Tampereen ammattikorkeakoulu

Automaatiotestauksen rajoitukset asiakasprojekteissa

Nina Karjalainen

OPINNÄYTETYÖ
Joulukuu 2019

Tietojärjestelmäosaamisen koulutusohjelma (ylempi AMK)

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaamisen koulutusohjelma (ylempi AMK)

KARJALAINEN, NINA:
Automaatiotestauksen rajoitukset asiakasprojekteissa

Opinnäytetyö 38 sivua, joista liitteitä 1 sivua
Marraskuu 2019

Tämän opinnäytetyön aiheena oli automaatiotestauksen rajoitukset asiakasprojekteissa. Tutkimuksen tavoitteena oli selvittää ja kerätä tietoa, mitkä ovat ne asiat, jotka tuovat ongelmia automaatiotestauksen käyttöön asiakasprojekteissa.

Tutkimuksessa etsittiin vastausta siihen, että onko automaation käyttö ohjelmistotestauksessa kannattavaa sekä miten automaatiota kannattaa parhaiten hyödyntää ohjelmistotestauksessa.

Alussa perehdyttiin hieman itse testaukseen ja sen jälkeen tutkittiin, mitä kaikkia kuluja yleisesti ottaen testaukseen kuuluu.

Kokemuksia automaation sisällyttämisestä projekteihin kerättiin projektipäälliköitä haastattelemalla. Kaikilla heillä on erilaisia projekteja, joissa on erilaisia käytäntöjä. Vaikka projektit olisivatkin keskenään samantyylisiä, se ei silti tarkoita automaattisesti sitä, että niissä testausta olisi automatisoitu.

Työssä tutkittiin myös, miten onnistuneesti testauksen automatisoiminen onnistui niissä projekteissa, joissa sitä haluttiin käyttää. Pohdinnassa mietitään, mikä testauksen merkitys on tänä päivänä yrityksille ja mikä sen merkitys tulee tulevaisuudessa olemaan.

Jatkotavoitteena on luoda tai kehittää menetelmiä, joilla voitaisiin hahmottaa projektin vaiheet ja kohdat, mitä voitaisiin automaatiolla auttaa ja kehittää.

Asiasanat: testaus, automaatiotestaus, asiakasprojekti

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Information System Competence

NINA KARJALAINEN:
Limitations in Using Automation Testing in Customer Projects

Master's thesis 38 pages, appendices 1 page
November 2019

The subject of this thesis is the limitations of automation testing in customer projects. The aim of the study is to find out and gather information on the issues, which cause problems in the use of automation testing in customer projects.

The research questions are whether it is worthwhile to use automation in software testing, and how to best use automation in software testing.

The thesis examines testing itself, and then focuses at what the costs for testing in general are.

Experiences of incorporating automation in projects were gathered by interviewing project manager, who all have different projects with varying practices. Even though the projects are similar in style, this does not automatically mean that testing is automated in all of them.

The thesis also investigates how the automation testing succeeded in projects where it was intended to be used. The reflection focuses on the importance of testing for businesses today, and what role it will play in the future.

The aim is to create or develop methods that will, in the future, outline the phases and points of a project that could be assisted and developed by automation.

Key words: testing, testing automation, customer project

SISÄLLYS

1 JOHDANTO

Työskentelen isossa kansainvälisessä konsulttitalossa ohjelmistotestaajana. Meidän yksikköömme Tampereella kuuluu noin 50 henkilöä. Olemme konsulttiyksikkö, joka tekee asiakasprojekteja asiakkaiden kanssa heidän vaatimusmäärittelyidensä mukaan. Testaajat tekevät useimpia projekteja saman aikaisesti.

Tänä päivänä suunta tuntuu olevan se, että testiautomaatiota haluttaisiin käyttää mahdollisimman laajasti ja manuaalista testausta väheksytään, vaikka usein manuaalinen testaus on järkevä vaihtoehto laadun kannalta. Testaaminen haluttaisiin usein automatisoida, vaikka ei edes ole kovin selvää käsitystä siitä, miten ja kuinka se tulisi toteuttaa.

Ohjelmistoprojektit ovat perinteisesti usein vaativia johdettavia, joiden aikataulua ja resurssien tarvetta voi olla hankala riittävän tarkalla tasolla hahmottaa projektin alkuvaiheessa. Testausvaihe on kuitenkin lopputuloksen laadun kannalta erittäin kriittinen osa ohjelmistokehitystä, ja näin ollen sille tulisikin varata tarpeeksi resursseja riittävän aikaisessa vaiheessa. Tässä työssä on pyritty selvittämään, kuinka mittavassa määrin testausvaiheen tehokkuuteen voidaan vaikuttaa automatisoimalla testausta.

2 MITÄ TESTAUS ON?

Ohjelmistotestaus on yksi osa ohjelmistotuotantoa. Siinä pyritään varmistamaan, että tehty tuote vastaa toiminnoiltaan, ominaisuuksiltaan sekä ulkoasultaan aiemmin niille tehtyjä määrittelyjä. (Kasurinen 2013, 10) Testaus on useasti se osa-alue, joka jätetään viimeiseksi välttämättömäksi pahaksi. Sille yleensä määritellään jo alun perinkin liian vähän aikaa ja mikäli jossain muualla tulee ongelmia, se on testaamisen aikataulusta pois.

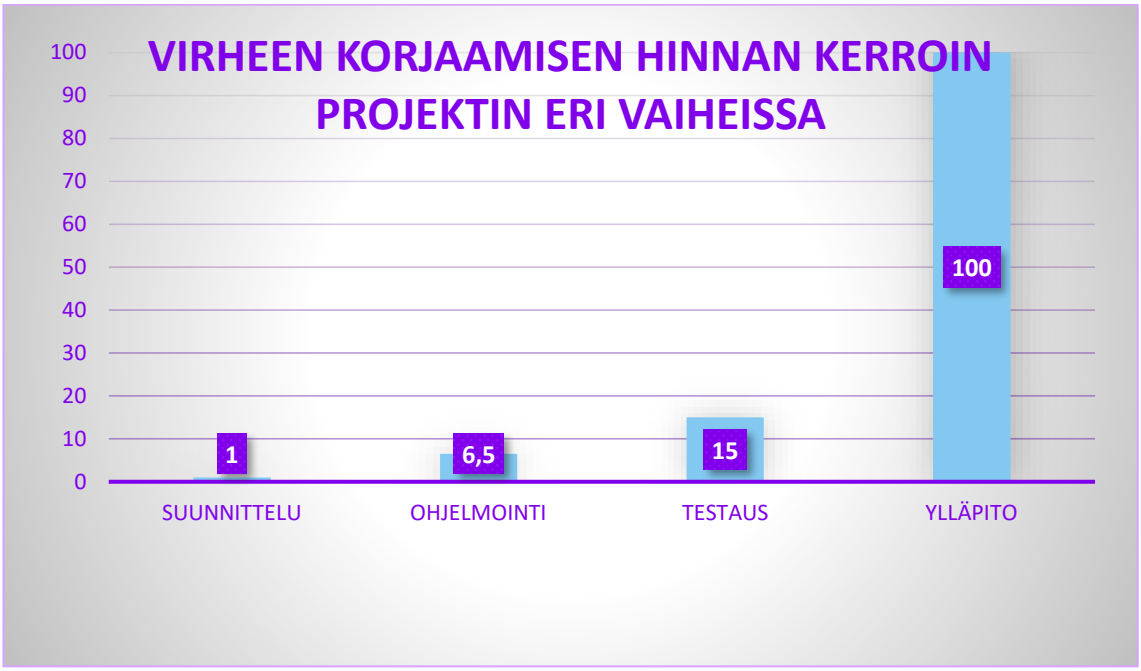
Ohjelmiston laadukas testaus vaatii kuitenkin paljon aikaa ja resursseja, ja siten on suunniteltava huolellisesti. Koska testaus vie paljon aikaa ja resursseja, suoritettavat testitapaukset kannattaakin suunnitella siten, että ne kattavat mahdollisimman paljon ohjelmiston toimintoja ja ominaisuuksia. Suunnitelmallisessa testauksella mahdolliset ohjelmavirheet voidaan löytää ja korjata ajoissa, ja siten välttyä toimittamasta asiakkaalle asti virheellistä koodia. Virheiden etsimisen ohella testauksen tärkeä tehtävä on myös varmistaa, että ohjelmisto toimii halutulla tavalla, ja määritellyt ominaisuudet ja toiminnot löytyvät ohjelmistosta. Ohjelmistotestaus ei siten ole pelkkää virheiden etsimistä (Lehto T. 2005).

Hyvin suunniteltu ja riittävän kattava testaus pystyy myös varmistamaan, että ohjelmisto toimii halutulla tasolla tietoturvallisesti. Tietoturvatestaus on tärkeä osa-alue ohjelmiston testausta, ja se usein ohjelmistosta riippuen suoritetaan erikseen ja usein jopa ulkopuolisen asiantuntijan suorittamana. Tietoturvatestausta on monen tasoista, ja projektiorganisaation onkin osattava valita juuri kyseiseen ohjelmistoprojektiin sopivan tasoinen tavoite tietoturvatestaukselle (Lehto T. 2005).

