Anna Tuominen

Ajanhallinta- ja raportointityökalun testaus

Opinnäytetyö
Syksy 2016
SeAMK Tekniikka
Tietotekniikan koulutusohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikka

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Sulautetut järjestelmät

Tekijä: Anna Tuominen

Työn nimi: Ajanhallinta- ja raportointityökalun testaus

Ohjaaja: Heikki Palomäki

Vuosi: 2016 Sivumäärä: 41 Liitteiden lukumäärä: 0

Tämän opinnäytetyön tarkoituksena oli testata Wapice Oy:n sisäiseen käyttöön tarkoitetun selainpohjaisen ajanhallinta- ja raportointityökalun toimintaa. Testattavan ohjelman tarkoituksena on tarjota käyttäjälleen graafista materiaalia ajanhallinnan ja raportoinnin tueksi. Ohjelmalla voidaan luoda useita erilaisia kaavioita käyttäjän valitsemien kriteereiden mukaisesti. Saatavilla olevia kaavioita ovat viivakaavio, lämpökartta, sankey-diagrammi, histogrammi ja puukartta. Tuotetut kaaviot voidaan tallentaa tietokoneelle Excel- tai PDF-tiedostoina. Testauksen tarkoituksena oli testata, toimiiko ohjelma tarkoitetulla tavalla, sekä selvittää, mitä ongelmia siinä mahdollisesti on ja millaisia virheilmoituksia se antaa missäkin tilanteessa.

Testi suoritettiin kokeilemalla ensin yksinkertaisesti ohjelman tarjoamia toimintoja, kuten kaavioiden luontia eri kriteerein, niiden tallentamista tietokoneelle, sekä ohjelman reagointia muutoksiin. Virheilmoituksiin päästiin käsiksi muun muassa antamalla ohjelmalle vajaita tietoja tulostettavasta kaaviosta. Tästä myös nähtiin, mitkä kohdat on vaadittu kaavion tulostamiseksi. Jokainen osio testattiin vähintään viisi kertaa ja eri kriteereitä käyttäen. Testaukset suoritettiin käyttämällä Google Chrome-, Mozilla Firefox- ja Internet Explorer -selaimia, jotta nähtiin ohjelman toimintaerot eri selainten välillä. Testauksesta kirjoitettiin raportti, josta ilmeni testisuunnitelma, testausmenetelmät, odotetut tulokset ja saadut tulokset.

Tuloksena saatiin selvitettyä, että ohjelman perustoiminnot toimivat odotetulla tavalla, vaikka joidenkin yksittäisten tapausten kohdalla ohjelma toimii hieman odotuksista poikkeavasti. Kaikki toiminnot eivät toimi joka näkymässä aivan odotetusti, mutta yleisesti ohjelma on pientä hiontaa vaille käyttövalmis.

Avainsanat: ohjelmistotestaus, ohjelmistotuotanto, raportointi, ajanhallinta, applikaatio, kaavio, testaus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Embedded Systems

Author: Anna Tuominen

Title of thesis: Testing of the reporting and time management tool

Supervisor: Heikki Palomäki

Year: 2016 Number of pages: 41 Number of appendices: 0

The aim of this thesis was to test the behaviour and functionality of a web based application for reporting and time management. The commissioner was Wapice Oy. This visualization application offers different types of graphical material for viewing reports. It can create multiple types of diagrams that show information about the hours spent on working with certain customers and projects, based on the criteria selected by the user. The types of diagrams currently available are line chart, heat map, Sankey diagram, histogram and tree map. The diagrams can be saved to the computer as Excel- or PDF-files. The purpose of this test was to reveal possible problems, errors and other dysfunctionalities in the application.

The test was executed by making the application create different kinds of diagrams and saving them to the computer. The aim was to see if all the features work correctly and how the application reacts to changes. It was also important to reveal what kinds of error messages it generates and in what kind of situations, and what does it take to crash the application. Each feature was tested at least five times by using different criteria every time. Everything was tested using the following web browsers: Google Chrome, Mozilla Firefox and Internet Explorer.

As a result, information on the functionality and problems was found. It turned out that the application has a few minimal flaws but generally it is working well. Though the application is not completely finished yet it still has potential as it generates useful graphical material for reporting.

All the results were documented into a test report that includes test case tables for each tested feature as well as the testing methods used. The tables indicate the test plan as well as the results of basic functionality, error situations and saving data.

Keywords: software testing, software engineering, reporting, time management, diagrams, application

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

Alfatestaus	Sisäinen hyväksymistestaus, jossa järjestelmä on kokonainen ja sisältää jo kaikki siihen suunnitellut ominaisuudet.
Betatestaus	Suuremman testiyleisön suorittama, viimeinen testausvaihe ennen tuotteen julkaisua.
Bugi	Häiriöön johtava poikkeama ohjelmistossa.
Integraatio	Valmiiden osien kokoaminen yhdeksi toiminalliseksi kokonaisuudeksi.
Iterointi	Samojen työvaiheiden toistaminen, kunnes saavutetaan haluttu päämäärä.
Kuormitustestaus	Järjestelmän kuormansiedon testaus.
Käytettävyystestaus	Käyttöliittymän ja sen toimivuuden testaus.
Laatikkomallit	Testausmenetelmiä, joista kukin kuvaa testaajan kykyä tarkastella järjestelmän sisäisiä toimintoja testauksen aikana.
Prosessimalli	Malli, joka kuvaa yleisperiaatetta toiminnan hallintaan ja ohjaukseen.
Regressiotestaus	Minkä tahansa menetelmän mukaisen testauksen suorittaminen moneen kertaan.
RUP	Ohjelmistotuotannon yleismalli, jossa yhdistyy suunnitelmalähtöisen ohjelmistotuotannon parhaaksi nähdyt puolet hyväksi havaittuihin toimintatapoihin (Rational Unified Process).
SCRUM	Tavallisesti pienemmissä organisaatioissa ja projekteissa käytetty ketterä ohjelmistotuotannon menetelmä.

SDLC	Ohjelmistokehityksen elinkaari (Software Development Life Cycle).
Sprintti	Scrum-mallissa projektin perusyksikkö.
Testitapaus	Kuvaus siitä, miten ohjelman kuuluisi toimia missäkin tilanteessa.
UML	Graafinen mallinnuskieli (Unified Modeling Language).
Vesiputousmalli	Perinteinen ohjelmistotuotannon malli, jossa edetään vesiputouksen tavoin vaihe kerrallaan ja testausta suoritetaan ainoastaan ennen käyttöönottoa.
V-malli	Vesiputousmallin kaltainen tuotantomalli, jossa testausta tehdään enemmän.

