Tampereen yliopisto

Jukka Pajarinen

WEB-KÄYTTÖLIITTYMÄN HYVÄKSYMISTESTAUKSEN PRIORISOINTI PAINOTETUN VERKON AVULLA

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Joulukuu 2019

TIIVISTELMÄ

Jukka Pajarinen: Web-käyttöliittymän hyväksymistestauksen priorisointi painotetun verkon avulla
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Joulukuu 2019

Avainsanat: hyväksymistestaus, painotettu verkko, priorisointi, jatkuva integraatio, testiautomaatio

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

ABSTRACT

Jukka Pajarinen: Web User Interface Acceptance Testing Prioritization with a Weighted Graph
Master's Thesis
Tampere University
Degree Programme in Information Technology
December 2019

Keywords: acceptance testing, weighted graph, prioritization, continuous integration, test automation

ALKUSANAT

Tampereella, 31. joulukuuta 2019

Jukka Pajarinen

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

$\mathbb{N}$	Luonnolisten lukujen joukko
ATDD	Hyväksymistestausvetoinen kehitys, englanniksi: acceptance test driven development
e2e	Päästä päähän testaus, englanniksi: end-to-end

1 JOHDANTO

Tässä luvussa esitetään lyhyesti työn keskeisin sisältö ja rakenne. Lisäksi pohditaan miksi työ on tarpeellinen ja miksi testitapauksien priorisoiminen on ongelmallista.

2 TUTKIMUSASETELMA

Tässä luvussa esitetään diplomityön taustaa, tutkimuskysymykset, käytetty tutkimusmenetelmä, tutkimuksen rajaus sekä tavoitteet. Tutkimuskysymykset liittyvät vahvasti yhteiseen priorisoinnin teemaan, johon tässä työssä erityisesti paneudutaan. Tutkimus on soveltavaa ja sen tarkoituksena on muodostaa selvitys tutkimusongelman ratkaisemiseksi. Tässä työssä se tarkoittaa erityisesti matemaattisen, toistettavissa olevan menetelmän kehittämistä tutkimusongelman ratkaisemiseksi. Tutkimuskysymyksistä itsessään voi päätellä tutkimuksen tarkoitusta ja tavoitteita, mutta tämä esitetään myös yksityiskohtaisemmin tavoitteet luvussa 2.5. Lisäksi työn lopussa on myös toteutuksellinen osuus, joka on tehty diplomityön asiakasyrityksen tarpeita varten. Toteutuksellisessa osuudessa on paljon muutakin sisältöä, joka on varsinaisen priorisointiteeman ulkopuolella, mutta pysyy kuitenkin työn kokonaiskontekstissa.

2.1 Tausta

Diplomityö tehtiin WordDive nimiselle yritykselle. WordDive on vuonna 2009 perustettu Tampereella toimiva, suomalainen kieltenoppimiseen keskittyvä yritys. WordDivellä oli kirjoitushetkellä kieltenoppimissovellus mobiilialustalle sekä web-alustalle. Tämän diplomityön sisältö koskettaa vain web-alustalla toimivaa sovellusta. Hyväksymistestauksen osalta mobiilisovellukselle oli yrityksessä jo toteutettu testiautomaatio, mutta web-alustalle sitä ei vielä oltu tehty.

Allekirjoittanut aloitti työt kyseisessä yrityksessä 2018 vuoden loppupuolella, jolloin diplomityön aihetta ei vielä ollut. Tarkoituksena oli tuolloin ensin töitä tekemällä tutustua yrityksen web-alustalla toimivaan sovellukseen ja yrityksen ohjelmistotuotantoprosessiin. Diplotyön aihe alkoi muotoutua vasta vuoden 2019 alkupuolella, kun tarvittava tietämys ohjelmistotuotteesta ja prosessista oli saavutettu. Asiakasyrityksessä sai hyvinkin vapaasti löytää itseään kiinnostavan, varsinaisten töiden ohella tehtävän, mutta kaikille osapuolille hyödyllisen aiheen. Aiheen löytämisen taustalla olivat hyvinkin konkreettiset tarpeet, jotka ohjelmistotuotannon työssä tulivat esille.

