Opinnäytetyö (AMK / YAMK)

Teknologiaosaamisen johtaminen

2018

Jarmo Nieminen

TESTAUSPROSESSIN KEHITTÄMINEN

– Sisäisten ohjelmistojen kehitys

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (YAMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Teknologiaosaamisen johtaminen

2018 | 54

Jarmo Nieminen

TESTAUSPROSESSIN KEHITTÄMINEN

- Sisäisten ohjelmistojen kehitys

Opinnäytetyön tarkoituksena oli kartoittaa toimeksiantajan sisäisen ohjelmistokehityksen testausprosessin nykytila ja laatia paremmin nykyvaatimukset täyttävä ja kehitysprosessiin soveltuva testausprosessi. Nykyinen testausprosessi on melko suppea ja testauksen kattavuus ja menetelmät riippuvat pitkälti kehittäjästä. Lisäksi testaaminen suoritetaan useasti vasta kehityksen loppuvaiheessa.

Työn teoreettisena viitekehyksenä on katsaus ohjelmistotestauksen menetelmiin ja mitä kaikkea testaus pitää sisällään. Teoriaosuudessa käydään pääpiirteittäin läpi eri testausmenetelmiä ja erilaisia malleja testauksen liittämiseksi osaksi testausprosessia. Lisäksi tutkitaan testauksen suunnittelua ja dokumentaatiota sekä testaustyökaluja.

Tutkimuksen aineiston keruu toteutettiin haastatteluin ja havainnoiden. Haastattelujen kautta testausprosessista pystyttiin koostamaan näkemys useamman eri henkilön ajattelun ja kokemusten pohjalta. Aineiston laadun parantamiseksi mukaan otettiin myös havainnoimalla tehtävää tutkimusta, jolloin prosessista voidaan löytää myös asioita, joita haastateltavat eivät huomioineet. Haastattelut suoritettiin ohjelmistokehitykseen liittyville ja erityisesti ohjelmistokehityksen operatiivisessa toiminnassa oleville henkilöille.

Työssä kehitettiin yritykselle uudistettu testausmenetelmä ja sen prosessikuvaus, jossa prosessi on kuvattuna alusta loppuun. Prosessikuvauksesta käy ilmi prosessin eri vaiheet, prosessin eri vaiheissa käytettävät testausmenetelmät sekä eri vaiheiden vastuuhenkilöt. Uudistetussa prosessissa testaus on liitetty vahvemmin osaksi kehitysprosessia ja testaamista suoritetaan kehityksen kaikissa vaiheissa, jolloin virheet löydetään mahdollisimman aikaisin. Lisäksi toteutettiin tarvittavat dokumenttipohjat prosessin eri vaiheisiin. Toteutettu tutkimus yhdessä prosessikuvauksen kanssa tarjoaa toimeksiantajalle ratkaisumallin testausprosessin kehittämiseen.

ASIASANAT:

Ohjelmistotestaus, prosessin kuvaus, kvalitatiivinen tutkimus

MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Technological Competence Management

2018 | 54

Jarmo Nieminen

TESTING PROCESS DEVELOPMENT

- Internal software development

The aim of the present master's thesis was to explore the current state of the testing process of the client's internal software development and to develop a better testing process that meets today's requirements as well as to improve its suitability for the software development process. The current testing process is rather limited and the testing coverage and the methods are dependent on the developer. In addition, testing is performed only in the final stages of the development.

The theoretical framework of this thesis is an overview of the software testing methods and the elements that are included in the testing. The theoretical part is an outline of various testing methods and different models for incorporating testing into the testing process. In addition, testing design and documentation as well as testing tools are being discussed.

The data for this qualitative research were collected with interviews and observations. Through the interviews, it was possible to sum up the views on the basis of thinking and experiences from more than one person involved in the test process. In order to improve the quality of the available information, data gathered through observation were taken into account in the research which made it possible to find the things that the interviewees did not take into account. The interviews were conducted with people in software development and especially in the operational development of software development.

As part of the study, a revised testing method and complete testing process description has been developed. The process description shows the different key phases, the test methods that are used at different stages and the persons who are responsible for the different phases. In the improved process, testing is more closely involved in the development process and testing is carried out at all stages of development thus revealing errors as early as possible. In addition, the necessary documentation was implemented at different stages of the process. The research carried out together with the process description provides the client with a solution for the development of the testing process.

KEYWORDS:

Software testing, process description, qualitative research

SISÄLTÖ

LIITTEET

KUVAT

KUVIOT

TAULUKOT

SANASTO

Analyysi	Tutkittavana olevan asian määrittäminen
Dokumentaatio	Ylös kirjattu tieto
Eclipse	Ohjelmointiympäristö (IDE, integrated development environment)
Ehtolause	Valintarakenne ohjelmoinnissa
End-to-end -testaus	Järjestelmän kokonaistoimivuuden testaaminen
Hyväksymistestaus	Valmiin ohjelmistokokonaisuuden testaus tilaajan toimesta
Integraatiotestaus	Ohjelmiston komponenttien yhteen toimivuuden testaamismenetelmä
Iteraatio	ohjelmistokehitykseen liittyvä pieni osaprojekti
Järjestelmätestaus	Valmiina ohjelmistokokonaisuuden testaus kehitystiimin toimesta
Komponentti	Ohjelmiston osa
Lasilaatikkotestaus	Testaustapa, jossa testaus suoritetaan tarkastelemalla järjestelmän sisäistä toimintaa (Myers, Sandler & Badgett 2011, 10)
Mustalaatikkotestaus	Testaustapa, jossa testaus suoritetaan ilman tietoa järjestelmän sisäisestä toiminnasta (Myers, Sandler & Badgett 2011, 8)
NetBeans	ohjelmointiympäristö (IDE, integrated development environment)
PhpStorm	ohjelmointiympäristö (IDE, integrated development environment)
Prosessi	Sarja suoritettavia toimenpiteitä (JUHTA 2012, 2)
Prosessi kaavio	Graafinen esitys prosessista (JUHTA 2012, 2)

Prosessin omistaja	Prosessin toiminnasta, tuloksesta ja kehittämisestä vastuussa oleva toimija (JUHTA 2012, 2)
Resurssi	Yrityksen käytössä oleva työvoima
RUP-malli	Ohjelmistotuotannon toimintamalli, joka pyrkii hyödyntämään ohjelmistokehityksen hyviä käytänteitä (RUP, Rational unified process)
Testitapaus	Yksittäinen tilanne testattavassa järjestelmässä
Toistorakenne	Ohjelmoinnin rakenne jota toistetaan, kunnes ehto täyttyy
Tukiprosessi	Avustaa ydinprosesseja ja luo edellytykset niiden toiminnalle. Liittyy yleensä sisäisiin asiakkaisiin (JUHTA 2012, 2)
UML-Kaavio	Graafinen kuvaus ohjelmistosta tai ohjelmiston osasta
V-malli	Vesiputousmallista johdettu prosessimalli, jossa jokaiselle vaiheelle on oma testausmenetelmä
Vaatimusmäärittely	Kuvaus ohjelmistoprojektin tavoitteista ja vaatimuksista.
Vesiputousmalli	Vaiheellinen prosessimalli
Ydinprosessi	Keskeinen osa organisaation toimintaa. Liittyvät suoraan ulkoisiin asiakkaisiin
Yksikkötestaus	Yksittäisen komponentin tai funktion testausmenetelmä
XUnit –ohjelmistokehys	Yksikkötestauksen suorittamiseen liittyvä ohjelmistokehys

1 JOHDANTO

1.1 Tausta

Toimeksiantaja on markkinoinnin palveluita tarjoava yritys. Toimeksiantajan päivittäisen toiminnan tukena on useista eri työkaluista koostuva ohjelmistokokonaisuus, jonka kehittäminen tapahtuu yrityksen sisäisinä projekteina. Ohjelmistokokonaisuuden käyttäjämäärän kasvaessa ja kehitysprojektien edetessä on ilmennyt tarve testausprosessin kehittämiselle.

Ohjelmistokokonaisuuden tavoitteena on ollut helpottaa työntekijöiden päivittäistä tekemistä, nopeuttaa ja suoraviivaistaa työn suorittamista ja vähentää sekä nopeuttaa työhön liittyvää, sekundaarista tekemistä. Työkalujen käyttöönotolla on lisäksi pyritty varmistamaan ja tehostamaan toimeksiantajan eri prosessien keskinäistä toimivuutta. Vaikka työkalut on tehty pelkästään sisäiseen käyttöön, toimintavarmuudella on henkilökunnan lisäksi vaikutusta myös asiakkaisiin, mikäli työkalut eivät toimi suunnitellulla tavalla.

Työkalujen kriittisyys päivittäisessä toiminnassa asettaa omat haasteensa testaamiselle. Toimeksiantajan kasvu on viime kuukausina ollut huomattavaa ja tarjottavien palveluiden sekä henkilöstön määrä on kasvanut nopeasti, mistä johtuen myös työkaluja on kehitetty nopeaa tahtia vastaamaan käyttäjien tarpeita. Laadukas ohjelmisto on toimeksiantajan tapauksessa ehdoton edellytys. Pahimmassa tapauksessa ohjelmiston tuotantoversiosta löytyvä virhe, joka lamaannuttaa osan toiminnoista, aiheuttaa rahallisia menetyksiä ja töiden viivästymistä. Toimintavarman ohjelmiston lisäksi hyvin toteutetulla testauksella on vaikutusta koko ohjelmistokehitysprosessiin. Laadukkaasti ja kattavasti suoritettu testaus kehityksen eri vaiheissa, mahdollistaa resurssien paremman hyödyntämisen ja laadukkaan ohjelmiston.

Työssä selvitetään toimeksiantajan sisäisten työkalujen testauksen nykytila ja toteutetaan uudistettu ja paremmin nykyvaatimukset täyttävä testausprosessi.

1.2 Työn tavoite & rajaus

On tärkeää, että käytössä olevat työkalut toimivat suunnitelmien mukaisesti eri tilanteet ja käyttäjätoimet huomioiden. Kehityksen aikana suurimmaksi ongelmaksi on koettu selkeän testausmallin ja toimintatapojen puuttuminen; kuinka voidaan varmistaa toisistaan riippuvien työkalujen 100% toimivuus ja testata uusien tai muokattujen ominaisuuksien vaikutus olemassa oleviin toiminnallisuuksiin? Millä menetelmillä työkaluja testataan, miten testaus suunnitellaan ja dokumentoidaan? Kuka on vastuussa testaamisen eri vaiheista ja miten testaamista voidaan tehostaa ja automatisoida?

Työn tavoitteena oli antaa vastaus edellä kuvattuihin ongelmiin ja kehittää ehdotus uudistetusta testausprosessista, joka tukee kehitystyötä, parantaa ja tehostaa työkalujen testaamista kehityksen eri vaiheissa sekä varmistaa julkaistujen ominaisuuksien korkean laadun. Nykytilannetta tutkittiin haastatteluin sekä tekemällä havaintoja työmenetelmistä, että olemassa olevista dokumentaatioista.

Työssä kartoitettiin testauksen nykytilaa ja siihen liittyviä tarpeita. Haastattelut suoritettiin tuotekehitykseen liittyville ja erityisesti tuotekehityksen operatiivisessa toiminnassa oleville henkilöille. Työ rajattiin koskemaan vain sisäisessä kehityksessä olevien työkalujen testausprosessia ja uusien menetelmien ja työkalujen käyttöönotto rajattiin työn ulkopuolelle. Työssä ei oteta kantaa kolmannen osapuolen työkalujen yhteensopivuuteen sisäisesti kehitettyjen työkalujen kanssa. Sikäli kun työkalujen toimivuudella on vaikutusta käyttäjiin ja välillisesti myös asiakkaisiin, testausprosessin toimivuutta tarkastellaan vain kehitykseen liittyvien tahojen näkökulmasta.

Työn teoriaosuuden viitekehyksenä on katsaus ohjelmistotestauksen menetelmiin ja siihen, mitä kaikkea testaus pitää sisällään. Teoriaosuudessa käydään läpi pintapuolisesti läpi eri testausmenetelmiä ja erilaisia malleja testauksen liittämiseksi osaksi testausprosessia. Lisäksi tutkitaan testauksen suunnittelua, dokumentaatiota sekä testaustyökaluja.

Työssä kehitettiin yritykselle uudistettu testausmenetelmä ja sen prosessikuvaus, jossa prosessi on kuvattuna alusta loppuun. Prosessikuvauksesta käy ilmi prosessin eri vaiheet, sen eri vaiheissa käytettävät testausmenetelmät sekä eri vaiheiden vastuuhenkilöt. Lisäksi toteutettiin tarvittavat dokumenttipohjat prosessin eri vaiheisiin. Toteutettu tutkimus yhdessä prosessikuvauksen kanssa tarjoaa toimeksiantajalle ratkaisumallin testausprosessin kehittämiseen.

Uudistetussa prosessissa testaus on liitetty vahvemmin osaksi kehitysprosessia ja testaamista suoritetaan kehityksen kaikissa vaiheissa, jolloin virheet löydetään mahdollisimman aikaisin. Mitä aiemmin virheet löydetään ja korjataan, sitä kustannustehokkaampaa se on.

2 OHJELMISTOTESTAUKSEN TEORIAA

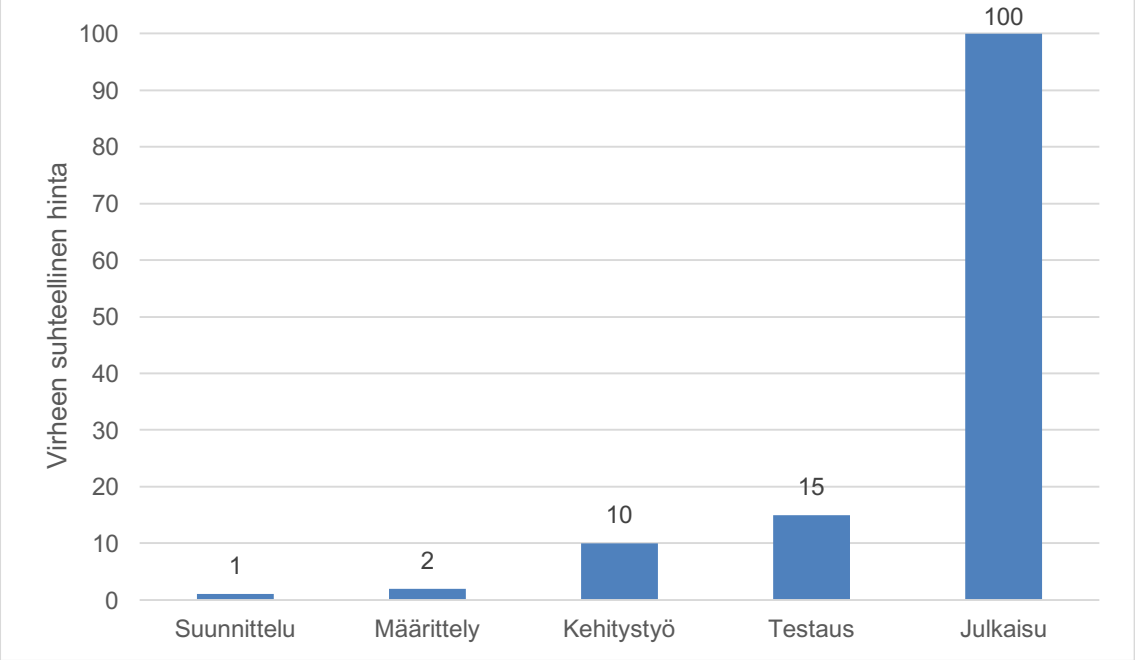
Ohjelmistotestaus on yksi ohjelmistosuunnittelun osa-alue. Sillä varmistetaan, että kehitettävästä ohjelmistosta tulee suunnitelmien mukainen ja kaikki ominaisuudet toimivat niin kuin on tarkoitettu. Sen tehtävänä on varmistaa, että toteutus vastaa suunniteltua ja ohjelmisto täyttää asiakkaan tarpeet. Testaus auttaa myös tunnistamaan ne kohdat ja tilanteet, joissa toteutus poikkeaa suunnitellusta.

Testausta suoritetaan eri vaiheissa ohjelmiston elinkaarta. Ohjelmiston komponentteja kehitettäessä suoritetaan yksikkötestausta. Komponenttien yhdistämisen jälkeen suoritetaan integraatiotestausta ja kun ohjelmistosta on koostettu valmis kokonaisuus, suoritetaan järjestelmätestaus. Ohjelmistokehitystä tehdään useasti jakamalla kehitys pienempiin osiin, iteraatioihin. Yhden iteraation voidaan ajatella olevan oma pieni julkaistavissa oleva projekti, johon liittyy aina myös testauksen eri vaiheet. Ohjelmiston julkaisun jälkeen, kun on siirrytty ohjelmiston ylläpitovaiheeseen, muutosten, korjausten ja päivityksen toteutuksen yhteydessä suoritetaan regressiotestausta, jolla varmistaa, että tehdyt muutokset eivät ole rikkoneet olemassa olevia toiminnollisuuksia (Tilley & Parveen 2012, 1).

