TAMPEREEN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma
Ohjelmistotuotannon suuntautumisvaihtoehto
Maria Sorvo

Opinnäytetyö

Testauksen liittäminen ohjelmistoprojektiin

Työn ohjaaja FL Paula Hietala, Tampereen ammattikorkeakoulu
Tampere 10/2010

TAMPEREEN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Tekijä: Maria Sorvo
Työnnimi: Testauksen liittäminen ohjelmistoprojektiin
Sivumäärä: 44
Valmistumisaika: 10/2010
Työn ohjaaja: FL Paula Hietala

TIIVISTELMÄ

Luotaessa uusia ohjelmistoja käyttäjät saattavat tietämättään joutua toimimaan testaajina. Mikäli testaus toteutetaan ainoastaan tällä tavalla, on se riskialtista toimintaa ohjelmiston tuottaneen yrityksen näkökulmasta. Yrityksen täytyisikin, riippumatta siitä tuottaako se ohjelmistot itse vai tilaa muualta, panostaa testaukseen ja nähdä se tärkeänä osana ohjelmistoprojektia.

Tämän opinnäytetyön tarkoitus on toimia opasteena ja tukena yrityksen ohjelmistoprojekteista vastaaville henkilöille tilanteessa, jossa yrityksessä ei ole systemaattista testausta ja sellainen tulisi saada. Opinnäytetyö tarjoaa perustiedot testauksesta, esittelee erilaisia toteuttamismalleja sekä ohjeistaa dokumentoinnissa. Luettuaan opinnäytetyön lukijalla on perusteet siitä, mitä testauksen toteuttaminen vaatii yritykseltä ja kuinka testaus otetaan mukaan projekteihin.

Opinnäytetyön lähteinä on käytetty artikkeleita sekä tutkimuksia. Lisäksi esimerkkeinä käytetään omakohtaisia kokemuksiani työharjoitteluajaltani, jossa toimenkuvaani kuului testauksen integrointi yrityksen ohjelmistoprosesseihin. Prosessi jatkuu yhä ja opin itsekin jatkuvasti lisää.

Avainsanat testaus, ohjelmistotestaus, projektinhallinta, dokumentointi

TAMK University of Applied Sciences
Degree Programme in Business Information Systems

Writer: Maria Sorvo
Thesis: Testing into the Software Project
Pages: 44
Graduation time: 10/2010
Thesis superviros: FL Paula Hietala

ABSTRACT

When new software is being developed, users may, without even noticing it, end up as software testers. Software testing done this way is a bit risky. Software companies should, even if the software is their own product or ordered from another company, invest on testing and consider it as an important part of software development.

The purpose of this thesis is to function as a guide and support for software developers and for people working as project managers. It gives the basics for testing, presents different execution models and guides the documentation process. This thesis gives the reader the basics of what software testing requires from companies and how testing should be made a part of the software project.

Different articles and researches have been used as sources in writing this thesis. I have also included examples from my own experiences from my practical training, where my job description was ‘integration of software testing into the company's software process’. This process is still ongoing.

Keywords testing, software testing, project management, documentation

Sisällysluettelo

1 Johdanto

Ohjelmistotestaus on mitä tahansa tekemistä, jonka tavoitteena on selvittää, ovatko valmiin ohjelman toiminnot sellaisia, että ne täyttävät ohjelmistolle asetetut vaatimukset (Hetzel 1988). Ohjelmistotestausta on ollut niin pitkään, kun on kehitetty ohjelmistojakin. Ohjelman toimiminen voidaan todentaa ohjelmistotestauksen kautta vertaamalla saatua tulosta odotettuun tulokseen. On kuitenkin varomatonta kokeilla ohjelma nopeasti läpi ja ottaa se käyttöön tähän nopeaan läpikäyntiin luottaen.

Opinnäytetyöni käsittelee ohjelmistotestausta osana ohjelmistoprojektia siltä lähtökohdalta, että testaus kuuluu projektiin ja on ainoastaan tekosyitä, joiden vuoksi testausta ei suoriteta aktiivisesti. Opinnäytetyöni toimii tukena, kun yrityksessä on aika keskustella testauksen integroimisesta prosessiin eikä kaikilla ole ymmärrystä, mitä ohjelmistotestaus on ja mihin kaikkeen sillä voidaan vaikuttaa. Lähteinä toimivat sekä omakohtaiset kokemukset harjoitteluyrityksessäni että artikkelit, kirjat ja muut tekstipohjaiset lähteet.

Toisessa luvussa aloitetaan perusasioista: mitä ohjelmistotestaus on ja mitä se ei ole. Pohditaan, miksi testaus ajautuu toissijaiseksi yrityksissä: onko syy tahallinen vai tahaton, ja mitkä ovat yleisimpiä syitä. Toisaalta käsitellään etuja, joita testauksesta on yritykselle sekä välittömästi että pitkällä aikavälillä. On myös syytä tarkastella tilanteita, jolloin testaus on tarpeetonta.

Kolmannessa luvussa käydään läpi erilaisia testaustyyppejä ja arvioidaan, milloin niitä on syytä käyttää. Luku antaa kattavan kuvan testaustyypeistä eikä ole tarkoitettukaan, että yrityksessä välittömästi ryhdyttäisiin käyttämään kaikkia tyyppejä. Tämän luvun luettuaan lukijalla on käsitys erilaisista testaustyypeistä ja hän osaa tehdä päätöksiä, minkä tyyppistä testausta käyttää omassa projektissaan.

Neljännessä luvussa esitellään testauksen dokumentointia: annetaan esimerkkitapoja, miten luoda dokumentaatiota. Esitellään IEEE-standardi testaussuunnitelman luomiseen sekä ohjeistetaan testitapausten kirjoittamisessa. Lisäksi neuvotaan, millainen on hyvä testausraportti, josta saadaan mahdollisimman paljon informaatiota jatkokehitystä ja virheiden korjausta varten.

Viimeisessä luvussa mennään lähemmäs itse testausta: käydään läpi projektihallintaa ja siinä huomioitavia seikkoja, kun testausta otetaan mukaan prosessiin. Lisäksi tutustutaan erilaisiin testausprosessimalleihin ja tarkastellaan niiden hyviä sekä huonoja puolia, ja arvioidaan, mikä sopisi missäkin tilanteessa käytettäväksi.

Viidennessä luvussa pohditaan myös testauksen ulkoistamista; missä tilanteissa se on järkevää ja mitä riskejä siihen liittyy. Lisäksi tarkastellaan testauksen aloittamista: missä projektin vaiheessa testaus on syytä aloittaa ja mitä edellytetään tehdyksi ennen sitä. Toisaalta käsitellään sitä, milloin testaus on syytä lopettaa. Lopuksi luvussa käsitellään testauksen riskejä ja seikkoja, joilla on suuri merkitys testauksen menestyksekkäälle onnistumiselle.

Jotta asiat olisi helpompi ymmärtää, on luvuissa mukana kuvitteellinen projekti. Tässä projektissa tavoitteena on suunnitella ja ohjelmoida verkkopankkijärjestelmä. Esimerkin tarkoitus on tuoda esitettyjä asioita käytännön tasolle havainnollistamaan, kuinka kerrottu sijoitetaan todelliseen projektiin.

Opinnäytetyöni luettuaan lukija pystyy perustelemaan johdolle, miksi testaus on syytä ottaa välittömästi mukaan projekteihin ja kuinka tämä tapahtuu joustavasti. Hän pystyy suunnittelemaan testauksen käyttöönottoa ja sijoittamista yrityksen muihin prosesseihin.

2 Testaus yleisesti

Ennen kuin testaus voidaan aloittaa, on tiedettävä, mistä testauksessa oikeastaan on kyse. Testata voidaan monin eri keinoin ja testauksen voi ymmärtää jokainen tavallaan. Tämän luvun tarkoituksena on tarkastella yleisesti testausta sekä pohdiskella syitä, miksi testaus nähdään vähäpätöisenä osana ohjelmistoprojektia. Luonnollisesti ei pidä unohtaa testauksen tuomia etuja ja niiden tarkastelua.

Ohjelmistotestaus, kuten myös koko ohjelmistosuunnittelu, lähtee laatuajattelusta, Software Quality Assurance (Galin 2004). Sen tarkoituksena on huolehtia siitä, että prosessin aikana käytetään tarvittavia standardeja, seurataan ohjeistusta ja laaditaan dokumentteja. Laatulähtöisessä ajattelumallissa huolehditaan, että virheitä etsitään aktiivisesti ja ne käsitellään oikealla tavalla löytymisen jälkeen. Testaus kuuluu olennaisena osana tähän laadunvarmistukseen.

Laadulla voidaan tarkoittaa virheistä vapaata ohjelmistoa, joka toimitetaan ajallaan ja jonka budjetti pitää (Hower n.d). Tällainen määrittely on kuitenkin lähes mahdoton mittari laadunvarmistukseen. Vuonna 1995 laadittua Chaos-raporttia varten The Standish Group lähetti kyselyn yritysten it-päälliköille koskien ohjelmistoprojekteja heidän yrityksissään (The Standish Group 1995). Vastauksia tuli kaiken kaikkiaan 365, jotka koskivat 8380:tä ohjelmistoa. Tutkimus osoitti, että projektissa budjetti ylittyi keskimäärin **182 % keskisuurissa yrityksissä** ja jopa **214 % pienissä yrityksissä**. On siis lähes mahdotonta pitää budjettia laatukriteerinä. Aikataulun suhteen raportti antoi yhtä lohduttoman kuvan: keskisuurissa yrityksissä aikataulu ylitettiin 202 % ja pienissä 239 %. Kun luvut näyttävät tältä, ei kappaleen alussa luotu määritelmä laatumäärittelystä tunnu pitävän paikkaansa. On esitetty, että laatu on yhteensopivuutta vaatimusten kanssa (Ruuska 2007). Tämä suoraviivainen määritys on helppo ymmärtää niin toimittajan kuin tilaajan ja sen todentaminen on huomattavasti helpompaa aiemmin annettuun määrittelyyn verrattuna.

Ohjelmiston yhdenmukaisuus annettujen määrittelyjen kanssa todennetaan testauksella. Seuraavassa tarkastellaan tarkemmin, mitä testaus on.

2.1 Mitä testaus on

Chaos-raportin ensimmäisessä luvussa todetaan lakonisesti: me teemme samat virheet uudelleen ja uudelleen. Ohjelmistotuotannossa tämä on karu totuus. Nopealla haulla internet tuo eteemme lukuisia kertomuksia epäonnistuneita ohjelmistoprojekteista niin 1980-, 1990- kuin 2000-luvuilta. Testauksen tarkoituksena on vähentää näitä epäonnistumisia.

Ei pidä ymmärtää väärin: virheetöntä ohjelmistoa ei ole eikä tule. Ei pidä luulla, että kyse on tekijöiden tahallisesta yrityksestä tehdä virheellistä koodia. Ohjelmistot vilisevät virheitä monista syistä: nykyään ohjelmistot ovat monimutkaisia ja laajoja kokonaisuuksia, joiden tekeminen on hajautettu useammalle ohjelmoijalle, puhumattakaan muista projektiin osallistuvista henkilöistä ja heidän toimistaan (Hower n.d). Kun tähän kompleksisuuteen yhdistetään kommunikaatio-ongelmat, tilaajan vaihtuvat vaatimukset sekä aikataulupaineet, on helppo uskoa, että virheitä tulee.

On kaikkien etu pyrkiä kohti virheetöntä ohjelmistoa. Tähän tavoitteeseen päästään monien polkujen kautta; ohjelmoijien tulisi pyrkiä yhtenäiseen laadukkaaseen koodiin, aikatauluarvioiden tulisi olla realistisia ja määrittelyjen kunnossa ennen ohjelmointia. Tärkeintä on ohjelmiston testaus. Testauksella voidaan pyrkiä saamaan haluttu tulos ja sillä voidaan tarkoituksella etsiä virheitä (Nyman 2002). Ideaalitilanne on, että testausta voitaisiin suorittaa molemmilla tavoilla. Testauksen ensisijainen tavoite on laadunvarmistus. On kuitenkin mahdotonta tehdä testauksesta kaiken kattavaa.