1 JOHDANTO

1.1 Työn tausta

Työssä testataan toimeksiantajan testausvaiheessa olevaa, yrityksen sisäiseen käyttöön tarkoitettua applikaatiota, joka on luotu avuksi raportointiin ja ajanhallintaan. Sen avulla voidaan tarkastella tehtyjä työtunteja kunkin projektin parissa. Kaaviotyypistä riippuen näitä tietoja voidaan tarkastella esimerkiksi työntekijän, tiimin tai ajan mukaan.

1.2 Työn tavoite

Työn ensisijaisena tavoitteena on testata applikaation ja sen osien käyttäytymistä. Tarkoituksena on selvittää, toimiiko ohjelma toivotulla tavalla, mitä ongelmia siinä on, sekä mitä vaaditaan sen kaatumiseen. Tärkeää on myös tutkia eri virhetilanteita ja applikaation antamia virheilmoituksia. Testausta varten laaditaan kirjallinen testisuunnitelma, josta ilmenee applikaation perustietojen ja tavoitteiden lisäksi testaustavat, testauksissa käytetyt kriteerit, sekä odotetut tulokset kunkin komponentin kohdalla. Kunkin osion testauksen jälkeen raporttiin kirjataan saadut tulokset, sekä mahdolliset korjausehdotukset.

1.3 Työn rakenne

Luvussa 2 tutustutaan yleisesti ohjelmistotestaukseen, sekä sen rooliin erilaisissa ohjelmistotuotannon toimintamalleissa.

Kolmannessa luvussa esitellään erilaisia ohjelmistotestauksen menetelmiä, joita käytetään, kun testattava tuote on kehitysvaiheessa.

Neljäs luku käsittää erilaisia ohjelmistotestauksen menetelmiä, joita käytetään ennen tuotteen julkaisua.

Viidennessä luvussa käydään läpi työssä testattua ohjelmaa, sekä testauksen etenemistä, menetelmiä ja tuloksia.

Kuudennessa luvussa on työn yhteenveto ja pohdintaa.

1.4 Wapice Oy

Wapice Oy on vuonna 1999 perustettu teollisuuden johtava teknologiapartneri, joka ohjelmisto- ja elektroniikka-asiantuntemustaan hyödyntäen tarjoaa asiakkailleen toimintaa tehostavia ohjelmistoja ja ratkaisuja. Se on yksityisesti omistettu ja AAA-luokiteltu yritys, jonka toiminta perustuu ISO 9001:2008- ja ISO 14001:2004 -laatusertifikaatteihin. Wapice Oy:n pääpaikka sijaitsee Vaasassa ja sen muita yksiköitä on Tampereella, Oulussa, Seinäjoella, Hyvinkäällä, sekä Jyväskylässä. (Wapice Oy [Viitattu 11.5.2016].)

2 OHJELMISTOTESTAUS

2.1 Testaus yleisesti

Ohjelmistotestaus on osa ohjelmistotuotantoa. Testauksilla pyritään varmistamaan, että toteutettu ohjelmistotuote toimii tarkoitetulla tavalla ja vastaa odotuksia. Testaustyöhön kuuluu myös vertailua suunnitellun tuotteen ja aikaansaadun tuotteen välillä. (Kasurinen 2013, 10.)

Ohjelmistotestaus on laaja ja vaihteleva kokonaisuus, joka testauksen tyypistä riippuen voi vaatia testaajalta monenlaista osaamista. Testaaja voi joutua työssään tekemään esimerkiksi ohjelmointia, dokumentaatioita ja haastatteluja. Kunkin testauksen painopiste riippuu siitä, millainen, kenelle ja mitä varten testattava ohjelmisto on luotu. Esimerkiksi peliyrityksissä painopiste on paljolti graafisissa ominaisuuksissa ja käyttäjän viihtymisessä, kun taas arkisten asioiden hoitamiseen suunnitelluissa ohjelmistoissa kiinnitetään huomiota helppokäyttöisyyteen. (Kasurinen 2013, 10.)

2.2 Testauksen suunnittelu ja tavoitteet

Testausta suunnitellessa on oleellista tehdä vaatimusmäärittelyt, joista ilmenee ohjelmalta odotetut toiminnot, sekä sen käytön rajoitteet. Testausta varten laaditaan myös testitapaukset, jotka kertovat, miten ohjelman kuuluisi toimia missäkin tilanteessa. (Kasurinen 2013, 63.)

Yhtenä testauksen tavoitteena on varmistaa, että tuote on tehty oikein ja vastaa vaatimusmäärittelyitä. Toisekseen testauksessa selvitetään, että tehty tuote on oikeanlainen käytettävissä oleva tuote, josta ei puutu sen käytön kannalta tärkeitä ominaisuuksia. Testauksessa pyritään myös löytämään kaikenlaiset bugit, joita voi olla syntaksisia tai semanttisia. Syntaksisella virheellä tarkoitetaan väärinkirjoitetun koodin aiheuttamaa ohjelman kaatumista, kun taas semanttisessa virheessä koodin kirjoitusasu on oikein, mutta sen tekemät toiminnot ei. Syntaksiset virheet ovatkin siis huomattavasti helpompia korjata. (Kasurinen 2013, 63.)

Esituotantovaiheessa suunnitellaan, kokeillaan sekä rakennetaan testiympäristö ja testaukseen tarvittavat työkalut, joiden toimiminen on tärkeää projektin edetessä kehitysvaiheeseen. Lisäksi esitestausvaiheessa valitaan testaajat, jotka tarvittaessa myös koulutetaan järjestelmän ympäristöön. Organisoidun testaustyön suunnitteluun kuuluvat myös määrittelyt muun muassa tehtävänjaoista ja viestinnän kulusta. (Kasurinen 2013, 64.)

2.3 Testaustasot

Testausta suoritetaan eri tasoilla ja näitä tasoja esittelee muun muassa ohjelmistotuotannon V-malli. Testaustasoilla mennään yksittäisestä komponentista koko järjestelmän testaamiseen, näitä tasoja ovat yksikkötestaus, integrointitestaus, järjestelmätestaus ja hyväksymistestaus. Näiden testaustasojen jälkeen tuote on valmis käyttöönotettavaksi. (Kasurinen 2013, 50-51.)

Yksikkötestaus kohdistuu johonkin toteutettavan järjestelmän yksittäiseen komponenttiin tai toimintoon. Sen suorittaa itse toteutuksen tehnyt henkilö, usein ohjelmoija, välittömästi toteutuksen jälkeen. Tarkoituksena on varmistaa, että kyseinen toteutus on toimiva, sekä pyrkiä korjaamaan viat ennen kuin toteutus on osa laajempaa tuotekokonaisuutta. (Kasurinen 2013, 51.)