Usein testaus mielletään yhdeksi työvaiheeksi ohjelmistokehityksessä. Todellisuudessa testaus koostuu useasta eri työvaiheesta ja tehtävästä, kuten suunnittelusta, ohjelmoinnista ja dokumentoinnista. Edellä mainittujen lisäksi testaukseen saattaa liittyä myös haastatteluja, riippuen testauksen toteuttamisesta. Haastatteluja voidaan hyödyntää esimerkiksi käytettävyystestauksen yhteydessä. Testaus onkin laajimpia ja haastavimpia osa-alueita ohjelmistosuunnittelun projekteissa ja se aiheuttaa merkittäviä haasteita varsinkin laajamittaisissa järjestelmissä. Lähes jokaisessa projektissa toistuvia haasteita on mm. resurssien riittävyys, päätökset siitä mitä testataan ja mitä ei testata sekä tietenkin testaukselle varattu aika (Tilley & Parveen 2012, 1).

Testaus on tärkeää toteuttaa huolella ja sille pitäisi olla varattuna tarpeeksi resursseja sekä aikaa kehityksen yhteyteen. Useasti ohjelmistoprojekteissa käy kuitenkin niin, että testaus suoritetaan kiireellä ja pienin resurssein projektin loppuvaiheessa. Testaus suoritetaan lähes ilman minkäänlaista suunnittelua ja toteutetaan muutaman henkilön kokeilemana, jotta kaikki näyttäisi toimivan (Kasurinen 2013, 16).

Projektin viivästyessä tai budjetin huomattavasti ylittyessä, testaus on usein se, josta tingitään, jotta projekti voidaan kustannustehokkaasti ja ajallaan saada päätökseen. Testauksesta tinkiminen on kuitenkin haitallinen menettelytapa. Virheen korjaaminen suunnitteluvaiheessa tai testauksen yhteydessä maksaa murto-osan verrattuna julkaisun jälkeen tehtyyn korjaukseen; suunnittelu- ja määrittelyvaiheessa noin 1-2 prosenttia ja testauksen yhteydessä n. 15 prosenttia (kuvio 1). Lisäksi on havaittu korrelaatio testauksen ja kannattavuuden välillä; kattavasti tuotteensa testanneet yritykset saavat tuotteelleen paremman katteen verrattuna yrityksiin, joissa testaus on vähäistä ja huonolaatuista. Lisäksi huonosti testattujen ohjelmistojen tarjoamisesta saattaa seurata asiakaskatoa ja pahemmissa tapauksessa maine epäluotettavia ohjelmistoja tekevänä yrityksenä (Kasurinen 2013, 12).



Kuvio 1. Eri kehitysvaiheissa löydetyn virheen suhteellinen hinta (Kasurinen 2013, 18)

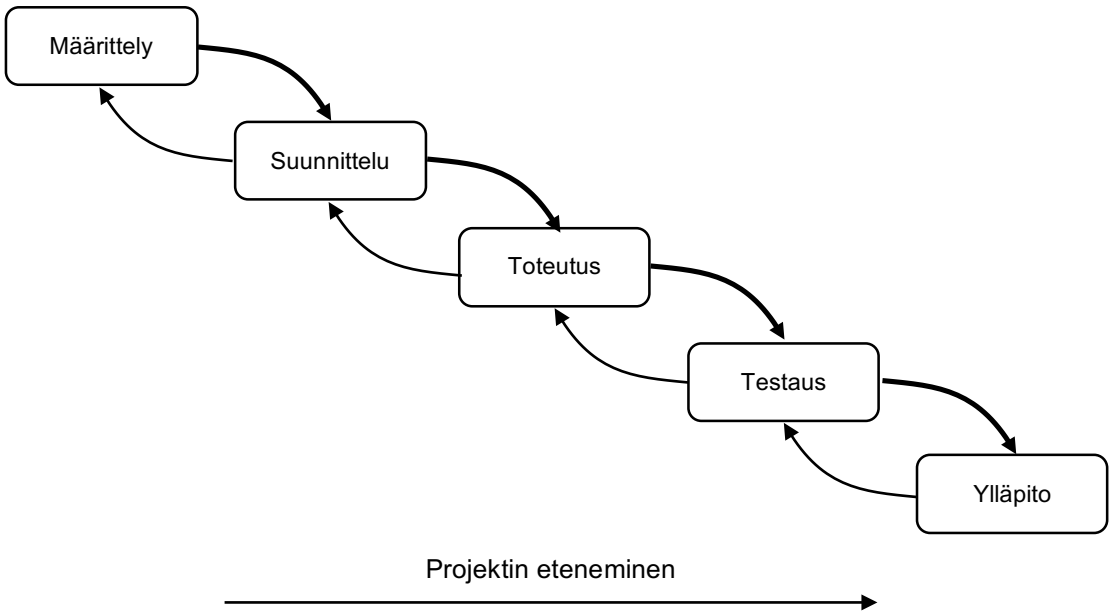
2.1 Testausmallit

Testaus voidaan toteuttaa eri projektimallien pohjalta. Yleisenä mielikuva on, että testaus on yksittäinen vaihe vaiheittain etenevässä ohjelmistoprojektissa. Tätä mallia kutsutaan vesiputousmalliksi, jossa prosessi etenee vaihe vaiheelta alaspäin. Malli on kuitenkin jo melko vanhanaikainen tapa toteuttaa testaus, koska se vaatii mm. lähes täydellistä vaatimusmäärittelyä. Paremmaksi tavaksi testauksen toteuttamiseksi on todettu V-malli.

V-malli toimii vesiputousmallin tavoin, mutta siinä testausta suoritetaan mallin kaikissa vaiheissa ja testauksella on muutenkin suurempi rooli osana kehitystä. Vesiputousmallin tavoin myös V-mallin heikkoutena on testauksen alkaminen liian myöhään. Testauksen kannalta olisi kuitenkin järkevää miettiä, kuinka testaus saadaan otettua mukaan jo alkuvaiheessa kehitystä. Kuinka järjestelmä kannattaa rakentaa, jotta ohjelmiston vaatimukset voi todentaa ja testaus voidaan aloittaa ennen kuin suurin osa kehityksestä on valmiina? Edellä kuvattu onkin otettu lähtökohdaksi RUP (Rational Unified Process) -malliin, jossa ohjelmaa testataan koko kehityksen ajan. RUP-mallilla tarkoitetaan yhtenäistettyä prosessia ja se perustuu ohjelmistotuotannossa hyväksi havaittuihin ajatuksiin. Näitä ajatuksia ovat esimerkiksi vaiheittainen kehittäminen, UML-kaavioihin perustuva visuaalinen suunnittelu, järjestelmällinen vaatimustenhallinta sekä jatkuva laadunvalvonta (Kasurinen 2013, 12-16).

2.1.1 Vesiputousmalli

Vesiputousmallissa ohjelmistoprojekti etenee vaiheittain ja se noudattaa yleensä tiettyä kaavaa; vaatimusmäärittely, suunnittelu, toteutus, testaus ja julkaisu (kuva 1). Vesiputousmallissa ohjelmiston kehittäminen nähdään askel askeleelta eteenpäin kulkevana prosessisina. Isojen ongelmien ilmetessä prosessissa voidaan palata aiempaan vaiheeseen. Tavoitteena kuitenkin on, että niin ei jouduttaisi missään vaiheessa tekemään (Kasurinen 2013, 13-14). Edellä kuvattuun on myös vastakkainen ajatus. Vuonna 1970 Royce julkaisi, vesiputousmallia käsittelevän artikkelin, jossa hän pitää mallin tärkeimpänä ominaisuutena taaksepäin tapahtuvia iteraatioita. Sallimalla iterointi ja vaiheiden päällekkäin toteuttaminen, vesiputousmallistakin saadaan toimiva malli moneen eri tilanteeseen (Royce 1970, Haikala & Mikkonen 2011,37 mukaan).



Kuva 1. Vesiputousmalli (Kasurinen 2013, 13)

Vaatimusmäärittelyssä kerätään yhteen kaikki saatavilla oleva tieto, kuten markkina-analyysi, kannattavuuslaskelmat ja lista asiakastarpeista. Näiden tietojen pohjalta tehdään lista vaatimuksista, jotka ohjelman on täytettävä. Vaatimuslistan perusteella tehdään suunnitelma, jossa kuvataan ohjelmiston rakenne ja toiminnallisuudet sekä tehdään loppuprojektin kattava projektisuunnitelma. Projektisuunnitelman avulla toteutetaan tarvittavat ohjelmistokomponentit, jotka lopuksi yhdistetään yhdeksi kokonaisuudeksi. Ohjelmistokokonaisuuden valmistumisen jälkeen aloitetaan testaus. Testausvaihetta toistetaan, kunnes kokonaisuudessa ei ole merkittäviä virheitä ja se täyttää vaadittavat toiminnallisuudet ja asiakkaan tarpeet. Testauksen jälkeen ohjelmisto otetaan käyttöön ja se siirtyy ylläpitoon. Ylläpitovaiheen aikana ohjelmistoon voidaan tarvittaessa tehdä korjauksia.

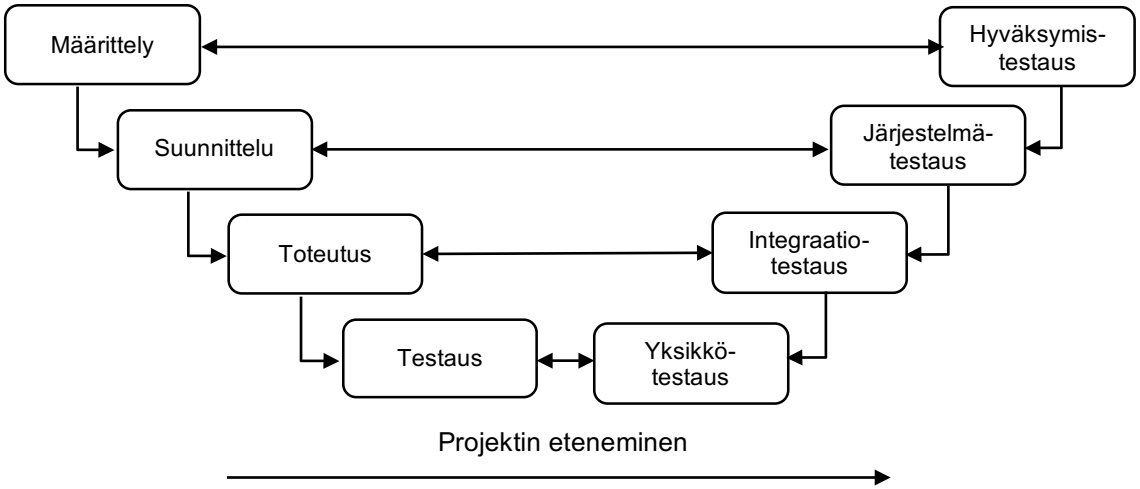
Ihanteellisessa tilanteessa vesiputousmalli toimii kuvatulla tavalla. Todellisuudessa usein on kuitenkin niin, että tietyn vaiheen suorituksen yhteydessä havaitaan virheitä aiemmassa vaiheessa, jolloin prosessissa on palattava taaksepäin. Lisäksi ongelmaa aiheuttaa mallin tapa kiinnittää tarkistukset tiukasti vaiheiden rajapinnoille sekä olettaen vaiheen loppudokumentin olevan syöte seuraavalle vaiheelle. Tästä johtuen aiempaan vaiheeseen palaaminen on työlästä ja resursseja kuluttavaa, koska se yleensä edellyttää kaikkien vaiheiden ainakin osittaista uudelleen toteutusta (Pohjonen 2002, 40). Pidemmissä projekteissa haasteita aiheuttaa myös takarajojen vähyys. Usein takarajat on

määritetty vain muutamien työvaiheiden valmistumiselle. Tästä johtuen projektin etenemistä ja aikataulussa pysymistä on hankala hahmottaa. Samoin resurssien tehokas käyttö on haastavaa. Kehittäminen menee helposti siihen, että siihen sisältyy turhan työn tekemistä sekä loppua kohden kiihtyvää työtahtia (Kasurinen 2013, 22).

Ongelmaksi on koettu myös etukäteen tehtävä vaatimusmäärittely. Heti alkuvaiheessa kattavasti tehty vaatimusmäärittely on useissa projekteissa lähes mahdotonta tai se vie huomattavan määrän resursseja. Harvoin on mahdollista etukäteen tietää tarkkoja vaatimuksia ohjelmalle. Puutteellista vaatimusmäärittelyä on hankala täydentää kesken projektin, sillä varsinaisia tuloksia pystytään esittelemään asiakkaalle vasta myöhäisessä vaiheessa projektia. Tulosten esittelyn tavoin, myös testaus ajoittuu projektin loppuvaiheeseen, jolloin suunnittelu ja kehitystyö on pääosin tehty (Pohjonen 2002, 40).

2.1.2 V-malli

Vesiputousmallia parempi malli kehitystyön ja testaamisen rajapinnoille on V-malli. V-mallin perusajatus on sama kuin vesiputousmallissa, mutta testausta esiintyy mallin eri vaiheissa (Kuva 2). V-mallissa testaus ei ole erillinen työvaihe, vaan jokaisessa vaiheessa on oma erillinen testauskategoria. Ohjelmointivaiheen toiminnot testataan yksikkötestauksella, suunnitelman toteutuminen integraatiotestauksella, määrittelyjen toteutuminen järjestelmätestauksella ja kokonaistoimivuus sekä vaatimusten mukaisuus hyväksymistestauksella. Kun kaikki eri testivaiheet on suoritettu onnistuneesti, ohjelmisto on valmis julkaistavaksi (Kasurinen 2013, 14).



Kuva 2. Esimerkki V-mallista ja vaiheisiin liittyvistä testausmenetelmistä (Mili 2015, 32)

Edellä kuvattu mahdollistaa testauksen suunnittelun aloittamisen projektin yksittäisen vaiheen valmistuttua. Esimerkiksi hyväksymistestauksen suunnittelu voidaan aloittaa heti vaatimusmäärittelyn ja ohjelmiston teknisten vaatimusten valmistuttua. Vaatimusmäärittelyä ja teknisiä tietoja hyödyntäen suunnitellaan hyväksymistestauksen testauskriteereitä ja vaatimuksia, mitä testauksen pitää täyttää (Mili 2015, 32).

Hyväksymistestauksen suunnittelun tavoin järjestelmätestauksen suunnittelu voidaan aloittaa heti ohjelmistoarkkitehtuurin perusajatuksen valmistuttua. Ohjelmistoarkkitehtuurissa kuvattujen tietojen pohjalta määritellään testidata sekä testien mallit. Määrittelyn tekemistä varten tiedossa pitää olla kuvaus ohjelmisto toiminnasta sekä tieto ohjelmistossa olevista toiminallisuuksi. Järjestelmätestauksen suunnittelussa lähtökohtana on, että löydettäisiin mahdollisimman suuri osa mahdollisista virheistä, jotta ne eivät ilmenisin enää hyväksymistestauksessa (Mili 2015, 32).

Hyväksymis- ja järjestelmätestauksen suunnittelussa keskitytään ulkoisiin tekijöihin ja niiden vaikutukseen ohjelmistossa. Integraatio- ja yksikkötestauksen suunnitelmat taas keskittyvät ohjelmiston sisäiseen käyttäytymiseen. Integraatiotestauksen suunnittelua voidaan tehdä samaan aikaan yksikkötestien suunnittelun kanssa. Integraatiotestauksen suunnitelmissa määritellään tavat ja vaatimukset, miten voidaan todentaa yksittäisten komponenttien ja osakokonaisuuksien keskinäinen toimivuus ja yhteen sopivuus, kuten on suunniteltu. Yksikkötestien suunnittelussa fokus on yksittäisen moduulin tai toimenpiteen testaamisessa (Mili 2015, 32).

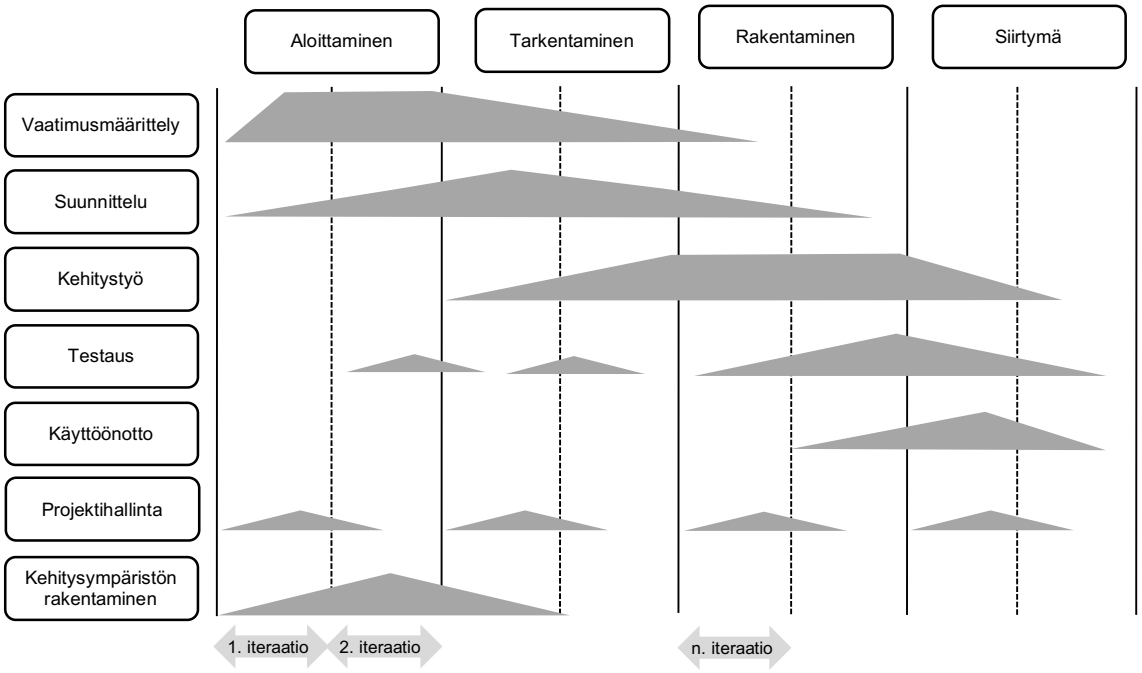
Eri vaiheiden testauksesta huolimatta, myös V-mallin ongelmana on itse testauksen aloittaminen projektin loppuvaiheessa. Vaikka testaus on suunniteltu jo hyvissä ajoin, on esimerkiksi vaatimusmäärittely ja järjestelmäarkkitehtuuri laadittu jo paljon ennen testauksen aloittamista. Tärkeää olisikin miettiä jokaisessa vaiheissa, miten vaiheiden onnistuminen voidaan todentaa ja miten ohjelmisto on mahdollista toteuttaa niin, että sen osakokonaisuuksia voitaisiin testata jo ennen kuin ohjelmisto on kokonaan valmis (Kasurinen 2013, 14).