Uusien ominaisuuksien ja koodimuutoksien tekemisen yhteydessä oli jatkuvasti tarve huolelliselle testaamiselle ja erityisesti asiakkaan näkökulmasta tärkeimpien sovelluksen ominaisuuksien toiminnan varmistamiselle. Tämä sai diplomityön aiheen suuntautumaan testiautomaatioon ja erityisesti hyväksymistestaukseen. Lisäksi yrityksessä oli jo toteutettuna päivittäisessä käytössä oleva hyväksymistestaus mobiilialustalle, joka auttoi hah-

mottamaan web-sovelluksen testiautomaation integroimista osaksi yrityksen ohjelmistotuotantoprosessia. Mobiilialustalle tehtyä hyväksymistestausta varten oli yrityksessä jo valittu tietyt hyväksi todetut sovelluskehykset ja työkalut testiautomaatiota varten, joten tässä työssä ei enää ollut tarvetta evaluoida eri työkaluja tarvittavan testiautomaation toteuttamiseksi. Web-sovelluksen testiautomaatio toteutettiin käyttäen pääpiirteittäin samoja sovelluskehyksiä ja työkaluja 7.4 kuin mobiilisovellukselle. Tästä syystä diplomityön aihetta ja tutkimusongelmaa lähdettiin etsimään muualta.

Tutkimusongelmaan miettiin aihioita etenkin alkuvaiheessa polkutestauksen ja ohjelmistotestaukseen liittyvien heuristiikkojen osalta. Polkutestaus oli yhtenä vaihtoehtona, mutta siitä luovuttiin, koska sen todettiin olevan paremmin soveltuvampi hyväksymistestausta alemmille testauksen tasoille. Heuristiikkojen hyödyntämistä mietittiin kahteen eri ongelmaan; testitapauksien muodostamiseen sekä kriittisten testitapauksien määrittämiseen. Näistä kahdesta, tässä työssä sivutaan heuristiikkojen käyttämistä testitapauksien muodostamiseen luvussa 4.4, mutta sitä ei käsitellä tutkimusongelmana.

Lopullisen tutkimusongelman löytämiseen vaikuttivat erityisesti konkreettiset tarpeet, jotka tulivat esiin vasta web-sovelluksen testiautomaation suunnitteluvaiheessa. Hyväksymistestauksen testiautomaatiota varten oli ensin määritettävä mitä testauksen kohteena olevasta sovelluksesta tulisi testata. Testiautomaation rakentamiseen allokoitavia resursseja oli rajallinen määrä, jonka lisäksi testikattavuuden suppeus sekä ylikattavuus nähtiin selkeänä ongelmana. Tämä ongelma voidaan esittää yksinkertaisemmin testitapauksien priorisointiongelmana, joka myös lopulta muotoutui työn tutkimusongelmaksi. Priorisointiongelman valitsemiseen ratkaisevasti johtavia asioita olivat kaksi allekirjoittaneen oivallusta aiheesta. Ensimmäiseksi hyväksymistestattavaa web-sovellusta keksittiin ajatella käyttöliittymän näkymä ja siirtymätasolla matemaattisena prioriteetein painotettuna verkkona. Toiseksi oivallukseksi keksittiin käyttää lyhimmän polun ongelmaan kehitettyjä algoritmeja prioriteetein painotetun verkon karsimiseen, jolloin alhaisen prioriteetin solmuja saatiin leikattua pois. Nämä oivallukset vaikuttivat lopulta varsinaisen tutkimusongelman eli testitapauksien priorisointiongelman valitsemiseen, koska ne loivat järkevän ja mielenkiintoisen pohjan tutkimusongelmaan vastaamiselle. Kokonaisuutena diplomityön aihe saatiin muodostettua sellaiseksi, että se esittää yleishyödyllisen menetelmän tutkimusongelman ratkaisemiseen sekä sitä hyödyntävän toteutuksen suunnittelemisen ja rakentamisen asiakasyritykselle.

2.2 Tutkimuskysymykset

Tutkimuksen tarkoituksena on pohjimmiltaan tarkoitus löytää ja kehittää toistettavissa oleva menetelmä hyväksymistestauksen testitapauksien priorisoimiseen. Testitapauksien laatimisen yleisenä ongelmanakohtana on erityisesti niiden priorisointi, joka usein johtaa liian suppean tai ylikattavan testiautomaation rakentamiseen. Tutkimuskysymykset on laadittu siten, että niihin vastaaminen antaa ratkaisun tähän edellä mainittuun testiautomaation ongelmaan.

Työlle asetettiin seuraavat tutkimuskysymykset:

- T1: *Miten painotettua verkkoa voidaan käyttää testitapauksien priorisoimiseen?*
- T2: *Mitkä muuttujat vaikuttavat web-käyttöliittymän hyväksymistestauksen testitapauksien priorisointiin?*
- T3: *Kuinka prioriteetein painotetusta verkosta valitaan toteutettavat testitapaukset?*
- T4: *Miten painotetun verkon avulla tehty priorisointi liitetään yhteen jatkuvan integraation ja testiautomaation kanssa?*

2.3 Tutkimusmenetelmä

Tutkimuskysymyksiin vastaamiseksi työn tutkimusmenetelmäksi valittiin tietotekniikan diplomitöissä yleisesti käytetty Design Science menetelmä. Menetelmän tarkoituksena on tuottaa teknologiaa hyödyntävä ratkaisu tutkimusongelmaan vastaamiseen. Design Science menetelmää käyttäessä pyritään tutkimaan uusia ratkaisumalleja ratkaisemattomiin ongelmiin tai kehittämään parempia ratkaisumalleja jo aiemmin ratkaistujen ongelmien tilalle. Tietotekniikan tutkimuksessa on aikojen saatossa kehitetty uusia tai parempia tietokonearkkitehtuureja, ohjelmointikieliä, algoritmeja, tietorakenteita ja tiedonhallintajärjestelmiä. Näiden osalta yhteistä on, että niissä on monesti käytetty usein jopa tiedostamatta Design Science menetelmää.