Testaajan työnkuva saattaa vaihdella hyvinkin paljon sen mukaan, millaisessa yrityksessä ja projektissa hän on mukana. Toisessa projektissa järjestelmän testaaminen saattaa vaatia ohjelmointitaitoa, kun toisessa taas dokumentoidaan testauksen etenemistä. Eri projekteissa testataan eri asioita, koska tuotteilla saattaa olla todella erilaiset käyttötarkoitukset. Joissakin tapauksissa esim. käyttöliittymän helppokäyttöisyys on tärkeintä, kun taas toisissa kohteissa pitää olla varma, että joku anturi antaa oikean arvon tai toiminnon välittömästi. (Kasurinen 2013, 10)

Jos testaus laiminlyödään, sen seurauksena menetetään ainakin rahaa, joskus työpaikkoja ja pahimmissa tapauksissa jopa ihmishenkiä (Guru99, 2017). Testauksen on tarkoitus varmistaa tuotteen oikeellisuus, valmiustaso sekä laatu. Mitä aikaisemmin virheet löydetään, sitä halvemmaksi ne tulee korjata.

Alla olevassa kuvassa havainnollisestaan (kuvio 1) miten The Systems Sciences Institute at IBM on todennut tutkimuksissaan, että mikäli virhe löydetään vasta ylläpito -vaiheessa, sen aiheuttamat korjauskustannukset ovat noin sata kertaa suuremmat, kuin mikäli virhe olisi ilmennyt jo määrittelyvaiheessa.



Kuvio 1: Virheiden korjaamisen kustannusten kertaantuminen vaiheittain. (The Systems Sciences Institute at IBM)

Vuonna 2017 australialainen yritys Tricentis teki laajan vertailun ohjelmistojen vioista. Ongelmien arvioitiin vaikuttavan 3,6 miljoonaan ihmiseen, tuoneen lisäkustannuksia 1,7 triljoonaa £ sekä hävittäneen 268 tuntia tehtyä työtä (Tricentis Report, 2019).

Kaikkia mahdollisia toimintoja ja variaatioita ei pysty testaamaan, joten testitapausten joukosta pitää valita ne tärkeimmät ja kriittisemmät (Guru99, 2017).

Jussi Pekka Kasurisen (Kasurinen 2013) mukaan testauksen elämänkaari voidaan tiivistää näin:

Projektin alussa testipäällikkö, kehittäjät ja testaajat tekevät testaussuunnitelman ja sen pohjalta testitapauksia. Samalla sovitaan työnjako. Kehittäjät tuovat järjestelmään yksikkötestattuja komponentteja ja luovat testaajien apuna testitapauksia, joilla pyritään kattamaan lopulta koko järjestelmä.

Testaajat suorittavat testitapauksia ja virhetilanteiden syntyessä tehdään niistä ilmoitus sovitulla tavalla sekä tyylillä. Raportoitujen virheilmoitusten avulla kehittäjät korjaavat löytyneet ongelmat.

Valmistuneita komponentteja tuodaan mukaan järjestelmään integraatiotestauksen kautta. Kun komponentit ovat valmistuneet ja integraatiotestaus on suoritettu hyväksytysti, alkaa järjestelmätestaus.

Kun järjestelmätestauksen kautta ei löydy enää merkittäviä vikoja, siirrytään hyväksymistestaukseen. Asiakkaan ollessa tyytyväinen järjestelmän toimintaan, todetaan järjestelmä valmiiksi.

Projekti päätetään sekä siitä koostetaan loppuraportti ja tarvittavat tiedot tallennetaan arkistoon mahdollista jatkokäyttöä ajatellen. Näistä on apua, mikäli myöhemmin järjestelmää joudutaan muokkaamaan tai täydentämään.

2.1 Testauksen tasot

Projektin alkaessa vaatimusmäärittelyissä määritellään, mitä eri osa-alueita ja tasoja testauksen tulee pitää sisällään. Toimitettavan ohjelmiston tai järjestelmän laadun varmistamiseksi on pohdittava tarkkaan, miten sen tärkeimmät ominaisuudet ja toiminnot saadaan mahdollisimman kattavasti testattua. Tarkka ja täsmällinen vaatimusmäärittely on erittäin tärkeässä osassa onnistuneessa projektissa, jossa lopputuloksena on järjestelmä tai tuote, minkä asiakas on halunnutkin (LIITE 1). Ohjelmiston luonteesta, käyttötarkoituksesta ja käyttäjäkunnasta riippuu, mitä ominaisuuksia pidetään tärkeänä ja halutaan testauksen avulla opti-

moida. Onko käyttäjien kannalta ensiarvoisen tärkeää, että käyttöliittymä on mahdollisimman helppokäyttöinen ja opastaa käyttäjää, vai onko käyttäjäkunta ammattilaisia, jotka saavat joka tapauksessa kattavan koulutuksen ohjelmiston käyttöön? Kuinka todennäköistä on, että ohjelmiston yhtäaikainen käyttäjäkunta saattaa ajoittain kasvaa hyvinkin suureksi?

2.1.1 Yksikkötestaus

Yksikkötestauksessa testataan vain pientä osa-aluetta, se saattaa olla joku osa koodista, joku yksittäinen toiminto tai komponentti. Yksikkötestaus on yleisesti kehittäjän itse suorittama testaus. Se on myös tavallisin ja yleisimmin käytössä oleva testauksen taso (Kasurinen 2013, 41).

Yksikkötestauksessa tarkistetaan, että testattavan moduulin koodit kääntyvät ja että se näyttää näennäisesti toimivalta. Yksikkötestauksessa testataan, että datan siirtyminen eri moduulien välillä toimii.

Omissa projekteissani tässä on käytetty koodikatselmointia, jossa muut kehittäjät tarkastavat koodin ja antavat siitä palautetta. Lisäksi kehittäjät suorittavat moduuleille erilaisia testejä, joiden tarkoituksena on testata kuinka hyvin moduulit vastaavat määrittelyjään. Testien tarkoituksena on varmistaa, että moduulit toimivat vaatimusten mukaisesti, sekä lisäksi että ne eivät myöskään suorita tarpeettomia tai virheellisiä toimintoja. Tällaisella käytännöllä myös varmistetaan, että jokainen yksittäinen osio on kattavasti testattu ja toimiviksi todettu.

Testattavan funktion, moduulin ja komponentin tyypistä riippuu, minkälaisia testejä sille tulisi tehdä ja minkä tyyppisiä virhetilanteita siltä on syytä odottaa. Eri yksiköiden testaus saattaa vaatia varta vasten testaukselle rakennettuja funktioita tai järjestelmiä.

2.1.2 Integraatiotestaus

Yksikkötestauksen jälkeen komponentteja ja funktioita yhdistetään ja testataan niiden toimivuutta keskenään. Tässä vaiheessa testaus tutkii eri osioiden yhteen-

sopivuutta sekä niiden toimintaa jo olemassa olevien komponenttien kanssa. Datan siirtyminen eri moduulien välissä pitää myös testata. Tästä syystä integraatiotestauksessa joudutaan usein rakentelemaan järjestelmän kaltaisia toimintoja, jotta komponenttien väliset yhteydet ja datan liikkuminen saadaan todennettua.

Integraatiotestausta voidaan suorittaa eri tavoilla (Guru99, 2014). Alla on esitelty yleisimmät mallit.

Kun testataan alhaalta ylöspäin, silloin ensimmäisenä testataan lähimpänä rautaa ja käyttöjärjestelmää olevat moduulit. Tämän päälle lisätään uusia moduuleita. Näin löytyy alemman tason ongelmat helposti.

Testatessa ylhäältä alaspäin, järjestelmän tärkeimmät moduulit tehdään ja testataan ensin. Tämä vaatii väliaikaisia ratkaisuja tarvittavien kantojen ja yhteyksien luomiseksi. Järjestelmästä saa selkeämmän yleiskuvan sekä testaaja havaitsee helpommin mikäli jotain toimintoja puuttuu.

Tyyliltään sekoitus kahdesta edellisestä on nimeltään voileipä. Koska moduuleita valmistuu kummastakin suunnasta, ei tarvitse rakennella teennäisympäristöjä. Mutta osien sovittaminen saattaa olla loppuvaiheessa haastavaa.

Hitain malli on big bang, jolloin odotetaan, että jokainen moduuli on valmis testattavaksi. Tämä tapa ei vaadi niin paljoa rakenneltuja väliaikaisratkaisuja, mutta vaatii odottelua. Virheiden löytäminen ei ole niin yksinkertaista, koska osat on tuotu testaukseen yhtä aikaa, eikä ole niin selkeää, että mistä moduulista vika johtuu.

Testauksen tavan valinta riippuu arkkitehtuurista sekä siitä, missä järjestelmän kriittisimmät moduulit sijaitsevat.