2.1.3 RUP-malli

Kuten jo aiemmissa kappaleissa todettiin, testausmallien ongelmana on testauksen aloittaminen projektin loppupuolella. Tämä ongelma on otettu huomioon RUP (Rational Unified Process) -mallin kehityksessä. Mallin lähtökohtana on ollut yhdistää kuusi ohjelmistotuotannossa hyvin toimivaa käytäntöä (Kasurinen 2013, 15,25):

1. Vaiheittainen kehittäminen
2. Vaatimuksia hallitaan, ylläpidetään ja seurataan koko projektin ajan
3. Ohjelman pitää rakentua itsenäisistä komponenteista
4. Mallien pitää olla visuaalisia, esim. UML-kaaviot
5. Laadunvalvonta on jatkuvaa
6. Muutokset tehdään hallitusti

RUP-malli koostuu neljästä eri päävaiheesta, joiden sisällä ohjelmistoa kehitettään pienemmissä vaiheissa (Kuva 3). Vaiheet ovat aloittaminen, tarkentaminen, rakentaminen ja siirtymä. Päävaiheita suoritetaan osittain lomittain muiden vaiheiden kanssa, jolloin työmäärä jokaisen vaiheen sisällä vaihtelee sen mukaan, missä kohtaa vaiheen suoritus on menossa (Kasurinen 2013, 15).



Kuva 3. RUP-malli (Kasurinen 2013, 15)

Ohjelmistoprojektin aloitusvaiheessa keskitytään liiketoiminnallisiin tarpeisiin ja vaatimusmäärittelyyn. Siinä tehdään teknologiavalinnat, määritellään projektin kesto ja resurssitarpeet sekä tehdään arvio, onko toteuttaminen ylipäätään mahdollista. Aloitusvaiheessa määritellään myös ohjelmiston olennaiset toiminnot ja mitä ohjelmistolta halutaan. Aloitusvaiheessa kerättyihin tietoihin pohjautuen voidaan tehdä jo ensimmäiset testitapaukset ja myöhemmässä vaiheessa tietoja voidaan käyttää pohjana testaussuunnitelmaan liittyviin päätöksiin (Kasurinen 2013, 26).

Tarkennusvaiheeseen siirryttäessä on tehty päätös ohjelmiston toteuttamisesta. Työvaiheen fokuksena on tunnistaa ongelmalliset ja riskialttiit työvaiheet sekä määritellä ohjelmiston arkkitehtuuri, jotta ohjelmiston toteuttaminen on mahdollista suorittaa rakennusvaiheessa. Tarkennusvaiheessa myös ohjelmointi ja testaus viedään laajemmalle tasolle. Yksinkertaisista konsepteista siirrytään varsinaisiin prototyyppeihin. Samoin testaus otetaan laajemmin mukaan, sillä osa prototyypeistä saattaa päätyä myös itse ohjelmistoon (Kasurinen 2013, 15, 26).

Rakennusvaihe aloitetaan lomittain tarkennusvaiheen kanssa. Kehitystyö aloitetaan niistä osista, jotka tarkennusvaiheessa määritettiin riskialttiimmiksi. Ohjelman toteutus tapahtuu vaihe kerrallaan siten, että testaus on koko ajan mukana. Testaus painottuu

tekniseen testaamiseen. Testaus alkaa yksikkötestauksesta edeten integraatiotestauksen kautta järjestelmätestaukseen. Lisäksi voidaan hyödyntää myös muita testaus menetelmiä, kuten käytettävyystestausta tai musta- ja lasilaatikkotestausta. Tässä vaiheessa voidaan tehdä myös hienosäätöä ohjelmiston rakenteeseen ja arkkitehtuuriin (Kasurinen 2013, 26).

Kun projektin toteutus on edennyt tarpeeksi pitkälle, tullaan siirtymävaiheeseen, jossa ohjelma otetaan hallitusti käyttöön. Ohjelmistosta saattaa puuttua vielä joitain toissijaisia ominaisuuksia tai siihen saatetaan tehdä vielä pieniä muutoksia. Siirtymävaiheessa tekninen testaus siirtyy taka-alalle ja testaus keskittyy mm. käytettävyys- ja hyväksymistestaukseen. Tässä vaiheessa mukaan voi tulla myös käyttäjillä tapahtuva testaus. Siirtymävaiheen testauksen keskeisenä tavoitteena on varmistaa ohjelmiston kokonaistoimivuus ja sille asetettujen vaatimusten täyttyminen (Kasurinen 2013, 26).

RUP-mallin suurin ero vesiputous- ja V-malliin on siinä, että määrittelyä, suunnittelua, toteutusta ja testausta tehdään mallin kaikissa vaiheissa. Näin ollen testaus tulee mukaan jo mahdollisimman varhaisessa vaiheessa kehitystä. Mallin keskeisenä ajatuksena oleva ohjelmiston vaiheittainen toteuttaminen riskialttiimmista osista alkaen varmistaa, ettei ohjelmiston tekeminen viivästy loppuvaiheessa huomatun ongelman vuoksi. Lisäksi kehityksen etenemistä ja laatua voidaan valvoa koko projektin ajan (Kasurinen 2013, 26).

2.2 Testaustavat

Yksikkötestaus

Yksikkötestaus on tavallisin testaustapa ohjelmistokehityksessä. Yksikkötestauksessa yksitäisen komponentin tai funktion toimivuutta tarkastellaan välittömästi toteutuksen yhteydessä, yleensä kehittäjän toimesta. Yksikkötestauksen tarkoituksena on varmistaa, että toteutettu ominaisuus tai olemassa olevaan järjestelmään tehty muutos toimii ja reagoi annettuihin arvoihin halutulla tavalla. Yksikkötestauksen testitapaukset perustuvat tilanteisiin, joiden voidaan olettaa olevan riskialttiita virheille ja ongelmille. Jos jokin kutsuista epäonnistuu, pystyy kehittäjä korjaamaan virheen ennen kuin komponentti siirretään itse järjestelmään (Tilley & Parveen 2012, 4).

Yksikkötestauksen heikkoutena on, että yksittäinen komponentti pystyy harvoin yksistään tekemään mitään, vaan komponentit toimivat keskinäisessä vuorovaikutuksessa.

Tällaisia tilanteita voidaan kuitenkin kiertää erillisten testikomponenttien avulla, joiden tehtävänä on simuloida varsinaisen järjestelmän toimintaa. Testikomponenttien ideana on varmistaa, että testit toimivat aina samalla tavalla ja testattavalle komponentille lähetettävät syötteet ovat tiedossa, jotta komponentilta saatuja vastauksia voidaan vertailla ja toimivuus varmistaa (Kasurinen 2013, 52).

Integraatiotestaus

Integraatiotestausta suoritetaan, kun järjestelmän komponentteja koostetaan pala kerrallaan isommaksi ja lopulta valmiiksi kokonaisuudeksi. Integraatiotestauksessa olemassa olevaan ja testattuun kokonaisuuteen kytketään uusi komponentti ja sen toimivuus testataan muun järjestelmän kanssa. Yksikkötestauksen tavoin myös integraatiotestauksessa voidaan joutua käyttämään erillisiä testauskomponentteja, jotta testit saadaan suoritettua. Integraatiotestauksen tavoitteena on käytännön tasolla todentaa, että yksittäiset komponentit toimivat yhtenä kokonaisuutena. Integraatiotestauksen testitapaukset ovat yksikkötestien testitapauksia laajempia, mutta eivät kuitenkaan koko järjestelmän kattavia testejä (Kasurinen 2013, 53).

Järjestelmätestaus

Järjestelmätestaus aloitetaan tavallisesti integraatiotestauksen valmistuttua. On kuitenkin tapauksia, jolloin järjestelmätestauksen raja on häilyvä ja testaus aloitetaan jo aikaisessa vaiheessa projektia. Järjestelmätestaus ei yksikkö- ja integraatiotestauksen tavoin tarkoita mitään tiettyä testaustapaa vaan se on yleisnimitys kokonaiselle järjestelmälle tehtävästä testauksesta. Testauksen tavoitteena on osoittaa, että järjestelmä toimii yhtenäisenä kokonaisuutena ja se täyttää sille määrittelyvaiheessa asetetut tavoitteet. Testauksen toteuttamiselle ei ole mitään tiettyä tapaa vaan se riippuu projektin luonteesta ja sen vaatimuksista. Tavallisimmin testaus voidaan suorittaa musta- tai lasilaatikkotestauksena, käyttäjätestauksena tai tutkivaa testausta käyttäen (Myers, Sandler & Badgett 2011, 119-120).

Järjestelmätestaus suoritetaan testiympäristössä ja se on menetelmänä lähellä hyväksymistestausta. Hyväksymistestauksesta poiketen järjestelmätestauksessa virheitä etsitään komponenttitasolta alkaen, kun taas hyväksymistestauksen painopiste on toiminnollisuuksien todentamisessa. Järjestelmätestausta tehdessä on myös mahdollista, että sen perusteella järjestelmään tehdään vielä muutoksia ja korjauksia (Kasurinen 2013, 57).

Hyväksymistestaus

Hyväksymistestaus on testaamisen viimeinen vaihe. Siinä testauksen painopiste on toiminallisuuksien todentamisessa, ohjelmiston laadukkuuden arvioinnissa ja siinä, että ohjelmisto täyttää vaatimusmäärittelyssä todetut vaatimukset. Toisin sanoen hyväksymistestauksen suorittaminen ja sen onnistuminen tarkoittavat, että asiakas on hyväksynyt toteutuksen, kehitystyö on päättynyt ja ohjelmisto otetaan käyttöön. Tavallisesti hyväksymistestaus suoritetaan sen kohdeympäristössä.

Regressiotestaus

Regressiotestaus ei suoranaisesti ole oma testaustapansa, vaan yleistermi kaikelle uudelleen testaukselle. Regressiotestausta tehdään aina kun ohjelmiston toimivaa osaa muutetaan ja halutaan todentaa, että ohjelmisto toimii myös muutoksen jälkeen ja mitään ei ole mennyt rikki. Regressiotestausta voidaan tehdä myös silloin, kun kehitystyössä saavutetaan jokin osatavoite ja halutaan varmentaa kaikkien, myös aiemassa kehitysvaiheessa testattujen toimintojen toimivuus. Regressiotestauksen suorittamiseen käytetään usein testausautomaatiota, koska sitä tehdään yleensä useampaan kertaan kehitystyön aikana (Tilley & Parveen 2012, 2).

Regressiotestauksen käyttö perustuu ajatukseen, että virheet sijoittuvat uusiin komponentteihin tai niitä käyttäviin komponentteihin. Jos jotain osaa ohjelmistossa muutetaan, se pitää testata, kuten jos se olisi uudistettu kokonaan. Lisäksi regressiotesteillä voidaan todentaa, että kehityshaarojen yhdistämisen jälkeen kaikki aiemmin korjatut virheet ovat poissa yhdistetystä versiosta ja versio toimii odotetulla tavalla (Kasurinen 2013, 68-69).

Muita testaustapoja

Edellä mainittujen testaustapojen lisäksi on olemassa myös muita testaustapoja, joita käytetään varsinkin järjestelmätestaus vaiheessa. Tällaisia testejä on mm. käytettävyystestaus, kuormitus- ja suorituskykytestaus, mallipohjainen testaus ja tutkiva testaus.

Käytettävyystestauksessa testauksen painopiste on ohjelmiston käyttöliittymän toimivuudessa ja intuitiivisuudessa. Käytettävyystestaus ei ole sidottu mihinkään tiettyyn testausvaiheeseen, vaan sitä voidaan tehdä jo suunnitteluvaiheessa erilaisten prototyyppien avulla. Käytettävyyttä voidaan testata erilaisin työkaluin, käyttäjäkokeiluin tai jopa haastatteluin (Myers, Sandler & Badgett 2011, 125).

Kuormitus- ja suorituskykytesteissä testataan järjestelmän toimivuutta suunnitellulla käyttäjämäärällä. Testeissä simuloidaan tietty määrä käyttäjä, jotka suorittavat testitapauksia, simuloiden normaalia toimintaa. Kuormitus- ja suorituskykytestauksella pyritään tunnistamaan järjestelmän pullonkaulat, selvittämällä miten järjestelmä selviää normaaleissa käyttöolosuhteissa ja millaisista käyttäjämääristä järjestelmän on mahdollista suoriutua (Myers, Sandler & Badgett 2011, 126).

Mallipohjaisessa testauksessa testitapaukset pohjautuvat ohjelmiston suunnitelmista luotuihin testitapauksiin. Testitapaukset testaavat, onko ohjelmisto toteutettu suunnitelman mukaisesti ja täyttääkö ohjelma mallissa esitetyt toiminnot. Jos testitapaukset toimivat oikein, voidaan varmistua, että ohjelmisto on toteutettu suunnitelmien mukaan (Kasurinen 2013, 75).

Tutkiva testaus on mielipiteitä jakava testaustapa. Osan mielestä se on hyvä tapa löytää ”typeriä” virheitä ja osan mielestä taas pelkkää ajanhukkaa. Tutkivassa testauksessa hyödynnetään testaajien ymmärrystä siitä, mistä virheet normaalisti löytyvät. Tutkivan testauksen lähtökohtana ei ole niinkään suunnitelmallisuus, vaan se pohjautuu enemmänkin riskiarvioiden kautta tehtävään testaamiseen. Suunnitelmallisuuden puutteesta huolimatta tutkiva testaus voi pohjautua olemassa oleviin malleihin ja siitä tehdään dokumentaatiot. Tutkivaa testausta voidaan käyttää löytämään ohjelmistosta ongelma-alueita, jotka eivät ole niin ilmeisiä tai helposti määriteltäviä (Kasurinen 2013, 74).

2.3 Testauksen suunnittelu

2.3.1 Testitapausten valinta

Testitapauksella tarkoitetaan tapahtumaa, joka kuvaa yhden yksittäisen toimenpiteen tai tapahtumaketjun, jonka perusteella ohjelmisto toteuttaa jonkin toiminnon. Testitapauksessa on määritelty kaikki tarvittavat vaiheet ja toimenpiteet, jotta testi voidaan toteuttaa ja toiminnon toimivuus voidaan varmistaa. Testitapauksia määritellään koko projektin ajan, jotta voidaan varmistua valmiin tuotteen toimivuudesta ja laadusta. Testitapausten määrittely aloitetaan heti projektin alussa, jolloin ne pohjautuvat ohjelmiston arkkitehtuuriin ja vaatimusmäärittelyyn. Tällä tavoin pystytään varmentamaan toiminnollisuuksien toteutuminen ja niiden muuttumattomuus kehityksen aikana. Lisäksi uusia testitapauksia

toteutetaan aina kun uusi ominaisuus valmistuu tai testauksessa havaitaan aiemmin ilmenemätön virhe (Kasurinen 2013, 118-120).

Virhe voi periaatteessa ilmetä missä tahansa kohtaa koodia. Kehitysympäristön kirjastot ja omat, aiemmin luodut kirjastot ovat todennäköisesti vähemmän virheherkkiä, johtuen aiemmasta testauksesta kuin kehityksen aikana luotu koodi. Kehityksen aikaisia virheitä voi aiheuttaa mm. uusi koodi, teknologia tai ominaisuus, muutokset ja viime hetken korjaukset, ulkopuolella tuotettu koodi tai uusien työntekijöiden kokemuksen määrä. Lisäksi virheitä voi aiheutua luonnollisista syistä, kuten tekijän huolimattomuudesta, heikosta johtamisesta tai ristiriitaisista ja epämääräisistä vaatimuksista (Kasurinen 2013, 120).