Kuvitellaan ohjelma, johon syötetään kaksi 32-bittistä lukua. Kokeiltavia vaihtoehtoja on silloin 2^64. Vaikka tämä testaus automatisoitaisiin, kuluisi silti tuhansia vuosia, ennen kun testaus olisi ohi (Pan 1999). Voidaan siis vain kuvitella, millaisen työmäärän esimerkkiohjelmamme verkkopankin monipuolinen testaus edellyttäisi. Viimeistään nyt on ymmärrettävä, ettei testaus koskaan kata kaikkea.

Osana laadunvarmistusta testauksen tehtävä on löytää ohjelmasta virheitä. Löytämättä

jääneet virheet voivat tulla kalliiksi; pelkällä hakusanalla *ohjelmistovirhe* löytyy Google.comista suomeksi 6600 osumaa. Ensimmäisellä sivulla on luettavissa esimerkki vuonna 2004 tapahtuneesta virheestä, jonka vuoksi kokonainen lentokenttä oli suljettava (Goodwins 2004). Myös edellisissä kuntavaaleissa 2008 mukana ollut sähköinen äänestys jouduttiin uusimaan ohjelmistovirheen vuoksi (Vihdin kunta 2009). Ei ole siis yhdentekevää, kuinka paljon resursseja ohjataan testaukseen.

Samalla tavalla kuin testauksella pyritään löytämään virheitä, on sen tarkoitus toisaalta varmentaa määrittelyjen toimivuutta. Tällöin oletuksena ei ole etsiä virheitä, vaan varmistaa ohjelman toimivan niin kuin se on suunniteltu. Hyvällä ohjelmistotestauksella pyritään tilaan, jossa ohjelmiston toimivuus oikeilla syötteillä on varmistettu (Beizer 1995). Lisäksi voidaan huolehtia siitä, että yleisesti tunnetut virhetilanteet on vältetty: kokenut ohjelmistotestaaja tietää jo ennen testauksen aloittamista, mitkä ovat perinteiset virheet, joita tällaisessa ohjelmassa ilmenee.

2.2 Miksi testaus ajautuu toissijaiseksi

Syitä, miksei yritys hoida ohjelmistotestaustaan kunnolla, on varmasti yhtä monta kuin on yrityksiä. Seuraavaan on koottu yleisimpiä verukkeita, joiden perusteella testausta ei ole tai se on puutteellista.

Täydelliset ohjelmoijat

Harjoitteluyrityksessäni keskustelin esimieheni kanssa siitä, minkä vuoksi testaus on jäänyt paitsioon. Hän tunnusti, että on pitänyt testausta nolona juttuna ottaa esiin asiakkaiden kanssa, koska sehän tarkoittaisi, että meidän ohjelmoijat tekevät hutiloitua työtä. Usein johdolla on käsitys, että ohjelma tai sen osa koodataan kerran ja se pitää koodata silloin kunnolla. On kuitenkin tosiasia, että ohjelmoijat ovat sokeita virheilleen (Marick 1997). Kun edessä on satoja ja taas satoja riviä koodia, yhden virheen löytäminen itse on kuin etsisi sitä kuuluisaa neulaa heinäkasasta. Kuka tahansa pitkää tekstinpätkää kirjoittanut voi yhtyä tähän: teksti on parempi antaa toisen arvioitavaksi.

Henkilöstöresurssit

Perustettaessa uutta ohjelmointiyritystä on selvää, että budjetti on tiukka ja palkattavan henkilöstön osaaminen on harkittava tarkkaan. Tällöin on tavallista, että palkataan ohjelmoijia ja projektijohtajia, mutta ei testaajia. Ohjelmoijathan testaavat samalla kun koodaavat. Tällöin päädytään samaan tilanteeseen kuin edellä: ohjelmoivat eivät löydä itse tekemiään virheitä. Toisaalta heillä ei välttämättä ole kokonaiskuvaa koko ohjelmasta: he ohjelmoivat moduulin kerrallaan toisistaan irrallisina kokonaisuuksina. Mikäli resurssit eivät aluksi mahdollista yhden ohjelmistotestaajan palkkaamista, on mahdollisuus, että testaus ulkoistetaan tarvittavilta osilta. On kiistatonta, että valmiissa ohjelmassa ilmi tullut virhe tulee huomattavasti kalliimmaksi, kuin jos virhe olisi jo kehitysvaiheessa löytynyt ja korjattu (Chou 2010).

Aikataulu ja budjetti

Chaos-raportin mukaan 52,7 % eli yli puolet projekteista ylittää budjettinsa ennen valmistumista, ja toisaalta 1/3 vastaajien projekteista ylitti aikatauluarvion 200-300-prosenttisesti. Ei ole ihme, että projektit elävät jatkuvassa paineessa pyrkiä tekemään nopeaa ja kustannustehokasta työtä. Koska on yleistä ensin tehdä ja sitten testata, on testaus usein helppo tiputtaa pois sovittujen aikataulurajojen ollessa liian lähellä. Kuitenkin testaamaton ohjelma ei mitä suuremmalla todennäköisyydellä toimi odotetulla tavalla, vaikka se valmistuisi aikataulujen puitteissa. Siksi on tehtävä päätös: tehdäkö ennemmin yliaikainen projekti, jossa testaus on olennaisten prosessien osalta tehty vai laitetaanko liikkeelle puolivalmis tuote, joka mitä todennäköisemmin palautuu takaisin toimimattomana. Useimmiten aikatauluongelmat johtuvat liian optimistisesta aikataulutuksesta ja mahdollisesti kokemattomasta aikataulujen laatijasta: hän ei välttämättä osaa arvioida, millainen aika minkäkin moduulin ohjelmoimiseen on varattava (Lehtimäki 2006).

Määrittely ja dokumentaatio

Testaus elää määritysten mukaan: niiden perusteella tiedetään, miten ohjelman tulisi toimia. Jos määrittelyt muuttuvat jatkuvasti tai ne eivät alunperinkään ole selvillä, on testaajan käytännössä mahdotonta tietää, mitä hänen tulisi saada ohjelmalla tehdyksi.

Kunnollinen dokumentaatio auttaa.

Liian usein yrityksillä tieto on henkilöiden sähköposteissa ja pään sisällä. Tällöin pelkkään tiedon hakemiseen menee testaajalla aikaa hänen selvitellessään, millaisia ominaisuuksia ohjelmaan on sovittu tehtäväksi. Hyvien dokumentointikäytäntöjen luonti auttaa myös yrityksen uusia työntekijöitä: he voivat selvittää dokumenteista, millaisia testaussuunnitelmia yrityksessä on ennen laadittu ja millaiseen dokumentaation tulisi pyrkiä. (Kasurinen 2010)

2.3 Testauksen edut yritykselle

Muutokset ovat hitaasti tapahtuvia prosesseja. Alkuvuonna 2010 Toyota joutui kutsumaan korjattavaksi miljoonia autoja kaasupoljinvian vuoksi (Kuningaskuluttaja, YLE 2010). IT-konsulttipalvelu Sogetin teki tutkimuksen, jossa kartoitettiin, onko tällä Toyotan kyseenalaisella esimerkillä ollut työntekijöiden tiimeissä vaikutusta suhteessa testaukseen ja laadunvarmistukseen (Sogeti UK 2010). Yllättävästi 86 % vastaajista totesi, ettei asialla ollut vaikutusta. Tämä johtunee siitä, ettei tilannetta nähty realistisena ohjelmistotaloissa eikä ymmärretä, että laadukkaalla testauksella Toyota olisi välttänyt ongelmansa ennen viallisten mallien esille tuontia.

Määrätietoisella ja pitkäjänteisellä testauksella saadaan aikaan positiivisia muutoksia: mitä useammin sama virhe löytyy saman ohjelmoijan koodista, sitä varmemmin hän oppii lopulta välttämään tämän virheen ja näin saadaan yksi korjauskierros pois prosessista. Jokaisella ohjelmoijalla on tapansa koodata ja tästä tavasta, tuotti se sitten millaista koodia hyvänsä, on vaikea poiketa muuten kuin testaajan pitkäjänteisellä toiminnalla, jolla osoittaa samat virheet uudestaan ja uudestaan. Tärkeää on jälleen dokumentointi: kun virheraportteja koostetaan huolellisesti, voidaan niistä myöhemmin hakea apua, kun mietitään mahdollisia virheitä, joita ohjelmissa on aiemmin ollut.

Samalla kun dokumentointi auttaa, myös testauksen mukana oleminen alusta asti vähentää turhan työn määrää. Olennaista on testata välittömästi, kun uusi ominaisuus on valmis

testattavaksi. Näin virheestä saadaan tieto nopeasti ohjelmoijille ja korjaus voidaan tehdä nopealla tahdilla. Tällöin testaaja on työllistetty pitkin projektia ja hänen työmääränsä on jakaantunut tasaisesti eikä niin, että hän saa kerralla testattavaksi valmiin ohjelman. Ohjelmat ovat nykyään erittäin monimutkaisia kokonaisuuksia, koska ne sisältävät paljon toiminnallisuutta. Kun testaaja pakotetaan testaamaan nopealla aikataululla ”kaikki”, riski vakavienkin virheiden löytämättä jäämisestä kasvaa.

Testaaminen on tehokas keino riskienhallintaan. Kun projektin alussa on määritelty riskit, voidaan testauksella pyrkiä siihen, että nämä riskit eivät toteudu. Riski voi liittyä tietoturvallisuuteen, käytettävyyteen tai asiakkaan tyytymättömyyteen. Kun riskit on tiedostettu ja kirjattu projektin alkaessa, testaaja tietää painottaa testejään ottaen huomioon myös riskit. (Koch 2010)

Ennen kaikkea testauksen etu on asiakkaan tyytyväisyys! Mikään ei turhauta enempää kuin puolivalmis ohjelmisto, jota asiakas ei voi ottaa käyttöönsä. Aikataulun ylityksetkään eivät haittaa yhtätestauksen priorisointi paljon kuin toimimaton ohjelma, joka palautuu takaisin tekijöille korjattavaksi. Vaikka asiakas testaisi ohjelman itse, kyseessä on useimmiten varmistava testaus, jossa asiakas varmistaa tuotteen toimivan halutulla tavalla eikä siis odota saatikka tarkoituksella etsikään virheitä (Hower n.d). Siksi on tärkeää, että tuottaja huolehtii negatiivisesta testauksesta eli tarkoituksella etsii virheitä, ennen kuin asiakas ne löytää.

2.4 Milloin testaus on tarpeetonta

Edellä on osoitettu, ettei testaus ole tarpeetonta. Miksi siis otsikko, jossa testauksen voidaankin olettaa olevan tarpeeton? Kyse ei ole siitä, että testaus olisi täysin tarpeetonta, ilman testausta otetaan suuri riski tuotteen toimimattomuudesta ja kustannusten noususta. On kuitenkin tilanteita, jolloin testausta ei vielä ehkä ole tarpeellista aloittaa.

Lähdetään siitä, että kaikkea ei voi testata (Hower n.d). Kuitenkin siihen voidaan halutessa käyttää määrättömästi aikaa ja resursseja. Yrityksellä on harvoin mahdollisuutta tarjota

tällaista tilannetta testaajalleen. Siksi on priorisoitava, mitä missäkin projektissa on testattava ensi sijaisesti. Priorisointiin vaikuttaa ohjelmiston käyttötarkoitus ja mitkä asiat koetaan erityisen tärkeiksi. Esimerkiksi kuormitusta mittaava testaus on tarpeellista, kun puhutaan tuhansista samanaikaisista käyttäjistä, mutta ei silloin, kun ohjelmaa käyttää kerrallaan muutama henkilö. Priorisointi tapahtuu yhdessä asiakkaan esittämien vaatimusten ja yrityksen laatimien määrittelyjen perusteella. Testaustyypeistä enemmän luvussa 3.