Integrointitestaus on yksikkötestauksen jälkeinen testausvaihe, jossa yksittäisistä osista rakennetaan yksi toimiva kokonaisuus. Tarkoituksena on testata, että nämä yksittäiset osat toimivat myös yhdessä tai että koko järjestelmä yhä toimii, kun siihen lisätään jokin uusi komponentti. Integrointitestauksessa testataan siis yksikkötestausta laajemmin, mutta koko järjestelmän käyttöön liittyviä testejä ei vielä voida suorittaa. (Kasurinen 2013, 54.)

Kun komponentit on yksikkötestattu ja koottu yhteen integrointitestauksessa, siirrytään järjestelmätestaukseen. Järjestelmätestauksessa testataan kokonaista järjestelmää ja tarkastellaan sen toimivuutta yhtenä kokonaisuutena. Testauksessa selvitetään, että järjestelmä toimii odotetulla tavalla ja sisältää kaikki siltä vaaditut ominaisuudet. Tässä vaiheessa tuotteelle voidaan vielä tehdä muutoksia ja korjauksia

tarpeen mukaan. Järjestelmätestausta voidaan suorittaa esimerkiksi musta laatikko- tai lasilaatikko-menetelmällä. (Kasurinen 2013, 56-57.)

Yksikkötestauksen, integrointitestauksen ja järjestelmätestauksen jälkeen siirrytään hyväksymistestaukseen, jossa tarkoituksena on simuloida lopputuotteen toimintaa sille tarkoitetussa kohdeympäristössä. Tämä on tuotteen viimeinen testaus, jolla varmistetaan, että tuote on valmis loppukäyttäjälle. Tuotteeseen ei siis tehdä enää merkittäviä muutoksia, vaan tarkoituksena on todeta tuotteen olevan valmis ja hyväksytty, sekä siirtää se asiakkaan omistukseen. (Kasurinen 2013, 57.)

2.4 Ohjelmistotuotannon mallit

Ohjelmistotuotantoon on kehitetty useita erilaisia malleja, jotka kuvaavat tietynlaisen prosessin etenemistä vaihe vaiheelta kohti valmista lopputuotetta. Mallin valintaan vaikuttaa paljolti se, millaista tuotetta ollaan tekemässä. Muun muassa organisaation, projektin ja budjetin suuruus, sekä projektiin käytettävän ajan pituus vaikuttavat sille sopivan tuotantomallin valintaan. (Kasurinen 2013, 24.)

2.4.1 Vesiputousmalli

Perinteisessä ohjelmistotuotannon vesiputousmallissa testaus on vain yksi työvaihe, joka suoritetaan ennen tuotteen käyttöönottoa. Vesiputousmallin mukainen prosessi käynnistyy määrittelyillä ja etenee vesiputouksen tavoin askel askeleelta alaspäin kohti ylläpitoa (Kuva 1). Malli on vanhanaikainen ja monesti ongelmallinen, mutta helposti ymmärrettävä ja yksinkertainen. Siksi se voi soveltua paremmin pienempien ja lyhytaikaisempien projektien toteutuksessa, mutta suuremmissa, pitkän aikavälin projekteissa se on varsin riskialtis ja epävarma tuotantomalli. (Kasurinen 2013, 12-13.)

Vesiputousmalli on varhaisimpia ohjelmistokehityksen elinkaaren (SDLC) malleja. Sen ajatuksena on edetä suoraan vaiheesta vaiheeseen siten, että seuraavaa vaihetta ei aloiteta ennen kuin edellinen on suoritettu loppuun. Näin vaiheita ei myöskään suoriteta päällekkäin. (Tutorials Point [Viitattu 13.4.2016].)

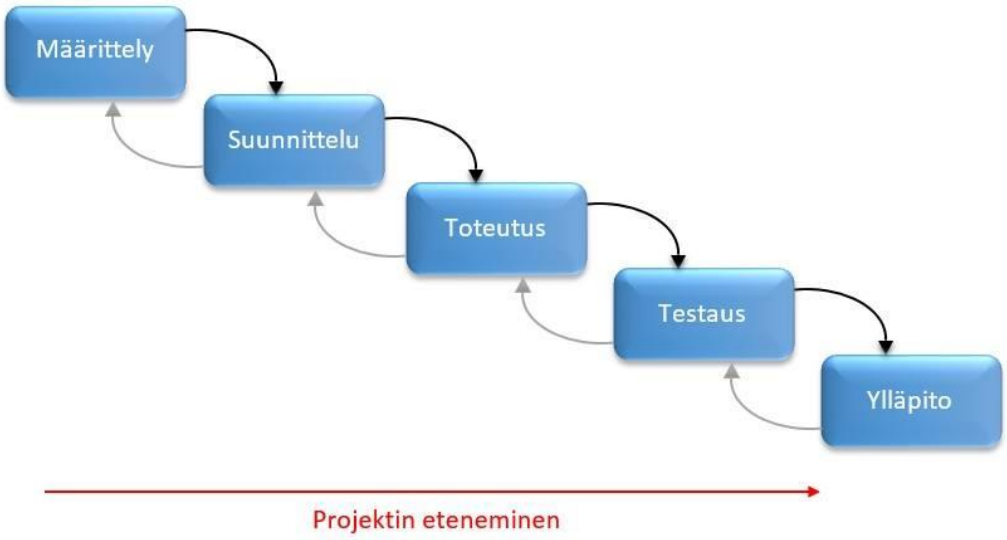
Ohjelmiston tekemisen ensimmäisessä vaiheessa tehdään vaatimusmäärittelyt. Ensin kootaan yhteen markkina-analyysi, tiedot kannattavuuslaskelmista ja alustavista asiakastarpeista. Näiden pohjalta listataan millaisia vaatimuksia lopullisen ohjelman tulisi täyttää. (Kasurinen 2013, 12-13.)

Kun vaatimusmäärittelyt on saatu huolellisesti päätökseen, siirrytään suunnitteluvaiheeseen. Tässä vaiheessa käydään läpi edellisessä vaiheessa tehdyt määrittelyt ja niiden pohjalta suunnitellaan laitteisto- ja ohjelmistoarkkitehtuuria, toiminnallisia vaatimuksia, sekä koko projektin etenemistä. (Kasurinen 2013, 12-13.)

Tehtyjen suunnitelmien pohjalta voidaan siirtyä toteutusvaiheeseen, jossa ohjelma ja sen vaatima laitteisto kootaan. Kun jokainen järjestelmän osa on saatu valmiiksi, niitä sovitellaan yhteen, tavoitteena on rakentaa niistä yksi toiminnallinen kokonaisuus. Tätä kutsutaan integraatioksi. (Kasurinen 2013, 12-13.)