2.1.3 Järjestelmätestaus

Järjestelmätestauksessa yksikkötestatut moduulit, joiden on todettu toimivan yhdessäkin, liitetään osaksi kokonaiseen järjestelmään. Tässä kohdassa testataan,

että kaikki moduulit, funktiot, koodit ja eri toiminnallisuudet toimivat niille määriteltyjen tasojen sekä toimintojen mukaan. Testaaja testaa järjestelmää niillä tavoin, kuten loppukäyttäjänkin on tarkoitus sitä käyttää. Järjestelmätestaus tapahtuu yleisimmin testausympäristössä, mikä jäljittelee oikeaa järjestelmää. Siellä ei kuitenkaan enää pitäisi olla mukana mitään teennäisiä tai rakenneltuja osioita. Järjestelmätestauksessa virheitä etsitään kuitenkin vielä myös yksittäisistä komponenteista, vaikka itse testaus pyrkiikin varmistamaan koko järjestelmän toimivuuden halutulla tavalla. (Kasurinen 2013, 44)

2.1.4 Hyväksymistestaus

Omissa projekteissani asiakas useimmiten suorittaa hyväksymistestauksen heidän omassa ympäristössään. Asiakas varmistaa, että järjestelmä toimii sille annettujen vaatimusten ja määräysten mukaan.

Hyväksymistestauksen voi suorittaa myös tietynlainen otanta järjestelmän tilaajan asiakkaita, jotka kokeilevat järjestelmän toimivuutta.

Erilaisia testitapoja on yli 150 (Guru99, 2011). Niitä kaikkia ei tarvitse eikä voikaan käyttää samassa projektissa, vaan testauksen tarpeellisuus sekä tyyppi määrittyy projektin luonteesta ja toiminnoista riippuen.

2.2 Testauksen erilaiset tyypit

2.2.1 Regressiotestaus

Regressiotestauksella varmistetaan jo testatun ”toiminnallisuuden” toimivuus kun uusia koodimuutoksia tehdään, eikä voida olla varmoja siitä, ettei muutokset vaikuta jo testattujen toiminnallisuuksien toimivuuteen. Regressiotestaus voidaan toteuttaa joko manuaalisesti tai automatisoidusti tai näiden yhdistelmänä. Regressiotestit voidaan jakaa kolmeen ryhmään (Pressman 2010, 462).

- testit, joilla testataan järjestelmään jo liitettyjä komponentteja ja toimintoja

- testit niille ominaisuuksille ja toiminnoille, joihin muutokset voivat vaikuttavat todennäköisimmin
- testit niille komponenteille, joita on muutettu.

2.2.2 Käytettävyystestaus

Käytettävyystestauksessa järjestelmän käyttöliittymä on yleensä jo lähes valmis. Testaaja testaa sen käytettävyyttä: ovatko kaikki toiminnot oikein ja sijaitsevat ne järkevillä paikoilla käyttöliittymässä. Käytettävyystestauksessa käyttäjä käyttää ohjelmistoa ja pyrkii tekemään sillä sellaisia asioita, joita ohjelmiston avulla pitäisi pystyä tekemään. Käytettävyystestauksessa usein nauhoitetaan käyttäjän toimintaa, ja näin päästään käsiksi mahdollisiin puutteisiin ja virheisiin ohjelmiston käyttöliittymässä ja toimintalogiikassa. Käytettävyystestauksella saadaan palautetta ohjelmiston helppokäyttöisyydestä.

Itse olen projekteissani ollut mukana teettämässä käyttäjäkokeiluja. Testaajat ovat koostunen ihmisistä, jotka eivät tunne tuotetta ennestään ja he antavat palautetta käyttöliittymän toimivuudesta, selkeydestä ja myöskin ulkonäöllisestä puolesta.

Usein ohjelmistoja testattaessa paras testaajaryhmä koostuu ohjelmiston loppukäyttäjistä. Etenkin asiantuntijakäyttöön suunniteltujen ohjelmistojen kohdalla arvokkaimmat havainnot saattavat tulla juuri loppukäyttäjiltä. Näin ollen monissa projekteissa tehdäänkin usein tiivistä yhteistyötä myös asiakkaan edustajista ja loppukäyttäjistä koostuvissa projektiryhmissä. Ohjelmiston varsinaiset käyttäjät osaavat loppukädessä arvioida, onko ohjelmisto ja sen toiminnot juuri suunniteltuun käyttöön sopivia ja riittävän helppokäyttöisiä. Määrittelyvaiheessa toki on pyritty määrittelemään vaatimukset mahdollisimman kattavasti, mutta muutostoiveita usein tulee kehitysvaiheessa joka tapauksessa ilmi. Tavoite joka tapauksessa on, että loppukäyttäjä saa käytettäväkseen juuri tarkoituksenmukaisen ohjelmiston.

Käytettävyystestauksessa automaation käyttö testauksen apuna saattaa olla haastavaa toteuttaa. Käyttäjäkokemus on vahvasti sidonnainen käyttäjään henkilökohtaisesti, ja käyttäjäkokemus voi olla joka käyttäjän kohdalla erilainen. Automaatiolla voidaan testata käyttöliittymän toimintaa, mutta ei välttämättä sitä,

onko käyttöliittymä käyttäjäystävällinen ja onko se toteutettu juuri kyseiselle käyttäjäryhmälle sopivaksi.

2.2.3 Savutestaus

Savutestauksessa tarkastetaan, että järjestelmän perustoiminnot toimivat. Niitä voi olla esimerkiksi, että järjestelmä käynnistyy, tallennus onnistuu, järjestelmä sammuu oikein. Kasurisen (2013, 57). mukaan kyse on minivaatimuksista, mitkä järjestelmässä pitää toimia.

Savutestaus pyrkii varmistamaan, että ohjelmiston versio on sillä tasolla, että sen siirtäminen varsinaiseen toiminnalliseen testaukseen on järkevää. Savutestaus ei vielä itsessään sinänsä ota kantaa siihen, toimiiko ohjelmisto varsinaisesti määrittelyjen mukaisesti, vaan savutestauksessa vasta tarkistetaan, ovatko ohjelmiston ydintoiminnot kunnossa. Savutestauksessa useimmiten ajetaan hyvin rajattu määrä testitapauksia, joilla perustoiminnallisuuksissa ilmenevät mahdolliset kriittiset puutteet tulevat ilmi. Savutestauksessa voidaan käyttää testausautomaatiota, ja usein savutestauksen testitapaukset ovatkin hyvin samanlaisia tai jopa täysin samat, joten automaatio voi olla tässä vaiheessa erittäin perusteltu vaihtoehto.

2.2.4 Kuormitustestaus

Kuormitustestauksessa testataan järjestelmän suorituskykyä erilaisissa tilanteissa. Yleensä järjestelmillä on tietty aika, missä ajassa niiden pitää pystyä toteuttamaan eri toimintoja. Testaus on aika haastavaa ja helpointa se onkin siihen suunnitelluilla työkaluilla, joilla kuormitetaan järjestelmää mahdollisimman paljon. Tällainen testaus ei ole ihan suora kopio todellisesta tilanteesta, mutta näin siitä saadaan kuitenkin jonkinlainen arvio.

Kuormitustestauksella selvitetään (Kasurinen 2013, 56) järjestelmän pullonkaulat sekä mikä on järjestelmän maksimikapasiteetti ja se, miten järjestelmä toimii normaalioloissa.

Kuormitustestauksessa luodaan testattavalle ohjelmistolle kuormaa, ja mitataan vasteaikoja. Ohjelmistolle on usein määrittelyvaiheessa kirjattu tietynlaiset vaatimukset jotka ohjelmiston on pystyttävä täyttämään. Riippuen millaiseen

tarkoitukseen ohjelmisto on suunniteltu, vaatimuksissa voi olla mainintoja muun muassa yhtäaikaisista käyttäjämääristä ja interaktioista, joita ohjelmiston on pystyttävä tietyssä ajassa käsittelemään.

Kuormitustestauksen haasteita on usein tuotannonkaltaisen tilanteen simuloinnin vaikeus. Kuormitustestauksessa ei myöskään välttämättä päästä käsiksi itse ongelmakohtiin mikäli puutteita kuormasta selviämisessä ilmenee. Automaatiotyökaluilla voidaan kuitenkin nykyään melko hyvin kuormittaa ohjelmistoa ja näin selvittää miten ohjelmisto kasvavaa kuormitusta pystyy käsittelemään.

2.2.5 Suorituskykytestaus

Suorituskykytestaus on kuten kuormitustestaus, mutta järjestelmää ei kuormiteta niin raskaasti. Tässä selvitetään, että järjestelmä suoriutuu toiminnoistaan niissä aikaraameissa, mitä sille on määrittelyissä sovittu. (Kasurinen 2013, 57)

Ohjelmiston luonteesta riippuen suorituskykytestaus saattaa olla tuotantokäytön kannalta hyvinkin kriittistä. Vaatimuksissa on mahdollisesti määritelty ohjelmistolle vasteajat, ja tämän lisäksi ohjelmiston käyttäjämäärä saattaa vaihdella suuresti. Ohjelmiston nopeus voi olla merkittävä tekijä käyttäjätyytyväisyydelle, ja asiantuntijakäyttöön suunnitelluissa ohjelmistoissa nopeus voi olla kriittinen tekijä työprosesseissa. Suorituskykytestausta tehdäänkin usein läpi ohjelmiston kehitysvaiheen, jotta nähdään miten ohjelmiston komponenttien muokkaus ja lisääminen vaikuttavat ohjelmiston suorituskykyyn. Kuten kuormitustestauksessa, myös suorituskykytestauksessa automaatiotyökalut ovat suuri apu testaajalle.