Design Science menetelmän vaiheet ovat yksinkertaistetusti seuraavat:

1. Tutkimusongelman tai kysymyksen määrittäminen
2. Ratkaisuvaihtoehtojen kartoittaminen ja perusteltu valitseminen
3. Ratkaisun toteuttaminen tai prototypointi
4. Ratkaisun toimivuuden evaluointi

Tutkimuksen tarkoituksena oli muodostaa uudenlainen toistettavissa oleva menetelmä tutkimuksen kohteena olevan ongelman ratkaisemiseksi. Tutkimusidean hahmottelemisen ja ratkaisua kaipaavan ongelman identifoinnin jälkeen, valittua tutkimusmenetelmää käyttäen ensin määriteltiin tutkimuskysymykset 2.2.

Seuraavaksi kartoitettiin ratkaisuvaihtoehto tutkimuskysymyksiin ja työn yleiseen priorisoinnin teemaan vastaamiseksi ja esitetään perustelut matemaattiseen toistettavissa olevaan ratkaisumenetelmään päätymiseen. Tutkimusta varten kerättiin teoreettista aineistoa, jonka tarkoituksena oli tukea menetelmän kehittämistä ja jonka avulla pyrittiin luomaan lukijalle mahdollisimman vahva teoreettinen pohja tutkimuskysymyksiin vastaavan ratkaisumenetelmän ymmärtämiseksi. Asiakasyrityksen ohjelmistotuotetta ja ohjelmistotuotantoprosessi huomioiden toteutettiin myös kokonaisratkaisu jossa toteutettiin testiautomaatio hyödyntäen kehitettyä priorisoinnin ratkaisumenetelmää.

Lopuksi vielä evaluoitiin menetelmän eli ratkaisun ja sitä hyödyntävän toteutuksen toimivuus käytännössä ja esitetään yhteenveto tutkimuksesta.

2.4 Tutkimuksen rajaus

Ohjelmistotestauksen tasojen osalta tutkimus rajoittuu hyväksymistestaukseen. Tämä rajaus pohjautuu ohjelmistotuotannon työssä konkreettisesti havaittuun tarpeeseen sekä yhdenmukaisen testiautomaation toteuttamiseen mobiili- ja web-sovelluksille asiakasyrityksessä.

Testitapauksien osalta on olemassa lukuisia eri testausalustoja, joita hyödyntäen testitapauksia voidaan toteuttaa. Tässä työssä testitapauksien toteuttaminen rajataan tietylle ennalta määräytyneelle Robot Framework alustalle. Tämä rajaus pohjautuu asiakasyrityksessä jo aiemmin valittuihin testauksen sovelluskehyksiin ja työkaluihin. Lisäksi Robot Framework on alustana yleisesti käytetty etenkin hyväksymistestauksen toteuttamiseen. Robot Framework on myös erityisen hyvin soveltuva tilanteissa, joissa halutaan koodia korkeampaa abstraktiotasoa.

Jatkuvan integroinnin osalta tutkimus rajoittuu perusteisiin ja tutkimuksen painoarvo pidetään testitapauksien toteuttamisessa ja niiden priorisoinnissa. Jatkuva integrointi on kuitenkin asiakasyrityksessä tärkeä osa testiautomaation ja jatkuvan käyttöönoton toteutuksessa. Jatkuvan integroinnin osalta ei tässä työssä esitetä muuta kuin testiautomaation toteutusosaan kokonaisuutena erityisesti liittyvät käsitteet ja ratkaisu. Tämä rajaus pohjautuu tutkimusongelman tarkempaan spesifioimiseen ja tutkimuksen kokonaislaajuuden hallitsemiseen.

Verkkoteorian osalta tutkimus rajoittuu perusteisiin ja painotettua verkkoa sekä kehitettyä menetelmää tukeviin käsitteisiin. Tämä rajaus pohjautuu työn kohdentamiseen ohjelmistotuotantoon ja diplomityön kirjoitusvaiheessa saatuun ohjauspalautteeseen, jossa matematiikan osuus oli kasvanut liiallisen suureksi.