Ohjelmoijan on turha jokaisen ohjelmoidun rivin jälkeen lähettää ohjelmaa testattavaksi testaajalle. Tärkeämpää olisi tunnistaa ja eriyttää yhtenäisiä kokonaisuuksia ohjelmasta ja lähettää ne testattavaksi. Tästä on ensinnäkin apua ohjelmoijalle, joka voi keskittyä yhteen kokonaisuuteen. Lisäksi virheitä korjatessa on epätodennäköisempää, että yhden moduulin virheen korjaaminen hajottaisi toisen moduulin toimivuutta. Projektin kannalta moduuleihin jakaminen tuottaa virstanpylväitä, joiden avulla ohjelman valmistumista on helpompi seurata. Testaajalle kokonaisuudet takaavat järkevää jatkuvaa työskentelyä ja odottaminen jää vähemmälle.

Kuten luvussa 2.2 sivuttiin, määrittelyiden ja dokumentoinnin puute näkyy vaillinaisena testauksena. Pahimmillaan dokumentit ovat vanhentuneita ja näiden pohjalta rakennettu testaus on täysin turhaa työtä (Kasurinen 2010). Voidaankin sanoa, että testaus on tarpeetonta, kun määrittelyt ja dokumentit eivät ole ajan tasalla: testaaja ei saa rakennettua tarvittavia testauksia kunnolla, kun hänellä ei ole tietoa, mitä olisi tarkoitus testata ja kuinka ohjelman tulisi toimia.

3 Erilaiset testaustyypit

Konkreettisesti testaus on erilaisten prosessien ajamista ohjelmalle. Kuitenkin näitä prosesseja eli tapoja testata on lukuisia. On tapauskohtaista, millaista testausta milloinkin on tarpeellista ja mielekästä aikataulun sekä budjetin puitteissa käyttää. Koska testattavaa on paljon, on myös testausprosesseja huima määrä. Niiden erittely sen mukaan, tarvitaanko kyseistä testausta juuri tässä projektissa voi olla vaikeaa.

Koska testausprosesseja on lukuisia ja useat niistä lähes joka kerta tarpeellisia, on testaajan tietotaitojen oltava myös laajat. Todellisuus on kuitenkin useimmiten toista: testaaja hallitsee tietyn tyyppiset testaukset ja pyrkii näin ollen ajamaan ja toteuttamaan niitä. Siksi onkin hyvä vaihtoehto ulkoistaa tarpeelliset testauksen ulkopuoliselle testausyritykselle mikäli ei omasta talosta osaamista löydy.

Jotta saataisiin yleiskuva, mitä kaikkea voidaan testata, käydään tässä luvussa läpi yleisimpiä testaustyyppejä sekä pyritään antamaan kattava kuva, millaista testausta käyttää missäkin vaiheessa. Jokaisessa kohdassa tuodaan verkkopankkiesimerkin kautta konkreettinen ehdotus, kuinka kyseisessä projektissa testaus toteutettaisiin. Esimerkit on sisennetty.

3.1 Toiminnallinen testaus

Toiminnallisuustestaus (functionality testing, feature testing)
Toiminnallisuustestauksella todennetaan ohjelman määrittelyjen toteutumista valmiissa ohjelmassa. Testaustyyppi toteutetaan, kun ohjelma on asennettu paikoilleen ja on valmiina käyttöön. Toiminnallisuustestaus on loppusilaus ennen käyttöönottoa. Toiminnallisuustestaus toteutetaan antamalla ohjelmalle syötteitä ja tarkistamalla, vastaako syötteen tuottama tuloste sitä, mitä ohjelman odotetaan tulostavan.

> Verkkopankkijärjestelmässä toiminnallisuustestausta voidaan toteuttaa esimerkiksi sisäänkirjautumisessa: on odotettavaa, että kun käyttäjä on syöttänyt tunnuksen ja salasanan, ohjelma näyttää käyttäjän oman sivun. Mikäli näin ei tapahdukaan,

toiminnallisuustestauksella on löydetty ongelma, joka on selvitettävä ennen käyttöönottoa.

Yhtäaikaisuustestaus (concurrency testing)

Yhtäaikaisuustestauksessa käyttäjät käyttävät ohjelmaa täsmälleen samalla tavalla täsmälleen samaan aikaan. Tällainen on vaikea toteuttaa manuaalisesti ja niinpä yhtäaikaisuustestaus on sellainen testaustyyppi, jonka toteuttamiseen kannattaa käyttää testausohjelmaa. Yhtäaikaisuustestauksen tarkoitus on paljastaa, aiheuttaako käyttäjien samanaikainen pyyntö ohjelmassa tarpeettoman pitkiä vastausaikoja.

Pankkijärjestelmässä yhtäaikaisuustestauksella voidaan varmistaa mitä tapahtuu kun useampi käyttäjä on kirjautumassa sisään samanaikaisesti.

Asennustestaus (installation testing)

Mikäli ohjelma vaatii asennusta, on syytä varmistaa, että käyttäjä pystyy asentamaan ohjelman ilman virhetilanteita. Eritoten kun kyse on kuluttajan omalle koneelleen asentamasta ohjelmasta, on tärkeää testata, että asentaminen onnistuu. Selainpohjaisissa ohjelmissa asennustestaus ei ole korkealla prioriteetilla pohdittaessa toteutettavia testauksia, koska ohjelman kerta-asennus palvelimelle riittää.

Aloitustestaus (build verification testing, smoke testing)

Aloitustestaus on perustavanlaatuinen testaus, joka suoritetaan, kun ohjelma alkaa ensimmäistä kertaa osoittaa valmiin ohjelman merkkejä: perusprosessit on ohjelmoitu ja ohjelma on valmis seuraavalle kehitysasteelle. Aloitustestaus toimii myös ensimmäisenä testaustyyppinä, joka ohjelmalle toteutetaan ennen muita testauksia. Tämän testaustyypin tarkoitus on paljastaa räikeät toimimattomuudet ohjelmassa. Testaus kattaa perusprosessit testattuna oletetulla toimintatavalla. Näin ollen testaaja käyttää ohjelmaa kuten on oletettukin eikä esimerkiksi tarkoituksella yritä antaa vääriä syötteitä saadakseen ohjelmaa hajoamaan.

Verkkopankkijärjestelmässä testataan, että jokainen avoin kenttä, johon on tarkoitus kirjoittaa tekstiä, myös ottaa tuon syötteen vastaan.

Konfiguraatiotestaus (configuration testing, hardware testing, portable testing)
Konfiguraatiotestauksessa pyritään varmentamaan, miten ohjelma reagoi erilaisiin laitteistoihin. Tyypilliset ohjelmat, joissa tämä testaustyyppi näyttelee suurta roolia, ovat pelit. Pelit testataan toisaalta sekä eri käyttöjärjestelmissä että konkreettisesti tietokoneen komponentteja muuttamalla (tyypillinen peleihin vaikuttava osa on näytönohjain).

3.2 Ei-toiminnallinen testaus

Kuormitustestaus (load testing)
Poiketen yhtäaikaisuustestauksesta, kuormitustestauksessa ei käyttäjien tarvitse tehdä samaa toimintoa täsmälleen samaan aikaan. Kuormitustestauksessa suuri käyttäjäjoukko toimii ohjelmassa tehden erilaisia toimenpiteitä ja välittäen erilaisia pyyntöjä. Kuormitustestauksen tarkoitus on varmentaa ohjelman riittävän nopea toiminnallisuus, kun sitä kuormitetaan erilaisilla pyynnöillä samanaikaisesti. Kuormitustestaus on ihanteellista toteuttaa automatisoidusti, sillä sen manuaalinen toteuttaminen on haasteellista edellyttäen suurta käyttäjäjoukkoa.

Rasitustestaus (stress testing)
Rasitustestauksessa viedään kuormitustestaus astetta pidemmälle. Siinä ohjelma altistetaan tarkoituksella oletettua suuremmalle käyttäjämäärälle ja pyritään viemään ohjelman suoritus yli rajojen. Tämän testaustyypin tarkoitus on todentaa, ettei ohjelma kaadu välittömästi, kun hallitsematon joukko pyyntöjä syötetään sille yhtäkkiä. Eritoten ohjelmille, joita käyttää jatkuvasti useampi käyttäjä, on tämän kaltainen testaus erittäin tarpeellinen, jotta voidaan huolehtia, ettei ohjelma kaadu äkillisestä käyttäjäryntäyksestä.

Tietoturvatestaus (security testing)
Tietoturvatestaus nousee ensisijaisesti testaustyypiksi ohjelmissa, jotka sisältävät salassa pidettävää tietoa. Tietoturvatestaus sisältää monta puolta ja on ehdottomasti sellainen testaus, jonka ulkoistamista kannattaa vakavasti harkita. Tietoturvaa ei voi ohjelmaan lisätä jälkeen päin ja niinpä siihen panostaminen on tehtävä ennen kuin ohjelmaa käyttää ensimmäinenkään oikea käyttäjä.

Verkkopankkijärjestelmässä tietoturva nousee suureen rooliin, kun kyse on käyttäjien tilitiedoista.

Käytettävyystestaus (usability testing)

Käytettävyystestauksessa huomio kiinnitetään ohjelman loogiseen käytettävyyteen: löytääkö käyttäjä tarvitsemansa tiedot helposti, onko ohjelman ulkoasu miellyttävä ja helppotajuinen, onko käyttäjää ohjeistettu riittävällä tavalla ja tietääkö hän jatkuvasti mitä on ohjelmassa tekemässä. Käytettävyys on toki henkilökohtainen asia: toisen helppokäyttöisyys on toiselle äärettömän vaikeaa. Käytettävyydestä on kuitenkin tehty erilaisia tutkimuksia ja on yleisiä käytäntöjä, joita on hyvä soveltaa myös omaan ohjelmaan. Käytettävyystestaus voidaan toteuttaa siihen erikoistuneiden testaajien toimesta tai he voivat suorittaa testauksen valvotussa tilanteessa loppukäyttäjien keskuudessa.

Verkkopankista uloskirjautuminen tapahtuu selainikkunan oikeasta yläkulmasta painamalla painiketta, jossa lukee ”Kirjaudu ulos”. Tämä on hyvä käytäntöä, sillä useat ohjelmat toimivat samalla tavalla ja tilanne on käyttäjälle tuttu.

4 Dokumentointi

Dokumentoinnilla, tai sen puutteella, on ratkaiseva merkitys kun testausta ryhdytään suunnittelemaan. Dokumentointi, niin testauksen kuin prosessin muiden vaiheiden, vie tietoa eteenpäin eikä tieto jää sähköposteihin ja palavereihin. Huolellinen dokumentointi paitsi tarjoaa tietoa tulevia projekteja varten, myös helpottaa uusien työntekijöiden perehdyttämistä talon tavoille. (Kasurinen 2010)

Kunnollisesta dokumentaatiosta käy ilmi, kuinka mikäkin vaihe on projektissa toteutettu ja mitä eri vaiheista on sovittu. Eritoten määrittelyt ja aikataulut ovat tärkeitä. Kummassakaan ei mitä todennäköisimmin pysytä, mutta mikäli asiakkaan taholta määrittelyt muuttuvat useaan otteeseen, voidaan lopulta vedota alkuperäiseen määrittelyyn ja joko karsia ominaisuuksia tai pyytää lisähintaa.

Testauksen dokumentointi on mukana projektin alusta alkaen. Samalla kun laaditaan projektisuunnitelma, laaditaan testaussuunnitelma (Ruuska 2007). Jo ennen testauksen aloittamista kirjoitetaan testitapaukset. Lopuksi on tärkeää dokumentoida suoritetut testit ja niiden tulokset. Testausdokumentit voivat olla hyvinkin eri näköisiä eri yrityksissä, mutta aloittaessa tyhjästä voidaan apuna käyttää IEEE 829 standardia (IEEE-SA 2004).