Seuraava vaihe on testaus, jonka tavoitteena on varmistaa, että tuote on toimiva ja suunnitelmien mukainen. Testausta jatketaan, kunnes ohjelmasta ei enää löydy merkittävän suuria virheitä ja se on tarpeeksi toimiva siirtyäkseen käyttöönotettavaksi. Mikäli tuote vaatii vielä pieniä muutoksia, tehdään ne tässä vaiheessa ennen siirtymistä ylläpitovaiheeseen. (Kasurinen 2013, 12-13.)

Ylläpitovaiheessa tuotteelle tehdään viimeiset korjaukset, mikäli siitä ongelmia vielä löytyy. Tarpeen vaatiessa siihen voidaan myös tehdä lisäyksiä, edellyttäen että ne ovat tarpeeksi yksinkertaisia toteuttaa. (Kasurinen 2013, 12-13.)



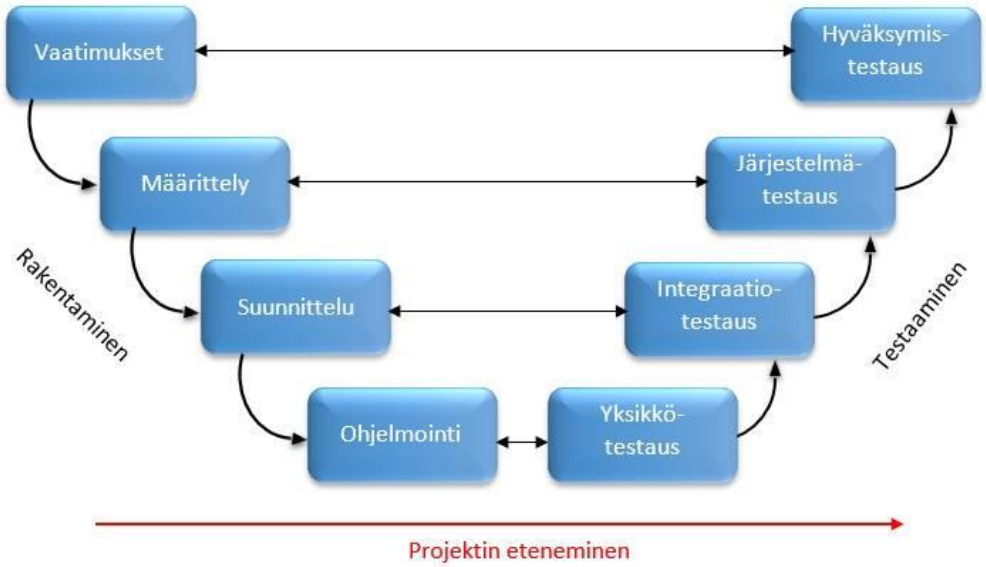
Kuva 1. Vesiputousmalli
(Kasurinen 2013).

2.4.2 V-malli

V-mallin mukaisessa tuotantoprosessissa edetään rakentamispuolella samoin kuin vesiputousmallissa, mutta testausta tehdään enemmän. Mallissa edetään ylävasemmalta alkaen alas ja jatketaan alhaalta yläoikealle (Kuva 2). Kun kuljetaan vasemmalla alas, jokainen vaihe on jonkinlaista tekemistä, kun taas mallin oikealla puolella ylöspäin edettäessä jokainen vaihe on testausta. Kukin testausvaihe kohdistuu oman rakennusvaiheen testaukseen eli siihen, mikä sen kanssa mallissa on samalla korkeudella. Ohjelmointityön tarkistus tehdään siis yksikkötestauksella, suunnitelmien toteutuminen integraatiotestauksella, määrittelyiden ja toteutusten vertailu järjestelmätestauksella ja vaatimusten mukaiset järjestelmän toteutukset hyväksymistestauksella. Mikäli ohjelma selviytyy näistä kaikista testauksista, on se valmis käyttöönotto- ja ylläpitovaiheeseen. (Lehtimäki 2006, 151.)

Vaikka V-mallissa testaustyötä tehdäänkin enemmän kuin vesiputousmallissa, ei sekään ole täysin ongelmaton. Se, että testausvaiheisiin siirrytään niin myöhään,

voi aiheuttaa ongelmia. Mitä aikaisemmin viat ja ongelmat havaitaan, sitä helpompaa ja edullisempaa niiden korjaaminen on. Myöhemmin havaitut viat taas voivat olla vaikeita ja kalliita korjata. (Lehtimäki 2006, 151.)



Kuva 2. V-malli
(Kasurinen 2013).

2.4.3 RUP-malli

Rational Unified Process eli RUP on Rational Softwaren vuonna 1996 kehittämä malli, joka hyödyntää kuutta ohjelmistotuotannon parhaiksi havaittua käytäntöä:

1. Iteratiivinen kehitystyö
2. Vaatimusten hallinta
3. Itsenäisesti toimivista komponenteista rakennettu ohjelma
4. Mallien visuaalinen suunnittelu UML-kaavioita käyttäen
5. Ohjelmiston laadunvalvonta
6. Hallitut muutokset.

(Rational Software 1998.)

RUP-mallissa prosessi kuvataan vaaka- ja pystyakselilla (Kuva 3). Vaaka-akseli ilmaisee ajankulkua, sekä prosessin dynaamista puolta, kuten päävaiheita, iteraatioita ja merkkipaaluja. Pystyakseli edustaa prosessin staattista puolta, joka ilmaisee, mitä tuotteelle tehdään kussakin päävaiheessa. Päävaiheita ovat aloittaminen, tarkentaminen, rakentaminen ja siirtymä. Kukin vaihe saadaan päätökseen merkkipaaluilla, jotka ilmaisevat, milloin tiettyjen päätösten on oltava tehtyinä tai saavutusten saavutettuina. (Rational Software 1998.)

Ensimmäinen päävaihe on aloittaminen. Tässä vaiheessa selvitetään liiketoiminnan mallia ja vaatimusmäärittelyitä. Tarkoituksena on luoda ja dokumentoida tuotteeseen haluttavia toimintoja, sekä niiden toteutukseen vaadittuja ominaisuuksia. Liiketoiminnalliset määrittelyt tehdään muun muassa arvioimalla riskejä ja tulosennusteita. Tässä vaiheessa luodaan projektisuunnitelma, josta ilmenee myös projektin aikatauluarvio. Mikäli projekti ei pääse tämän vaiheen yli, saatetaan se joutua uudelleenarvioimaan tai jopa perumaan. (Rational Software 1998.)