Priorisointiin vaikuttavien muuttujien osalta tutkimus rajoittuu muuttujien kartoittamiseen, mutta niiden määrittäminen jätetään työn ulkopuolelle. Tämä tarkoittaa käytännössä sitä, että jokainen menetelmää hyödyntävä taho hankkii itse varsinaiset numeeriset arvot muuttujille. Esimerkiksi liiketoiminnallisen vision arvo on yksinomaan menetelmää käyttävän tahon päätettävissä.

Painotetun verkon lyhimmän polun etsimiseen on olemassa lukuisa määrä erilaisia algoritmeja, mutta tässä työssä hyödynnetään vain perinteistä Djikstran algoritmia. Tämä rajaus pohjautuu työssä kehitetyn priorisointimenetelmän käyttämisen perimmäiseen tarkoitukseen, jossa ei algoritmin tehokkuudella tai lisäominaisuuksilla ole suurta merkitystä. Lisäksi Dijkstran algoritmi on selkeä, paljon tutkittu ja käytetty ratkaisu lyhimmän polun etsimiseen, joka tekee siitä ihanteellisen tässä työssä muodostettavaan painotettuun verkkoon tehtävien leikkauksien identifoimista ajatellen.

2.5 Tavoitteet

Tutkimuksen tavoitteena on löytää toistettavissa oleva menetelmä hyväksymistestauksessa tarvittavien testitapauksien priorisoimiseen.

Lisäksi tavoitteena on pystyä todentamaan kehitetyn menetelmän toimivuus käytännössä asiakasyritykselle tehtyä toteutusta evaluoimalla.

3 TESTIAUTOMAATIO

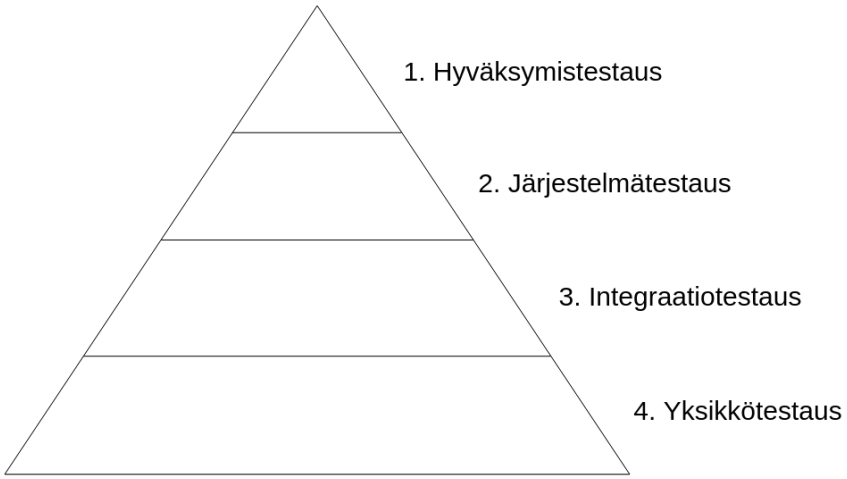
Tässä luvussa esitetään perusteet ja tarvittavat tiedot testiautomaatiosta, jotka liittyvät työn laajempaan teoreettiseen kehykseen. Testiautomaation perusteiden ymmärtämistä tarvitaan työn myöhemmässä vaiheessa, jossa esitetään varsinainen testitapauksien priorisointi painotetun verkon avulla.

3.1 Testiautomaation tarkoitus

Testiautomaation tarkoitus on pohjimmiltaan mahdollistaa ohjelmistotuotteen jatkuva ja vaivaton laadunvarmistus, nyt ja tulevaisuudessa. Ohjelmistojen testaamisella itsessään pyritään löytämään ohjelmistotuotteesta virheitä, anomalioita ja varmistamaan että se toimii asetettujen vaatimusten mukaisesti. Testauksen automatisoiminen vapauttaa henkilöresursseja manuaalisesta testaamisesta muihin tuotantotehtäviin sekä parantaa toistuvien testien luotettavuutta poistamalla manuaalisessa testauksessa tapahtuvat inhimillisen virheet. Laadunvarmistuksen osalta testiautomaatiolla voidaan kattaa erilaisia ohjelmistotuotteen laadullisia ominaisuuksia, kuten toiminnallisuus, luotettavuus, käytettävyys, tehokkuus, ylläpidettävyys ja siirrettävyys (ISO:9126-1 2001).