IEEE (Institute of Electrical and Electronics Engineers) on kansainvälinen tekniikan alan järjestö. Yksi sen tavoitteista on luoda standardeja, yhtenäisiä ohjeistuksia, joita on helppo ymmärtää ja soveltaa eri yrityksissä. Standardi 829 (standardi testausdokumentaatiosta) tarjoaa apua testausdokumenttien laadintaan. Standardi antaa ehdotuksia, mutta sen käyttäminen ei edellytä sen käyttämistä sellaisenaan, vaan sitä voi muokata haluamaansa suuntaan. (IEEE-SA 2004)

Kuten missä tahansa dokumentaatiossa, myös testausdokumentaatiossa on tärkeää ajantasainen tieto. Määrittelyjen tai aikataulujen muuttuessa on tärkeää, että nämä muutokset tallennetaan myös dokumentteihin. Mikäli näin ei toimita, on riskinä vääränlaisen testauksen suorittaminen väärään aikaan. Kaiken kaikkiaan

väärinymmärrysten välttämiseksi on dokumentaatio pidettävä ajan tasalla. (Kasurinen 2010)

4.1 Testaussuunnitelma

Testaus ja sen suunnitelma laaditaan samaan aikaan projektisuunnitelman kanssa, sillä ne täydentävät toisiaan (Ruuska 2007). Suunnitelma sisältää yleiskatsauksen siitä koska ja kuinka testaaminen tapahtuu, kuka sen tekee ja millä työvälineillä. Myös riskianalyysi sisällytetään suunnitelmaan (Hower n.d). Suunnitelman laatimisen etuina ovat määrittelyjen tarkentaminen sekä ymmärrys projektin työntekijöillä siitä, millaista testausta ollaan suorittamassa. Suunnitelma onkin syytä tehdä sellaisella tasolla, että testauksesta ymmärtämätönkin henkilö tajuaa lukemaansa. Asiakas voi myös vaatia testaussuunnitelman näkemistä, niinpä sisältö on tarkasteltava myös tältä kannalta: mitä tietoja halutaan sisällyttää dokumenttiin asiakkaan nähtäväksi?

Standardi IEEE 829-2008 määrittelee testaussuunnitelman sisältöä. Seuraavassa käydään tämä sisältö kohta kohdalta läpi nojaten esimerkkiin verkkopankkijärjestelmästä. Esimerkit on sisennetty erotuksena leipätekstistä. On syytä huomioida, että tämä on ehdotus ja jokaisen yrityksen on syytä henkilöstön kesken miettiä, millainen testaussuunnitelma palvelisi heidän tarkoituksiaan parhaiten.

Esittely

Käydään läpi testausdokumentin tarkoitus. Dokumentti on syytä myös versioida. Jatkossa jokaisen muutoksen jälkeen versionumeroa kasvatetaan ja esitetään tehdyt muutokset lyhyesti sekä muutoksen tekijä.

> Dokumentti käsittelee verkkopankkijärjestelmä XYZ:n testausta.
> Versio 1.0 Testaussuunnitelman laatiminen, laatija: Virtanen
> Versio 1.1 Täydennetty esittelyä, laatija: Virtanen

Testattava ohjelmisto

Seuraavaksi esitellään testattava ohjelmisto: määritetään sen tarkoitusta sekä keitä ovat

oletetut käyttäjät. Tähän lukuun voidaan sisällyttää määrittelyyn kirjattuja tavoitteita ja asiakkaan tahtoa, kuinka ohjelman tulisi yleisellä tasolla toimia.

> Verkkopankkijärjestelmä XYZ:n tarkoituksena on tarjota pankin asiakkaille selainpohjainen järjestelmä, jonka kautta he voivat maksaa laskuja sekä hallinnoida tilejään.

Ominaisuudet, jotka testataan

Ominaisuuksilla viitataan pääsääntöisesti ohjelman prosesseihin, mutta ominaisuuksia voi olla myös esimerkiksi käytettävyys. Ominaisuudet kirjataan otsikkotasolla niin, että ne ovat ymmärrettävissä. Ominaisuudelle annetaan identifioiva tunnus, jolloin laadittaessa testitapauksia voidaan näihin testattaviin ominaisuuksiin viitata. Ominaisuuksille on syytä myös tehdä priorisointi numeroilla tai sovituin termein. Tämä tarjoaa testitapauksien laatijalle ja itse testaajalle tiedon, mitkä ominaisuudet ovat ensisijaisia ja mitkä toissijaisia. Voi tuntua hankalalta priorisoida ominaisuuksia, sillä ymmärrettävästi kaikki tuntuvat tärkeältä. Priorisointiin voi hakea apua asiakkaalta.

1.1 Käyttäjän kirjautuminen järjestelmään	priority high
1.2 Maksun suorittaminen	priority high
1.3 Sähköpostin lähettäminen pankille	priority major

Ominaisuudet, joita ei testata

Aina tulee olemaan ominaisuuksia, joita ei syystä tai toisesta voida testata. Nämäkin ominaisuudet on syytä kirjata ylös. Ensinnäkin, mikäli aikaa tulisikin, nähtäisiin tästä kohdasta heti ne ominaisuudet, jotka ovat jääneet testaamatta. Toisaalta tämä kohta tarjoaa arvokasta tietoa asiakkaalle: he saavat selkeän tiedon mitä ei tulla testaamaan. Vaikka näistä asioista saatetaan sopia sanallisesti, myöhemmin, varsinkin epäselvissä tapauksissa, kirjattuun tietoon on helpompi vedota.

> 2.1 Graafiset ominaisuudet
>
> 2.2 Pankkikortin teeman valinta

Lähestymistapa

Tähän lukuun kirjataan testaustyypit, joita aiotaan käyttää.

Asteikko

Käytettävät asteikot on syytä sopia etukäteen. Varsinkin ulkomaalaisten asiakkaiden kanssa esimerkiksi pituusmääreet voivat olla erilaiset ja väärinkäsitysten välttämiseksi on syytä sopia etukäteen, mitä määreitä käytetään. Tarpeellisin asteikko on se, jota sovitaan käytettävän ajetuista testeistä. Näin ajettujen testien dokumentin lukija tietää, mitä mikäkin termi tarkoittaa ja mitkä jatkotoimenpiteet sellaisesta virheestä seuraavat.

A = testiajo onnistui

B = prosessi keskeytyi

C = prosessi saatiin ajettua, tulos ei ole oikein

D = prosessia ei saatu käynnistettyä

Keskeyttämiskriteerit

Luvussa 2.4 viitattiin, ettei kaikkea voida testata. On kaikkien etu laatia yhteiset pelisäännöt siitä, mitkä toimivat testauksen keskeyttämisen kriteereinä, toisin sanoen: millainen on testituloksen oltava, jotta testaus lopetetaan. Tämä luku on tarjoaa tietoa pääsääntöisesti itse testaajalle, mutta voi olla myös tapauksia, jolloin testauksen keskeyttäminen koskee myös asiakasta.

> Testaus keskeytetään, mikäli testiajoista saadaan pelkkää D:tä (katso luku Asteikko). Testaus voidaan myös keskeyttää, mikäli tulee tilanne, jolloin ei olla varmoja, kuinka ohjelman tulisi käyttäytyä tässä tilanteessa. Ongelman ratkaisemiseksi tarkistetaan määrittelydokumentit, mutta mikäli nämä eivät tarjoa tietoa, otetaan yhteys asiakkaaseen.

Testauksen tulosteet

Kirjataan dokumentit, joita testausprosessissa muodostuu. Tähän kuuluvat yleisesti testaussuunnitelma, testitapaukset sekä testiajojen raportti. Myös muut testaukseen liittyvät dokumentit kirjataan.

A Yleinen testaussuunnitelma

B Käytettävyyden testaussuunnitelman

C Käytettävyyden testitapaukset

D Ajettujen testien raportti

Testausympäristö

Kuvataan ohjelmat ja asennukset, joilla testaus suoritetaan. Selainpohjaisessa järjestelmässä kirjataan selaimet.

Järjestelmä testataan Internet Explorer 6 sekä 8 selaimilla. Lisäksi testataan Firefox 3.6 selaimella.

Henkilöstö- ja koulutustarve

Nimetään käytettävä henkilöstö. Mikäli testaus vaatii erityistä kouluttautumista, varataan sille vaadittava aika.

Testausprosessin toteuttaa N.N sekä M.N.

M.N tarvitsee koulutusta testiympäristön rakentamiseen. Koulutukselle varataan aikaa 7,5 tuntia.

Vastuut

Kirjataan tarkemmin, millaiset vastuut testauksen henkilöstöllä on. Vastuut on syytä jakaa henkilöstön tietojen sekä taitojen mukaan tehokkaasti. Pelkkä kirjaaminen ei luonnollisesti riitä, vaan nämä vastuut on käytävä yhdessä henkilöstön kanssa läpi.

N.N laatii testitapaukset käyttäen laadittuja määritelmiä. Epäselvissä tapauksissa otetaan yhteys projektipäällikkö P.P:hen. M.N pystyttää testausympäristön ja suorittaa testien ajon. Ajettujen testien raportin M.N toimittaa N.N:lle, joka koostaa niistä yhteenvedon P.P:lle.

Aikataulu

Aikataulu laaditaan mahdollisimman tarkaksi huomioiden kuitenkin riskitekijät.

Testitapaukset laadittuna viikolla 25.

Testiympäristö pystyssä viikolla 25.

Testiajot aloitetaan viikolla 26.

Testaus lopetetaan viikolla 30.

Riskianalyysi

Riskianalyysilla pyritään valmistautumaan testausprojektin aikana esiin tulleisiin ongelmiin. Riskien esiin tuominen edistää projektia: kun riskeistä uskalletaan puhua ääneen, niiltä voidaan myös paremmin välttyä. Riskeistä enemmän luvussa viisi.

Riskien kirjattaessa kirjataan itse riski, annetaan todennäköisyys sille esimerkiksi numeroilla sekä tarjotaan vaihtoehto toisaalta ehkäistä riski ja toisaalta tarjota ratkaisu, kun riski toteutuu.

Henkilön sairastuminen

todennäköisyys 50%

ehkäisy: ei voida ehkäistä

Ohjelman toimiminen ohjelmistotestauksen vertaamalla tulosta odotettuun tulokseen.ratkaisu: huolehditaan niin, että kaikki ovat perillä toistensa tekemisistä (avoin toiminta) sekä pidetään dokumentit ajan tasalla

4.2 Testitapaukset

Testitapaukset ovat käytännön tasolle meneviä kuvauksia ohjelman prosesseista. Testitapauksissa kuvataan askel askeleelta kuinka ohjelma missäkin vaiheessa toimii. Lisäksi kirjataan, mitä tuloksia tulisi saada ja mitkä tilanteet sen sijaan ovat virheellisiä (Lehtimäki 2006). Testitapauksien laatiminen melko aikaisessa vaiheessa projektin alussa selventää ohjelman toiminnallisuutta ja mahdolliset avoimet kohdat voidaan selvittää ennen varsinaisen ohjelmoinnin alkamista (Hower n.d).

Testitapauksien laadintaan ei ole olemassa yhtenäistä standardia, niinpä alla tarjotaan yksi esimerkki kuinka testitapauksen voi dokumentoida. Tärkeää on, että testitapauksen dokumentti tarjoaa testaajan tarvitseman tiedon testauksen itsenäiseen toteuttamiseen mahdollisesti niin, ettei muihin dokumentteihin ole tutustuttu.

Testitapauksien kirjaamisen etu on se, että selkeytetään prosesseja ja puutteelliset kohdat tulevat aikaisessa vaiheessa esille. Itse testaus on helppo aloittaa dokumentin pohjalta

välittömästi kun testattavaa on. Mikäli projektiin tulee ulkopuolinen työntekijä, on hänen testitapauksien avulla helppo suorittaa testausta, vaikkei hän tietäisi ohjelman rakenteesta mitään (Lehtimäki 2006).

Testitapauksia laadittaessa on tärkeää hahmottaa suuria kokonaisuuksia. Ohjelmien kompleksisuuden vuoksi testausta tulisi optimoida niin, että yhden testauksen ajo vastaisi useampaan kysymykseen. Tällöin on tarkasteltava vaadittuja ominaisuuksia ja mietittävä, mitkä niistä käyvät ilmi missäkin tilanteessa. Tällä tavalla saadaan selville voidaanko useampi määrittely todeta samassa tilanteessa.