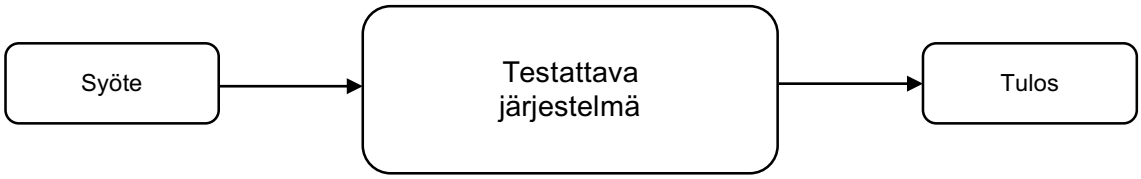
Testitapausten valinta voidaan toteuttaa kolmen eri lähestymistavan kautta; mustalaatikko-, lasilaatikko- tai harmaalaatikkotestaus. Lasilaatikkotestauksessa testitapaukset perustuvat ohjelman toteutukseen ja ohjelman koodiin, mustalaatikkotestauksessa puhtaasti ohjelmiston teknisiin vaatimuksiin ja vaatimusmäärittelyyn ja harmaalaatikkotestauksessa tietoon ohjelman toteutusperiaatteista (Haikala & Mikkonen 2011, 209).

Mustalaatikkotestauksessa testaus perustuu annettuun syötteeseen ja saatuun vastaukseen. Mustalaatikkotestauksen toiminta on kuvattu kuvassa 4. Mustalaatikkotestausta voidaan käyttää koko projektin ajan alkaen yksikkötesteistä ja päättyen hyväksymistestaukseen. Mustalaatikkotestauksessa on tärkeää, että annettujen syötteiden perusteella voidaan todentaa toiminnon toimivuus kaikissa odotettavissa tilanteissa. Mustalaatikkotestauksessa testaajan tehtävänä on lähinnä varmistaa annettujen syötteiden ja saatujen vastausten oikeellisuus ja mahdollisissa virhetilanteissa kirjata huomautus testausraporttiin (Myers, Sandler & Badgett 2011, 8).

Kuva 4. Mustalaatikkotestaus: Järjestelmälle annetaan syöte ja tuloksesta arvioidaan ohjelman toimivuus

Lasilaatikkotestauksessa (kuva 5) testaus toteutetaan tarkastelemalla ohjelmiston sisäistä toimintaa. Ohjelmistolle annetaan erilaisia syötteitä, tutkitaan miten ohjelmisto syötteeseen reagoi ja mitä ohjelmiston sisällä tapahtuu. Lasilaatikkotestauksessa pysty-

tään varmentamaan, että saatu tulos ei ollut sattumaa, koska on tiedossa, miten annettua syötettä käsitellään. Mahdollisen virheen ilmetessä voidaan virhe jäljittää kooditasolle asti. Vaikka lasilaatikkotestauksessa löydetään myös kooditason virheet sillä ei pystytä havaitsemaan huonosta vaatimusmäärittelystä tai puuttuvista ominaisuuksista aiheutuvia ongelmia. Lasilaatikkotestauksessa testaajan pitää ymmärtää ohjelmoinnista ja järjestelmän logiikasta, jotta hän voivat varmuudella todeta järjestelmän toimivuuden (Myers, Sandler & Badgett 2011, 10).



Kuva 5. Lasilaatikkotestaus: Järjestelmälle annetaan syöte ja tuloksesta sekä järjestelmän sisäisestä toiminnasta arvioidaan järjestelmän toimivuus

Harmaalaatikkotestaus on yhdistelmä musta ja lasilaatikkotestauksen parhaista ominaisuuksista; mustalaatikkotestauksen vaatimusmäärittelyyn perustuvat testitapaukset ja lasilaatikkotestauksen kyky tarkastella ohjelmistoa myös sisäisesti. Voidaankin sanoa, että harmaalaatikkotestaus on kokonaisvaltainen sekä vaatimukset ja koodin testaava menetelmä. Harmaalaatikkotestausta voidaan käyttää esimerkiksi silloin, kun kokonaisvaltaisten lasilaatikkotestien tekeminen ei ole mahdollista, mutta ohjelmiston paikallisiin komponentteihin päästään käsiksi. Oma ohjelmisto tunnetaan ja se on testattu lasilaatikkomenetelmää käyttäen, mutta liittyvät ohjelmistot voidaan testata vain mustalaatikkotestauksena (Kasurinen 2013, 68).

Testaus on aina kompromissi testauksen kattavuuden ja resurssien välillä. Usein testattavia asioita on enemmän kuin resurssien puolesta on mahdollista toteuttaa ja tämän seurauksena testauksesta joudutaan karsimaan. Jotta kaikki tärkeät ja kriittiset toiminnot saadaan testattua, voidaan testitapauksien valinnassa käyttää projektin määrittelyvaiheessa tehtyä riskikartoitusta. Ensimmäisenä suoritetaan ne testitapaukset, joiden toimimattomuus on haitallisimpia ohjelmiston toiminnan kannalta ja joiden toimimattomuus voi estää myös muiden testitapauksen suorittamisen. Testitapausten valinnassa voidaan käyttää myös valintamenetelmää, kuten suunnitelma- tai riskilähtöinen valintamenetelmä (Kasurinen 2013, 121).

Suunnitelmalähtöinen testitapausten valinta perustuu ohjelmiston toimivuuden ja annettujen laatuvaatimusten mahdollisimman tehokkaaseen täyttämiseen. Idea on kustannustehokkaasti pyrkiä osoittamaan, että ohjelmisto täyttää sille asetetut laatuvaatimukset. Testitapausten valinnalla pyritään minimoimaan tehtävän työn ja käytettävien resurssien määrä, koska testausta ei voida lopettaa ennen kuin kaikki vaatimukset on saatu täytetyksi. Suunnitelmalähtöinen testaustapojen valinta edellyttääkin, että resursseja on riittävästi tai ainakin kohtuullisesti saatavilla. Menetelmää käytetään yleensä isommissa projekteissa, joissa ohjelmiston julkaiseminen ja toimintavarmuus ovat tärkeitä ja julkaisun jälkeinen työmäärä halutaan minimoida sen hankaluuden ja kustannusten vuoksi (Kasurinen 2013, 122).

Riskilähtöinen testitapausten valinta pyrkii kohdentamaan käytössä olevat resurssit ohjelmiston ydintoimintojen toimivuuden ja käytettävien tekniikoiden tarkistamiseen ja varmistamiseen. Ideana on käyttää rajalliset resurssit mahdollisimman tehokkaasti hyödyksi minimoimalla ohjelmistoon jääneiden vikojen aiheuttamat ongelmat. Testitapaukset priorisoidaan siten, että ohjelmistoon jäävät virheet ovat mahdollisimman huomaamattomia ja yksinkertaisia sekä helposti korjattavia, jotta julkaisun jälkeinen korjaustyö olisi mahdollisimman halpaa. Menetelmä onkin yleinen organisaatioissa, joissa resurssien määrä on rajallinen (Kasurinen 2013, 123).

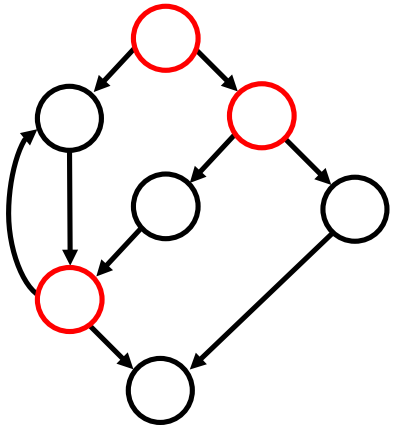
Vaikka testitapauksien valintaan on olemassa erilaisia menetelmiä, on tärkein valintakriteeri yksinkertainen: aloitetaan tärkeimmistä toiminnallisuuksista ja edetään kohti vähemmän tärkeitä toiminnallisuuksia. Näin toimittaessa on ohjelmistosta testattu aina kaikkien tärkeimmät toiminnallisuudet mitä saatavilla olevilla resursseilla oli mahdollista testata (Kasurinen 2013, 123).

2.3.2 Testauksen lopettaminen

Testaamista ei voida jatkaa loputtomiin ja hyvästäkin testauksesta ja onnistuneesta projektista huolimatta ohjelmistoon jää aina virheitä. Ideaalisessa tilanteessa virheiden pitäisi olla harvinaisia ja niin mitättömiä, että niiden korjaamisella ei saavuta taloudellista hyötyä ja suurella todennäköisyydellä käyttäjät eivät näihin virheisiin missään vaiheessa törmää. Mikäli virhe kuitenkin ilmenee, todennäköisesti se on hetkellistä hidastumista, harvinaisia yhteyshäiriöitä tai käyttökokemusta hetkellisesti haittaavaa toimintaa, kuten hetkellisesti toimimaton painike (Kasurinen 2013, 124).

Testauksen määrä on kompromissi käytössä olevien resurssien, varmuuden luotettavuudesta ja myöhästymisen aiheuttamien kustannusten välillä. Testauksen määrää on hankala määritellä ja se ei tarkoita samaa kuin testauksen tehokkuus. Hyvin suunnitellut testitapaukset voivat johtaa laadukkaampaan ja tehokkaampaan testaamiseen kuin umpimähkäinen useita päiviä kestävä testaus. Tarvittavan testauksen määrää voidaan kuitenkin pyrkiä hahmottamaan erilaisilla mutkikkuusmitoilla ja testauksen riittävyyttä kattavuusmittareilla sekä virheitä kylvämällä (Haikala & Mikkonen 2011, 210-211).

Mutkikkuusmittauksessa koodia analysoimalla pyritään paikallistamaan ohjelmiston paljon testausta vaativat moduulit. Mutkikkuusmittojen määrittämiseen on useita eri menetelmiä, joista tunnetuin on McCaben syklomaattinen luku. Luku kuvaa funktion monimutkaisuutta ja se saadaan lisäämällä funktion kontrolliverkon haarautumiskohtien (ehtolauseet ja toistorakenteet) lukumäärään ykkönen. Testauksen näkökulmasta saatua arvoa voidaan pitää minimimääränä testitapauksia joilla ko. funktiota tulisi testata. Jos saatu luku on kuitenkin yli 10, kannattaa moduuli suunnitella uudelleen, jotta moduulin toteutus ja testaus pysyvät hallinnassa. Syklomaattisen luvun laskeminen on havainnollistettu kuvassa 6, jossa punaiset ympyrät ovat kontrolliverkon haarautumiskohtia, joita käytetään syklomaattisen luvun laskemiseen. Tässä tapauksessa syklomaattiseksi luvuksi saadaan 4 (Haikala & Mikkonen 2011, 210; McCabe 1976, 308).



Kuva 6. Havainne kuva funktion kontrolliverkosta ja syklomaattisen luvun laskemisesta (McCabe 1976, 308)

Kattavuusmittareilla pyritään varmistamaan ohjelmiston kaikkien osien kattava testaus sekä varmuus siitä, että testausta on suoritettu riittävästi. Kattavuusmittareita voidaan käyttää sekä koodikattavuuden mittaamiseen, että toisinaan myös toiminnallisen katta-

vuuden mittaamiseen. Toiminnallinen kattavuus tarkoittaa tiettyjen ominaisuuksien testaamista, kun taas koodikattavuudella viitataan kooditasolle. Tässä keskitytään vain koodikattavuuden mittareihin (Kasurinen 2013, 211).

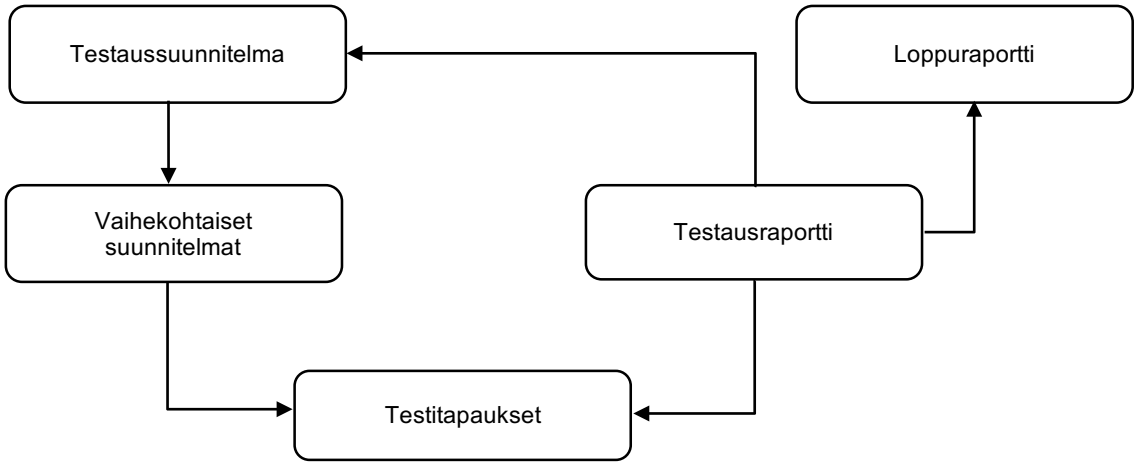
Koodikattavuuden mittareita ovat mm. lausekattavuus, päätöskattavuus, ehtokattavuus, moniehtokattavuus ja polkutestaus. Lausekattavuus mittaa, että jokainen ohjelman lause testataan vähintään kerran. Päätöskattavuus edellyttää, että molemmat arvot, tosi ja epätosi on testattu. Ehtokattavuudessa edellytetään, että kaikki ehtorakenteen päätöksen variaatiot on testattu, kun taas moniehtokattavuudessa kaikkien ehtojen kaikki päätökset pitää olla testattu. Polkutestauksessa taas pyritään käymään ohjelmaa läpi mahdollisimman monen eri suorituspolun kautta (Kasurinen 2013, 211).

Testauksen kattavuutta voidaan pyrkiä selvittämään virheitä kylvämällä, mutta menetelmää ei juurikaan käytetä johtuen sen aiheuttamasta lisätyöstä, mahdollisista koodiin jääneistä virheistä ja epävarmuudesta testauksen validisuuteen. Menetelmässä ohjelmakoodiin lisätään tahallisia virheitä ja testauksesta löydettyjen virheiden määrästä pyritään päättelemään ohjelmaan jääneiden virheiden määrä. Menetelmä perustuu olettamaan, että todellisten virheiden määrä suhteessa kylvettyihin virheisiin on sama kuin testauksessa löydettyjen todellisia ja kylvettyjen virheiden suhde (Kasurinen 2013, 211).

Testauksen lopettamiselle tulisi siis aina asettaa kriteerit eli lopetusehto, joiden perusteella projektin johto voi tehdä päätöksen testauksen lopettamisesta. Lopetusehtona voidaan käyttää esimerkiksi kumulatiivisesti löydettyjen virheiden määrää, projektin yleisiä laatuvaatimuksia, testaukseen varattujen resurssien kulutusta tai jopa julkaisuajankohtaa (Haikala & Mikkonen 2011, 210).

2.4 Testauksen dokumentointi

Testaus on monivaiheinen prosessi ja sen seurauksena syntyy paljon dokumentoitavaa informaatiota. Pelkästään projektitasolla testaamisen keskeisiä dokumentteja on testaussuunnitelma, vaihekohtaiset suunnitelmat, testitapausten dokumentaatio, testausraportti ja loppuraportti. Kuvassa 7 on kuvattuna projektin tuottamien testaukseen liittyvien dokumenttien suhde toisiinsa (Kasurinen 2013, 104).



Kuva 7. Testausdokumenttien suhde toisiinsa (Kasurinen 2013, 104)

Testauksen resurssien oikealaisen käytön, dokumentoinnin ja tiedon oikeellisuuden helpottamiseksi voidaan käyttää valmiita dokumenttipohjia. Dokumenttipohjien avulla voidaan varmistaa, että arvokas tieto saadaan kirjattua talteen ja testaajat pystyvät keskittymään itse testaamiseen eikä aikaa kulu turhaan dokumenttien muotoiluun. Dokumenttipohjat helpottavat myös tietojen hakemista dokumenteista, sillä tietojen pitäisi olla aina tietyssä, sille varatussa paikassa. Samoin raporttien koostaminen on nopeaa, koska tieto on aina samassa paikassa ja tietyssä muodossa (Kasurinen 2013, 88).

Testauksen täydellisessä dokumentaatiossa testausdokumentteja tulee huomattava määrä. Jokaiselta testaustasolta tehdään aina suunnitelmat ja raportit; järjestelmätestaussuunnitelma ja -raportti, jokaisen integraatiotestin suunnitelma ja raportti sekä jokaisen yksikkötestin suunnitelma ja raportti. Projektin koosta riippuen dokumentteja voidaan yhdistää suuremmiksi kokonaisuuksiksi ja esimerkiksi pienemmissä projekteissa riittää yksi yleinen suunnitelma testauksesta sekä raportti, joka kattaa kaikki tasot (Haikala & Mikkonen 2011, 216-217).

2.4.1 Testaussuunnitelmat

Testaussuunnitelma on dokumentti, jossa määritellään mitä testataan, milloin testataan ja miten testataan. Testaussuunnitelma on yleensä testaus- tai projektipäällikön tai testauksesta päättävän henkilön tekemä yhtenäinen dokumentti, jossa on määriteltynä testauksen pääkohdat. Määriteltäviä asioita on mm. vaatimusmäärittelyssä esitellyt tarpeet, asiakkaan vaatimukset, projektin tavoite ja laajuus, käytössä olevat resurssit, testauksen

lopetusehdot, testaustyökalut ja –menetelmät sekä kuka testauksesta on vastuussa (Kasurinen 2013, 105, 116).