3.2 Testauksen tasot

Testauksen tasoja on useita ja usein ohjelmistojen kattavaan testaamiseen on suositeltavaa käyttää ohjelmistotuotantoprosessissa eri tasojen yhdistelmää. Ohjelmistotestaus usein jaotellaan kolmeen erilaiseen menetelmään, jotka myös vaikuttavat eri testauksen tasojen käytettävyyteen. Erilaisia menetelmiä ovat mustalaatikkotestaus, harmaalaatikkotestaus ja valkolaatikkotestaus, jotka eroavat toisistaan yleisesti ottaen siinä, otetaanko tieto ohjelmistotuotteen sisäisestä toteutuksesta mukaan testaamiseen. Testauksen tasot voidaan jakaa neljään eri tasoon, jotka usein kuvataan pyramidin muotoon. Pyramidimuodossa alimpana kuvataan yksikkötestaus, joka luo vahvan pohjan kokonaisvaltaiselle testaamiselle. Noustessa pyramidissa ylöspäin testattavana olevan kohteen laajuus kasvaa. Ylimpänä pyramidissa on hyväksymistestaus, joka on tarkoituksellista toteuttaa vaatimusmäärittelyn täyttävää valmista järjestelmää vastaan.



Kuva 3.1. *Testauksen tasot pyramidin muodossa*

3.2.1 Yksikkötestaus

Yksikkötestauksen ajatuksena on testata ohjelmistotuotteen lähdekoodista löytyviä yksiköitä, kuten luokkia, funktioita tai moduleita. Yksikkötestaus toteutetaan ohjelmiston toteuttavia pienempiä yksikköjä varten. Yksikkötestaus eroaa muista testauksen tasoista siinä, että sen suorittavat ohjelmistokehittäjät tai muut ohjelmiston lähdekoodiin perehtyneet henkilöt. Yksikkötestauksella pyritään varmistamaan, että ohjelmiston pienimmät osat toimivat tarkoituksenmukaisella tavalla. Yksikkötestausta hyödynnetään usein myös ketterien menetelmien aihepiirissä, jossa ohjelmistotuotantoa toteutetaan niin sanotulla testivetoisella kehityksellä. Testivetoisessa kehityksessä ohjelmistokehittäjät laativat ensisijaisesti yksiköiden yksikkötestit ennen niiden toteuttamisen aloittamista.

3.2.2 Integraatiotestaus

Integraatiotestauksen ajatuksena on testata ohjelmistotuotteen toteuttavien eri komponenttien yhteentoimivuutta niiden rajapintojen osalta. Integraatiotestaus toteutetaan ohjelmiston suunnitelmaa ja mallia vastaan. Integraatiotestauksen onnistuminen luo perustan ohjelmiston toimimiseen ja koostamiseen kokonaisena, eri komponenteista koostuvana järjestelmänä. Integraatiotestauksen yhteydessä puhutaan usein myös niin sanotusta savutestauksesta, jonka tarkoituksena on koostaa päivittäinen koontiversio ohjelmistosta ja testata sen kriittisten komponenttien yhteentoimivuus.

3.2.3 Järjestelmätestaus

Järjestelmätestauksen ajatuksena on testata kokonaista ja toimivaa järjestelmää, yhtenä suurena yksikkönä. Järjestelmätestaus toteutetaan usein eräänlaisena tulikokeena, erityisesti ohjelmiston vaatimuksia vastaan. Järjestelmätestaukseen liittyy laajasti erilai-

sia testattavia laadullisia ominaisuuksia, kuten esimerkiksi toiminnallisuus, luotettavuus, käytettävyys, tietotruva, tehokkuus ja siirrettävyys.

3.2.4 Hyväksymistestaus

Hyväksymistestauksen ajatuksena on varmistaa toteutettavan ohjelmiston vaatimusten toimivuus erityisesti käytännön tilanteissa. Hyväksymistestaus toteutaan ohjelmiston toimintoja kuvaavaa vaatimusmäärittelyä vastaan. Hyväksymistestaus on tarkoituksenmukaista laatia sellaiseen muotoon joka testaa lopullisten käyttäjien toimintaa vastaavia käyttötilanteita. Samassa asiayhteydessä puhutaan usein myös niin sanotusta päästä päähän testauksesta (englanniksi: e2e, end-to-end). Testiautomaatio on erittäin hyödyllinen hyväksymistestausen osalla, koska sillä voidaan automatisoida ohjelmiston validointi ja hyväksyminen sekä estää puutteellisesti toimivan ohjelmiston julkaiseminen.

3.3 Jatkuva integrointi

3.4 Testausvetoinen kehitys

4 HYVÄKSYMISTESTAUS

Tässä luvussa esitetään perusteet ja tarvittavat tiedot hyväksymistestauksesta..