2.2.6 Ad hoc -testaus

Ad hoc -testaus on testausta, mitä saatetaan suorittaa projektissa matkan varrella erilaisissa tilanteissa. Nämä testaukset eivät ole juuri mitenkään organisoituja, eikä niistä laadita yleensä mitään raporttejakaan. Tulokseksi riittää, että toimiiko järjestelmä vaiko ei. (Kasurinen 2013, 60).

Ad hoc testauksessa ei yleensä käytetä mitään tiettyä testaustekniikkaa eikä välttämättä suunnitella testitapauksia, vaan pyritään satunnaisen testikäytön avulla

saamaan esiin ohjelmiston ongelmakohtia. Ad hoc testaus ei kuitenkaan välttämättä löydä ohjelmistovirheitä yhtä tehokkaasti kuin paremmin strukturoitu testaus.

2.2.7 Tutkiva testaus

Tutkivassa testauksena päämääränä on löytää mahdollisimman paljon vikoja (Kasurinen 2013, 59).

Omalla kohdallani tutkivaa testausta on suoritettu niin, että rinnakkaisen tiimin jäsenet ovat testanneet meidän tekemää osuutta. Näin ollen he tietävät jo alusta alkaen miten tiimissä työskennellään, mitä on mahdollisesti tehty ja mistä virheitä todennäköisesti löytyy.

Tutkivassa testauksessa pyritään hyödyntämään testaajien omaa kokemusta ja tietotaitoa ongelmien ja virheiden kaivamiseen. Tutkivassa testauksessa testaaja ei luo varsinaisia testitapauksia etukäteen, vaan suunnittelee testitapaukset suurelta osin jo tehtyjen testien tulosten perusteella. Näin testaaja pystyy usein paremmin syventymään toimintoihin, joissa näkee olevan mahdollisia puutteita. Testaajalla saattaa myös kokemuksensa perusteella olla hyvä käsitys toiminnoista ja käyttötapauksista, joissa todennäköisimmin löytyy haasteita. Myös erilaisten syötteiden kokeileminen ja niiden perusteella jatkotestien suorittaminen hyötyy usein testaajan monipuolisesta testauskokemuksesta.

2.3 Testaus lukuina

Ohjelmiston kehityskaari koostuu useasta eri osa-alueesta. Alla olevasta osiosta nähdään lukuina (Kasurinen, 2013), kuinka iso rooli testauksella on ohjelmistokehityksessä. Lukujen perusteella voidaan todeta, että testauksen laiminlyönti, testauksen suorittaminen väärässä ohjelmiston elinkaaren vaiheessa, tai testauksen suorittaminen puutteellisella suunnittelulla ja systemaattisuudella aiheuttaa helposti merkittäviä suoria ja epäsuoria taloudellisia tappioita.

Vuonna 2002 tehdyn tutkimuksen mukaan yhdysvaltalaiset ohjelmistotalot menettivät 21,2 miljardia dollaria testauksen puutteiden takia. Kun tähän laskettiin mukaan testauksen läpi päässet viat, jotka toivat lisäongelmia asiakkaille, summa

nousi 59,5 miljardiin dollariin. Tähän summaan ei ole laskettu menetyksiä, joita yritykset ja asiakkaat ovat menettäneet maineensa kärsimisestä tai asiakkaiden menetyksestä.

Arviolta n. 50% projektin budjetista menee testaukseen. Näin ollen testaus on selkeästi kallein osuus kehitysprojektissa. Se on myöskin tärkein osa projektissa, sillä hyvin tehtynä se on tuottoisin työvaihe. Kun myyty tuote tai järjestelmä on laadukas, on sillä parempi kate sekä sitä on helpompi myydä. Asiakkaat ovat tyytyväisempiä ja palaavat asiakkaiksi uudelleen.

2.4 Testauksen periaatteet

Testaukseen ei ole olemassa varsinaisesti mitään ohjenuoraa, koska jokainen projekti on luonteeltaan ja ympäristöltään erilainen. Jyväskylän yliopiston mukaan kuitenkin seuraavat ovat yleisimmin alalla esiintyvät periaatteet (Jyväskylän Yliopisto. N/A).

Testitapauksen haluttu lopputulos tulee määritellä tarkasti. Testin pitää olla selkeä ja edetä siten, että testaajalla ei ole epäselvyyttä sen suorittamisessa. Järjestelmää tulee testata ajoissa ja usein. Testauksen pitäisi olla mukana jo järjestelmän määrittelyvaiheessa.

Kirjoita testit niin, että niitä voidaan uudelleen käyttää muuallakin järjestelmän testauksessa. Jos joku osa-alue on hyödyllistä automatisoida, siihen kannattaa käyttää hieman aikaa.

Katselmointeja on syytä järjestää niin koodeille, kuin testitapauksillekin. Ulkopuolisen näkemys komponentin tai testin toimivuuteen on erittäin suotavaa.

Ei riitä, että järjestelmä tekee, mitä sen pitäisikin. Se ei myöskään saa tehdä mitään ylimääräistä taikka väärää. Aikaisemmin löydetty virhe on aina halvempi ja helpompi korjata, kuin että se löytyy vasta valmiista tuotteesta. Testaus ei voi kuitenkaan olla täydellistä. Testaaminen on aina kompromissi. On valittava ne kaikkein riskialttiimmat kohdat.

2.5 Testauksen sijoittuminen erilaisissa projektimalleissa

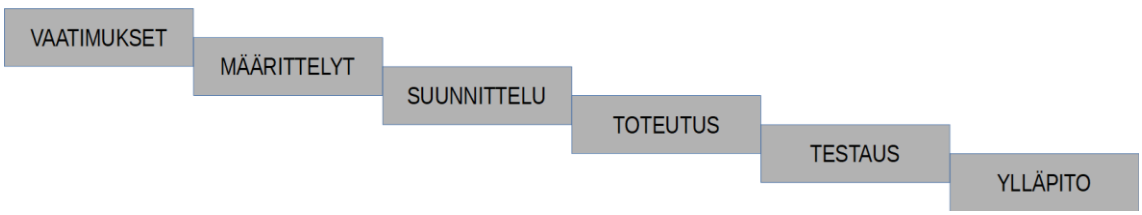
Projektin luonteesta riippuen, siihen liittyvät prosessit voidaan hoitaa eri tavalla. Jo ennen vaatimusmäärittelyä pitäisi miettiä näitä asioita ja valita toimivin projektimalli (Pulkkanen N/A):

- Organisaatiosi ydinarvot
- Projektisi tärkeimmät bisnestavoitteet
- Projektisi tärkeimmät rajoitteet (aika, resurssit, laatu)
- Sidosryhmät
- Riskit
- Monimutkaisuus
- Projektin koko

Eri malleissa on hieman eroja. Toiset ehkä tukeutuvat enemmän tarkkoihin aikatauluihin ja budjettiarvioihin, toisissa taas riskien tunnistaminen on tärkeässä roolissa ja mahdollisiin ongelmatapauksiin pitää pystyä reagoimaan nopeasti.

2.5.1 Perinteinen vesiputous -malli

Vesiputous -mallissa projektin prosessit etenevät vaihe kerrallaan (kaavio 1). Yksi osa-alue tehdään valmiiksi, jonka jälkeen siirrytään seuraavaan. Vesiputousmallin ohjelmistoprojekti saattaa alkaa esitutkimuksella, jonka perusteella lähdetään määrittelemään ohjelmistolle asetettavia vaatimuksia. Määrittelyjen perusteella aloitetaan projektin toteutus, jonka jälkeen voidaan siirtyä testausvaiheeseen. Kun ohjelmiston testaus on suoritettu, ohjelmisto siirtyy ylläpitovaiheeseen.