Testaussuunnitelma voidaan toteuttaa joko yleisenä koko projektin kattavana suunnitelmana tai menetelmäkohtaisena suunnitelmana, jolloin puhutaan vaihekohtaisesta suunnitelmasta. Vaihekohtaisessa suunnitelmassa testauksen tekeminen ja vaatimukset kuvataan testaustasoja mukaillen yksi taso kerrallaan. Testaussuunnitelman rakenne ja kattavuus vaihtelevat projektin luonteesta ja organisaation käytännöistä riippuen ja siihen ei ole olemassa yhtä ja ainoaa oikeaa tapaa (Kasurinen 2013, 105, 116-117).

Esimerkiksi pienemmissä organisaatioissa testaussuunnitelmassa ei välttämättä ole kuvattuna erillistä testausstrategiaa. Yleensä testaussuunnitelmista kuitenkin löytyy tietyt perusosat, jotka määritellään lähes kaikissa projekteissa. Testaussuunnitelma voidaan toteuttaa monellakin eri tavalla. Testaussuunnitelma voi noudattaa testausstandardia, esimerkiksi ISO/IEC 29119 -testausstandardi (Taulukko 1) tai menetelmää, esimerkiksi SPACE DIRT (Taulukko 2) (Kasurinen 2013, 116-117).

Taulukko 1. ISO/IEC29119-testausstandardin testaussuunnitelmassa olevia perusasioita (Kasurinen 2013, 117)

Osakokonaisuus	**Selite**
Projektin kuvaus	Yleistiedot projektista. Mitä projektissa on tarkoitus tehdä. Testaukseen liittyvät päivämäärät
Kuvaus testattavasta ohjelmistosta	Mitä projektissa on tarkoitus testata. Kuvaus pääkomponenteista ja niiden yhteyksistä. Mitä komponenttien on tarkoitus tehdä
Testauksen laajuus	Mitä osia testataan. Tiedossa olevat ongelmat ja rajoitteet
Testausstrategia	Testauksen lähestymistapa. Kuka, milloin, missä ja miten suorittaa testauksen
Aikataulu ja työnjako	Testauksen aikataulutus; Deadlinet, sovitut katselmoinnit ja välietapit
Riskikartoitus & toimintasuunnitelma	Ohjelmistoon liittyvät riskit ja miten ne voidaan välttää. Lista vaatimusmäärittelyssä olevista vaatimuksista
Henkilöstölistaus	Testaustiimiin kuuluvat henkilöt ja heidän vastuunsa

Taulukko 2. SPACE DIRT –menetelmän perusasiat (Kasurinen 2013, 118)

Osakokokonaisuus	**Selite**
Laajuus	Mitä testataan ja mitä ei testata
Ihmiset	Testaajien koulutus, vastuut ja aikataulut
Lähestymistapa	Käytettävät testausmenetelmät
Kriteerit	Millä kriteereille testaus aloitetaan ja lopetetaan sekä tarvittaessa keskeytetään ja jatketaan
Ympäristö	Millainen testiympäristö testausta varten tarvitaan
Tuotokset	Mitä testausprosessi tuottaa kehitysprosessille
Satunnaiset	Testaukseen liittyvät erikoisominaisuudet ja poikkeukset Kuka saa tarvittaessa muuttaa testaussuunnitelmaa
Riskit	Testaukseen liittyvät riskit ja niiden torjunta
Tehtävät	Testausprosessiin liittyvät tehtävät

2.4.2 Testitapaukset

Yksinkertaistettuna testitapauksessa kuvataan, mitä testitapauksessa tehdään ja miten järjestelmä reagoi siihen. Usein kuitenkin testitapaukset kannattaa kuvata tarkemmin, jotta voidaan varmistua siitä, että kaikki toimii, kuten on suunniteltu. Testitapauksessa kannattaakin kuvata järjestelmän lähtötila, ehdot ja vaatimukset testin toteutuksesta, sidokset muihin testitapauksiin ja mitä testillä tavoitellaan. Lisäksi testaajaa varten on hyvä kuvata myös, mitä järjestelmän odotetaan tekevän (Kasurinen 2013, 120).

Kasurisen (2013, 120) näkemys yhden testitapauksen kuvaamisesta:

- Testitapauksen suunnittelija
- Miten testitapaus liittyy järjestelmään
- Mihin osaan järjestelmää testitapaus vaikuttaa
- Testitapauksen tavoite
- Testitapaukseen sidoksissa olevat muut testitapaukset

- Testitapauksen erikoisehdot, kuten poikkeus tai virhetilan mallintaminen
- Millaisessa ympäristössä testitapaus suoritetaan
- Annettavat syötteet ja miten järjestelmä niihin reagoi
- Miltä järjestelmän pitäisi näyttää testitapauksen jälkeen
- Millä ehdoin testitapaus on onnistunut
- Missä tapauksessa testitapauksen epäonnistuminen voidaan jättää huomiotta

2.4.3 Testausraportit

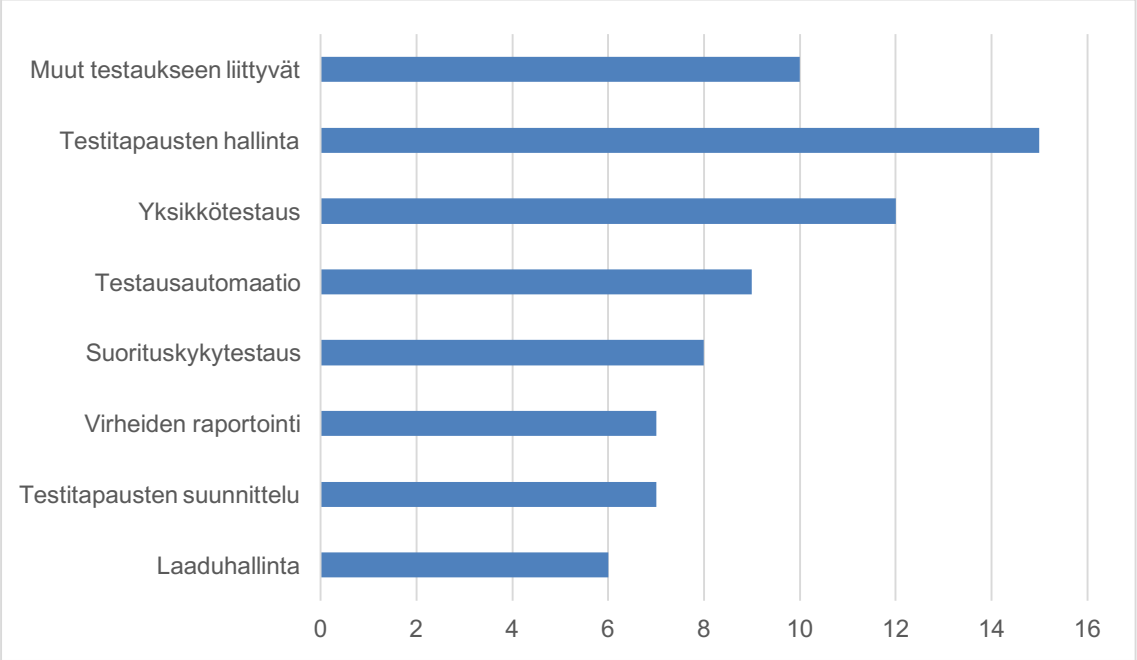
Testausraporttien tarkoituksena on antaa tietoa projektin kehityksen ja testauksen sujuvuudesta projektin johdolle. Testausraporttien pohjalta voidaan itse projektin tai sen tietyn vaiheen päätyttyä koostaa loppuraportti, jossa on kuvattuna tärkeimmät mittarit ja havainnot projektin kulusta. Testausraportteja tuotetaan testauksen kaikilla tasoilla, alkaen yksikkötestauksesta ja päätyen hyväksymistestaukseen (Kasurinen 2013, 105).

Testausraportissa kuvataan testauksen kulku ja käytetyt menetelmät sekä välineet. Kasurinen (2013, 88) on listannut raportin sisällöstä seuraavaa:

- Perustiedot testauksesta; testausajankohta, testaaja, testausalusta ja käytetyt testausvälineet.
- Testauksessa käytetyt syötteet ja tehdyt toimenpiteet.
- Testauksen tulokset; onnistuneet ja epäonnistuneet testitapaukset.
- Epäonnistuneista testeistä virheilmoitukset, kuvaus mitä tapahtui ja mahdolliset kuvan kaappaukset. Arvio virheen kriittisyydestä.
- Jäikö jotain testaamatta. Jos jäi, niin mitä ja miksi.

2.5 Testaustyökalut

Testauksessa tavallisimpia työkaluja ovat kehitysympäristö, yksikkötestauksen työkalut ja testausautomaatioon liittyvät järjestelmät. Lisäksi käytössä voi olla erilaisia testauksen suunnitteluun, raportointiin ja laadunvalvontaan liittyviä työkaluja. Kuviossa 2 on nähtävissä testauksen työkaluja eri ohjelmistotaloissa. Ohjelmistoprojektin luonteesta ja testausmenetelmistä riippuen testaustyökalut voivat olla osa valmista ohjelmistokehityspakettia tai organisaation itse kehittämiä testaustyökaluja (Kasurinen 2013, 84-85).



Kuvio 2. Testaustyökalujen käyttö suomalaisissa ohjelmistotaloissa. Otos sisältää 31 eri kokoista ja eri toimialoilla toimivaa yritystä (Kasurinen 2013, 84)

Kehitysympäristöllä tarkoitetaan kehitykseen käytettävää työkalua, kuten Eclipse, PhpStorm tai Netbeans. Työkalusta riippuen, yksikkötestaus ja staattinen testaus tehdään usein samoilla työvälineillä kuin itse kehitystyökin. Kehitysympäristöstä riippuen staattinen testaus voi olla yksinkertaisemmillaan syntaksin tarkistusta, avainsanojen korostamista ja väärin kirjoitettujen muuttujien merkitsemistä. Kehittyneimmissä ympäristöissä voi olla lisäksi integraatio virhetietokantaan ja versiohallintaan, vaaralliset koodirakenteet ja ristiviittaukset paljastava työkalu tai työkalu yksikkötestauksen tekemiseksi. Joissakin ohjelmointikielissä on sisäänrakennettuna tuki yksikkötestauksen tekemistä varten, jolloin testauksen tekeminen suoraan kehitysympäristöstä on helpompaa. Jos ohjelmistokieli ei suoraan tue yksikkötestejä, voidaan testaamiseen käyttää xUnit ohjelmistokehystä, joka on saataville useimmille kaupallisessa käytössä oleville ohjelmointikielille (Kasurinen 2013, 85-86).

Testausautomaatiolla tarkoitetaan sarja-ajettavia komentoja, jotka lähettävät komponenteille viestejä, kirjaavat ylös mitä komponentissa tapahtui ja vertaa vastaako tulos odotettua. Testausautomaatiota voidaan käyttää myös käyttöliittymän testaamiseen, jolloin automaatiossa simuloidaan hiiren painalluksia, lomakkeisiin kirjoitettua tekstiä jne. ja seurataan, miten ohjelma toimenpiteisiin reagoi (Kasurinen 2013, 86).

Testausautomaation tarkoituksena on vähentää käsin tehtävän testauksen määrää, jolloin testaajien resursseja voidaan vapauttaa muihin tehtäviin sekä varmistaa, että aiemmin toimineet osat eivät ole rikkoontuneet kehityksen aikana. Testausautomaation käyttö on suositeltavaa, jos testiä toistetaan kehityksen aikana useita kertoja. Tavallisimmin testausautomaatiota käytetään yksikkö- ja integraatiotestauksessa, mutta myös regressiotestauksessa ja järjestelmätestauksessa sen käyttö on suotavaa. Testausautomaatiota ei ole tarkoitettu uusien ominaisuuksien testaamiseen tai sellaisten testien suorittamiseen, jotka suoritetaan vain kerran (Haikala & Mikkonen 2011, 213).

Puhtaasti testaukseen liittyvien työkalujen lisäksi voidaan käyttää virheiden raportointiin kehitettyä työkalua, jolloin testaajan tai kehittäjän on helppo raportoida löydetyt virheet. Työkalua voidaan käyttää kehittäjien työlistana ja sen avulla voidaan seurata testauksen etenemistä sekä koostaa lista seuraavaan versioon tehtävistä korjauksista. Lisäksi työkalun tietokannasta voidaan seurata testauksen tehokkuutta ja kehityksen sekä ohjelman tarkastelun sen hetkistä laatua (Kasurinen 2013, 86).

3 TESTAUSPROSESSIN KEHITTÄMINEN

Työn tutkimusongelmana oli sisäisen ohjelmistokehityksen testausprosessin kehittäminen. Tutkimus koostuu nykytilanteen kartoittamisesta ja testausprosessin kehittämisestä vastaamaan paremmin nykyvaatimuksia.

Testausprosessin nykytilan kartoittamisen tutkimusmenetelmäksi valikoitui laadullinen eli kvalitatiivinen tutkimus ja tutkimussuuntaukseksi tapaustutkimus. Tutkimuksen aineiston keruu toteutettiin haastatteluin ja sitä täydennettiin havainnoimalla. Haastatteluissa ja havainnoinnissa keskityttiin yrityksen omassa käytössä olevien työkalujen kehittämiseen liittyvän testaamisen nykytilanteen kartoittamiseen. Asiakkaille myytävien palveluiden ohjelmistot ovat kolmannen osapuolen toteuttamia, jolloin niihin liittyvä testaus on toimittajan vastuulla.

3.1 Laadullinen tutkimus

Laadullisessa tutkimuksessa on aina kyse merkityksen käsitteestä ja merkityksellisen toiminnan tutkimuksesta (Alasuutari 2011, 31). Sen avulla parannetaan, kehitetään tai uudistetaan tutkittavaa kohdetta. Laadullinen tutkimus on aineiston keruusta ja analyysistä koostuva kokonaisuus, josta edellä mainittuja osia ei voida erottaa toisistaan (Tuomi & Sarajärvi 2009,68). Näin ollen laadullinen tutkimus soveltuukin hyvin myös tämän tutkimuksen tutkimusmenetelmäksi. Testausprosessi voidaan mieltää merkityksellisenä toimintana, jota tässä tutkimuksessa pyritään parantamaan ja uudistamaan.

Laadullisessa tutkimuksessa aineistoa voidaan kerätä usein eri tavoin. Aineistoa voidaan kerätä esimerkiksi haastatteluin, kyselyin, havainnoiden tai dokumenteissa olevaa tietoa analysoiden. Aineiston keruu voidaan toteuttaa tutkimusasetelmasta riippuen, käyttäen yhtä tai useampaa tapaa, joko rinnan tai eri tavoin yhdisteltynä. Yleisin aineistonkeruu tapa lienee kyselyt ja haastattelut. Kyselyissä ja haastatteluissa voidaan selvittää ihmisten ajattelua, kokemuksia ja motivaatiota liittyen tutkittavaan aiheeseen. Myös havainnointi on melko yleisesti käytössä oleva aineiston keruu menetelmä, vaikkakin analyysin kannalta se on haasteellisin. Havainnoinnissa seurataan tutkimuskohdetta ja tehdään havaintoja tutkimukseen liittyen (Tuomi & Saarijärvi 2009, 71).

Tutkimuksen aineiston keruutavaksi muodostuivat yleisimmin käytettävät haastattelut ja havainnointi. Haastattelujen kautta testausprosessista pystyttiin koostamaan näkemys useamman eri henkilön ajattelun ja kokemusten pohjalta. Haastattelut itsessään antoivat jo hyvän kuvan prosessin nykytilasta, mutta aineiston laadun parantamiseksi mukaan otettiin myös havainnoimalla tehtävää tutkimusta, jolloin prosessista voidaan löytää myös asioita joita haastateltavat eivät huomioineet.

Kerättyä aineistoa tarkastellaan analyyttisesti yhtenä kokonaisuutena tavoitteena ymmärtää kokonaisuutta. Analyyttisesti ajateltuna laadullinen analyysi koostuu kahdesta vaiheesta, havaintojen pelkistämisestä ja arvoituksen ratkaisemisesta, mutta käytännössä ne ovat yhteen kietoutunut kokonaisuus. Havaintoja pelkistettäessä aineistoa tarkastellaan tietystä näkökulmasta kiinnittäen huomiota teoreettiseen viitekehykseen ja kysymysasettelun kannalta olennaiseen tietoon. Havaintojen erottamiseksi tutkimuksen tuloksista pitää olla valittuna viitekehykseen soveltuva tutkimusmetodi. Tutkimusmetodi määrittää käytännöt ja operaatiot, joiden perusteella aineistosta voidaan tuottaa havaintoja, miten havaintoja voidaan tulkita ja miten niiden merkitystä tutkimukselle voidaan arvioida (Alasuutari 2011, 32-33, 63).

Teoreettinen viitekehys toimii ohjaavana tekijänä aineiston keruussa ja analyysissa. Sen perusteella tehdään päätös, millaista kerättävän aineiston tulee olla ja millä menetelmällä aineistoa analysoidaan. Teoreettisella viitekehyksellä yhdessä tutkimusmetodin kanssa on kuitenkin suuri vaikutus tutkimuksen tekemisessä, sillä riippuen valinnoista, tutkimusta on mahdollista tarkastella monesta eri näkökulmasta. Edellä mainitusta syystä laadulliselle tutkimukselle onkin ominaista, että kerättävä aineisto on sellaista, että se mahdollistaa monen tyyppiset tarkastelut mahdolliseksi (Alasuutari 2011, 63-64).