Testitapauksesta käy ilmi ensinnäkin yksilöivä tunnus. Tunnus voi olla otettuna testaussuunnitelmasta tai se voidaan luoda yrityksen oman käytännön perusteella. Sen jälkeen kirjataan testitapauksen nimi. Jälleen tämä nimi voi olla suoraan otettu testaussuunnitelmasta. Ensimmäisenä itse testitapauksesta kirjataan sen tarkoitus: miksi testitapaus suoritetaan. Esityökohdassa käydään läpi vaadittavia toimenpiteitä, jotka on suoritettava ennen testitapausta. Sen jälkeen käydään kohta kohdalta läpi testitapauksen vaiheet mahdollisimman tarkasti. Jälleen korostan, että testitapaus on kirjattava kuin tekisi sellaiselle henkilölle, joka ei ole nähnyt ohjelman dokumentaatiota eikä ole käyttänyt ohjelmaa aikaisemmin, ikään kuin käyttöohjeet. Jotta voidaan todentaa, onko testitapaus onnistunut, täytyy tietää millainen on odotettu lopputulos kun vaiheet on käyty läpi. Lopuksi kirjataan kaikki mahdolliset (odotetut) virhetilanteet, jolloin on toimittava testaussuunnitelman luvussa ***Keskeyttämiskriteerit*** määritetyllä tavalla.

1.1 testitapaus: Käyttäjän kirjautuminen järjestelmään

Tarkoitus	Testataan, että käyttäjä pystyy kirjautumaan järjestelmään onnistuneesti.
Esityö	Käyttäjätiedot viety tietokantaan, testaajalle annettu käyttäjätunnus
Vaiheet	1. Kirjoita kohtaan ”Tunnus” sinulle annettu tunnus 2. Kirjoita kohtaan ”Salasana” sinulle annettu salasana 3. Napsauta ”Kirjaudu”-painiketta
Odotettu tulos	Sivusto ohjautuu käyttäjän omalle etusivulle.

Virheitä jotka johtavat jatkotoimenpiteisiin	1. Tunnus tai salasana ei kelpaa 2. Ohjelma vie muualle kuin käyttäjän omalle etusivulle 3. Ohjelma vie jonkun muun käyttäjän omalle sivulle 4. Ohjelma antaa jonkun muun virhetekstin

4.3 Ajettujen testien raportointi

Kun testaus on suoritettu, on vielä kirjattava, mihin tulokseen päästiin. Sen lisäksi, että on tärkeää ilmoittaa onnistuneet testaukset, vielä olennaisempaa on ilmoittaa epäonnistuneet testitapaukset. Testauksen raportointiin on lukuisia eri tapoja ja erottava tekijänä voitaneen pitää vastaanottajaa: projektivastaavalle ei ole olennaista tietää virheen teknisiä tietoja vaan ainoastaan virheen laajuus ja sen vaikutus aikatauluun. Ohjelmoijan sen sijaan on saatava seikkaperäisempää tietoa voidakseen jäljittää virheen (Hower n.d).

Ohjelmoijalle annettavassa tiedossa on kerrottava tarkoin, missä virhe ilmeni ja kuinka tähän tilanteeseen tultiin. Tärkeää on kertoa mahdollisimman tarkasti, mitä testaaja teki ennen virhetilannetta ja antoiko ohjelma virheestä virheilmoitusta. Mikäli kyseessä on selainpohjainen järjestelmä, kirjataan ylös osoiterivillä näkyvä sivun osoite sekä käytetty selain. Toisinaan virhe voi olla selaimessa. Lisäksi, mikäli käytössä on erilaisia käyttäjärooleja, kerrotaan rooli, jota käyttäessä virhe ilmeni. Lopuksi voi kuvata virheen laatua: estyikö ohjelman käyttö kokonaan vai pystyikö käyttöä jatkamaan? Joskus virhetilanteista voi olla tarpeellista ottaa kuvakaappaus ja liittää se mukaan raporttiin, näin helpotetaan entisestään virheen löytämistä. (Hower n.d)

Henkilölle, jolle ohjelmakoodi ei ole tuttua, ei ole tarpeellista kertoa näin seikkaperäisesti virheen laatua. Olennaisempaa on kuvata virhe, kuinka kriittinen se on ja millä prioriteetilla sitä lähdetään työstämään. Tämän jälkeen on suotavaa vielä pyytää aikatauluarvio ohjelmoijalta virheen korjaamiseksi. Näin pystytään asiakkaalle viestimään paremmin projektin etenemisestä. Tärkeää on priorisoida virheet myös niin, ettei niiden korjaaminen venytä aikataulua liiaksi. On siis tarkasteltava, estävätkö virheet käyttöä vai eivät. Aikataulun lähetessä loppuaan voidaan näin jättää epäolennaisemmat virheet korjattaviksi myöhemmässä vaiheessa.

5 Testaus mukaan projektiin

Kun perustiedot on hankittu, voidaan ryhtyä tuomaan testausta mukaan prosesseihin. Testaus ei ole vain loppusilaus projektissa: se on mukana aina ensimmäisistä määrittelyistä asiakkaalle luovuttamiseen asti. Testaus ei myös ole jotakin, jonka voi valmiista prosessimallista kaapata mukaan ja luottaa, että pelkkä lisääminen prosessiin takaa onnistuneen testauksen ja projektin.

Onnistuneen testauksen avaimet ovat kunnollinen määrittely, realistiset aikataulut, riittävä testaus, suunnitelman noudattaminen sekä kommunikaatio (Hower n.d). Projektipäällikön tehtävä on hallita nämä erilliset osat ja sulauttaa ne yhteen. Vaikka tässä opinnäytetyössä tarkastellaan ohjelmistotestausta ja sen roolia, ei ole ollenkaan tarpeetonta muistuttaa, että paraskin testaaja on pulassa, kun projekti ei ole kontrollissa muilta osin. Riskienhallinta onkin olennainen osa projektin toteuttamista ja on projektipäällikön vastuulla huolehtia, että riskit on otettu huomioon ja niiden estämiseksi on tehty tarvittavat toimenpiteet (Lehtimäki 2006). Myös testaus itse vaatii riskien hallintaa. Testauksen riskejä käsitelläänkin luvun lopuksi.

5.1 Testaus prosessimalleissa

Projektipäälliköllä on iso vastuu projektin läpiviennistä: hän vastaa ylemmille tahoille projektin suunnittelusta, etenemisestä ja päättämisestä (Ruuska 2007). Lähdetään oletuksesta, että projektien suunnittelu ja läpivienti on lukijalle jo tuttua. Nyt tuohon kaikkeen suunnitteluun on lisättävä mukaan ohjelmiston testaaminen.

Projekteista laaditaan projektisuunnitelma. Jokaisella yrityksellä on tuohon suunnitelmaan oma sisäinen mallinsa, mutta pääsääntöisesti projektisuunnitelman voidaan olettaa sisältävän seuraavia asioita: elinkaarimalli, projektiorganisaatio, riskienhallinta sekä projektin vaiheet. Nyt näihin lukuihin tulisi osata sisällyttää mukaan testaus. Riskien hallintaan olikin luvun johdannossa jo viitattu; testaus sisältää, aivan kuten mikä tahansa muukin projektin vaihe, lukuisia riskejä, joihin on osattava varautua ennen kuin projektia

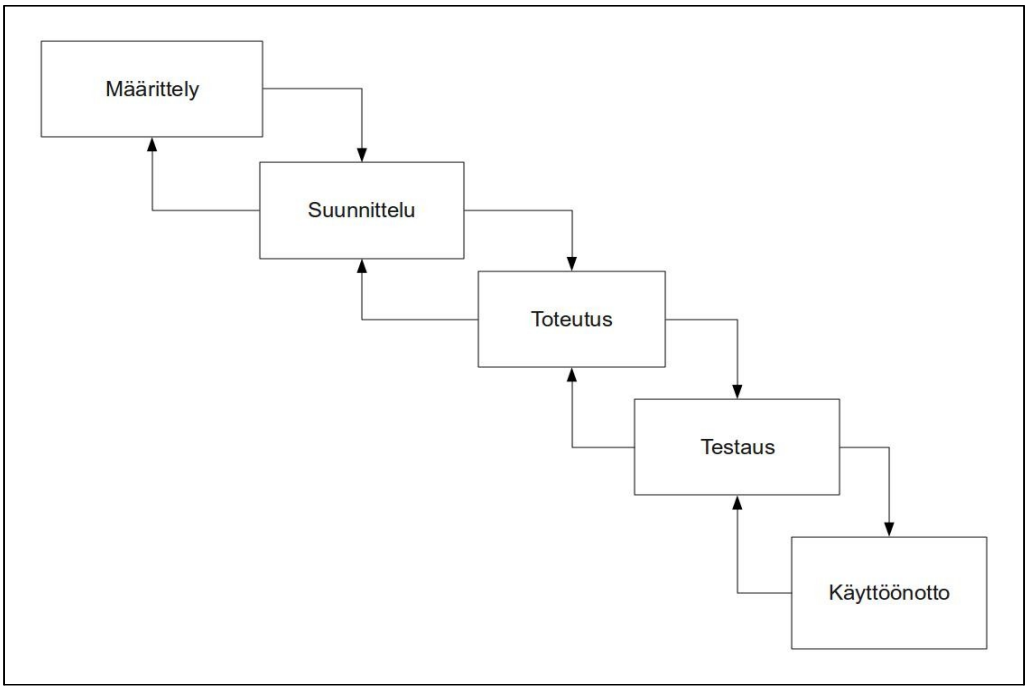
käynnistetään. Riskien tiedostaminen auttaa esimerkiksi aikataulun tekemisessä. Projektiorganisaatiolla viitataan projektissa mukana olevaan niin tilaajan kuin tuottajan henkilöstöön. Testaus saattaa tuoda mukanaan uusia henkilöitä projektiin tai lisätä jo käytettävissä olevien henkilöiden työmäärää. On testausprosessimallista kiinni, onko testaus osana jokaista vaihetta vai täysin omana vaiheenaan.

Prosessimallin tarkoitus on havainnollistaa projektin vaiheita. Malleja ja niiden muunnelmia on yhtä monta kuin tekijääkin, mutta olemassa olevat perusmallit tarjoavat muutamia vaihtoehtoja sisällyttää testaus mukaan prosessiin. Prosessimallit eivät takaa hyvää projektia, mutta ne auttavat jäsentämään projektin vaiheita. Seuraavassa esittelen yleisimmät mallit, joita voidaan hyödyntää testauksen integroimisen ensi vaiheessa.

Vesiputousmalli

Perinteisin prosessimalli on nimeltään vesiputousmalli (kuvio 1). Tässä mallissa ohjelmistoprojektin vaiheet kulkevat lineaarisesti vaiheesta toiseen. Vesiputousmallin perinteisessä muodossa testaus on viimeisimpiä vaiheita ennen käyttöönottoa. Mallin ideana on, että edellisiin vaiheisiin ei palata kesken projektin, vaan jokainen vaihe päättyy katselmukseen, jossa vaiheen toteuma tarkistetaan ja vaihe päätetään. Vesiputousmallin etu on sen tunnettuus; malli on tuttu kaikille alaa opiskelleille ja näin asiakkaan kanssa on helppo päästä yhteisymmärrykseen projektin vaiheista. Parhaimmillaan vesiputousmalli tarjoaakin projektimallin, jonka vaiheet ovat ennakoitavissa ja eteneminen on johdonmukaista.

Vesiputousmallissa korostuu huolellinen suunnittelu jokaisessa vaiheessa; koska vaihe on tarkoitus tehdä kerran, on se tehtävä kerralla kunnolla. Kunnollinen suunnittelu tuottaa oikein toteutettuna säästöjä. Lisäksi vesiputousmallin etuina nähdään vaatimus kunnollisesta dokumentoinnista; jotta seuraava vaihe voidaan toteuttaa huolellisesti, vaaditaan edellisistä vaiheista täydellinen dokumentaatio.



Kuvio 1: Perinteinen vesiputousmalli

Vesiputousmalli on kehitetty 1970-luvulla ja se on edelleen suosittu malli. Niinpä siitä on yleensä kokemusta kaikilla ohjelmistoalan ihmisillä ja toisaalta myös siihen saatavat laadunvarmistus ja muu tuki on helposti saatavilla. Malli on selkeä sekä helposti ymmärrettävissä pienellä vaivalla.