KAAVIO 1. Vesiputous -malli

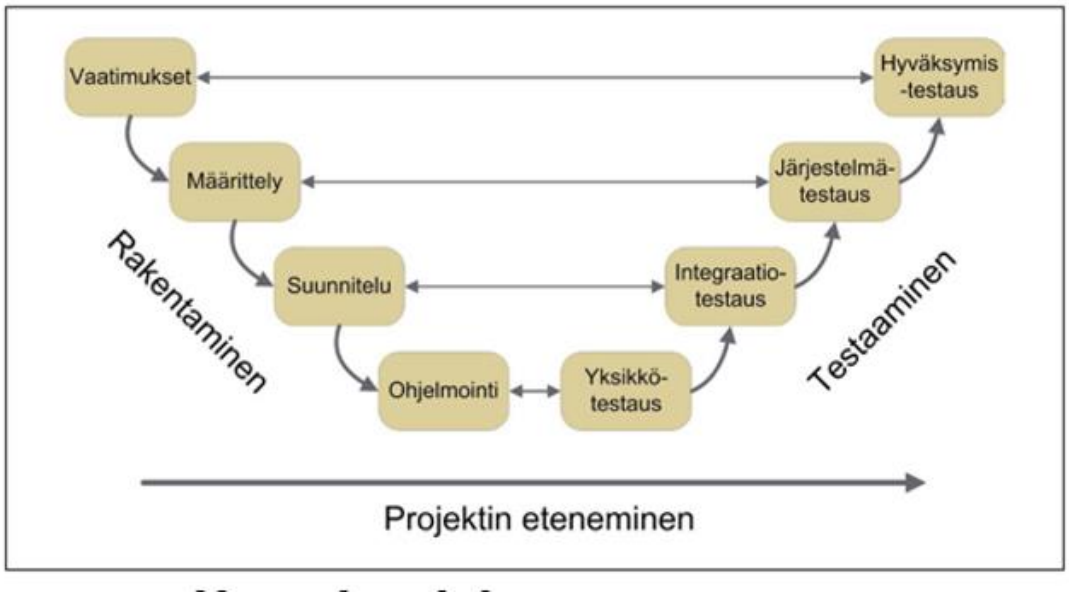
Vesiputous -mallissa tarkoitus on tehdä jokainen osa-alue niin tarkasti, ettei prosessissa tarvitse mennä taakse päin. Tämä malli ei sovi projekteihin, joissa määrittelyt muuttuvat useasti tai asioita tapahtuu nopealla syklillä. Vesiputous -malli sopii hyvin sellaisten prosessien käyttöön, jossa jo suoritettu osa-alue ei tule muuttumaan myöhemmin, esimerkiksi auton kasaaminen tehtaalla tapahtuu tietyssä järjestyksessä, eikä siihen jälkikäteen lisätä lisää penkkejä tai rattia toiseen paikkaan.

Vesiputous -mallissa testaus suoritetaan projektin loppupäässä. Mikäli tässä kohtaa havaitaan jotain ongelmia, ne ovat yleensä hitaasti korjattavissa ja todella kalliita muutoksia. Mikäli testauksessa huomataan, että määrittelyt eivät ole olleet oikeat, valmistunut tuote ei vastaa sitä mitä haluttiin.

Mikäli prosessit sujuvat ongelmitta, on vesiputous -mallissa etuna se, että siinä on selvillä tarkemmat aikataulut sekä budjetti.

2.5.2 V-malli

V-mallissa jokaisella osa-alueella suoritetaan oma testauksensa, kuten alla olevasta kuviosta nähdään (kaavio 2). Kun koodia on tuotettu, sille ajetaan yksikkötestit. Näin nähdään, että koodit itsessään ovat toimivia. Integraatiotestauksessa varmistetaan, etteivät koodit keskenään aiheuta vikatilanteita eikä ongelmia eri rajapinnoissa. Kun tehdyt muutokset tai lisäykset tuodaan järjestelmään, testataan, että ne toimivat siellä oikein. Samalla verrataan järjestelmää määrittelyihin. Hyväksymistestauksessa asiakas vertaa saamaansa tuotetta antamiinsa vaatimuksiin ja päättää, onko tuote oikeanlainen.



KAAVIO 2 V-malli (Kasurinen 2013)

V-mallissa testaamisen portaallisuus helpottaa virheiden aikaista löytämistä. Koska jokainen osa-alue on testattu erikseen ja myös erillään, se myös helpottaa mahdollisten ongelmien paikantamista.

2.5.3 Ketterät mallit (Agile)

Ketterissä malleissa projektin prosessit ovat pilkottu pienempiin osiin. Jokaisen osion sisällä pyörii oma vesiputous -mallin tyylinen prosessi, jossa osa-alueet suoritetaan ja hyväksytään sovitussa määräajassa.

Tärkeintä ketterien mallien toimivuudessa on toimiva, nopea ja jatkuva yhteistyö tekijöiden ja sidosryhmien välillä (Pulkkanen, N/A) Näin esiin nouseviin kysymyksiin ja ongelmatilanteisiin saadaan nopeasti selvityksiä sekä päätöksiä. Kanssakäyminen pyritään pitämään lyhyenä ja selkeänä. Yleensä ketterissä projekteissa on päivittäinen palaveri, joka kestää n. 15min. Siinä käydään nopeasti läpi mitä on tehty ja mitä seuraavaksi ollaan tekemässä.

Ketterät mallit toimivat projekteissa, joissa lopputulos ei ole tarkoin määritelty. Prosesseissa voidaan matkan aikana visioida parempia tapoja sekä keksiä lisää ideoita. Tämä malli ei toimi kovinkaan hyvin raskaissa ja työläissä projekteissa,

kuten esim. talon rakentamisessa. Jatkuvat muutokset tulisivat kalliiksi sekä vaikuttaisivat aikatauluihin, jos vaikka seinien paikkaa tai muita jo tehtyjä kiinteitä asioita ruvettaisiin muuttamaan tai purkamaan.

Projektin työvaiheet ovat ketterän menetelmän mallissa jaettu pienempiin osiin. Näistä valitaan työtehtävät tietyn mittaista ajanjaksoa varten. Tässä ajassa nuo valitut työt ohjelmoidaan, testataan, raportoidaan ja yhdistetään jo aikaisempiin tuotoksiin. Kun koodeja tuotetaan ja testataan koko ajan, saadaan niistä palaute nopeammin, kuin että testaus tehtäisiin vasta projektin lopussa. Tämä helpottaa myös asiakkaan kanssa tehtäviä muutoksia. Mikäli asiakas ei ole tyytyväinen suoritettuihin muutoksiin, ne ovat helpompia muokata aikaisemmassa vaiheessa uudelleen.

3 MANUAALINEN TESTAUS VS AUTOMAATIOTESTAUS

3.1 Manuaalinen testaus

Manuaalinen testaus on ihmisen suorittamaa testausta. Testaaja on fyysisesti paikalla suorittamassa testattavia toimintoja. Manuaaliseen testaukseen sitoutuu testaajan aikaa ja yrityksen rahaa. (Guru99, 2013)

Manuaalisessa testauksessa inhimilliset virheet ovat mahdollisia, varsinkin jos testattava asia pysyy koko ajan samana. Näin testaukseen turtuu, eikä virheitä välttämättä huomaa.

Manuaalinen testaus on välttämätöntä sellaisissa tapauksissa, joissa tulokset pitää nähdä ja arvioida sen visuaalisen ilmeen mukaan. Testitapaus, jonka määrittelyt ja vaatimukset muuttuvat useasti, on järkevämpi suorittaa manuaalisesti.

3.2 Automaatiotestaus

Automaatiotestauksessa testauksen hoitaa siihen tarkoitukseen valittu työkalu, joka käyttää ihmisen rakentamaa ohjeistusta testaamiseen. Automatisoidussa testauksessa ihmisen ei tarvitse olla läsnä ja se voidaan suorittaa ohjelmoidusti vaikka keskellä yötä (Guru99, 2013).

Automaatiotestaus on paikallaan silloin, kun testattava datamäärä on suuri tai samoja asioita toistetaan moneen kertaan. Samat, useasti toistuvat testitapaukset ovat järkevää automatisoida (Edureka!, 2017). Testiautomaatio on tarkempaa, koska jo testiä automatisoidessa on selvillä testin lähtökohdat sekä lopputulos. Nykypäivänä automaatiotestauksen avulla pystytään testaamaan erilaisia alustoja (mobiili vs pöytäkone, IOS vs Android, MAC vs Windows) sekä konfiguraatiota ja asetuksia.

Automaatiotestauksesta saadaan tulokset nopeasti ja niihin voidaan reagoida välittömästi. Tulokset vaativat kuitenkin ihmisen niitä tulkitsemaan. Raportti osaa

kyllä kertoa onko ajettu testitapaus mennyt oikein sen mukaan mitä tietoja siihen on annettu, mutta ihmisen on päätettävä mitä saatu tulos tarkoittaa. Automaatiotestauksen tärkeimmät hyödyt ovatkin nopeus ja tehokkuus (VALA Group. 2017).

Automaatiotestauksen vaiheet ovat jokainen oma prosessinsa. Ensin automatisoitujen testien suorittamiseen on valittava sopiva työkalu. Tarjolla olevien työkalujen toimintoja sekä ominaisuuksia on verrattava projektin tarpeisiin.

Automatisoitaviksi sopivat testitapaukset valitaan ja kirjoitetaan määrittelyjen mukaan. Tämän jälkeen suoritetaan testitapausten katselmoiminen.

Testiympäristön valmistelu käyttöön voi vaatia erilaisia asennuksia, oikeuksia ja jopa koulutuksia, jotta ympäristö saadaan toimivaksi.

Lopulta, kun ympäristö on valmis ja testitapaukset saatu sinne mukaan, on vuorossa testien ajaminen ja syntyneiden raporttien tarkastelu sekä näiden pohjalta tehtyjen päätöksien tekeminen. Jatkossa myös automatisoituja testitapauksia sekä testiympäristöä pitää ylläpitää ja päivittää tarvittaessa.

Näissä prosesseissa sekä jatkossa automaation kanssa on tehtävä erilaisia päätöksiä ja valintoja, eikä ne ole aina itsestään selviä. Erilaisia ongelmia, sekä huonoja valintoja voi tulla eteen (Swiss TestingDay. 2012). Mikäli asiakkaalla tai toimittajallakaan ei ole oikeaa käsitystä automaatiosta tai järjestelmän vaatimuksista, edessä on pettymyksiä vääristä luuloista johtuen. Muun muassa osaamisen puute, liian suuret odotukset, automaation kalleus sekä ylläpidon työläys voi tulla yllätyksenä (VALA Group. 2017).