Mikäli prosessin ensimmäinen vaihe saadaan kunnialla päätökseen, voidaan siirtyä mallin seuraavaan vaiheeseen eli tarkentamiseen. Tässä työvaiheessa suunnitellaan tuotteen teknisiä ominaisuuksia ja ohjelmiston arkkitehtuuria, sekä näiden pohjalta myös rakentamaan varsinaisia prototyyppejä. Tämän työvaiheen aikana tulisi todentaa kaikki käyttötapaukset ja toimijat, sekä tunnistaa projektin ongelmalliset ja riskialttiit yksityiskohdat. Vaiheen lopussa selvitetään valmiudet siirtyä seuraavaan vaiheeseen eli varmistetaan, ovatko tuotteen arkkitehtuuri ja suunnitelmat mukautettavissa mahdollisiin muutoksiin, ovatko rakentamisvaiheen suunnitelmat tarpeeksi yksityiskohtaisia, sekä onko yleisesti koko projektisuunnitelma loppuun asti toteutettavissa. (Rational Software 1998.)

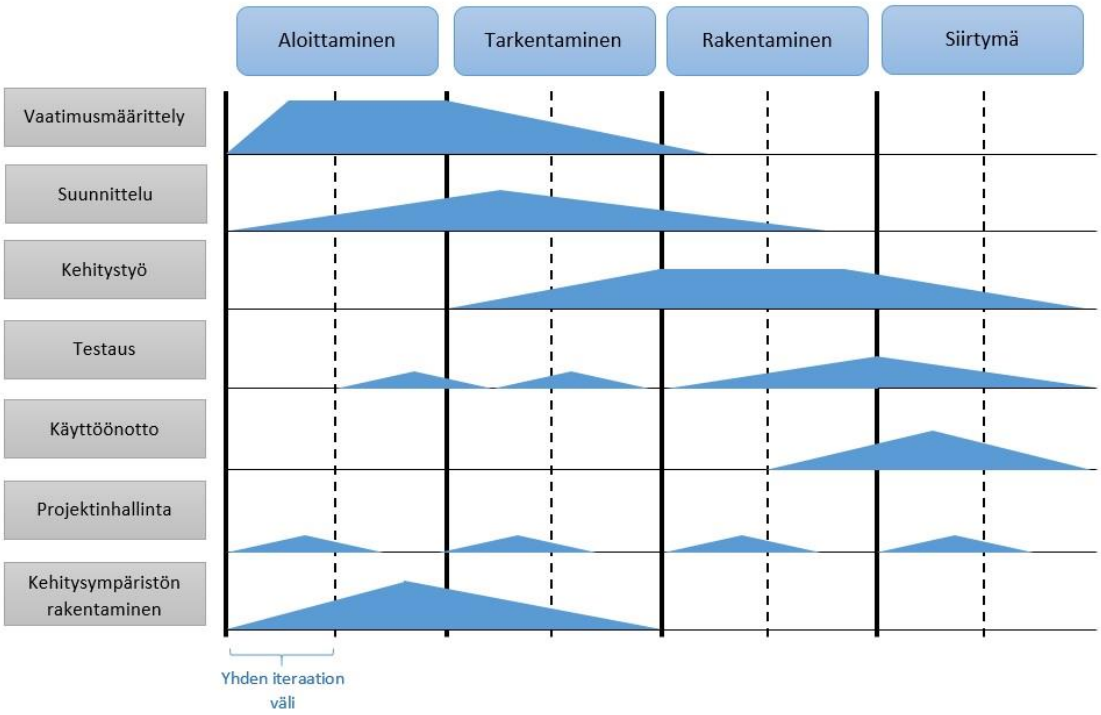
Kolmannessa työvaiheessa eli rakentamisessa kehitetään kaikki komponentit ja applikaation ominaisuudet, integroidaan ne tuotteeseen, sekä testataan huolellisesti. Riskialttiimmat osiot tehdään ensin ja ohjelma toteutetaan vaihe vaiheelta siten, että viimeisimmät hienosäädöt voidaan vielä toteuttaa. (Rational Software 1998.)

Rakennusvaiheessa suoritetaan useita eri testauksia, kuten yksikkötestauksia, integrointitestauksia ja järjestelmätestausta. Tässä vaiheessa voidaan testausten

puolesta aloittaa myös kehitysvaiheen testausmenetelmillä testaaminen, esimerkiksi lasilaatikkomenetelmää käyttäen. Kyseessä on siis itse tuotantoprosessi, jonka aikana tuote saadaan siihen kuntoon, että se voidaan siirtää loppukäyttäjälle. (Kasurinen 2013, 26.)

Mallin viimeisessä vaiheessa, siirtymässä, tarkoituksena on saada ohjelmistotuote käyttöön. Käytännössä tuote on siis riittävän valmis ja laadun puolesta hyväksyttävällä tasolla siirtyäkseen loppukäyttäjän kokeiltavaksi. Tuotteelle suoritetaan beetatestaus, jonka avulla saadaan selville tuotteen vikoja, virheitä ja ongelmia, joita sitten beetatestauksista saatujen dokumentointien pohjalta lähdetään korjaamaan. Siirtymisvaiheen lopussa selvitetään, ovatko kaikki tavoitellut päämäärät saavutettu, onko käyttäjä tyytyväinen, sekä onko käytettyjen ja suunniteltujen resurssikustannusten suhde hyväksyttävällä tasolla. (Rational Software 1998.)

Tärkein ominaisuus, jonka avulla RUP-malli eroaa muista, on se, miten siinä eri työtehtäviä suoritetaan tuotannon jokaisessa vaiheessa. Testaus ei ole vain kertaluontoinen vaihe työn lopussa, vaan sitä tehdään jatkuvasti ja useita eri menetelmiä käyttäen. Testaukset aloitetaan, kun ensimmäinen prototyyppi on valmis ja niitä toistetaan, kunnes tuote on julkaisuvalmis. Toinen RUP-mallin keskeinen ominaisuus on sen iteratiivinen kehitystyö, jonka mukaan tuotteen riskialttiimmat osat hoidetaan ensimmäisenä. Tällä varmistetaan se, että tuote todella saadaan julkaisukelpoiseksi, eikä projekti kaadu loppuvaiheissa löydettäviin suuriin vikoihin. RUP-mallin mukaan tehdyissä tuotantoprosesseissa laadunvalvonta ja -arviointi ovat paremmin hallittavissa. (Kasurinen 2013, 26-27.)



Kuva 3. RUP-malli
(Kasurinen 2013).