Tutkimussuuntaukseksi valikoitui tapaustutkimus, koska tutkimusongelmana on olemassa olevan toiminnan parantaminen. Pitkärannan (2014, 29) mukaan tapaustutkimus on käytännön havainnointiin perustuva tutkimus, joka hyödyntää monipuolisesti eri tavoin hankittua tietoa nykyisen tapahtuman tai toiminnan analysointiin tietyssä ympäristössä.

Tapaustutkimuksessa voidaan tutkia yhtä tai useampaa tapausta ja siinä on tavoitteena tapauksen holistinen ja syvällinen ymmärtäminen sekä tapauksesta oppiminen. Yhtä tapausta tarkastellaan silloin, kun on mahdollisuus harvinaiseen ja vaikeasti lähestyvään tapaukseen. Useita, eli rinnakkaisia tapauksia tutkitaan, kun halutaan varmentaa edelli-

sessä tapauksessa saatuja tuloksia tai tuottaa ennakoidusti vastakkaisia tuloksia. Olennaista on, että tapaukset ovat tutkijan käsityksen mukaan samanlaisia. Tapaustutkimus on tyypillisesti tapauksen tarkkaa kuvailua ja havainnointia; mikä tapauksissa on yhteistä ja mikä erityistä. Tapaustutkimuksessa pyritään selventämään reaalielämän ilmentymien monimutkaisia syy-seuraussuhteita ja vastaamaan kysymyksiin, miten ja miksi tai etsimään uusia näkökulmia (Sillius & Tervakari 2006).

Tässä tutkimuksessa tutkittiin yhtä tiettyä tapausta, testausta. Tutkimuksen tavoitteena oli kartoittaa testauksen nykytilaa eri näkökulmista ja saada vastaus siihen, miten testausprosessia voidaan kehittää vastaamaan paremmin tämän hetkisiin tarpeisiin. Tutkimuksen teoreettisena viitekehityksenä olivat testausmenetelmät, testausmallit ja testauksen suunnittelu. Näiden pohjalta suunniteltiin tutkimuksen aineiston keruu ja toteutettiin aineiston analyysi.

3.1.1 Aineiston keruu

Haastattelut

Ensisijaiseksi aineiston keruutavaksi valikoitui haastattelut, sillä sen avulla pystytään kartoittamaan ohjelmistokehityksessä mukana olevien henkilöiden mielipide testauksen nykytilasta. Haastattelut ovat hyvä keino tietää, mitä ihminen ajattelee tai miksi hän toimii tietyllä tavalla. Lisäksi haastattelun etuna on sen joustavuus. Haastattelija pystyy tarkentamaan kysymystä, tekemään tarkentavia kysymyksiä tai oikaisemaan väärinkäsityksiä. Lisäksi haastatteluissa kysymykset voidaan esittää haastattelijan parhaaksi näkemässä järjestyksessä (Tuomi & Saarijärvi 2009, 72).

Haastattelut toteutettiin teemahaastatteluna, jossa haastateltaville esitettiin samat kysymykset, riippumatta hänen toimenkuvastaan. Haastattelut tehtiin kehitystiimin jäsenille sekä projektipäällikölle (Taulukko 3.). Haastatteluissa haluttiin selvittää, millä tavoin haastateltava näkee testauksen tämän hetkisen tilan. Haastattelun aihealueet olivat testausstrategia, testauksen suunnittelu ja dokumentointi, testausautomaatio, -menetelmät ja –työkalut sekä muut testausprosessiin liittyvät asiat (Liite 1).

Taulukko 3. Tietoja haastatelluista ja heidän taustoistaan

Haasteltava	**Toimenkuva**	**Kokemus alalta**
A	Johtava sovelluskehittäjä	10+ vuotta
B	Sovelluskehittäjä	5 vuotta
C	Ohjelmistosuunnittelija	20 vuotta
D	Projektipäällikkö	5+ vuotta

Teemahaastattelu eli puolistrukturoitu haastattelu etenee etukäteen valittujen teemojen ja tarkentavien kysymysten varassa. Etukäteen valitut teemat perustuvat teoreettiseen viitekehykseen, mutta haastattelun luonteesta riippuen kysymysten suhde viitekehykseen vaihtelee intuitiivisten ja havaintojen sallimisesta varsin tiukkaan, etukäteen päätetyissä kysymyksissä pitäytymiseen. Teemahaastattelussa haastateltavan tulkinnat ja heidän asioille antamana merkitys on korostetussa roolissa. Teemahaastattelussa pyritään löytämään merkityksekkäitä, ongelma-asettelun mukaisia vastauksia (Tuomi & Saarijärvi 2009, 75).

Havainnointi

Havainnointi on haastattelujen tavoin yleinen menetelmä laadullisessa tutkimuksessa. Tässäkin tutkimuksessa haastatteluista saatua aineistoa rikastettiin havainnoimalla tutkimusongelmaan liittyviä dokumentteja ja menetelmiä.

Havainnointi yksinään on haasteellinen tiedon keruu menetelmä, mutta yhdessä toisen, tässä tapauksessa haastattelujen kanssa, sen käyttö on monesti hyvinkin hedelmällistä. Havainnoinnin käyttö on perusteltua varsinkin silloin, kun tutkittavasta aiheesta tietoa on vähäisesti saatavilla ja haastattelun toteuttaminen vähäisen tiedon takia on hankalaa. Havainnoilla voidaan myös monipuolistaa tutkittavasta ilmiöstä haluttua tietoa tai yhdistää muista aineistonkeruumenetelmistä saatua tietoa oikeisiin konteksteihin. Havainnointia käytettäessä on kuitenkin hyvää muistaa, että se on suuritöinen ja aikaa vievä menetelmä. (Tuomi & Saarijärvi 2009, 81)

Havainnointia voidaan suorittaa eri menetelmin; piilohavainnointi, havainnointi ilman osallistumista, osallistuvana havainnointi tai osallistavana havainnointi. Piilohavainnoinnissa tutkija osallistuu kohteiden elämään, mutta tutkittavat eivät tiedä osallistuvansa

tutkimukseen. Menetelmä on harvoin käytetty ja se sisältää pahoja eettisiä ongelmia, koska tutkittavat eivät tiedä osallistuvansa tutkimukseen (Tuomi & Saarijärvi 2009, 81).

Havainnointi ilman osallistumista ja osallistuvan havainnoinnin erottamisella ei ole selkeä rajaa. Tutkimukseen osallistuvat tietävät osallistuvansa tutkimukseen ja siihen on heidän lupansa. Suurin ero ilman osallistumista tapahtuvassa havainnoinnissa ja osallistuvassa havainnoinnissa on tutkijan vuorovaikutus tutkittavaan. Havainnoinnissa ilman osallistumista tutkija tarkkailee toimintaa ilman vuorovaikutusta, kun taas osallistuvassa havainnoinnissa hän on vuorovaikutuksessa tutkittavien kanssa. Tässä tapauksessa havainnointia suoritettiin ilman osallistumista tutkimalla olemassa olevaa testausdokumentaatio sekä seuraamalla testausta yleisesti (Tuomi & Saarijärvi 2009, 81-82).

Samoin osallistuvan ja osallistavan havainnoin erottaminen on hankalaa niiden samankaltaisuuden vuoksi. Osallistavassa havainnoinnissa painotus on yhteisen hoidon aspekteihin, jossa pyritään tutkimukseen osallistuvat osallistuttamaan siten, että toiminta jatkuisi samanlaisena myös tutkimuksen päätyttyä (Tuomi & Saarijärvi 2009, 82).

Olemassa olevasta ohjeistuksesta sekä testauksen suorittamisesta poimittiin oleellisia, tutkimuksen viitekehykseen liittyviä tietoja. Dokumentaatiota havainnoimalla pystyttiin muodostamaan ulkopuolinen näkökulma testauksen suunnittelusta, ohjeistuksesta ja dokumentaation tasosta. Itse testauksen suorittamista havainnoimalla taas saatiin näkemys siitä, miten testaus liittyy ohjelmistokehitykseen, mitä testausmenetelmiä on käytössä ja miten vastuut testauksen eri vaiheista on jaettu.

3.1.2 Tulosten analysointi

Laadullisessa tutkimuksessa on useita eri menetelmiä suorittaa aineiston analyysi, mutta käytetyin ja lähes kaikkiin tutkimuksiin soveltua tutkimusmenetelmä on sisältöanalyysi. Sitä voidaan käyttää itsenäisenä metodina tai väljänä teoreettisena kehyksenä. Sisältöanalyysi menetelmänä voidaan dokumentteja, kuten haastattelut, kirjat, puhe, raportit jne., analysoida systemaattisesti ja objektiivisesti. Tässäkin tutkimuksessa aineiston analyysi toteutettiin sisältöanalyysina (Tuomi & Saarijärvi 2009, 92,103).

Sisältöanalyysissä aineiston analyysimuotoja on useampia; aineistolähtöinen analyysi, teoriaohjaava analyysi ja teorialähtöinen analyysi. Analyysien erot pohjautuvat tutkittavaa ilmiötä kuvaavaan teorian vaikutukseen aineiston hankinnassa, analysoinnissa ja

raportoinnissa (Taulukko 4). Teoria- ja aineistolähtöisessä analyysissä aineiston hankinta on vapaampaa, kun taas teoriaohjaavassa se perustuu jo tiedettyyn tietoon. Raportoituna taas teorialähtöinen ja –ohjaava analyysi eivät juurikaan eroa toisistaan, kun aineistolähtöinen taas saattaa erota suurestikin muiden analyysien tuloksista (Tuomi & Saarijärvi 2009, 98)

Tulosten analyysin muodoksi valikoitui teorialähtöinen analyysi, jossa analyysi pohjautuu testauksen teoriaan ja sitä ohjaa valmis aiemman tiedon pohjalta luotu kehys. Teorian perusteella hahmoteltiin kategoriat, joihin aineisto suhteutetaan.

Taulukko 4. Laadullisen tutkimuksen analyysimuodot (Tuomi & Saarijärvi 2009, 99)

	Viitekehys	**Aineiston hankinta**	**Aineiston analyysi**	**Raportointi**
Aineistolähtöinen	• metodologia • tutkittavasta jo tiedetty	• metodologia ohjaava • vapaa	aineistolähtöinen	aineistolähtöinen
Teoriaohjaava	• metodologia • tutkittavasta jo tiedetty	• metodologia ohjaava • vapaa	• teoriaohjaava • aineistolähtöinen, johon liitetään teoriaohjaava	teoriaohjaava
Teorialähtöinen	• metodologia • tutkittavasta jo tiedetty	teorialähtöinen	teorialähtöinen	teorialähtöinen

Analyysin teko noudatti pitkälti Timo Laineen (1993, Tuomi & Saarijärvi 2009, 92 mukaan) kuvaamia vaiheita analyysin etenemiseksi:

1. Valittiin tarkasti rajattuna tutkittu ilmiö
2. Kerätyn aineiston koodaaminen
3. Aineiston ryhmittely ja teemojen etsiminen
4. Analyysin yhteenveto

Tutkittuvaksi ilmiöksi valittiin testausprosessi organisaatiossa ja muut haastattelussa esiin tulleet ilmiöön liittymättömät tiedot jätettiin tässä tutkimuksessa taka-alalle. Haastatteluista oli kirjattuna yksityiskohtaiset vastaukset, joista kaikki tutkittavaan ilmiöön liittyvät vastaukset ja aineisto koodattiin myöhempää ryhmittelyä varten. Haastattelujen lisäksi tehdyistä havainnoista poimittiin tutkittavaan ilmiöön liittyvät huomiot.

Koodauksen pohjalta aineisto ryhmiteltiin haastatteluissa käytettävien teemojen alle ja vastauksia alettiin tutkia tarkemmin. Ryhmitellystä aineistosta vertailtiin eri teemojen

esiintymisistä vastauksista ja näiden tietojen pohjalta muodostui kokonaiskuva tutkittavasti ilmiöstä. Lopuksi tehdystä analyysistä koostetiin yhteenveto, jonka perusteella prosessia voidaan lähteä kehittämään.

3.2 Nykytilanne ja kehitystarpeet

Riippumatta haastateltavien ammattinimikkeistä ja työtehtävistä, haastatteluissa esille tulleet mielipiteet olivat melko yhtenäisiä. Haastateltavien mielestä nykyisessä prosessissa on jo hyviä käytänteitä ja tapoja, mutta myös kehitettäviä kohteita tuli esille. Kehitettävien kohteiden tärkeydessä oli havaittavissa painottumista eri osa-alueisiin, riippuen haastateltavan työtehtävistä ja kokemuksesta.

Haastattelujen lisäksi tehty havainnointi vahvisti haastatteluissa esille tulleita asioita. Tämän hetkiseen testausprosessiin liittyvästä dokumentaatiosta ja testauksen suorittamisesta tehdyt havainnot täydensivät haastatteluista saatua tietoa. Havaintojen perusteella ei kuitenkaan löydetty uusia, haastatteluissa piiloon jääneitä kehityskohteita. Haastattelujen ja havaintojen perusteella saatiin selkeä käsitys testausprosessin nykytilasta. Esille tuli sekä hyvin olevat asiat, että kehityksen tarpeessa olevat asiat.

Tutkimuksen viitekehyksen pohjalta haastattelut koostuivat kolmesta eri teemasta: testausstrategia, testauksen suunnittelu ja dokumentointi sekä testausmenetelmät ja -automaatio. Näitä teemoja käytettiin hyödyksi aineiston analysoinnissa. Haastatteluissa esille tulleet asiat sekä tehdyt havainnot ryhmiteltiin edellä mainittujen teemojen alle ja näistä tiedoista koostettiin tutkimuksen raportti. Kaikkiin haastattelun teemoihin saatiin kattavasti vastauksia ja pääsääntöisesti vastaukset olivat melko yhtenäisiä haastateltavien kesken. Samoin tehdyt havainnot olivat linjassa haastatteluissa saatujen tulosten kanssa.

Testausstrategia

Testausstrategiasta nousi päällimmäisenä esiin epämääräisyys testauksen suunnitelmallisuudessa. Varsinkin kehityksen aikana tehtävien testitapausten määrittely on suurimmaksi kehittäjän vastuulla. Tämä aiheuttaa sen, että testien tasossa ja kattavuudessa on eroja riippuen testien toteuttajasta ja hänen mielenkiinnostaan testaamiseen. Tilanteissa, joissa kehittäjän lisäksi myös johtava kehittäjä määrittelee testitapauksia, testien taso ja kattavuus ovat tasaisempia.

Suunnitelmallisuuden epämääräisyyden lisäksi esille nousi käytänteiden ja ohjeistuksen puute testattavien asioiden ja testitapausten määrittelyyn. Haasteltavien mielestä selkeä ohjeistus testauksen suorittamiseksi parantaisi testauksen laatua. Esille nousi myös epämääräisyys vastuusta. Tällä hetkellä testauksen kehittäminen ja suunnittelu sekä testitapausten määrittely eivät ole virallisesti kenenkään vastuulla.

Testauksen nykyinen taso arvioitiin olevan parhaimmillaan tyydyttävällä tasolla. Osa testaamisesta suoritetaan huolellisesti ja kattavasti, mutta paljon on tilanteita, jolloin testaamisesta tingitään. Näissä tilanteissa ajanpuuteen ja kiireen vuoksi testaus tehdään nopeasti ja ilman suunnitelmallisuutta. Testaus on usein optimistisista ja se pohjautuu liikaa odotettuun käytökseen, ei poikkeustapauksiin. Testaamisen tason nostamiseksi esille nousi testauksen yhdenmukaistaminen riippumatta testaajasta. Testaukseen tulisi ottaa mukaan toistuvat käytänteet ja testit tulisi määritellä ja suorittaa aina tietyllä tavalla.

Testauksen resursoinnista mielipiteet jakautuivat eniten. Yhteenvetona voidaan kuitenkin todeta, että nykyisenlaiseen testaukseen varatut resurssit ovat pääsääntöisesti riittäviä. Jos testauksen määrää ja kattavuutta nykyisin menetelmin lisätään, testauksen resurssit riittyvyys saattaa muodostua ongelmaksi. Haasteltavien mielestä testaamisen paremmat määrittelyt ja suunnitelmallisuus mahdollistaisivat paremman testauksen nykyisilläkin resursseilla. Tällöin testauksen tasoa ja laatua voidaan parantaa ja heikosta testauksesta johtuvien korjausten tekemiseen hukatut resurssit ainakin osittain siirtyisivät testaukseen. Testaukseen kuluvista resursseista ei ole käytössä selkeää mittaria, mistä syystä resurssien riittävyyden arviointi pohjautui haastateltavien omaan arvioon.

Testauksen suunnittelu ja dokumentointi

Testauksen suunnittelusta ja dokumentoinnista haasteltavien vastaukset olivat melko yhteneviä. Testitapauksista ei erikseen ole suunnitelmia tai määrittelyitä. Testitapaukset toteutetaan pääosin testin suorittajan oman arvion pohjalta. Jonkin verran testitapauksia voidaan tehdä ominaisuuden vaatimusmäärittelyn ja teknisen määrittelyn perusteella. Vaatimusmäärittelyn ja teknisen määrittelyn käyttö testien suunnittelussa riippuu kuitenkin pitkälti niiden toteutuksen laajuudesta. Osassa tapauksissa määrittelyt ovat suppeita ja osassa kattavampia. Lisäksi laajempien kokonaisuuksien yhteydessä pitäisi testien määrittelyt laatia jonkun muun kuin kehittäjän toimesta.