4 LAADUN HINTA

Testauksesta kertyviä kuluja ovat mm. palkat, työtilat – sekä välineet, testausympäristö, ohjelmistojen lisenssit sekä koulutus.

Laadun tuottamiseen, ylläpitoon ja näiden kulujen laskemiseen on olemassa erilaisia malleja. Laadun kulut koostuvat määrittelyistä, ennaltaehkäisystä, virheiden löytämisestä sekä toimenpiteistä, mitä virheiden korjaamisesta syntyy. Laadun hinta ei ole sama, kuin laadukkaan tuotteen hinta, vaan se on kustannus siitä, että tuote ei ole ollut laadukas alusta alkaen. (Asq.com)

Feigenbaumin PAFF -mallin mukaan kulut jaotellaan neljään eri osa-alueeseen: Ennaltaehkäiset ja tutkimuskulut sekä suorat ja epäsuorat virheistä tulleet maksut (Sanjeev Deshmukh, 2015).

Ennaltaehkäisevät kulut koostuvat asioista, joilla pyritään estämään virheiden pääsy tuotteeseen. Näitä ovat projektin suunnittelu sekä erilaiset vaatimusmäärittelyt, työntekijöiden tarvittava koulutus, komponenttien sekä testitapausten katselmoinnit ja mahdolliset data-analyysit.

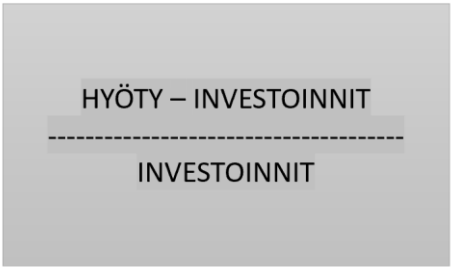
Tutkimusosion kustannukset kertyvät asioista, joilla ylläpidetään laatua. Näistä tärkeimmät ovat testauksen suunnittelu ja suorittaminen, itse testausvälineet ja niiden ylläpito sekä työntekijöiden koulutus.

Suorat kustannukset ovat taas erinäisiä kuluja, jotka koostuvat mahdollisten virheiden tutkimisesta, korjaamisesta sekä uudelleen testaamisesta. Mikäli tuote on virheellinen, voi asiakas vaatia hinnanalennusta. Asiakkaat työllistävät myös asiakaspalvelua.

Tämän lisäksi tuotteen virheellisyys voi aiheuttaa yritykselle myös sakkoja ja muita takuu- sekä vastuukuluja. Yleisesti myös yrityksen maine kärsii ja asiakkaita saatetaan menettää. Nämä ovat epäsuoria kustannuksia.

4.1 Automaation kulut

Ennen kuin projektin testausta aletaan automatisoimaan, pitää miettiä, että onko return on investment (ROI = pääoman tuotto) riittävä (Smartbear, N/A). Se voidaan laskea jakamalla ajateltu tuotto sen saavuttamiseksi käytettyjen investointien summalla (kuva 1).



Kuva 1. Kaava, jolla pääoman tuotto lasketaan

4.2 Automaation hyödyt

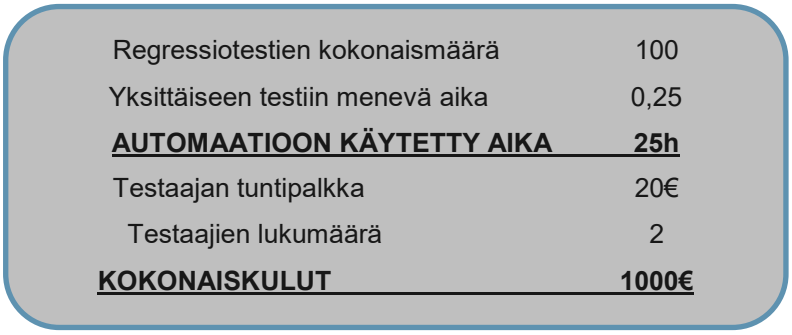
Hyötyjen ja investointien laskentaan on olemassa kuusi (6) mittaria, jolla tarkastellaan sekä lyhyen että pitkän aikavälin rahallista vaikutusta.

1. Automatisoitavien testien määrä. Päätetään mitä testitapauksia automatisoidaan ja mitä ei. Lasketaan, kuinka kauan testin kirjoittaminen, suorittamien ja ylläpito vie aikaa. Mukaan lasketaan myös testaajien määrä kerrottuna tuntipalkalla (kuva 2).

Testitapausten kokonaismäärä	100
Yksittäiseen testiin menevä aika	0,5
AUTOMAATIOON KÄYTETTY AIKA	**50h**
Testaajan tuntipalkka	20€
Testaajien lukumäärä	2
KOKONAISKULUT	**2000€**

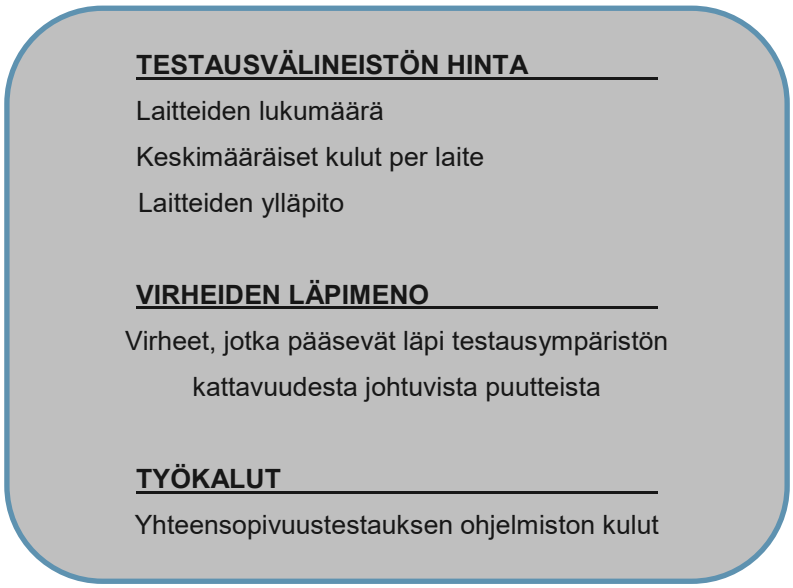
Kuva 2. Kaava, jolla lasketaan automatisoinnin kulut

2. Valitaan tärkeimmät testitapaukset, joilla suoritetaan jatkossa regressiotestausta. Näillä selvitetään tietyn ajan välein, että tuotteen perustoiminnallisuus pysyy ehjänä matkan varrella tehtyjen muutoksien ja lisäyksienkin jälkeen (kuva 3).



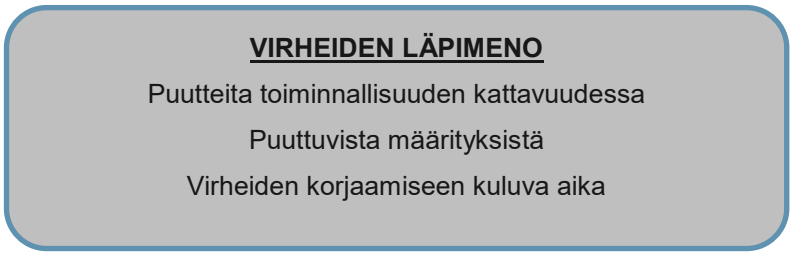
Kuva 3. Kaava, jolla lasketaan regressiotestien kulut

3. Yhteensopivuustestaus on tärkeää nykypäivänä. Testitapausten tulisi toimia myös erilaisilla järjestelmä-, selain- ja laiteyhdistelmillä. Monien testiympäristöjen ylläpitäminen tuo kuluja. (kuva 4).