Vesiputousmalli sisältää ohjelmistoprojektin olennaisimmat vaiheet. Määrittelyvaihetta seuraa suunnittelu, suunnittelua toteutus ja toteutus testataan. Testauksen jälkeen suoritetaan käyttöönotto eli valmiin tuotteen luovutus asiakkaalle.

Vesiputousmalli ei kuitenkaan ole aukoton; nimenomaan sen joustamattomuus muutoksien edessä tuottaa sille pitkän miinuksen. Varsinkin ohjelmistotuotannossa on yleistä, että määrittelyt muuttuvat kesken projektin (Hower n.d). Tällaiseen tilanteeseen vesiputousmalli ei tarjoa paluuta määrittelyyn vaan se lähtee oletuksesta, että määrittely tehdään kerralla kunnolla alkuvaiheessa (Murphy n.d). Tilanteissa, joissa ohjelmoidaan täysin uutta ohjelmaa, projektin venyminen on suuri riski, kun asiakkaalle annetaan

mahdollisuus kommentoida ohjelmaa vasta sen ollessa valmis. Tämä on selkeä epävarmuustekijä myös asiakkaalle; pahimmassa tapauksessa valmis ohjelma hylätään kokonaan ja projekti joudutaan aloittamaan alusta.

Kun päätökset ohjelmiston toiminnasta on tehtävä jo ensimmäisessä vaiheessa, sitoutetaan isot rahat toteutumiseen. Riskinä on kustannusten suuri kasvu jonkin prosessin vaiheen epäonnistuessa. Vesiputousmallin yksi perusajatus on aikatauluarvion helppo tekeminen ja pienikin viivästys jossakin mallin vaiheessa vie aikaa seuraavilta vaiheilta, useimmiten viimeisimpien joukossa olevalta testaukselta.

Vesiputousmalli voi siis parhaimmillaan olla selkeä malli, jota seuraamalla projekti etenee johdonmukaisesti vaiheesta toiseen. Ohjelmistoprojektien epävarmuustekijät, kuten asiakkaan toivomukset, eivät kuitenkaan tule huomioiduksi riittävästi ja näin ollen tämä malli voi pahimmillaan moninkertaistaa projektille varatun ajan ja budjetin.

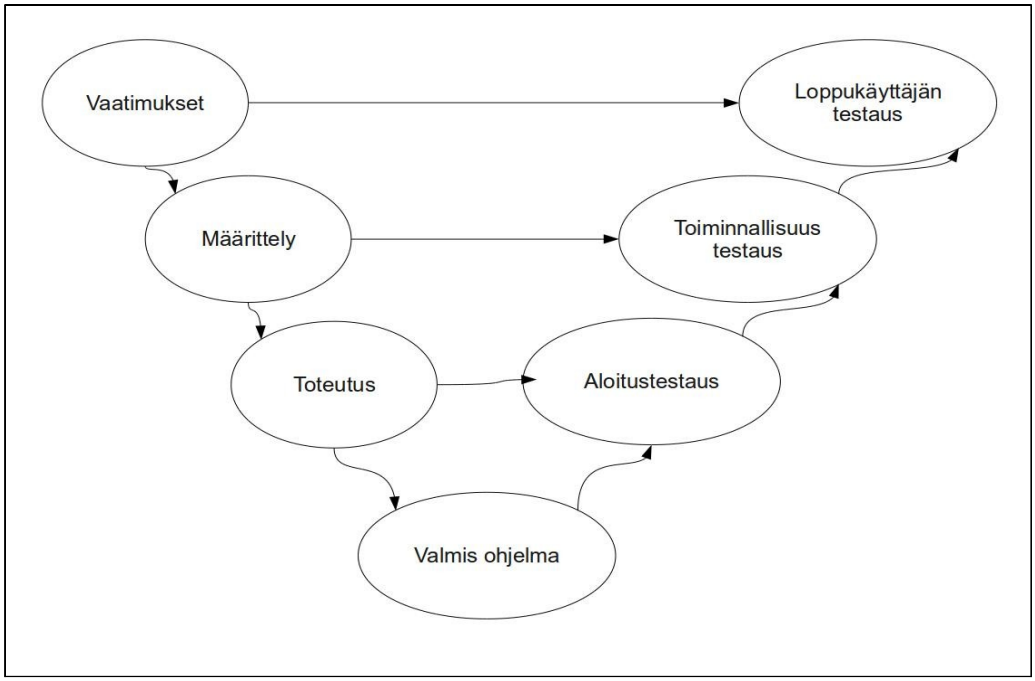
V-malli

Vesiputousmallissa testaus sisältyi omana vaiheenaan viimeisimpien vaiheiden joukkoon. Tämä kuitenkin on juuri sellainen tapa, mistä ohjelmistomaailmassa pyristellään eroon ja niinpä rinnalle on tullut v-malli. Kuten vesiputousmalli, v-mallikin toimii lineaarisesti vaiheissa jokaisen vaiheen sisältäessä testauksen. V-mallissa prosessi on jaettu kahdelle linjalle: suunnittelu ja toteutus sekä testaus.

Koska v-malli on johdannainen vesiputousmallista, molemmat sisältävät samat edut. Dokumentointi nousee v-mallissa tärkeämmäksi, sillä jopa dokumentointi tarkistetaan huolellisesti joka vaiheen jälkeen. V-mallin omia etuja ovat testauksen läsnäolo projektin ensivaiheista lähtien. Lisäksi vesiputousmallista tuttu riski turhan työn tekemisestä kun tuote ei vastaa asiakkaan toiveita, on v-mallissa huomattavasti pienempi jatkuvan testauksen ansiosta. Eri tyyppiset virheet löydetään joka vaiheen jälkeen (ohjelmistovirhe ohjelmoinnin jälkeen, suunnitteluvirhe suunnittelun jälkeen).

V-malli voidaan soveltaa ja muokata pitkälle. Alla on esitetty yksi mahdollisuus (kuvio 2).

Vasemmalla sijaitsevat vaiheet ovat tuttuja vaiheita ohjelmistoprojektista. Jokaisen vaiheen jälkeen kyseinen vaihe tarkistetaan. Lisäksi, kun ohjelma on valmis, noustaan vielä kerran oikea puoli alhaalta ylös: testataan ohjelman toimivuus, varmistetaan että määrittelyt pitävät paikkaansa ja jokainen vaatimus on toteutettu ohjelmassa. V-mallissa testaus siis suoritetaan kahteen kertaan, prosessin edetessä sekä lopuksi.



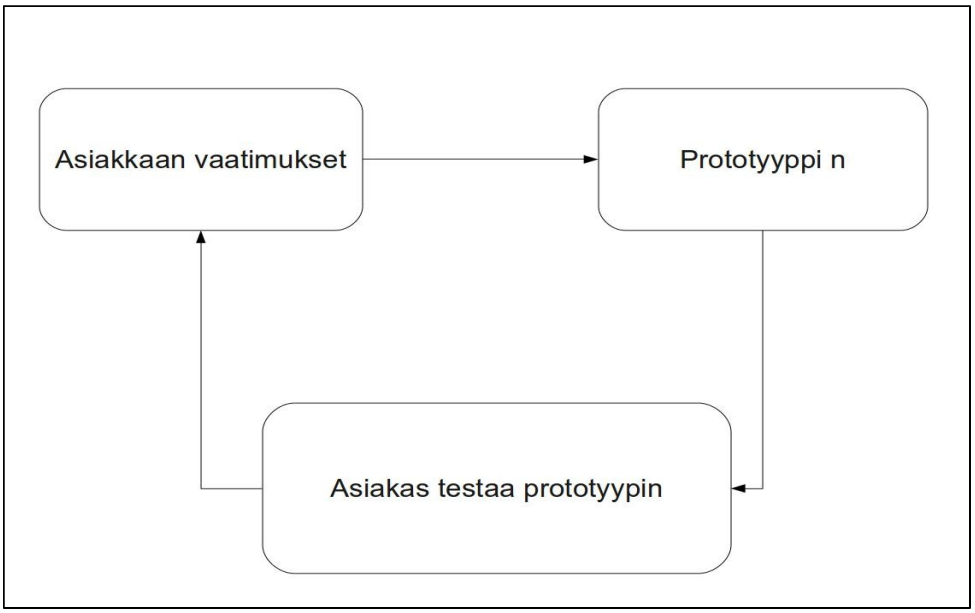
Kuvio 2: V-malli

Kuten vesiputousmalli, myös v-malli kärsii ongelmista. Vesiputousmallista tutut ongelmat asiakkaan kannalta ovat olemassa myös v-mallissa. Asiakas on mukana projektissa ainoastaan projektin aloitus- ja lopetusvaiheissa ja näin asiakkaan vaatimusten kunnollinen määrittely korostuu. Lisäksi v-mallin noudattaminen nostaa kustannuksia vesiputousmallia noudattavaa projektia suuremmiksi dokumentoinnin korostuessa ja testauksen ollessa mukana joka vaiheessa. Toki tällainen vähentää riskiä projektin epäonnistumisesta, mutta pahimmassa tapauksessa projektin budjetti moninkertaistuu verrattuna vesiputousmallia toteuttavaan projektiin.

Evoluutiomalli

Evoluutiomalli (tunnetaan myös *prototyyppimallina*) poikkeaa aiemmin esitellyistä

malleista. Siinä ei oteta niinkään kantaa suunnitteluun tai dokumentointiin. Mallin idea on tuottaa jatkuvasti pieniä osia ohjelmasta valmiiksi, luoda niin sanottu prototyyppi ja testauttaa nämä osat välittömästi (kuvio 3). Saadun palautteen perusteella ohjelmaa kehitetään eteenpäin.



Kuvio 3: Evoluutiomalli

Evoluutiomalli vastaakin siihen ongelmaan, joka vesiputous- ja v-mallilla oli: asiakas ei välttämättä tiedä, mitä hän haluaa. Tämä on erittäin yleistä, kun kyseessä on asiakkaalle uusi ohjelma, josta hänellä ei ole olemassa valmista ratkaisua ennestään. Evoluutiomallin avulla asiakkaan tarpeet saadaan ajoissa esille, kun hän testaamalla prototyyppejä voi kertoa, mihin suuntaan ohjelmaa tulisi kehittää. Prototyyppien käyttö tukee myös varsinaisen ohjelman käyttöönottoa: ohjelma on käyttäjälle tuttu eikä kouluttamiseen mene aikaa.

Evoluutiomalli näyttää yksinkertaiselta ja sitä se onkin. Niin tuottaja kuin asiakas ymmärtää sen nopeasti. Mutta yksinkertaisuus paljastaa myös ongelman: tämä malli ei ota kantaa ohjelmointia edeltävään aikaan. Mallissa ei ole prosessin vaiheita syvemmällä

tasolla. Se ei painota riittävästi kunnollista suunnittelua. Asiakkaan testattua prototyypin on tärkeää, että hänen ilmaisemansa muutosehdotukset kirjataan ylös ja käsitellään riittävän tarkasti. Ongelmana saattaa olla, että asiakas haluaa liian suuria muutoksia. Koska aikataulu- ja kustannusarviot annetaan ennen projektin alkamista, on asiakkaalle painotettava sitä, että alussa annetut vaatimukset kulkevat mukana jatkuvasti ja pienemmissä määrittelyseikoissa kuten käytettävyydessä, prototyyppien kommentointi otetaan huomioon.

Evoluutiomallin riski on siinä, ettei projektia koskaan saada päätökseen. Lukuisten versioiden jälkeen asiakas saattaa edelleen löytää ohjelmasta muutettavaa ja näin ollaan päättymättömässä ketjussa. Onkin tärkeää, että asiakkaan muutosehdotukset käydään läpi ennen uuden prototyypin tekemistä ja arvotetaan, mitkä muutokset ovat realistisia ja toteutettavissa ja mitkä sen sijaan selkeästi ylimitoitettuja annettuun aikatauluun ja budjettiin.