2.4.4 Scrum-malli

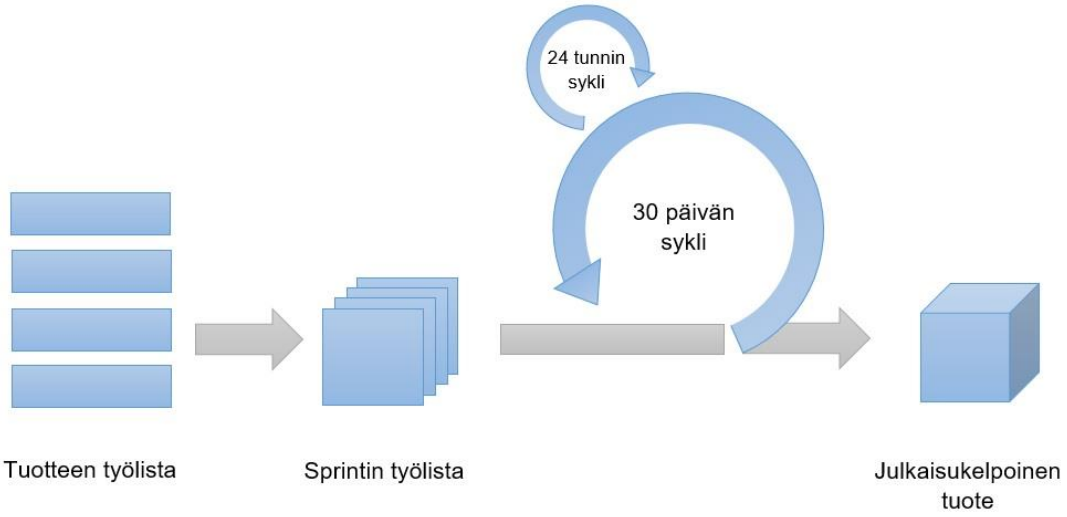
Scrum-malli (Kuva 4) on ketterä menetelmä. Tämä tarkoittaa, että mallissa pyritään vähentämään hallinnollista työtä ja näin menetellen voidaan reagoida nopeammin projektissa tapahtuviin muutoksiin. Kun suunnitelmalähtöisissä menetelmissä pyritään varautumaan kaikkiin mahdollisiin ongelmiin, ketterissä menetelmissä pyritään reagoimaan nopeasti niihin ongelmiin, joita projektin aikana oikeasti tulee vastaan. Scrum-mallissa myös ryhmän jäsenien yhteistyö ja kommunikointi ovat tärkeässä asemassa ja tarkoituksena on pitää päivittäisiä palavereja, joissa sovitaan viikon aikana tehtävät asiat kunkin jäsenen osalta, sekä keskustellaan mahdollisista ongelmista. (Kasurinen 2013, 27.)

Scrum-mallissa aika on jaettu lyhytkestoisiin, tavallisesti viikon tai kahden pituisiin, työkokonaisuuksiin eli sprintteihin. Nämä sprintit muodostetaan vaatimusmäärittelyiden ja suunnitelmien pohjalta. Jokaisen sprintin lopussa projektiin osallistuvat kokoontuvat tarkastelemaan aikaansaatua tuotosta ja keskustelevat sen seuraavista vaiheista. (James [Viitattu 22.5.2016].)

Sprintit ovat kuin pieniä projekteja, sillä projektien tapaan sprinttejä käytetään jonkin tavoittelemiseen tai aikaansaamiseen. Jokaisella sprintillä on määritelmänsä siitä, mitä tehdään ja miten, sekä millaisiin lopputuloksiin tulisi päätyä. Sprintin aikana ei tehdä mitään muutoksia, jotka voisivat vaarantaa kyseisen sprintin tavoitteisiin pääsyn. (Schwaber & Sutherland 2014.)

Scrum-tiimi koostuu tuotteen omistajasta, kehitystiimistä ja Scrum-mestarista. Tuotteen omistaja vastaa tuotteen työlistasta, josta ilmenee tuotteelta vaaditut ominaisuudet, sekä varmistaa, että kehitystiimin jäsenet ymmärtävät työlistan kohdat. Kehitystiimi koostuu asiantuntijoista, joiden tehtävänä on yhdessä rakentaa tuote, sekä julkistaa se mahdollisimman julkaisukelpoisena jokaisen sprintin lopussa. Scrum-mestari taas muun muassa varmistaa, että Scrum on kaikkien osalta ymmärretty ja sen kaikkia ominaisuuksia noudatetaan. (Schwaber & Sutherland 2014.)

Scrum-malli alkaa visioinnilla. Tässä määritellään, mitä ollaan tekemässä ja millainen lopputuotteen kuuluisi olla. Tämän jälkeen tehdään tuotteen työlista, josta käy ilmi lopputuotteelta vaaditut ominaisuudet. Seuraavaksi suunnitellaan eri tehtävät ja toiminnot, joita sprintin aikana tulisi tehdä. Sprintin toteutuessa tiimi suorittaa kyseiselle sprintille suunnitellut tehtävät siten, että haluttuun päämäärään päästään. Päivittäisissä, noin 15 minuuttia kestävissä palavereissa käydään läpi projektin etenemistä ja mahdollisia ongelmatilanteita. Kunkin sprintin lopussa tuotteen tulisi olla periaatteessa julkaisukelpoinen, kun se esitetään tuotteen omistajalle. Tuotteen omistaja päättää voidaanko tuote ottaa käyttöön vai suoritetaanko seuraava sprintti. Mikäli päädytään jatkamaan seuraavaan sprinttiin, koko Scrum-tiimi keskustelee yhdessä edellisen sprintin hyvistä ja huonoista puolista, jotta näiden pohjalta voidaan laatia uusi toimivampi sprinttisuunnitelma. (Tolvanen 2013.)



Kuva 4. Scrum-malli
(Kasurinen 2013).

3 KEHITYSVAIHEEN TESTAUSMENETELMÄT

Järjestelmän testausta voidaan suorittaa staattisesti tai dynaamisesti. Staattisessa testauksessa tutkitaan järjestelmän arkkitehtuuria ja koodia itse järjestelmän käyttämisen sijaan. Siinä pyritään varmentamaan, että järjestelmän logiikka on kunnossa ja selkeästi havaittavat virheet saadaan korjattua ennen tarkempaa järjestelmän käyttöön perustuvaa testausta. Staattinen testaus voidaankin suorittaa jo hyvin varhaisessa vaiheessa tuotantoa, sillä se ei vaadi järjestelmän käyttöä. Tämän vuoksi myös havaitut ongelmat on helpompi ja halvempi korjata, kuin dynaamisessa testauksessa havaitut. Staattista testausta voi olla esimerkiksi lasilaatikkomenetelmillä suoritetut testaukset. (Kasurinen 2013, 65.)