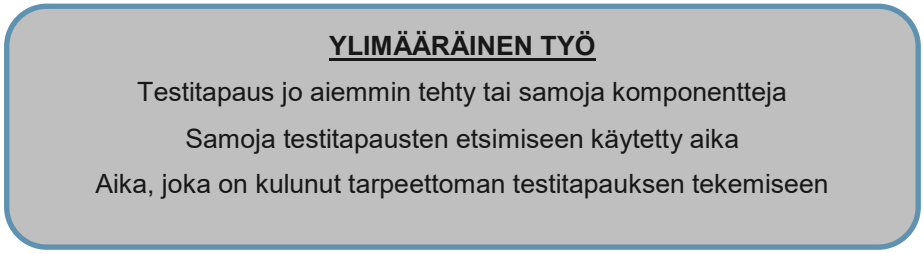
Kuva 4 Kaava, jolla lasketaan yhteensopivuustestauksen kulut

4. Virheiden läpimeno. Kaavalla lasketaan virheiden määrä, jotka pääsevät tuotteeseen (kuva 5).



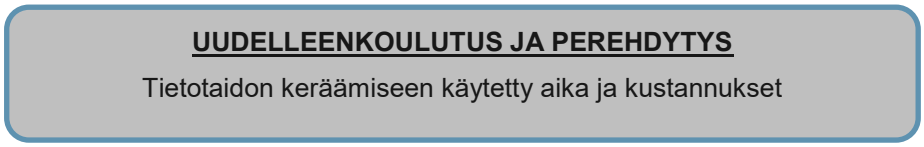
Kuva 5 Kaava, jolla lasketaan tuotteeseen jääneiden virheiden kulut

5. Testitapausten tulisi olla mahdollisimman monikäyttöisiä ja uudelleenkäytettäviä. Näin parannetaan testisyklejä sekä testaamisen nopeutta. Turhien testitapausten kuluihin vaikuttavat seuraavat asiat (kuva 6)



Kuva 6. Kaava, jolla lasketaan turhan työn kulut

6. Tietotaidon häviäminen. Mikäli testiautomaatiota ylläpitänyt henkilö lähtee yrityksestä, seuraavan henkilön kouluttaminen vie aikaa ja rahaa. Uudelleenkoulutus sekä perehdytys on yksi menoeristä (kuva 7)



Kuva 7. Kaava, jolla lasketaan tietotaidon poistumisesta aiheutuneet kulut

4.3 Investoinnit

Investoinneissa otetaan huomioon se, että automaatioon tarvittavien työkalujen valinta sekä niiden käyttöönotto vaatii ainakin jonkin verran työtunteja. Ja myöskin, että tiimin jäseniä on koulutettava automaation käyttämiseen tai sitten jopa palkattava työntekijä erikseen tätä varten.

4.4 Yleisimmät ROI virheet

Alle on listattu yleisimmät virheet, mitä ROI:n laskennassa tapahtuu. Mikäli näistä osa-alueista jokin tai useampikin jää huomioimatta, ei analyysin tulos ole totuudenmukainen.

Keskitytään vain automatisoituihin testeihin, vaikka myös manuaalista testausta tarvitaan. Näin ollen myös ne testitapaukset on ajettava ja ylläpidettävä. Ei oteta huomioon sitä prosenttiosuutta testitapauksista, jotka pitää tehdä manuaalisesti. Testejä, jotka vaativat ihmisen osallistumista ja havainnointia, ei voi automatisoida.

Ei huomata tietotaidon puutteellisuutta. Että automatisoiminen onnistuisi, tiimin pitäisi tuntea sekä tuote että mitä sen automatisoiminen vaatii.

Ei oteta huomioon automatisoimisen pitkän aikavälin hyötyjä. Automatisoimiseen satsaaminen ei ole välttämättä tuottavaa alussa, mutta pidemmällä aikavälillä hyödyt tulevat esille.

5 ESTEET TESTAUKSEN AUTOMATISOIMISELLE

Projektin testauksen pitäminen manuaalisena voi johtua useammastakin eri syystä. Osa niistä voi olla käytännön asioihin liittyviä ja osa tietämättömyydestä johtuvia.

Yleisiä rajoitteita manuaalisen testauksen automatisoimiselle on muun muassa se, että asiakas ei halua investoida. Joissain tilanteissa saattaa olla, että toimittaja ei syystä tai toisesta ole halukas tarjoamaan automatisointia. Monissa projekteissa järjestelmä muuttuu jatkuvasti, mikä rajoittaa osaltaan automatisointimahdollisuuksia. Projektissa saattaa myös olla työn alla eri järjestelmiä yhdessä, tai eri toteutuksia, mikä myös rajoittaa automatisointia. Projektit, joissa ei ole tarvetta testitapauksien toistamiselle, eivät myöskään kannusta automatisointiin. Sama koskee projekteja, joissa rakennetaan monimutkaista graafista käyttöliittymää.

Uusien ohjelmistojen käyttö projektissa on myös selkeä rajoite ottaa käyttöön automaatiota. Joskus projektin osapuolilla saattaa olla liian epärealistinen käsitys automatisoinnista, mikä osaltaan rajoittaa sen tehokasta käyttöä. Myös koko ohjelmistotestauksen osuus projektissa saattaa olla häilyvää, jolloin automaation käyttö on todennäköisimmin myös rajoittunutta. Automaation vaatima ylläpito voidaan myös kokea hankalaksi. Yhtenä rajoitteena voi olla aikataulujen yhteensovittamisen vaikeus ohjelmistokehittäjien ja testausasiantuntijoiden kesken. Monissa projekteissa myös todetaan, että automaatiotestaus ei läheskään aina tuo projektiin lisähyötyä.

Alla on käyty läpi paremmin muutamia meidän yrityksellemme tyypillisimpiä ongelmakohtia.

5.1 Rahoituksen puute

Maksavan tahon puuttuminen lienee se yleisin syy testauksen vajavaisuuteen tai jopa puuttumiseen. Meidän projekteissamme asiakkaat eivät kaikissa tapauksissa osta hyväksymistestausta, vaan suorittavat sen itse.

Osassa projekteista asiakkaana on yhdistys, mutta projektin maksaja on joku muu taho. Näissä ei yleensä haluta testaukseen käyttää alun perinkään pientä budjettia.

5.2 Järjestelmä muuttuu jatkuvasti

Joidenkin asiakkaiden projektit ovat muutoksissaan niin nopeatempoisia, että testien automatisoiminen olisi turhaa. Testitapauksia sekä ehkä jopa testiympäristöä pitäisi päivittää ja muuttaa koko ajan.

5.3 Projektin koko

Jotkut järjestelmät ovat niin pieniä, että niiden testaaminen on nopeaa ja helppoa manuaalisestikin. Niissä ei ole satoja testitapauksia toistettavana eikä isoja määriä dataa käytettävänä. Itse testitapausten suunniteluun ja kirjoittamiseen menee sama aika mikäli automatisoidessakin, mutta mikäli automaatioympäristöä ei ole jo valmiina käyttöön, on testaus nopeampaa manuaalisesti.

5.4 Järjestelmässä paljon sidosryhmiä ja eri järjestelmiä

Tänä päivänä monessa projektissa on mukana useita sidosryhmiä sekä näiden kautta eri järjestelmiä ja rajapintoja. Näissä tapauksissa testejä ei pysty automatisoimaan suoraan, koska eri järjestelmiin ei pääse ottamaan yhteyttä.

Järjestelmien välillä saattaa olla myös erilaisia tunnistautumisprotokollia tai muita salasanoja vaativia rajapintoja. Saattaa myös olla, että data järjestelmästä toiseen ei olisi yhteensopivaa.

5.5 Projektin aineisto ei sovi järkevästi automaatioon

Data voi olla sellaista, että se muuttuu testauksen edetessä, eikä se näin ollen olekaan seuraavan testikierroksen alussa sama, mitä testitapaus edellyttää. Tämän tyyppiset testitapaukset kyllä onnistuvat tehdä, mutta se vaatii enemmän työtä ja osaamistakin.

5.6 Projektissa graafinen käyttöliittymä

Graafinen käyttöliittymä on hidas automatisoida. Näissä yleensä vielä käyttöliittymän ulkonäölläkin on väliä ja sen ihminen pystyy havainnoimaan paljon nopeammin.

Käyttöliittymätestaus voidaan kyllä itsessään automatisoida, mutta se olisi syytä huomioida jo koodausvaiheessa. Näin käyttöliittymän eri elementeille voidaan antaa jo alusta alkaen tarvittavat tunnisteet ja muut hyödylliset tiedot.

6 PROJEKTIN ONNISTUNUT AUTOMATISOINTI

Minkä tyyppisiä projekteja meillä on automatisoitu?

Asiakaskuntamme koostuu hyvin erilaisista organisaatioista. Myös projektit koostuvat eri kokoisista ja luonteisista työvaiheista, ja rakennettavien ohjelmistojen käyttäjäkunta vaihtelee tyypiltään myös. Usein ohjelmiston testaus kuuluu toimittajan tarjoamaan palveluun, mutta usein asiakas haluaa huolehtia testauksesta kokonaisuudessaan itse.

Yrityksessämme automaatiotestaus on pääsääntöisesti käytössä projekteissa, joissa järjestelmä on yhteiskunnallisesti merkittävä, järjestelmä on auditoinnin piirissä ja sen elinkaari on pitkä. Varsinkin näissä projekteissa on myös tärkeää, että regressio löydetään mahdollisimman aikaisin.

Projektit, joissa testaus on työlästä tehdä käsin, on usein automatisoitu. Mikäli testitapauksia on paljon ja niissä käytettävä data on laajaa, on projektissa käytössä usein automaatiotestaus.

Joissakin projekteissa testaukseen halutaan tarkoituksella panostaa ja asiakkaalla on halua sekä varaa kehittää myös testausta. Saattaa kuitenkin olla, että asiakkaan oma IT ja erityisesti testauksen parissa toimiva henkilöstökapasiteetti on rajallinen.