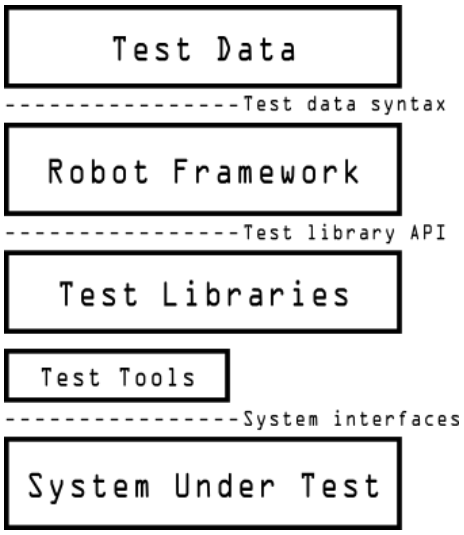
4.1 Hyväksymistestauksen tarkoitus

4.2 Hyväksymistestausvetoinen kehitys

Hyväksymistestausvetoisen kehityksen (englanniksi: ATDD, acceptance test driven development) tarkoituksena, kuten testausvetoisessakin kehityksessä on toteuttaa ohjelmistotuotannollinen prosessi testaaminen edellä. Tämä tarkoittaa käytännössä, sitä että ohjelmistokehittäjät laativat ohjelmiston vaatimusten ja suunnitelman mukaisia iteratiivisesti suoritettavia testitapauksia, ennen niitä käyttävän varsinaisen ohjelmakoodin toteuttamista. Hyväksymistestausvetoisessa kehityksessä luodaan ennen toteutusta tarvittavat ohjelmiston asiakasvaatimuksia palvelevat hyväksymistestit, joiden ohjelmiston on tarkoitus läpäistä. Tarvittavat ohjelmiston hyväksymistestit suoritetaan iteratiivisesti ohjelmistokehitysprosessin aikana, ja se tarkoittaa käytännössä jatkuvan integraation ottamista käyttöön ohjelmistokehityksessä. Hyväksysmistestausvetoinen kehitys on erittäin hyödyllinen ohjelmistokehityksessä käytetty menetelmä, sillä kehitysvaiheessa on aina tarkasti tiedossa vastaako ohjelmiston silloinen tila asiakasvaatimuksia ja kuinka hyvin. Hyväksymistestausvetoisessa kehityksessä toteutettavat hyväksymistestit testaavat ohjelmistoa kokonaisena järjestelmänä tarkoituksenamukaisesti, siten kuten se esiintyy loppukäyttäjille.

4.3 Robot Framework

Robot framework on geneerinen avoimen lähdekoodin testausalusta hyväksymistestaukseen, hyväksymistestausvetoiseen kehitykseen ja robotisten prosessien automaatioon.



Kuva 4.1. *Robot framework alustan arkkitehtuuri*

4.4 Testitapauksien määrittäminen

Yleisiä testitapauksien määrittämiseen käytettäviä heuristiikkoja ovat muun muassa:

4.5 Web-käyttöliittymien erityispiirteet

Web-käyttöliittymillä on myös omia erityispiirteitä, jotka vaikuttavat testitapauksien laatimiseen.

- Navigointi
- Syötteet
- Syntaksi
- Selainasetukset

4.5.1 Selenium

4.5.2 Moni-selaimellinen testaus

4.6 Priorisointiongelma

Testitapauksien priorisointi on kustannussyistä tai resurssien optimoinnin kannalta erittäin tärkeää. Ohjelmistotestauksessa on hyvä tiedostaa, että ohjelmistotuotetta ei usein voida testata täydellisesti, joka nostaa esiin tarpeen tärkeimpien testitapauksien löytämisestä. Testitapauksia voidaan priorisoida monella tavalla, joihin tämä diplomityö tuo yhden uudenlaisen painottua verkkoa hyödyntävän lähestymistavan.

- Painotetun verkon hyödyntäminen
- Muut priorisointitavat

5 VERKKOTEORIA

Tässä luvussa käsitellään työhön keskeisesti kuuluvan verkkoteorian perusteet käydää huolellisesti läpi erityisesti työssä käytettävät osat. Työssä sovelletaan erityisesti verkkoteorian painotettua verkkoa sekä verkkoteoriassa esiintyvän lyhimmän polun ongelmaan kehitettyjä ratkaisualgoritmeja. Verkkoteoria itsessään on osa diskeettiä matematiikkaa.

5.1 Matemaattisten verkkojen tarkoitus

Matemaattisten verkkojen tarkoituksena on mallintaan parittaisia riippuvuuksia verkkomaisessa objektijoukossa. Verkkoteoriassa peruskäsitteitä ovat itse *verkko* eli *graafi*, joka muodostuu *solmuista* ja niiden välisiä riippuvuuksia esittävistä *kaarista* tai *nuolista*. Verkkoteorialla on lukuisia käytännön sovellutuksia. Verkkoteoriaa sovelletaan muun muassa tietokonetieteissä, kielitieteissä, fysiikan ja kemian sovellutuksissa, sosiaalisissa tieteissä ja biologiassa. Alun perin verkkoteoria katsotaan syntyneen 1700-luvulla esiintyneestä niin sanotusta Königsbergin siltaongelmasta, johon Leonhard Euler esitti todistuksensa.