Aikataulu- ja budjettiongelmat ovatkin yksi evoluutiomallin heikkouksista. Koska ensimmäistä prototyyppiä voidaan alkaa työstää hyvinkin köykäisin määrittelyin, on lähes mahdotonta antaa realistista arviota siitä työmäärästä, mitä ohjelman tekeminen tulee vaatimaan. Evoluutiomallissa saatetaan pahimmillaan tehdä runsaasti turhaa työtä asiakkaan hylätessä ja vaatiessa uusia ominaisuuksia. Siksi evoluutiomalli on paras silloin, kun asiakkaalla on selkeä kuva ohjelmasta jo ennen ensimmäistä prototyyppiä, ohjelmisto on yksinkertainen eikä aikataulu ole liian tiukaksi määritetty.

Ketterät menetelmät

Ohjelmistokehityksessä on yleistynyt projektimallina ketterät menetelmät. Poiketen tunnollista dokumentaatiota vaativista prosessimalleista, ketterissä menetelmissä pääpaino on projektiryhmän sisäisellä, parhaimmassa tapauksessa kasvokkain tapahtuvalla kommunikaatiolla. Ketterissä menetelmissä ohjelmisto tuotetaan niin kutsuttujen iteraatioiden kautta. Yhdessä iteraatiossa toteutetaan osa ohjelmistosta alkaen määrittelyistä päättyen testaukseen. Iteraation kesto on pääsääntöisesti yhdestä neljään viikkoa.

Ketterät menetelmät ovat keino hallita muuttuvia vaatimuksia: kun ohjelmaa ei alun perinkään suunnitella heti valmiiksi, tällöin muutoksien tullessa on varaa joustaa ja toteuttaa tämä muutos. Lisäksi ketterissä menetelmissä testataan jatkuvasti. Jokainen iteraatio pitää sisällään testausta. Näin on toimittava, jotta jokaisen iteraation päätteeksi projektiryhmällä olisi jotakin valmista.

Ketterien menetelmien etu on nopeassa reagoinnissa, kun ohjelmassa havaitaan tekninen ongelma tai asiakas ilmaisee haluavansa muutoksen. Lisäksi ketterät menetelmät perustuvat tiiviiseen yhteydenpitoon tekijäryhmän kesken, mikä paitsi kehittää työyhteisön ilmapiiriä positiiviseen suuntaan, takaa myös sen, että kaikilla projektin jäsenillä on ajantasainen tieto tehtävästä työstä.

Ketterät menetelmät voivat kuitenkin olla huono ratkaisu tuottaa ohjelmisto yrityksessä, jossa päätökset tehdään monella tasolla ja vastausten saamiseksi on odotettava pitkä aika suhteutettuna iteraatioon. Kyse voi olla sekä yrityksestä, joka tuottaa ohjelmiston tai yrityksestä, jolle ohjelma tuotetaan. Siirtyminen ketteriin menetelmiin ei tapahdu hetkessä ja yrityksen tulee pohjustaa tuota siirtymää huolella.

5.2 Testaus

Sen lisäksi, että testausta valmistellaan osaksi prosessia, on konkreettista testausta valmisteltava monella tapaa. Valmisteluihin liittyvät myös dokumentit, joita käsiteltiin luvussa neljä.

Kuten aiemmin on todettu, testaus ei ole vain prosessin lopuksi tehtävää kokeilua, vaan se vaatii järjestelmällistä suunnittelua, jossa joka projektin kohdalla arvioidaan erikseen tarve testaukseen. Jokainen projekti ei automaattisesti vaadi suuria testauksia, vaan testauksen tarve on arvioitava projektikohtaisesti suhteuttaen testaus ohjelmoitavan ohjelmiston kokoon. Testaus sisältää myös valmisteluja, jotka on tehtävä huolellisesti, jotta itse testaus olisi tehokasta ja tietoista toimintaa.

Tässä alaluvussa käsitellään myös testauksen riskejä. Kyse ei niinkään ole siitä mitä riskejä itse testauksessa on, vaan mitkä riskit uhkaavat testauksen onnistumista. Testaus ei ole irrallinen prosessi projektissa, vaan se kiinnittyy kiinteästi jokaiseen vaiheeseen ja jokainen vaihe vaikuttaa testaukseen. Siksi testaus elää jokaisen vaiheen ja muuttuvien määrittelyjen mukaan.

5.2.1 Sisäinen vai ulkoinen testaus

Testauksen voi ulkoistaa. Sisäisellä testauksella tarkoitetaan nimenomaan ohjelman tuottavan ohjelmistotalon omaa testaustoimintaa, jonka toteuttaa yrityksen oma henkilöstö. Ulkoisessa testauksessa ohjelma lähetetään testattavaksi eri yritykseen. Nykyään testaukseen erikoistuneita yrityksiä on runsaasti ja niiden tarjoamat testauspalvelut helposti räätälöitäviä ja monipuolisia.

Ulkoisessa testauksessa tuottajasta tulee asiakas ja tällöin on oltava selvää, mitä vaatimuksia esitetään palvelun tuottajalle. Kuten mainittua, testauksella ei ikinä voida tehdä ohjelmaa täydelliseksi, joten testauksen tekijältä ei myöskään voida odottaa kaikkien mahdollisten virheiden löytämistä. On tärkeää määrittää tavoitteet, joita haluaa asettaa ja jotka tuottajan odotetaan täyttävän. Tällaisia odotuksia voivat olla erilaisten testaustyyppien käyttäminen tai tietty määrä testausajoja. Hyvin määritellyt vaatimukset ovat tukena, kun testaukset on ajettu ja tuloksia aletaan tarkastelemaan. Tällöin saadaan selville ovatko asetetut vaatimukset täyttyneet.

Testauksen voi ulkoistaa kokonaan tai osissa. Tilanteessa, jossa ohjelmassa käytetään valmiita kirjastoja, mutta luodaan myös itse uutta, voi olla realistista testauttaa nämä uudet ominaisuudet koko ohjelman sijaan. Ulkoinen testaus pystyy tarjoamaan runsaasti etuja verrattuna testauksen sisäiseen toteuttamiseen: ulkoisessa testauksessa ammattitaito on edellytys eikä näin ollen ole samanlaista riskiä ammattitaidottomuudesta, kuin jos yritys palkkaisi uuden työntekijän vastaamaan testauksesta. Usein ulkoista testausta suorittavilla yrityksillä on runsaasti kokemuksia erilaisista ohjelmistoista.

Testauksen ulkoistaminen auttaa kustannusten hallinnassa sekä aikataulutuksessa: kun testauksen suorittavan yrityksen kanssa on sovittu koska valmis ohjelmisto on heillä testauksessa, on paine pysyä tässä sovitussa ajassa kova. Myös tilaaja voi luottaa, että aikataulu ei testauksen suorittavan yrityksen vuoksi veny; kun tämä yhteistyökumppani otetaan määrittelyistä lähtien mukaan projektiin, he osaavat arvioida tarvittavan ajan testauksen suorittamiseen.

Toki ulkoistaminen sisältää riskejä, jotka on syytä arvioida tarkkaan. Vaikka oletuksena testausta suorittava yritys omaa kokemusta ja asiantuntijuutta, on tämä asia syytä varmentaa ennen sopimuksen tekemistä. Onko heillä asiakkaina tunnettuja yrityksiä? Onko mahdollista ottaa yhteys asiakkaisiin ja tiedustella työn tulosta? Yhtä lailla kuin ohjelmistoja tuottavat yritykset, myös testausta tuottavat yritykset voivat painia aikatauluongelmien kanssa. Ongelma voi myös olla niiden ajamissa testeissä; saadaanko niistä haluttua tietoa vai paljastuuko projektin päätyttyä, että on testattu jotakin aivan muuta kuin on sovittu.

Kun huolehditaan, että testauksen suorittava yritys on alansa asiantuntija ja tekee laadukasta työtä, on myös itse huolehdittava, että tekee tarpeeksi selväksi mitä odottaa testaukselta. Tämän vuoksi perustiedot testauksesta olisi hyvä omata, vaikkei itse testauksia suorittaisikaan. Testauksen ulkoistaminen ei myöskään saa olla syy tehdä huolimattomampaa koodia tai kehnompaa dokumentaatiota. Testauksen ulkoistaminen ei saa olla tekosyy hyväksyä omat puutteet eikä se saa estää kehitystä.

Testauksen ulkoistaminen on varteen otettava vaihtoehto eri tilanteissa. Vaikka riskinä on, että testaus ajautuu jälleen prosessin viimeiseksi vaiheeksi, jolloin korjaukset tulevat kalliiksi, voidaan ulkoisella testauksella kuitenkin saada arvokasta asiantuntijaetua sekä pidettyä aikataulu ja kustannukset aisoissa.

5.2.2 Riittävä määrittely

Ennen kuin testausta voidaan alkaa toteuttamaan, on tärkeää huolehtia valmisteluista. Dokumenttien laadinta on yksi osa testauksen suunnittelua, mutta niitäkään ei voida tehdä tyhjästä. Keskeinen osa testauksen suunnittelussa ovat ajantasaiset ja monipuoliset määrittelyt.

Määrittelyt ovat asiakkaan kanssa sovittuja ominaisuuksia, toimintoja tai seikkoja, joita ohjelmiston on toteutettava (Japenga n.d). Kyse voi olla konkreettisesta toiminnasta: ohjelmaan on kyettävä syöttämään nimi ja puhelinnumero. Ominaisuuksia voivat olla ulkoiset seikat, esimerkiksi värimaailma. Määrittelyihin voidaan sisällyttää abstrakteja toiveita, kuten hyvä käytettävyys tai käyttäjäystävällisyys. Yhtä lailla kuin on suunnittelijan tehtävä suunnitella näiden määrittelyjen pohjalta ohjelma ja sen ominaisuudet, on testaajan hyvä olla tässä vaiheessa mukana.

Testauksen suunnittelu rakennetaan määrittelyjen varaan. Jos määrittelynä on käyttäjäystävällisyys, mitä testaajan tulisi tällöin testata? Koska tämän kaltainen määrittely ei kerro ohjelmasta itsestään mitään, on testaajan rooli olla myös määrittelyjen testaaja. Mikäli määritellyistä ominaisuuksista ei voida työstää selkeää testitapausta, on testaussuunnittelijan kerrottava tästä eteenpäin ja vaatia asian konkretisoimista. Vaikka suunnitteluvaihe tämän johdosta venyisi, takaa se kuitenkin tyytyväisemmän asiakkaan projektin päätyttyä.

Yleisesti on esitetty, että jos virheen havaitseminen ja korjaaminen ohjelman suunnitteluvaiheessa maksaa yhden rahayksikön, tulisi havaitseminen juuri ennen testausta maksamaan noin 6, testauksen aikana noin 15 ja ohjelman julkaisun jälkeen 60-100 rahayksikköä. Virheiden yksi lähde on nimenomaan huono määrittely (Hower n.d). Kun määrittelyyn panostetaan jo suunnitteluvaiheessa, sujuu projektin muu osuus tehokkaasti. Ohjelmoijan ei tarvitse jatkuvasti kysyä, mitä milläkin vaatimuksella tarkoitetaan. Toisaalta testaaja pystyy suunnittelemaan testauksen tehokkaaksi tietäessään, mitä ominaisuuksia ohjelma sisältää ja kuinka sen tulisi toimia.

Luonnollisesti kun sekä suunnittelijoiden kuin muidenkin projektin jäsenten taidot karttuvat, he oppivat tunnistamaan huonoon tilanteeseen johtavat heikot määrittelyt ja näin vaatimaan asiakkaalta vastauksia ennen kuin huono määrittely johtaa ohjelman myöhästymiseen.

5.2.3 Lopetus

Testaukselle on aina liian vähän aikaa (The Standish Group 1995). Siksi priorisointi ominaisuuksien kesken on tärkeää; mitkä ominaisuudet ovat ohjelman kannalta kriittisiä eli pakko testata. Priorisoinnissa auttaa laadukkaasti toteutettu riskianalyysi. Priorisointi on tarpeellista myös sen vuoksi, ettei kaikkea voida testata. Priorisointi tulisi tehdä jo suunnitteluvaiheessa, ei missään tapauksessa enää silloin kun testaus on määrä aloittaa.

