemmän Jiraan ja Confluenceen. Testiskriptien koodissa säilöttiin kuvaus siitä, mitä testin on tarkoitus tehdä, ja kommentteja siitä, mitä koodin on tarkoitus tehdä.

Testimetodien kaikki dokumentaatio oli itse testimetodin lähdekoodissa. Javan dokumentaatiomerkkijonossa oli selitystä parametreista ja testin toiminnasta.

5 Työskentelyn arviointi

5.1 Ohjelmointiosuus

Pythonilla tuotantotasaisen koodin kirjoittaminen oli nopeaa ja sujuvaa. Alusta asti kiinnitettiin huomiota hyvien Pythonin PEP-oppaassa (10) määriteltyjen ohjelmointityyliohjeiden noudattamiseen. Tyylioppaan noudattaminen teki koodista helposti luettavaa, ja siihen pystyi palaamaan nopeasti vielä huomattavasti kirjoittamisajankohdan jälkeen.

Eniten aikaa kului joidenkin poikkeustilanteiden selvittämiseen. Esimerkiksi XML-tiedostossa saattoi olla käytetty sitaatin sijaan merkintää *"*; tai jossakin menetelmässä oli erikoismerkkejä. Näiden ratkaiseminen tapahtui useimmiten vasta koeajon jälkeen, koska ne olivat suhteellisen harvinaisia.

Parsimen koodaamisella pystyttiin välttämään toistavien ja samaan aikaan haastavien tehtävien tekemistä, kun testiskriptejä siirrettiin järjestelmästä toiseen.

5.2 Projektiviestintä

Kommunikaatio oli erittäin tärkeä osa projektin onnistumista. Joitain ristiriitoja aiheutti alan käytäntöjen eroavaisuus yrityksen kulttuurin ja alan yleisten käytäntöjen välillä. Koko projektissa kommunikaatio sujui hyvin, kun oppi ymmärtämään ja seuraamaan asiakkaan työkulttuuria.

Tämä tarkoitti jonkin verran vähemmän optimaalisten ratkaisujen noudattamista, koska työskentelyyn käytetyt virtuaalikoneet olivat hyvin tiukasti rajoitetuilla käyttöoikeuksilla kehittäjän näkökulmasta. Jokaisesta muutoksesta tuli luoda tiikki projektijärjestelmään ja odottaa sen toteutumista.

Kommunikaatio projektissa itse parsinten osalta oli lähinnä automaatiotestaajien välistä, eli sain palautetta siitä, millaisia ominaisuuksia toinen

automaatiotestaaja toivoi näkevänsä lopputuotoksessa. Tässä oli tärkeää esittää kysymyksiä ja ottaa vastaan palautetta.

5.3 Tilaajan tekemä arvio

Asiakas oli tyytyväinen luovutettuihin työn tuloksiin. Asiakas ei suoraan arvioinut parsintaohjelmiin liittynyttä osaprojektia, mutta kokonaisprosessista saatu palaute oli yksinomaan positiivista.

Kehitystyötä kehuttiin nopeaksi. Epäsuorasti tämä tarkoitti myös parserin kehitystä ja sen käyttöä, koska ilman parsimia työ olisi edistynyt huomattavasti hitaammin.

6 Yhteenveto

Asiakas tilasi testiskriptikokoelman, jolla testata Finastra Fusion Markets Summit -ohjelmistoa. Testiskriptejä ja testimetodeja oli valmiina sadoittain, ja niiden siirtäminen XML-muodosta Silk Test -ohjelmistolla käytettäväksi testeiksi oli keskeinen osaprojekti ja tämän opinnäytetyön aihe. Metodeille ja skripteille luotiin molemmille omat parserinsa.

Testiskriptien parsiminen oli näistä helpompi tehtävä ja sillä sai valmiin Java-koodatun skriptin, johon ei paljoa muutoksia tarvinnut tehdä. Ainoastaan vertailu odotusarvon ja saadun tuloksen välillä täytyi aina tehdä manuaalisesti.

Java-koodiksi parsittu testiskripti sisälsi kirjastojen tuomiset, parametrit sisältävät metodikutsut ja koodilohkoiksi jakamisen Java-koodina ja myös kuvauksen siitä, mitä skriptin oli tarkoitus tehdä.

Testimetodien parsiminen vaati enemmän jälkikäsitteilyä. Windows-elementtien automaatiotunnisteet vaativat usein pientä uudelleenkirjoitusta ja jokaisen parsitun Java-koodirivin yhteydessä piti selvittää, mitä rivillä oli tarkoitus tehdä.

Tässä avuksi oli Silk Test -ohjelmiston Record-toiminto.

Parsittuja skriptejä ja testimetodeja käytettiin luomaan regressiotestien skriptikoelma, jolla testattiin Finastra Fusion Markets Summit -nimistä ohjelmistoa.

Lähteet

- 1 Certified Tester Foundation Level (CTFL) Syllabus. 2018. Verkko-pdf. International Software Testing Qualifications Board. Päivitetty 1.7.2021. Luettu 15.1.2023.
- 2 Certified Tester Advanced Level Syllabus: Test Automation Engineer. 2016. Verkko-pdf. International Software Testing Qualifications Board. Päivitetty 21.10.2016. Luettu 15.1.2023.
- 3 A Brief History of Java. Verkkoaineisto. Baeldung.com. <<https://www.baeldung.com/java-history>>. Luettu 8.2.2023.
- 4 What is Python? Executive Summary. Verkkoaineisto. Python.org. <<https://www.python.org/doc/essays/blurb/>>. Luettu 8.2.2023.
- 5 Silk4J User Guide: Silk4J. Verkkoaineisto. Microfocus. <<https://www.microfocus.com/documentation/silk-test/200/en/silk4j-help-en/SILKTEST-53E47F79-WELCOME-TOP.html>>. Luettu 2.2.2023.
- 6 JUnit 4. Verkkoaineisto. JUnit.org. <<https://junit.org/junit4/>>. Luettu 6.2.2023.
- 7 Confluence Basics. Verkkoaineisto. Atlassian. <<https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview>>. Luettu 6.2.2023.
- 8 Your personal guide to Jira Software. Verkkoaineisto. Atlassian. <<https://www.atlassian.com/software/jira>>. Luettu 6.2.2023.
- 9 Ramalho, Luciano. 2022. Fluent Python: Clear, Concise and Effective Programming. O'Reilly Media: Sebastopol.
- 10 PEP 8 – Style Guide for Python Code. Verkkoaineisto. Python.org. <<https://peps.python.org/pep-0008/>>. Luettu 8.2.2023.