5.2 Perusmerkinnät ja käsitteet

Verkkoteoriassa käytetään seuraavia perusmerkintöjä:

- $V := \{v_1, v_2, v_3\}$ Solmujoukko joka sisältää *solmut* v_1, v_2 ja v_3.
- $E := \{e_1, e_2, e_3\}$ Kaarijoukko joka sisältää *kaaret* e_1, e_2 ja e_3.
- $\phi(e_1) := \langle v_1, v_2 \rangle$ Kaariparin v_1 ja v_2 yhdistävän *kaaren* e_1 kuvaaja.

Verkkojen solmujen välisiä yhteyksiä, eli kaaria esitetään usein myös yhteys- tai painomatriisina.

$$M_G = (a_{ij})_{3\times 3} = \begin{array}{c} G \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix} \end{array}$$

Verkkoteoriassa käytetään myös muun muassa seuraavia käsitteitä:

- Solmun asteluku, $d_G(x)$, eli solmuun liittyvien *kaarten* määrä.
- Surkastunut verkko, $E = \varnothing$, eli verkko jossa ei ole *kaaria*.
- Täydellinen verkko, eli jokaista solmuparia $v_1 \neq v_2$ yhdistää ainakin yksi *kaari*.

- Aliverkko, $G_2 \subset G_1$, eli *verkko* G_2 joka koostuu osasta *verkon* G_1 *solmuja* ja *kaaria*.
- Verkon komplementti, G', eli sellainen *verkko*, jossa on kaikki ne *kaaret* joita *verkossa* G ei esiinny.
- Verkon yhtenäisyys, $v_1 \neq v_2, v_1 \rightarrow v_2$, eli jokaiselle solmuparille $v_1 \neq v_2$ on olemassa niitä yhdistävä *kaari*.
- Polku, $\{v_0, v_1, ..., v_n\}, v_0 \rightarrow v_n$, eli *suunnattu solmujono* jota pitkin voidaan kulkea *solmusta* v_0 *solmuun* v_n.
- Eristetty solmu, $d_G(v_1) = 0$, eli *solmu* jonka *asteluku* on nolla.
- Silta, $v_1 \rightarrow v_2, d_G(v_1) = 1 \vee d_G(v_2) = 1$, eli *kaari* johon yhdistyvän *solmun asteluku* on yksi ja jonka poistaminen epäyhteinäistää *verkon*.

5.3 Painotettu verkko

- $\alpha := V(G), E(G) \rightarrow \mathbb{N}$ Painofunktion yleinen kuvaus.

5.4 Verkon leikkaaminen

5.5 Lyhimmän polun ongelma

- $d_G^{\alpha}(v_1, v_2) = min\{\alpha(P)|P : v_1 \rightarrow v_2|v_1, v_2 \in V(G)\}$ Lyhimmän polun ongelma.

5.5.1 Dijkstran algoritmi

6 PRIORISOINTI PAINOTETUN VERKON AVULLA

Tässä luvussa esitetään tutkimuksen tärkein sisältö ja kokonaisuutena vastaus tutkimuskysymykseen *T1*, eli toistettavissa oleva menetelmä testitapauksien priorisoimiseen. Priorisointiin vaikuttavat muuttujat luvussa 6.1 esitetään myös suora vastaus tutkimuskysymykseen *T2*. Lisäksi painofunktiot 6.2 ja verkon karsiminen 6.4 esittää vastaukset tutkimuskysymykseen *T3*. Testitapauksien muodostaminen verkosta 6.6 antaa osittaisen vastauksen myös tutkimuskysymykseen *T4*.

Priorisointia varten esitetään harkintaa käyttäen lähdeaineistosta suodatetut priorisointiin vaikuttavat muuttujat, painofunktio, testitapauksien näkymäperusteinen koostaminen ja painotetun verkon laatiminen. Lisäksi menetelmää käyttäen tuotetun painotetun verkon sisältämää informaatiota käytetään prioriteeteiltaan tärkeiden polkujen löytämiseen ja testikattavuuden arviointiin.