Dynaamisessa testauksessa taas on oleellista, että järjestelmää voidaan käyttää. Siinä seurataan, miten järjestelmä reagoi annettuihin syötteisiin käytännössä. Dynaamiseen testaukseen kuuluvat muun muassa yksikkötestaus, integrointitestaus ja kuormitustestaus. (Kasurinen 2013, 65.)

3.1 Musta laatikko -testaus

Musta laatikko -testaus on yksi tunnetuimpia ja käytetyimpiä ohjelmistotestauksen menetelmiä. Se on toiminnallinen testaus, joka perustuu ulkoisesti havaittavien toimintojen tarkasteluun. Nimensä mukaisesti tässä testauksessa testattava ohjelma on kuin musta laatikko, jonka sisään testaaja ei näe (Kuva 5). Testaajalla ei siis ole pääsyä ohjelmakoodiin, vaan hänen tehtävänään on antaa ohjelmalle syötteitä ja vertailla saatua lopputulosta odotettuihin tuloksiin, näkemättä mitä ohjelman sisällä tapahtuu. (Kasurinen 2013, 65-66.)

Testauksessa määritellään testitapaukset, joista ilmenee annettavat syötteet tai tehtäväsarjat, sekä se, miten ohjelman tulisi näihin reagoida. Ohjelmalle voidaan myös syöttää virheellisiä tai haitallisia syötteitä ja tarkastella sen reagointia niihin. Testauksessa voidaan kokeilla eri käyttötapauksia, kuten tiedostojen avaamista ja tallentamista, lomakkeiden syöttämistä, sekä eri painikkeista tapahtuvia toimintoja. (Kasurinen 2013, 65-66.)

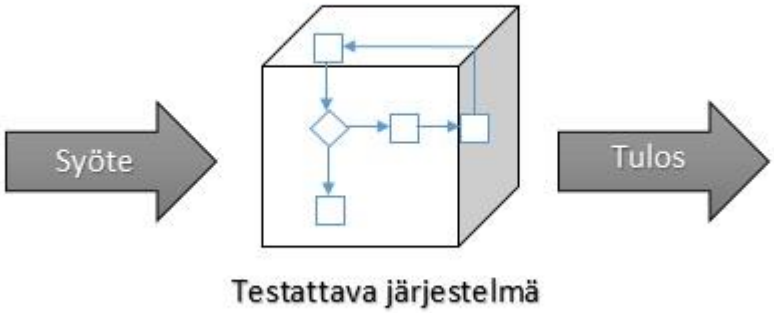
Musta laatikko -testausta voidaan suorittaa lähes missä tahansa työvaiheessa, edellyttäen, että olemassa on jo jotain toimintoja suorittava ohjelma (Kasurinen 2013, 65-66).

Kuva 5. Musta laatikko -testaus
(Rongala 2015).

3.2 Lasilaatikkotestaus

Musta laatikko -testauksesta poiketen lasilaatikkotestauksessa tarkastellaan syötteiden ja tulosten lisäksi myös järjestelmän sisäisiä toimintoja testauksen aikana. Tässä testausmenetelmässä ohjelma on kuin läpinäkyvä laatikko, jonka sisäisiä toimintoja voidaan tarkastella ja arvioida osana testausta (Kuva 6). Testaaja antaa ohjelmalle syötteitä ja näkee, kuinka kutakin syötettä käsitellään järjestelmän sisällä. Tästä johtuen mahdolliset virhetilanteet voidaan jäljittää lähdekooditasolle asti, jolloin saadaan varmistettua, että ohjelma todella toimii oikein, eivätkä tulokset ole sattumia. (Kasurinen 2013, 67-68.)

Lasilaatikkotestaus on siis musta laatikko -testauksen täydellisempi testausmuoto ja vaatii testaajalta enemmän tietämystä. Testaajan tulee tuntea järjestelmä ja sen logiikka tarpeeksi hyvin voidakseen olla varma ohjelmiston toimivuudesta ja saatujen tulosten oikeellisuudesta. Lähdekooditasolle asti pääseminen vaatii testaajalta myös tietämystä ohjelmoinnista. (Kasurinen 2013, 67-68.)



Kuva 6. Lasilaatikkotestaus
(QaQuench 2016).

3.3 Harmaa laatikko -testaus

Harmaa laatikko -testaus on yhdistelmä musta laatikko- ja lasilaatikkotestauksen parhaista puolista. Musta laatikko -testauksessa testaaja ei näe järjestelmän sisäisiä toimintoja, kun taas lasilaatikkotestauksessa järjestelmän sisäiset toiminnat ovat nähtävillä ja osana testausprosessia. Harmaa laatikko -testauksessa taas kyseiset toiminnot ovat osittain testaajan nähtävillä testitapausten suunnittelua varten. (Kasurinen 2013, 68.)

Harmaa laatikko -testaus sopii tilanteisiin, joissa järjestelmää ei lasilaatikkotestauksella voida kokonaan testata, mutta joissa kuitenkin vaaditaan pääsy joihinkin järjestelmän komponentteihin. Tällaisia ovat esimerkiksi useat verkkopalvelut, joissa oma järjestelmä voidaan testata lasilaatikkona, mutta sen alla olevat ulkoiset palvelut ja rajapinnat voidaan testata ainoastaan musta laatikko -testauksena. (Kasurinen 2013, 68.)

3.4 Regressiotestaus

Regressiotestaus ei itsessään ole oma testausmenetelmänsä vaan termillä tarkoitetaan uudelleentestaamista. Se on siis yleisnimitys millä tahansa testausmenetelmällä suoritetulle testaukselle, jota suoritetaan useaan otteeseen. Periaate on siis

se, että järjestelmään tehtyjen muutosten jälkeen testataan, että järjestelmä yhä toimii halutulla tavalla. Mikäli jotain järjestelmän osaa muutetaan, suhtaudutaan siihen kuin koko järjestelmä olisi rakennettu uudelleen. Regressiotestauksella halutaan varmistaa, että tehtyjen muutosten jälkeen mitään ei ole hajonnut, eivätkä aiemmin korjatut ongelmat toistu. (Kasurinen 2013, 68-69.)

4 TESTAUSMENETELMÄT ENNEN JULKAISUA

4.1 Käytettävyystestaus

Käytettävyystestauksessa testataan pääsääntöisesti järjestelmän käyttöliittymää ja sen toimivuutta. Perusideana on varmistaa, että tehty käyttöliittymä toimii oikein ja toivotulla tavalla. Käyttöliittymäsuunnitelmien pohjalta voidaan rakentaa yksinkertaisia prototyyppejä ilman muuten toimivaa järjestelmää, jolloin käyttöliittymää voidaan testata jo hyvinkin varhaisessa vaiheessa. (Kasurinen 2013, 70.)