Testaukselle on syytä määritellä erikseen, millaisissa tapauksissa testaus lopetetaan. Kun nämä säännöt on määritelty ennen testauksen aloittamista, tietävät testausta suorittavat työntekijät, mitä ominaisuuksia tarkkailla. Testauksen kannalta selkein syy lopettaa on silloin, kun ohjelma on kerta kaikkiaan käyttökelvoton. Tämä tarkoittaa tilannetta, jossa mikä tahansa toiminta johtaa virheeseen. Ei ole mielekästä jatkaa testaamista ennen virheiden korjaamista.

Testaukselle voidaan määritellä tapauskohtainen onnistumisprosentti (Hower n.d). Tällöin jokaiselle testitapaukselle asetetaan vaatimus, kuinka monta testiajoa suoritetaan ja kuinka monen niistä on mentävä läpi. Kun rajat on sovittu etukäteen, kokematonkin testaaja tietää, että mikäli jokin toiminto onnistuu tarpeeksi monta kertaa, voidaan sen testaaminen jättää. Tällainen menetelmä ei kuitenkaan aina ole aukoton; nykypäivän ohjelmistot ovat monimutkaisia kokonaisuuksia, joissa yhden ominaisuuden muuttaminen voi hajottaa jo toimivia ominaisuuksia. Siksi olisikin suotavaa suorittaa kokonaisvaltaista testaamista jokaisen muutoksen jälkeen.

On mahdollista määrittää virhetaso, jonka alle mentäessä ohjelman testaus päättyy (Hower

n.d). Jokaisella testikierroksella löytyy tietty määrä virheitä eli niin kutsuttuja bugeja. Kun uuden testikierroksen päätteeksi löydetään vähemmän bugeja kuin määritelty virhetaso, ohjelman testaus voidaan lopettaa. Tällainen määritys tunnustaa sen tosiasian, että virheitä on eikä kaikkia voi löytää. Moni ei kuitenkaan olisi valmis määrittämään virhemäärän tasoksi muuta kuin nollan. Toisaalta on myös runsaasti virheitä, joita ei koskaan löydetä. Tämän tasoiset virheet voivat olla haitattomia. Niinpä on naiivia määrittää virhetasoa nollaksi, koska se luo kuvan, että kun nolla on saavutettu, ei virheitä enää olisi.

Useimmiten testaus loppuu siihen, että se on vain lopetettava (Hower n.d). Tällöin tilanne on sellainen, että testaukselle varattu aika on lopussa ja ohjelma pitäisi saada eteenpäin. Tämän kaltainen lopetus on usein sellainen, että testauksella ei enää ole vähään aikaan löydetty suuria käyttöä estäviä virheitä eikä näin ollen ole mielekästä jatkaa, ainakaan aikataulun puitteissa.

5.2.4 Riskit

Riskit ovat epäonnistumisen uhkia, jotka jokaiseen projektiin on määriteltävä (Lehtimäki 2006). Riskianalyysi sisältää riskit sekä arvion siitä, kuinka todennäköinen riski on ja ohjeistuksen siitä, kuinka riski vältettäisiin ja kuinka sen sattuessa toimitaan. Näin voidaan ensinnäkin ehkäistä riskiä, mutta myös riskin sattuessa saadaan riskianalyysista se hyöty, että on olemassa ohjeistus, kuinka toimia vahinkojen välttämiseksi.

Testauksen yleisin riski on aikatauluongelmat. Tämä tarkoittaa, että tiukan aikataulun vuoksi testauksesta joudutaan tinkimään, eikä testausta voida suorittaa suunnitellulla tavalla. Aikatauluongelma toteutuu riskinä usein ja siihen ei oikeastaan voida varautua kuin riittävän laajalla ja joustavalla aikataululla (Hower n.d). Valitettavasti yrityksillä on tavoitteena voittaa tarjouskilpailut matalilla kustannuksilla ja kustannukset pidetään matalalla tehokkaalla työnteolla. Tällöin realististen aikataulujen tekeminen ei ole mahdollista. Tätä riskiä on erittäin vaikea hallita, sillä asiakkaan voi olla vaikeaa ymmärtää miksi pitäisi valita tarjouksista sellainen, joka on valmis kolmessa kuukaudessa, kun muut tarjoavat samaa työtä kuukauden aikataululla.

Toinen riski on henkilöstö. Tällä voidaan tarkoittaa henkilöstön hyvinvointia eli riskiä sairauslomiin tai loukkaantumisiin. Toisaalta tarkoitetaan myös ammattitaitoa tai pikemminkin sen puutetta. Sairauslomiin voidaan varautua henkilöstön työkykyä ylläpitävillä päivillä ja työpäivien oikealla kuormittavuudella, mutta riski on suuri ja heikosti estettävissä. Työnantaja kun ei kuitenkaan voi painostaa työntekijöitään elämään työpaikan ulkopuolella terveellisesti ja turvallisesti. Loukkaantumisriskit eivät ohjelmistotaloissa liity niinkään työssä tapahtuviin tapaturmiin vaan esimerkiksi vapaa-ajan harrastuksiin. Nämäkin ovat valitettavasti sellaisia riskejä, ettei niihin voi vaikuttaa. Lomat sellaisenaan ovat riskitekijä eli mikäli työntekijälle on myönnetty talviloma juuri projektin loppupuolelle, on tärkeää laatia riskisuunnitelma kuinka toimitaan kun loma alkaa. Tällaiset riskit ovat varmasti tapahtuvia ja sen vuoksi huolellinen riskitilanteen suunnitelma on tarpeen. Näitä riskejä voidaan ehkäistä neuvottelemalla työntekijöiden kanssa loman pitämisestä sitten, kun projekti on valmis.

Työntekijöiden ammattitaidottomuus on riski. Tämä tarkoittaa, että testaus tuo eteensä uusia tilanteita, joita ei aiemmissa projekteissa ole toteutettu. Esimerkiksi erilaiset turvallisuuteen liittyvät testit voivat olla aivan uusia työntekijälle, joka on tottunut tekemään käytettävyystestausta. Tällaiset ongelmat havaitaan toivottavasti jo suunnitteluvaiheessa ja näin pystytään ajoissa aloittamaan pohdinta kuinka toimitaan. Nopea ratkaisu on ulkoistaa se osa testauksesta, jota ei itse pystytä toteuttamaan. Toisaalta jos vastaavanlaista testausta on tiedossa myöhemmissäkin projekteissa, henkilöstön kouluttaminen on hyvä vaihtoehto.

Vaihtuvat määrittelyt ovat erittäin yleinen riski testauksessa. Ei ole tavatonta, että alussa annetut määrittelyt muuttuvat ohjelmaa kehitettäessä useaan otteeseen. Jos asiakas on kiinteästi mukana ohjelman suunnittelussa ja toteutuksessa ja hän saa usein nähtäväkseen prototyyppejä, tällöin on oletettavissa, että asiakkaalta tulee myös muutostoiveita tai kokonaan uusia ominaisuuksia, jotka hän haluaa sisällyttää ohjelmaan (Hower n.d). Testauksen näkökulmasta muuttuvat ominaisuudet ovat rasittavia; jokaisen muutoksen jälkeen testausdokumentit on käytävä läpi ja muutettava tarvittavilta osin. Oikeastaan

muuttuvat ominaisuudet eivät sinällään ole oma riski, vaan ne aiheuttavat useita riskejä: aikataulu venyy eikä testaus vastaakaan sitä mitä odotetaan. Jälkimmäisellä tarkoitetaan tilannetta, jossa testataan ominaisuutta, jonka toiminnallisuus on muutettu testaajan tietämättä. Pahimmassa tapauksessa ohjelmoija korjaa ohjelman testaajan virheilmoituksen perusteella ja näin ohjelma ei toimi asiakkaan odottamalla tavalla. Riski ehkäistään pitämällä määrittelydokumentit kaikkien saatavilla ja ajantasaisina.

6 Yhteenveto

Testaus on keino parantaa ohjelman laadukkuutta ja varmentaa perusprosessien toiminnallisuutta. Testaukseen voidaan käyttää ääretön määrä resursseja, mutta testauksen voi suorittaa myös tehokkaasti. Testaus parhaimmillaan säästää kustannuksia, kun ohjelmien vakavat virheet havaitaan sisäisessä testauksessa, ei asiakkaan käytössä.

Testauksen aloittaminen voi aluksi vaikuttaa haasteelliselta ja monimutkaiselta, mutta aikaa myöten siitä tulee yhtä rutinoitunutta toimintaa kuin ohjelmoinnista. Alussa voi olla tarpeellista käyttää valmiita malleja ja istuttaa niitä oman yrityksen prosesseihin muokattuina. Kuten prosessit, myös testaus kehittyy jatkuvasti ja olennaista onkin ottaa ensimmäinen askel.

Opinnäytetyö on esittänyt, että testaus ei ole vain prosessin lopussa hätäisesti tehty toiminto, vaan sen toteuttaminen vaatii huolellista suunnittelua ja motivoituneita tekijöitä. Edellä on käsitelty testausta monesta näkökulmasta ja teksti on toivon mukaan inspiroinut projektijohtajia ja päättäviä tahoja tuomaan testausta aktiivisemmin mukaan ohjelmistoprojekteihin.

Lähteet

Beizer, Boris 1995. *Black-Box Testing: Techniques for Functional Testing of Software and Systems,* John Wiley & Sons

Chou, Andy 2010. *Static analysis improves efficiency, reduces downstream integration costs,* [online][viitattu 21.10.2010] http://www.mil-embedded.com/articles/id/?4508

Galin, Daniel 2004. *Software Quality Assurance,* Pearson Education

Goodwins, Rubert 2004. *Microsoft software caused air traffic shutdown* [online][viitattu 22.07.2010]. http://www.silicon.com/technology/software/2004/09/20/microsoft-software-caused-air-traffic-shutdown-39124122/

Hower, Rick n.d. *SoftwareQATest.com* [online][viitattu 21.10.2010] http://www.softwareqatest.com

IEEE 2004. *Standards description 829-1983* [online][viitattu 07.10.2010] http://standards.ieee.org/reading/ieee/std_public/description/se/829-1983_desc.html

Japenga, Robert n.d. *How to write a software requirements specification*[online] [21.10.2010] http://www.microtoolsinc.com/Howsrs.php

Kasurinen, Jussi 2010. *Dokumentoinnista* [online][viitattu 21.10.2010] http://www.ohjelmointikurssit.com/blogi/2010/05/15/dokumentoinnista/

Koch, Alan 2010. *Quantifying Risk: The Purpose of Testing* [online][viitattu 21.10.2010] http://www.cmcrossroads.com/the-road-to-quality/13269-quantifying-risk-the-purpose-of-testing

Kuningaskuluttaja n.d. *Toyota –autojen kaasupoljinviat* [online][viitattu 07.10.2010]. http://kuningaskuluttaja.yle.fi/node/2539

Lehtimäki, Timo 2006. *Ohjelmistoprojektit käytännössä,* Readme.fi

Marik, Brian 1997. *Classic Testing Mistakes.* [online][viitattu 21.10.2010] http://www.exampler.com/testing-com/writings/classic/mistakes.html

Murphy, Cliff. *Reducing Risk and Increasing the Probability of Project Success* [online] [viitattu 21.10.2010]. http://www.projectsmart.co.uk/reducing-risk-increasing-probability-

of-project-success.html

Nyman, Jeff 2002. *Positive and Negative Testing,*[online][viitattu 21.10.2010] http://www.sqatester.com/methodology/PositiveandNegativeTesting.htm

Ruuska, Kai 2007. *Pidä projekti hallinnassa,* Talentum

Pan, Jiantao 1999. *Software Testing,* Carnegie Mellon University

Sogeti UK 2010. *Sogeti survey: Does management see software testing as 'a costly necessity'?*[online][viitattu 23.07.2010]. http://www.sourcewire.com/releases/rel_display.php?relid=55536

The Standish Group 1995. *The Standish Group Report: Chaos* [online][viitattu 25.07.2010]. www.projectsmart.co.uk/docs/chaos-report.pdf

Vihdin kunta 2009. *Uusintavaalit - vuoden 2008 kunnallisvaalien uusiminen.* [online] [viitattu 21.10.2010] http://www.vihti.fi/ajankohtaista/1/kunnallisvaalien_uusiminen