Testauksen suunnittelu pitäisi ottaa huomioon jo määrittelyiden yhteydessä sekä kehityksen aikana. Näin toimittaessa kirjoitettu koodi olisi toteutettu niin, että se olisi mahdollisimman helppoa testata. Samoin testauksen kehittäminen sekä testauksen suunnittelu

ja määrittely olisi hyvä antaa jonkun henkilön vastuulle. Tällä hetkellä testaus ja siitä vastaaminen ei ole kenenkään vastuulla.

Haasteltavien mielestä tämän hetkisessä kehitystavassa suunnitelmien ei tarvitse olla tarkka kuvaus siitä mitä testataan. Suunnitelmaksi riittää yksinkertainen ja selkeä lista, jossa on määriteltynä lyhyesti mitä asioita halutaan testata. Jokaista testitapausta ei ole tarvetta kuvata yksityiskohtaisesti. Riittää, että määriteltynä on mitä ominaisuuden pitäisi tehdä, miten sen kuuluisi toimia ja mitä poikkeuksia pitäisi huomioida.

Suunnitelmallisuuden puute heijastuu myös testauksen dokumentointiin. Dokumentointi on suppeaa, lähinnä kommentteja ohjelmallisesti suoritettavien testitapausten ohjelmakoodissa. Haastatteluissa kuitenkin ilmeni, että dokumentaation laaja-alaista toteuttamista ei koeta järkeväksi. Dokumentaatioksi riittää testaussuunnitelma, jossa on lyhyesti kuvattuna tarvittavat testiskenaariot. Dokumentaation ei tarvitse kattaa yksityiskohtaisesti jokaista testausvaihetta ja testitapausta vaan se olisi muistilista tyyppinen tekninen dokumentaatio, jossa on kuvaus siitä, mitä testataan ja miksi.

Testausmenetelmät ja –automaatio

Tällä hetkellä testaus koostuu kehityksen aikana testausohjelmistolla suoritettavista yksikkötesteistä ja manuaalisesti suoritettavista järjestelmä- ja hyväksymistestauksesta. Regressiotestausta tehdään yksikkötestejä hyödyntäen ja isompien muutosten yhteydessä regressiotestausta tehdään myös manuaalisesti. Testaus on painottunut pitkälti manuaalitestaamisen ja käyttöliittymän osalta testaus suoritetaan pelkästään manuaalisesti.

Haastateltavat olivat pitkälti samaa mieltä testausmenetelmien ja automaation käytön tasosta. Testausmenetelmissä olisi kehitettävää ja automaation lisääminen helpottaisi ja parantaisi testaamista. Varsinkin käyttöliittymän osalta testausmenetelmät ovat vajavaiset ja testausautomaatio ei ole käytössä. Manuaalitestaamisella on oma paikkansa testauksessa, mutta automaation lisääminen mahdollistaisi laadukkaamman regressiotestauksen sekä vähentäisi testaajan vaikutusta testien suorittamiseen.

Testausmenetelmien kehittäminen ja automaation lisääminen järkevissä määriin koettiin tarpeelliseksi, jotta testauksen kattavuutta voidaan lisätä. Uusien menetelmien käyttöönotto tehostaisi testaamista ja parantaisi varmuutta järjestelmän toimivuudesta. Varsinkin käyttöliittymän osalta automatisoitu end-to-end -testaus parantaisi huomattavasti tes-

tauksen tasoa. Automaation lisääminen myös tehostaisi pidemmässä juoksussa testauksen resurssien käyttöä. Automaation avulla manuaalitestaukseen käyttöä voidaan vähentää ja säästyvät resurssit suunnata esimerkiksi testauksen kattavuuden lisäämiseen.

Järjestelmä on laaja kokonaisuus ja se laajenee koko ajan. Ideaalisessa tilanteessa toimivien testausmenetelmien ja automaation avulla suoritetun testauksen jälkeen voisi olla luottavainen, että ainakin tärkeimmät omaisuudet toimivat.

3.3 Kehitettävät osa-alueet

Kerätyn aineiston pohjalta voidaan todeta, että testausprosessin kehittämiselle on tarvetta. Hyvin määritellyn testausprosessin avulla testaamisen käytettyjen resurssien käyttöä voidaan tehosta ja sitä kautta testaamisen kattavuutta ja laatua voidaan parantaa. Tärkeimmät kehityksen kohteet ovat testauksen suunnittelussa ja testauksen automatisoinnissa. Pienemmälle huomiolle voidaan jättää testauksen dokumentointi ja testausmenetelmien muuttaminen.

Testausprosessin kehittämiseen oman haasteensa tuo organisaatiossa tehtävän kehitystyön luonne. Kehitystyö tapahtuu osakokonaisuus tai ominaisuus kerrallaan. Kehittäminen noudattaa samaan kaavaa kuin tavanomainen ohjelmistokehityskin; ominaisuus määritellään, suoritetaan kehitystyö ja testaus ja ominaisuus otetaan käyttöön. Tavanomaisesta ohjelmistokehityksestä poiketen kehityksen iteraatiot ovat yleisesti huomattavasti nopeampia ja suppeampia. Ohjelmisto kehitys pohjautuu käyttäjien tarpeeseen ja vastaan tulee tilanteita, jolloin ominaisuus pitää saada nopeasti käyttöön. Jokaisesta ominaisuudesta, varsinkin pienimmistä, ei ole järkevää toteuttaa laajoja dokumentaatioita tai suunnitelmia. Testauksen liittyvät suunnitelmat ja dokumentaatio voidaan toteuttaa osana ominaisuudelle tehtäviä määrittelyitä.

Testauksen nykytilannetta ja kehitystarpeita tarkasteltaessa havaittiin, että testauksen suunnitelmallisuus koettiin vajavaiseksi. Jotta testauksen suunnitelmallisuutta voidaan parantaa ja selkeyttää, pitää testauksen suunnittelulle määritellä selkeät käytänteet, jota kaikki testaukseen liittyvät osapuolet noudattavat kehityksen aikana.

Testauksen suunnittelun tueksi tullaan laatimaan ohjeistus, jonka pohjalta jokainen testaussuunnitelma toteutetaan. Ohjeistuksessa kuvataan testausmenetelmittäin, minkä tyyppistä testausta ja millä tavoin testaus suoritetaan. Ohjeistuksessa on kuvattuna millä

tavoin testitapaukset esitetään suunnitelmassa ja mitkä asiat pitää ottaa huomioon testitapauksia suunniteltaessa. Ohjeistuksessa on kuvattuna myös muut testaukseen liittyvät käytänteet, jolloin testaus tapahtuu aina samalla tavalla riippumatta testaajasta.

Testausmenetelmien käyttöä tullaan laajentamaan, jolloin testaamisesta saadaan kattavampaa. Yksikkö- ja integraation testien lisäksi testausmenetelmät tullaan laajentamaan kattavampaan järjestelmä- ja hyväksymistestaukseen. Edellä mainittujen lisäksi tarvetta on myös end-to-end -testaukselle, joka parantaa varsinkin regressiotestauksen laatua yksikkö- ja integraation testien kanssa. Mahdollisuuksien mukaan end-to-end -testauksen toteutukseen otetaan testausautomaatiota, jolloin pidemmässä juoksussa regressiotestauksesta vapautuu resursseja muuhun testaamiseen tai itse kehitystyöhön.

Vaikka tärkeimmät kehityskohteet ovat testauksen suunnittelussa ja testausmenetelmissä, myös dokumentaatiota tullaan kehittämään. Suunnittelun ja testaamisen dokumentointia varten laaditaan dokumenttipohja, mihin on helppo koostaa kuvaus testattavasta ominaisuudesta, testaukseen liittyvät vastuut, vaadittavat testit eri menetelmissä ja merkitä testauksessa ilmenneet virheet ja puutteet. Dokumentaation pohjalta testaajien on helppo havaita minkä tyyppisiä testejä pitää tehdä ja projektijohto taas pystyy helposti seuraamaan testauksen etenemistä.

3.4 Prosessin kehittäminen

Prosessin kehittämisellä tähdätään pääsääntöisesti toiminnan tehostamiseen, toiminnan laadun ja palvelutason parantamiseen, ongelmatilanteiden hallintaan sekä kustannussäästöjen aikaansaamiseen. Prosessin kehittäminen liittyy aina organisaation kehittämiseen ja sitä ohjaa samat visiot, strategiat ja toimintaperiaatteet, kuin organisaation muutakin toiminta. Prosessin kehittämisen laajuus vaihtelee jatkuvista muutoksista isoihin kehityshankkeisiin (JUHTA 2012, 3).

Prosessin kehittäminen lähtee usein liikkeelle ongelmasta ja sen tavoitteena on parantaa prosessin mitattavuutta, parantaa prosessin käytettävyyttä tai luotettavuutta. Muutos ei kuitenkaan saa olla kertaluonteinen vaan, sen tulee pyrkiä jatkuvaan kehittämiseen ja vaikutusten arviointiin. Prosessia kehitettäessä ei kuitenkaan yhdellä kertaa kannata muuttaa liian montaa asiaa ja muutoksille pitää olla hyvät perusteet. Prosessin kehittämisellä on myös tärkeää varata riittävästä aikaa ja resursseja (JUHTA 2012, 3).

Tässäkin työssä prosessin kehittäminen lähti liikkeelle nykyisessä testausprosessissa havaituista puutteista. Työssä pyrittiin löytämään nykyisen prosessin hyvät ja huonot käytännöt ja niiden pohjalta luotiin suuntaviivat uudistetulle prosessille. Prosessia kehitetään aluksi osana tätä kehitystyötä ja kehitystyön valmistumisen jälkeen prosessin kehittämistä jatketaan organisaation kehitystiimin ja projektijohdon kanssa.

3.4.1 Prosessin tunnistaminen

Prosessin tunnistaminen tarkoittaa prosessin rajaamista muista prosesseista. Prosessit luokitellaan yleisesti ydin- ja tukiprosesseihin. Ydinprosessit ovat organisaation ydintehtäviä, joita varten organisaatio on olemassa. Ydinprosessit ilmaisevat miten organisaatio pyrkii tavoitteeseen, mitä varten se on olemassa. Tukiprosessit taas mahdollistavat ydinprosessien toiminnan. Tukiprosessit ovat yhtä tärkeitä kuin ydinprosessitkin, mutta ydinprosessista eroten, ne ovat olemassa organisaation toimintaa varten, kun taas organisaation voidaan ajatella olevan toteuttamassa ydinprosesseja (Virtanen & Wennberg, 2007, 118).

Ohjelmistokehitysprosessi on yksi toimeksiantajan ydinprosesseista ja testausprosessi yksi siihen liittyvistä tukiprosesseista. Muita ohjelmistokehitysprosessiin liittyviä tukiprosesseita on mm. määrittelyprosessi, ohjelmointiprosessi ja ylläpitoprosessi. Tässä työssä kuitenkin keskitytään vain testausprosessiin, mistä syystä muita prosesseita ei tarkastella tarkemmin.

Prosessin tunnistamisen nyrkkisääntönä voidaan pitää ajattelua, että prosessi alkaa ja päättyy asiakkaaseen. Asiakas käsitteenä on kuitenkin monitahoinen ja prosessin välitön asiakas ei välttämättä ole asiakas, jota pyritään palvelemaan (Virtanen & Wennberg 2007, 116-117). Testaus prosessin osalta välittömänä asiakkaana voidaan pitää testattavan ominaisuuden tilaajaa, mutta todellisuudessa asiakkaita ovat ohjelmistoa käyttävät henkilöt, joiden työskentelyyn testausprosessin toimivuudella on vaikutusta. Asiakkaan kautta tapahtuvan tunnistamisen lisäksi prosessin tunnistamisen sääntönä on, että prosessi alkaa toimenpiteen suunnittelusta ja päättyy seurantaan. Tunnistaminen ei kuitenkaan saa lähteä liikkeelle tuotoksista, vaan vaikutuksista, joita prosessilla pyritään tuottamaan. Vaikutusten kautta tunnistettu prosessi varmistaa, että prosessin kehittämisen suuntautumisen vaikuttavuuteen (Virtanen & Wennberg 2007, 117-118).

3.4.2 Prosessin määrittely ja kuvaaminen

Prosessien tunnistamisen jälkeen siirrytään prosessien kuvaamiseen, jossa prosessin sisältöä täsmennetään. Prosessista esitetään siihen liittyvät keskeiset vaiheet ja niiden keskinäiset yhteydet, vastuut ja prosessiin vaikuttavat kriittiset suorituskykytekijät. Valittu prosessi rajataan, jotta voidaan varmistua, että prosessin alku ja loppu on määritelty oikein. Rajaus voidaan tehdä esimerkiksi niin, että prosessi alkaa ja päättyy aina, asiakkaaseen. Rajauksessa pitää muistaa huomioida prosessin tarkoituksen mukaisuus ja hallittavuus. Prosessin rajaamista kannattakin miettiä huolella, sillä jos prosessi on rajattu liian löysästi, prosessia voi olla vaikea hallita ja hahmottaa, mutta myöskään liian tiukalla rajaukselle ei saada prosessikuvaukselle lisä arvoa (JUHTA 2012, 4).

Ennen aloittamista on tärkeää miettiä, miksi kuvaus tehdään ja millä tasolla prosessi on tarpeellista kuvata. Ei ole tarkoituksen mukaista tehdä liian yksityiskohtaista kuvausta. Riippumatta prosessikuvauksen tarkkuudesta, prosessien määrittely koostuu aina samoista toimenpiteistä, jotka ovat kuvattuna taulukossa 5. Prosessia kuvattaessa on koko ajan oltava tiedossa, minkä tasoista kuvausta ollaan tekemässä ja mihin ylätason prosessiin kuvaus liittyy. Samoin on tärkeää muistaa, että ennen kuin prosessin kuvausta ja määrittelyä voidaan tehdä, pitää olla tiedossa prosessin sisältö ja työnkulku. Prosessia kuvattaessa pitää hahmottaa, miten prosessi vaiheistetaan, ketkä vaiheisiin osallistuvat ja millainen kaavio prosessista laaditaan (JUHTA 2012, 5; Virtanen & Wennberg 2007, 121-123).

Taulukko 5. Prosessin määrittelyn keskeiset vaiheet (Virtanen & Wennberg 2007, 122)

Ominaisuus	**Määrittelyyn liittyvä tehtävä**	**Tavoite**
Prosessi on loogisesti toisiinsa liittyvien toimintojen sarja	Tunnista prosessin keskeiset vaiheet	Prosessin osaprosessit on tunnistettu ja järjestetty toimintojen sarjaksi
Prosessi koostuu useista osaprosesseista ja toiminnoista, joilla on omat vastuuhenkilönsä	Tunnista osaprosessien keskeiset toiminnot ja näistä vastaavat henkilöt	Osaprosessien sisältö on määritelty ja vastuutettu
Prosessin tuloksena syntyy suoritteet, joiden kautta vaikutukset syntyvät	Tunnista prosessin keskeiset suoritteet liittyen prosessin kuhunkin vaiheeseen	Prosessien keskeiset suoritteet tunnistettu ja nimetty
Prosessin suorituskykyä mitataan	Määrittele sidosryhmäkohtaiset vaatimukset, näitä kuvaavat mittarit ja tavoitearvot	Selkeä käsitys prosessin suorituskykytavoitteista

Prosessikuvaus rakentuu toisiaan täydentävistä prosessin perustiedoista, sanallisesta kuvauksesta ja kaaviosta. Perustietojen avulla voidaan tunnistaa prosessin lähtökohdat. Perustiedoista pitää selvitä, mihin tarkoitukseen prosessia mallinnetaan ja mitkä ovat prosessin keskeiset tiedot. Sanallinen kuvaus kuvaa tarkemmalla tasolla prosessiin liittyviä tehtäviä. Siinä esitetään yksityiskohtaisesti prosessin vaiheet, toiminnot, tehtävät, toimijat sekä lähtö- ja tulostila. Kaavio, taas on graafinen esitys prosessista ja se laaditaan samaan aikaan kuin sanallinen kuvaus (JUHTA 2012, 5).

Prosessin kuvantamisen tason päättää prosessin omistaja. Tason valinnassa perusperiaatteena voidaan pitää sitä, että mitä muodollisemmiksi kuvaukset muuttuvat sitä tarkemmalla tasolla prosessi kuvataan. Kuvauksen tarkkuus vaikuttaa varsinkin kaavion ja prosessissa kulkevan tiedon esittämisen tarkkuuteen. Prosessin kuvaamisen tasot jaetaan yleensä neljään eri kuvaustasoon; prosessikartta, prosessin toimintamalli, prosessin työnkulku ja prosessin toimintataulukko. Taulukossa 6 on kuvattuna tasojen välisiä eroja. Eritasojen kuvaukset menevät osittain päällekkäin ja niiden erot voivat olla pieniä, riippuen mm. kuvausten käyttötarkoituksesta tai tehtävien monipuolisuudesta.