Automaation onkin nähty olevan iso apu regressiotestauksessa projekteissa, joissa ohjelmiston kehityskaari on pitkä, ja uusia versioita ohjelmistosta julkaistaan säännölliseen tahtiin. Regressiota varten suunnitellun testikierroksen suorittaminen manuaalisesti saattaa kestää useita tunteja, ja mikäli testaaja on joka kierroksella sama henkilö, saattaa jopa virheiden huomioimistarkkuus kärsiä. Näin ollen onkin huomattu tehokkaaksi vaihtoehdoksi suorittaa regressiotestaus automaation avulla. Testit käynnistetään edellisenä iltana, ja yön aikana automaatiotyökalu ehtii hyvin suorittamaan koko testisetin. Aamulla on testaushenkilökuntaa varten valmiina työkalun tuottama raportti ohjelmistoversion tilasta, ja sen perusteella voidaan heti alkaa suunnittelemaan seuraavia työvaiheita.

Projektit ovat olleet ammattilaiskäyttöön tarkoitettuja verkkopalveluja, joissa on useita kyselylomakkeita täytettävänä. Tällaisen toiminnallisuuden testaaminen automaatiotyökalulla onkin suhteellisen yksinkertainen toteuttaa ja tällaisissa testitapauksissa automaatiotyökalu onkin mielestäni parhaimmillaan. Lomakkeiden kaikkien kenttien ja painikkeiden läpikäynti on usein manuaalisesti työlästä eikä välttämättä kovin mielekästä, joten automaation käyttäminen siihen tuntuu luontevalta.

Mainituissa projekteissa on testauksen aiheuttamat kulut saatu pidettyä hyvin budjetissa ja automaation tuottamaan apuun oltu tyytyväisiä. Luotuja automaatiotestejä ja niiden rakennetta on melko helppo uudelleenkäyttää, joten kerran hyvin suunnitelluista automaatiotesteistä hyötyy helposti myös seuraavat projektit.

7 POHDINTA

Tämän opinnäytetyön tarkoituksena oli selvittää, mitä rajoituksia asiakasprojekteista löytyy heidän järjestelmänsä testaamisen automatisoimiseen. Tutkimuksena aikana huomattiin moneen kertaan, että järjestelmän automatisoiminen ei ole yksiselitteinen asia. Siihen ei ole suoraan mitään ohjekirjaa, vaan se riippuu paljonkin projektiin ja järjestelmään liittyvistä tahoista. Toisissa projekteissa automatisoiminen olisi hyvinkin suositeltava vaihtoehto, mutta sitten taas testausympäristö tai tietotaito eivät siihen riittäisi.

Jokainen polku tuntui päättyvän samaan ongelmaan: rahan puutteeseen. Jokainen asiakas varmasti haluaisi, että heidän oma järjestelmänsä olisi testattu jokaisessa mahdollisessa vaiheessa, jokaisella mahdollisella sekä mahdottomallakin tavalla. Kun tutkittiin, että onko automaation käyttö ohjelmistotestauksessa kannattavaa, siitäkin alan ihmisillä tuntui olevan erilainen käsitys, ihan työntekijän näkökulmasta riippuen. Kannattavuuden arvioimiseen liittynee paljon ihmisen oma käsitys automatisoimiseen tarvittavasta työstä sekä vaatimuksista. Ne, joille koodaus ja ympäristöjen pystyttäminen oli tavallista arkipäivää, olivat positiivisemmin mielin automatisoinnin suhteen. Mikäli taas asiasta ei ollut niin selvää kuvaa, manuaalinen testaus tuntui helpommalta tavalta sekä vähemmän työläältä.

Osalla asiakkaista on tarvittavaa tietotaitoa tehdä heidän testauksensa itse. Tämä on todella hyvä asia, koska asiakas kuitenkin tietää, kuinka järjestelmän tulisi toimia ja mitä sillä halutaan tehdä. Toki tällaisessakin tilanteessa järjestelmää on testattu useammassakin kohdassa sen kehitysvaihetta.

Testaus yleisesti ottaen on edelleen osittain ”välttämätön paha”. Useasti mikäli projektissa on ongelmia aikataulujen kanssa, on testaus se osio, mistä nipistetään. Kuten muutamassakin kohtaa tätä opinnäytetyötä on todettu, se ei todellakaan kannattaisi. Virheen löytäminen mahdollisimman aikaisessa vaiheessa olisi todella tärkeää.

Meidän täytyy omalta osaltamme nostattaa testauksen imagoa asiakkaiden silmissä ja tuoda ilmi sen tärkeellisyyttä. Toisaalta meidän on kuitenkin syytä myös

lisätä testauksen osuutta asiakkaalle tehtäviin tarjouksiin. Vaikka asiakas ei välttämättä ostaisikaan virallista testausta, on toimitetun tuotteen laatu silti tärkeä asiakkuuden jatkuvuuden kannalta.

Joillakin asiakkailla vaatimusmäärittelyihin vaikuttavat suoraan erilaiset lait ja määräykset. Näiden perusteella vaaditaan jo tietynlainen taso järjestelmältä.

Tutkimuksen perusteella ei kehitetty mitään järjestelmää, jolla voisi eri projekteja tutkia ja vertailla automaation näkökulmasta. Jokainen asiakasprojekti on niin erilainen, ettei mikään valmis pohja välttämättä oikein toimisi. Perustimme kuitenkin erillisin oman sisäisen projektimme, jossa teemme tarkempaa määrittelyä testauksellemme yleisesti ja pyrimme tätä kautta yhtenäistämään asiakasprojekteissa läpi viedyt käytännöt.

LÄHTEET

Asq.com. N/A. COST OF QUALITY (COQ) https://asq.org/quality-resources/cost-of-quality Luettu 15.11.2019

Edureka! 2017 Selenium tutorial for beginners. Video. Youtube-videopalvelu, julkaistu 14.3.2017 https://www.youtube.com/watch?v=5FUdrBq-WFo Katsottu 19.3.2019

Edureka! 2019. How to Build a Test Automation Strategy? Video. Youtube-videopalvelu, julkaistu 20.3.2019 https://www.youtube.com/watch?v=jlJbg9bBEPs&list=PL9ooVrP1hQOFP9H8Y15DVGCA6GavhgJ8a&index=22 Katsottu 4.4.2019

Edureka! 2019 Software Testing Tutorial For Beginners. Video. Youtube-videopalvelu, julkaistu 11.2.2019 https://www.youtube.com/watch?v=T3q6QcCQZQg Katsottu 26.3.2019

Guru99. 2011. What is Non-Functional Testing? Video. Youtube-videopalvelu, julkaistu 5.8.2011 https://www.youtube.com/watch?v=n2A9OakDYcY&list=PLDC2A0C8D2EC934C7&index=9) Katsottu 14.11.2019

Guru99. 2013. Automation Testing Tutorial for Beginners. Video. Youtube-videopalvelu, julkaistu 22.1.2013 https://www.youtube.com/watch?v=RbSlW8jZFe8 Katsottu 27.3.2019

Guru99. 2014. What is integration testing? Video. Youtube-videopalvelu, julkaistu 3.6.2014 https://www.youtube.com/watch?v=QYCaaNz8emY Katsottu 13.11.2019

Guru99. 2017. Software testing tutorials for beginners. Video. Youtube-videopalvelu, julkaistu 13.3.2017 https://www.youtube.com/watch?v=goaZTAzsLMk Katsottu 18.3.2019

Haikala, I & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum

IBM SSI https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/ Luettu 14.11.2019

Jyväskylän Yliopisto. N/A. Testaus http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus Luettu 15.11.2019

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo

Lehto T. 2005. Ohjelmistojen testaus. Luettu 15.3.2019 https://www.tivi.fi/Arkisto/2005-04-18/Ohjelmistojen-testaus-3089028.html

Pressman R.S. 2010. Software Engineering : A Practitioner's Approach http://dinus.ac.id/repository/docs/ajar/RPL-7th_ed_software_engineering_a_practitioners_approach_by_roger_s._pressman_.pdf Luettu 14.11.2019

Pulkkanen A. N/A. Sinunkin kannattaa valita: 6 yleistä menetelmää projektityöhön. https://www.agendium.com/post/agile-waterfall-kanban-6-projektinhallintamenetelmaa Luettu 26.4.2019

Sanjeev Deshmukh. 2015. https://www.slideshare.net/Sanjeev-Deshmukh/mel420-m3costofquality Luettu 14.11.2019

Smartbear. N/A. 6 Ways to Measure the ROI of Automated Testing https://smartbear.com/resources/ebooks/6-ways-to-measure-the-roi-of-automated-testing/ Luettu 15.11.2019

Swiss TestingDay. 2012. Test Automation - 10 (sometimes painful) Lessons Learned. Video. Youtube-videopalvelu, julkaistu 16.3.2012 https://www.youtube.com/watch?v=tJ0O8p5PajQ Katsottu 26.3.2019

Tricentis Report. 2019. Software failure caused $1.7 trillion in financial losses in 2017 https://www.tricentis.com/wp-content/uploads/2019/01/Software-Fails-Watch-5th-edition.pdf Luettu 14.11.2019

VALA Group. 2017. Ohjelmistotestauksen trendit sekä testiautomaation höydyt ja kompastuskivet vuonna 2017. https://www.valagroup.com/fi/2017/10/ohjelmistotestauksen-trendit-seka-testiautomaation-hyodyt-ja-kompastuskivet-vuonna-2017/ Luettu 21.5.2019

LIITTEET

Liite 1. Vaatimusmäärittelyn tärkeys projektissa.