6.1 Priorisointiin vaikuttavat muuttujat

- Liiketoiminnallinen arvo
- Liiketoiminnallinen visio
- Käyttäjäpalaute
- Projektin muuttumisen volatiliteetti
- Kehittämisen kompleksisuus
- Vaatimusten taipumus virheellisyyteen

6.2 Painofunktiot priorisointiin

Painofunktion yleinen kuvaus verkossa G, solmuille V ja kaarille E.

$$\alpha := V(G), E(G) \rightarrow \mathbb{N}$$

Painofunktio yksittäiselle solmulle v, eli näkymälle.

$$\alpha(v) = value + vision \pm feedback - volatility - complexity - errorness$$

Painofunktio yksittäiselle kaarelle e, eli siirtymälle.

$$\beta(e) = value - volatility - complexity - errorness$$

Painofunktion polulle P solmusta v_1 solmuun v_2.

$$\gamma(P) = \sum_{v \in P} \alpha(v) + \sum_{e \in P} \beta(e)$$

6.3 Verkon rakentaminen

$$M_G = (a_{ij})_{3\times 3} = \begin{array}{c} G \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{pmatrix} \end{array}$$

6.4 Verkon karsiminen

Painotetun verkon karsiminen eli leikkaaminen on prioriteeillä painotetun verkon tärkeä ominaisuus. Verkkoteorian soveltaminen prioriteettien avulla painotettuun verkkoon on erityisen hyödyllistä, kun verkon kaarissa korkea paino tarkoittaa suurta prioriteettia. Tällaisessa tapauksessa on mahdollista soveltaa lyhimmän polun ongelman ratkaisemiseen kehitettyjä algoritmeja, jolloin ne toimivat etsien alhaisimman prioriteetin polkuja. Lyhimmän polun etsimiseen on tarkoituksenmukaista valita aina aloitus ja lopetuspisteet, joiden välille lyhin polku verkossa voidaan etsiä. Prioriteetein painotetun verkon karsimistarkoitukseen olisi järkevää valita sellaiset aloitus- ja lopetuspisteet, joiden välillä ei näyttäisi olevan korkean prioriteetin solmuja. Voidaan kuitenkin menetellä myös siten, että valitaan aloitus- ja lopetuspisteeksi sellaiset solmut, jotka ovat painoltaan verkon alhaisimmat $v_1 = min(V)$ ja $v_2 = min(V \setminus \{v_1\})$ ja esimerkiksi verrata niiden lyhimmän polun kokonaisprioriteettia muuhun verkkoon.

6.4.1 Dijkstran algoritmin soveltaminen

- Pienimmän prioriteetin solmuparin etsiminen, eli $v_1 = min\{\alpha(V)\}$ ja $v_2 = min\{\alpha(V \setminus \{v_1\}\}$.
- Dijkstran algoritmin käyttö lyhimmän (prioriteetiltaan pienimmän) polun löytämiseen, eli $s = min(\gamma(P)), P \in G$.
- Leikkauksien tekeminen ja toistaminen n-kertaa.

6.4.2 Leikkauksien tekeminen

- Poistetaan yhden yksittäiseen solmuun johtavat sillat, jossa $\alpha(v) < aivanliianalhainen$.
- Poistetaan kaikki yksittäiset eristetyt solmut, eli asteluku on nolla $d_G(X) = 0$.
- Poistetaan Dijkstran lyhimmän polun kaaret, jossa $\beta(e) < liianalhainen$.

6.5 Verkon ja testitapauksien yhteys

6.6 Testitapauksien muodostaminen verkosta

7 TESTAUKSEN SUUNNITTELU JA TOTEUTUS

Tässä luvussa esitetään työn perusteella tehty esimerkkitoteutus sekä käytännön sovelluskehyksen ja työkalut. Tämän luvun tarkoituksena on todistaa menetelmän toimivuus oikeassa ohjelmistotuotannon ympäristössä toteuttaen samalla testiautomaatio asiakasyritykselle.

7.1 Käyttöliittymän näkymät ja siirtymät

7.2 Painotetun verkon rakentaminen

7.3 Painotetun verkon käyttö priorisointiin

7.4 Sovelluskehykset ja työkalut

7.4.1 Docker

7.4.2 GoCD

7.4.3 Robot Framework

7.4.4 Selenium

7.5 Testitapauksien toteuttaminen

7.6 Testitapauksien suorittaminen

Tässä kappaleessa esitetään vastausta tutkimuskysymykseen *T4*, keskittyen jatkuvaan integroinnin ja testitapauksien priorisoinnin yhteyteen.

8 TULOSTEN TARKASTELU JA ARVIOINTI

Tässä kappaleessa esitetään yhteenveto tutkimuksen tuloksista ja muun muassa pohditaan kuinka hyvin toistettavissa oleva kyseinen kehitetty menetelmä on.

8.1 Tutkimuksen konkreettiset tulokset

8.2 Menetelmän evaluointi

8.3 Toteutuksen evaluointi

8.4 Jatkokehitysehdotukset

9 YHTEENVETO

Tässä kappaleessa esitetään yhteenveto tehdystä työstä.

LÄHTEET

ISO:9126-1 (2001). *ISO/IEC 9126-1:2001*. en. URL: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/02/27/22749.html.

A ESIMERKKILIITE