Taulukko 6. Prosessikuvauksen tasojen hierarkia (JUHTA 2012, 6-10; Virtanen & Wennberg 2007, 127)

Prosessikartta	• Yleiskuvaus organisaation toiminnasta ja sitä tukevasta prosessirakenteesta • Jäsentää prosessien kokonaisuuden • Korvaa prosessiorganisaatiossa perinteisen organisaatio kaavion
Prosessin toimintamalli	• Yleiskuva prosessin sisällöstä • Prosessin keskeiset työ- ja osaprosessit tunnistettu • Kuvaa prosessihierarkian ja sitoo prosessit yhteen
Prosessin kulku	• Kuvaa toiminnan periaatteet • Kuvaa, millaisten työvaiheiden kautta prosessit etenevät organisaatioyksikön sisällä • Kuvaa prosessin toimintojen ja tehtävien lisäksi myös niiden suorittajat.
Prosessin työnkulku	• Kuvaa toiminnan työvaiheet vaihevaiheelta • Näyttää yksilöllisen työn • Näyttää vaiheeseen liittyvät toimenpiteet

Prosessikartta on prosessikuvausten ylin, pelkistetyin taso, jossa esitetään kokonaiskuva organisaation toiminnasta. Prosessikartassa esitetään ydin ja tukiprosessit sekä pelkistetty organisaatio ja toimintaympäristö, prosessin välisiä suhteita ja riippuvuuksia ei kuvata. Ydinprosesseilla ilmaistaan organisaation tavoitteet ja tukiprosessit luovat edellytykset ydinprosessien toiminnalle. Prosessikartta on kokonaiskuva organisaation

toiminnasta, jota hyödynnetään ulkoisen viestinnän ja päätöksenteon apuvälineenä (JUHTA 2012, 6-7).

Prosessin toimintamallissa organisaation toimintaa kuvataan prosessikarttaa tarkemmin. Toimintamallissa on kuvattuna prosessien hierarkia, eli niiden jakautuminen osaprosesseiksi, prosessien väliset vuorovaikutukset ja riippuvuudet sekä rajapinnat. Lisäksi esitetään prosessien omistajat ja vastuut, tavoitteet ja mittarit. Toimintamalli antaa kokonaiskuvan organisaation toiminnasta ja siinä on nähtävissä prosessien kulku ja prosesseihin vaikuttavat tekijät (JUHTA 2012, 7-8).

Prosessin kulussa tiedot esitetään yksityiskohtaisemmin kuin toimintamallissa. Tasolla kuvataan toiminnan työvaiheet, toiminnot ja vastaavat toimijat. Prosessin kuvaamisessa kiinnitetään huomioita prosessin jakautumiseen osaprosesseiksi, toiminnoiksi ja tehtäviksi. Osaprosessit, toiminnot, tehtävät ja syötteet ovat nimettyjä ja niiden tarkoitus ja tiedot kuvataan, osaprosessit ja tehtävät myös numeroidaan. Prosessin kulussa on kuvattuna osaprosessien välinen vuorovaikutus ja osaprosesseista kirjataan omistajat ja vastuut. Tehtävistä kirjataan suorittajien roolit ja asiakas nimetään. Kaaviossa on kuvattuna myös sidosryhmät, liittymät muihin prosesseihin sekä prosessin toteuttamiseen osallistuvat tietojärjestelmät (JUHTA 2012, 8-9).

Kehitystyössä testausprosessista kuvattiin prosessin kulkua. Prosessin toimintamalli olisi ollut kuvauksena liian suppea, sillä siitä puuttui mm. prosessin ei vaiheiden suorittajat. Prosessin työnkulun kuvaaminen taas olisi ollut liian laaja ja yksityiskohtainen. Tässä vaiheessa ei ollut järkevää kuvata yksityiskohtaisesti eri työvaiheita, vaan nyt prosessin kehittämisessä keskityttiin prosessin vaiheistukseen ja vastuisiin. Prosessin prosessikuvauksessa (liite 2) määriteltiin työvaihekohtaisesta toiminnot, tehtävät ja toimijat. Prosessikuvauksen tueksi laadittiin prosessikaavio (liite 3), jossa on graafisesti esitetty prosessin toiminnot, tehtävät ja toimijat.

4 ARVIOINTI JA POHDINNAT

4.1 Pätevyys ja luotettavuus

Tutkimusta tehtäessä on tärkeää arvioida tehdyn tutkimuksen luotettavuutta. Muista tutkimustavoista poiketen laadullisen tutkimuksen kohdalla on erilaisia käsityksiä siitä, miten tutkimuksen luotettavuutta arvioidaan. Tutkimusmenetelmien luotettavuutta käsitellään useasti validiteetin ja reliabiliteetin kautta. Laadullisessa tutkimuksessa näiden käyttäminen kuitenkin kohtaa kritiikkiä, sillä käsitteenä ne vastaavat vain määrällisen tutkimuksen tarpeisiin. Laadullisen tutkimuksen oppaissa suositetaankin käsitteiden hylkäämistä tai korvaamista laadullista tutkimusta arvioitaessa. Lähteestä riippuen laadullisen tutkimuksen luotettavuustarkastelua voidaan kuvata hyvinkin eri tavoin. Tämän seurauksena saattaa herätä kysymyksiä, onko laadullisessa tutkimuksessa yhtenäistä käsitystä tutkimuksen luotettavuudesta. Laadullisen tutkimuksen osalta esille nousee helposti kysymykset totuudesta ja objektiivisesta tiedosta. Samoin koetaan, että näkemys totuuden luonteesta vaikuttaa suhtautumiseen tutkimuksen luotettavuuteen (Tuomi & Saarijärvi 2009, 134-137).

Vaikka laadullisen tutkimuksen luotettavuuden arvioinnille ei ole olemassa yksiselitteistä ohjetta, voidaan useimmiten arviointi toteuttaa pohtimalla esimerkiksi (Tuomi & Saarijärvi 2009, 140-141):

- Mitä on tutkittu ja miksi?
- Miksi tutkimus on tärkeä?
- Miten aineiston keruu on tapahtunut menetelmänä ja tekniikkana?
- Millä perusteella tiedonantajat valittiin?
- Miten aineisto analysoitiin?
- Miksi tutkimusraportti on luotettava?

Työhön liittyvässä tutkimuksessa tutkimuksen kohteena oli ohjelmistokehityksen testauksen nykytila. Tutkimus toteutettiin, jotta testausprosessia voitiin kehittää vastamaan paremmin tämän hetkisiä tarpeita. Tutkimus oli tärkeä osa prosessin kehittämistä, sillä ilman tutkimusta prosessin kehittäminen olisi ollut hankalaa. Tutkimuksen avulla saatiin kattavasti tietoa prosessin nykytilasta, useasta eri näkökulmasta.

Aineisto kerättiin haastatteluiden ja havaintojen pohjalta. Haastateltaviksi valikoitui sisäisessä kehityksessä mukana olevat henkilöt projektijohdosta kehittäjiin. Haastateltavat valikoitiin siten, että kaikki kehitykseen ja varsikin testaukseen liittyvät roolit olivat edustettuna. Lisäksi aineistoa rikastettiin havainnoimalla nykyistä prosessia ja dokumentaatiota. Vaikka tutkimuksen aineistoa kerättiin useasta eri näkökulmasta, haastattelun kautta kerätty aineisto olisi syytä kerätä isommalta joukolta. Tutkimuksen luotettavuus olisi parempi, jos haastattelut olisi toteutettu isommalle joukolle. Tällöin samoja työtehtäviä tekevistä olisi saatu keskinäistä vertailupohjaa tutkimuksen tuloksiin.

Aineisto analysoitiin tarkasti rajattua ilmiötä tutkimalla. Aineiston analyysissä keskityttiin pelkästään toimeksiantajan sisäisen kehityksen testausprosessiin. Analyysissa pyrittiin ymmärtämään tutkimuksen kohdetta haastateltavien näkökulmasta, jolloin tutkimuksen raportti vastaa parhaiten esitettyyn ongelmaan. Aineiston ryhmittelyssä käytettiin pohjana haastattelussa olleita teemoja, joiden alle aineisto jaoteltiin. Analyysin pohjalta tehty raportti testauksen nykytilasta pohjautuu vertailuun aineiston ja testauksessa yleisesti vakiintuneiden käytönteiden välillä. Haastatteluista saatuja vastauksia verrattiin kirjallisuudesta saatavaan tietoon ja tämän pohjalta voitiin tehdä johtopäätöksiä analyysin raporttiin.

4.2 Tavoitteiden saavuttaminen

Tutkimuksen tavoitteena oli selvittää, miten testausprosessia voidaan kehittää, jotta voidaan varmistua ohjelmiston laadukkuudesta ja toimivuudesta. Tutkimuksessa kerätty aineisto antoi kattavan tiedon testauksen nykytilasta ja tämän tiedon pohjalta testausprosessia oli mahdollista kehittää parempaan suuntaan. Tutkimuksessa saatiin kattavasti vastauksia eri näkökulmista, jolloin voidaan todeta tutkimuksen täyttäneet sille asetetut tavoitteet, selvittää kattavasti testauksen nykytila. Kattavista vastauksista huolimatta tutkimusta kuitenkin pitää tarkastella kriittisesti johtuen haastateltavien vähäisestä määrästä. Jos tutkimus olisi voitu suorittaa laajemmalla otannalla, olisi tutkimukseen mahdollisesti saatu vielä uusi näkökulmia.

Tutkimuksessa kerätyn aineiston pohjalta rakennettu testausprosessi on kompromissi testauksen yleisten käytänteiden ja olemassa olevien resurssien välillä. Rajoitettujen resurssien vuoksi uusi prosessi ei noudata kaikkia testaukseen liittyviä periaatteita. Prosessi on suoraviivaisempi, käytettävissä oleviin resursseihin ja toimeksiantajan tarpeisiin

parhaiten vastaava ratkaisu. Prosessista laadittiin prosessikuvaus (liite 2) ja prosessikartta (liite 3). Prosessikuvauksessa on kuvattuna työvaihekohtaisesta toiminnot, tehtävät ja toimijat. Prosessikuvauksen tueksi laadittiin prosessikaavio, jossa on graafisesti esitettynä prosessin kulku vaiheittain.

Prosessiin on sisällytetty jonkin verran supistettuna testaukseen tärkeimmät osa-alueet; määrittely, suunnittelu, toteutus ja raportointi. Vaikka prosessi on supistettu versio testauksen olemassa olevista käytänteistä, laadittu prosessi ohjeistuksineen ja dokumentteineen vastaa asetettuihin tavoitteisiin ja antaa ratkaisun testausprosessin parantamiseen. Laadittua prosessia on helppo jatkokehittää sitä mukaa kun tarvetta ilmenee ja käytettävissä olevat resurssit antavat siihen mahdollisuuden.

Prosessin tueksi laadittiin dokumenttipohja (liite 4) testauksen suunnittelua ja dokumentointia varten. Valmista pohjaa käyttäen testauksen vastuut ja testattavat ominaisuudet tulee dokumentoitua aina samalla tavalla, riippumatta siitä kuka suunnittelun ja dokumentoinnin toteuttaa.

4.3 Jatkokehitys

Uudistettu prosessi ja prosessissa kuvatut menetelmät tullaan ottamaan käyttöön vaiheittain. Uusia ominaisuuksia kehitettäessä testauksessa tullaan noudattamaan uutta prosessia suunnittelun ja dokumentoinnin sekä testaustapojen osalta. Myöhemmin käyttöön otetaan lisää testaukseen liittyvää automaatiota. Lisäksi resurssien salliessa olemassa oleviin kriittisimpiin ominaisuuksiin implementoidaan prosessissa kuvattuja testejä, jotta voidaan kustannustehokkaasti varmistaa ominaisuuksien toimivuus kehityksen jatkuessa. Prosessin toimivuutta tullaan tarkastelemaan kehitystiimin ja projektinjohdon kanssa säännöllisesti. Mikäli tarkastelussa huomataan puutteita tai ongelmia, tehdään prosessiin tarvittavia muutoksia.

Työ rajattiin koskemaan pelkästään sisäisen kehityksen testausprosessin parantamista. Haastatteluissa ja havainnoissa esille nousut tarve testausautomaation käytön lisäämiselle sekä testaukseen on huomioitu prosessissa, mutta itse testausautomaatioon liittyvien työkalujen valinta ja käyttöönotto tullaan toteuttamaan myöhemmässä vaiheessa. Testausautomaation työkalujen valinta ja käyttöönotto toimeksiantajan käyttöön on laajempi kokonaisuus, johon pitää erikseen varata resurssia useammalta työntekijältä, jotta löydetään parhaiten soveltuvat työkalut. Automaation käyttöönoton tavoin ohjeistuksen

laatiminen jätettiin työn ulkopuolelle. Ohjeistus laaditaan prosessiin käyttöönoton yhteydessä yhdessä kehittäjien kanssa.

Uudistettua testausprosessia on mahdollista hyödyntää myös muissa yhteyksissä, kuin sisäisen kehityksen kehitysprosessissa. Testausprosessi kannattaa implementoida soveltuvin osin koskemaan myös kolmannen osapuolen toteutuksia. Esimerkiksi hyväksymistestaus voidaan suorittaa käyttäen testausprosessissa kuvattuja määrityksiä ja dokumentteja, jolloin testaus noudattaa soveltuvin osin samaa kaavaa, kuin sisäisessä kehityksessä suoritettava testaus. Tämä mahdollistaa testauksen yhtenäisen toteutuksen ja dokumentaation riippumatta kehityksen toteuttajasta.

LÄHTEET

Alasuutari, P. 2011, *Laadullinen tutkimus 2.0,* Vastapaino, Tampere.

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

JUHTA 2012, JHS 152 Prosessien kuvaaminen.

Kasurinen, J.P. 2013, *Ohjelmistotestauksen käsikirja,* Docendo Oy, Jyväskylä.

Laine, T. 1993, *Aistisuus, kehollisuus ja dialogisuus,* Jyväskylän Yliopisto, Jyväskylä.

McCabe, T.J. 1976, "A Complexity Measure", *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* vol. 2.

Mili, A. 2015, Software testing: concepts and operations, Wiley, Hoboken, New Jersey.

Myers, G.J., Sandler, C. & Badgett, T. 2012, *The art of software testing,* John Wiley & Sons, Hoboken, N.J.

Pitkäranta, A. 2014, *Laadullinen tutkimus opinnäytetyönä,* e-Oppi Oy, Jokioinen.

Pohjanen, R. 2002, *Tietojärjestelmien kehittäminen,* Docendo Oy, Jyväskylä.

Royce, W.W. 1970, Managing the Development of Large Software Systems.

Sillius, K. & Tervakari, A. 2006, 7.2.-last update*, 6.luento MATHM-57550 Kvalitatiiviset tutkimusmenetelmät 5 op*. Viitattu 2.1.2018 http://matriisi.ee.tut.fi/hmopetus/kval-tutk/2005/luennot2005/liitteet/kvalit070206.pdf.

Tilley, S.R. & Parveen, T. 2012, *Software testing in the cloud: migration and execution,* Springer, Heidelberg ; New York.

Tuomi, J. & Sarajärvi, A. 2009, *Laadullinen tutkimus ja sisältöanalyysi,* Kustannusosakeyhtiö Tammi, Helsinki.

Virtanen, P. & Wennberg, M. 2007, *Prosessijohtaminen julkishallinnossa,* Edita Prima Oy, Helsinki.

Haastattelukysymykset

1. Testausstrategia
   1.1. Miten testaus liittyy työhösi?
   1.2. Kuinka testattavat tapaukset päätetään?
   1.3. Miten testitapaukset on määritelty?
   1.4. Minkälaisena koet tämän hetkisen tilanteen testaamisen tasosta?
   1.5. Onko testaamiselle tarpeeksi resursseja? Miksi/miksi ei?
   1.6. Mitä haluaisit parantaa/muuttaa testauksessa?
2. Testauksen suunnittelu ja dokumentointi
   2.1. Millä tasolla testaussuunnitelmat ja testitapaukset on dokumentoitu?
   2.2. Kenen vastuulla on testauksen suunnittelu, testitapausten määrittely ja dokumentointi?
   2.3. Minkälaisen dokumentoinnin ja suunnittelun tason koet sopivaksi ja miksi?
   2.4. Mitä haluaisit parantaa/muuttaa testauksen suunnittelussa ja dokumentoinnissa?
3. Testausautomaatio, -menetelmät ja –työkalut
   3.1. Mitä testausautomaatioita on tällä hetkellä käytössä? Pitäisikö testausautomaation käyttöä lisätä?
   3.2. Suoritetaanko testaamista manuaalisesti?
   3.3. Minkälaisena koet tällä hetkellä käytössä olevien testausmenetelmien tason?
   3.4. Mitä haluaisit parantaa/muuttaa testausmenetelmien käytössä?
   3.5. Koetko tarpeelliseksi uusien testausmenetelmien ja -työkalujen käyttöönoton/käytön lisäämisen? Miksi?
4. Muut asiat
   4.1. Mihin osa-alueeseen/alueisiin testausprosessin kehityksessä pitäisi mielestäsi panostaa? Miksi?
   4.2. Muuta testausprosessiin liittyvää?

Prosessikuvaus (ei julkinen)

Testausprosessin prosessikaavio (ei julkinen)

Testaussuunnitelmadokumentti (ei julkinen)