Etenkin ulkopuolisen tekemänä käytettävyystestauksella voidaan löytää huomattavasti paremmin käytettävyyteen liittyviä vikoja ja ongelmia, kuin itse testaamalla. Itse järjestelmän rakentajalle oma tuote on jo niin tuttu, että voi olla hyvinkin haastavaa osata arvioida järjestelmän toimivuutta ulkopuolisen käyttäjän silmin. (Testing Lab [Viitattu 21.4.2016].)

Käytettävyystestaukseen on useita eri menetelmiä, kuten käyttökokeilu, haastattelututkimukset, asiantuntijatestit ja kohdekäyttäjäryhmällä tehty tutkiva testaus. Käytettävyystestauksissa voidaan myös käyttää erilaisia työkaluja, joiden avulla voidaan selvittää sellaisia ongelmia, joita käyttäjä ei välttämättä itse muista tai halua kyselyissä mainita. Tällaiset ongelmat käyvät ilmi testaajan tahattomista ilmeistä ja eleistä, joita voidaan seurata esimerkiksi nauhoituksien ja katseenseurantamonitorien avulla. (Kasurinen 2013, 70.)

4.2 Kuormitustestaus

Kuormitustestauksessa jo nimensäkin mukaan on tarkoituksena kuormittaa järjestelmää ja selvittää näin, mihin kaikkeen se oikeastaan pystyy ja mitä rajoitteita sillä on. Kuormitustestauksessa pyritään simuloimaan järjestelmän normaalikäyttöä, jotta nähdään, miten se siitä suoriutuu. Tavoitteena on selvittää, miten järjestelmä sietää kuormaa, sekä paikantaa järjestelmän ongelmia aiheuttavat kohdat, joita myös pullonkauloiksi kutsutaan. (Kasurinen 2013, 71.)

Kuormitustestauksen tarkka suoritustapa riippuu testattavasta järjestelmästä. Siinä kuitenkin toimitaan käytännössä suorittamalla järjestelmän eri toimintoja yhtä aikaa ja toistuvasti, useamman käyttäjän toimesta. Esimerkiksi useita käyttäjiä samanaikaisesti palvelemaan tarkoitetun verkkosivun kuormitustestauksessa voidaan sen normaalikäyttöä simuloida luomalla tarkoitukseen riittävän määrän virtuaalikäyttäjiä, joista jokainen suorittaa jotain tiettyä sivun toimintoa. Toiset voivat avata ja sulkea sivua, toiset selailla sitä ja niin edelleen. (Kasurinen 2013, 71.)

Kuormitustestauksesta on olemassa vielä raskaampi muoto, rasitustestaus, joka toimii muuten samoin kuin kuormitustestaus, mutta siinä kuormitus on raskaampaa. Siinä pyritään löytämään ongelmia, joita ei normaalikäyttöä simuloivassa kuormitustestauksessa löydy. Kuorman, esimerkiksi virtuaalikäyttäjien, määrää kasvatetaan entisestään ja tavoitteena on näin saada asetettua järjestelmälle myös maksimirajat sen käyttöön. (Kasurinen 2013, 71-72.)

4.3 Alfa- ja beetatestaus

Alfa- ja beetatestaukset ovat testausvaiheita, joita suoritetaan itsenäisinä ohjelmapaketteina myytäville tuotteille, esimerkiksi toimisto-ohjelmistoille tai tietokonepeleille (Kasurinen 2013, 73).

Alfatestauksessa testiryhmä simuloi oikeaa käyttäjäympäristöä suorittamalla tehtäviä, joita loppukäyttäjä tekisi. Testauksen tarkoituksena on löytää ja korjata mahdollisia virheitä, joita muissa testauksissa ei ole löydetty (Techopedia 2016a.)

Alfatestausvaiheessa järjestelmä on kokonainen ja sisältää jo kaikki siihen suunnitellut ominaisuudet, mutta vaatii vielä muun muassa eri toimintojen virheettömyyden tarkastamista ja käyttöliittymän viimeistelyä. Kun todetaan, että järjestelmä on riittävän toimiva, voidaan edetä suuremman testiyleisön suorittamaan beetatestaukseen. (Kasurinen 2013, 73.)

Beetatestaus on viimeinen testausvaihe ennen tuotteen julkaisua. Siinä testataan sekä tuotteen toimivuutta, että mahdollisten asiakkaiden kiinnostusta tuotteeseen. Mikäli tuote ei kerää riittävästi kiinnostusta beetavaiheessa, voidaan siihen tehdä

vielä tarvittavia muutoksia tai tarpeen tullen hylätä koko tuote ilman kalliita jälkiseurauksia. (Kasurinen 2013, 73.)

Beetatestauksen suorittaa joukko potentiaalisia asiakkaita, jotka hoitavat testauksen kotoaan käsin. Testaajat kokeilevat tuotetta normaaleissa olosuhteissa ja ilmoittavat tuotteen kehittäjille löytämistään vioista ja epäkohdista ennen tuotteen julkistamista. Ennen beetatestausvaihetta suurin osa virheistä ja ongelmista pitäisi jo olla hoidettuna ja jäljellä olla ainoastaan pienimmät virheet, sillä tässä vaiheessa suurimpia virheitä voi olla lähes mahdotonta korjata. (Techopedia 2016b.)

5 AJANHALLINTA- JA RAPORTOINTITYÖKALUN TESTAUS

5.1 Applikaation kuvaus

Testattava applikaatio on selainpohjainen ajanhallinta- ja raportointityökalu, joka tarjoaa käyttäjälle monenlaista graafista materiaalia raportoinnin ja ajanhallinnan tueksi. Kyseessä on Wapice Oy:n sisäiseen käyttöön tarkoitettu ohjelmisto, jonka avulla voidaan tarkastella yrityksen meneillä olevia projekteja, kunkin projektin osallisia, sekä työtuntien määriä kutakin projektia kohden. Kaavioiden tarkasteltavia tietoja voidaan kaaviotyypistä riippuen rajoittaa asiakkaan, projektin, projektin aikavälin, laskutettavuuden, henkilön, segmentin tai tiimin mukaan. Tuotetut kaaviot voidaan tallentaa tietokoneelle Excel- tai PDF-tiedostoina.

Applikaatiossa kaavioita voidaan tarkastella kuudessa eri näkymässä, joista jokainen on omalla välilehdellään. Ensimmäisellä kahdella välilehdellä voidaan tarkastella päivittäisiä työtunteja viivakaavion avulla. Toisella näitä tarkastellaan kumulatiivisesti. Muissa näkymissä voidaan työtunteja tarkastella lämpökarttana, sankey-diagrammina, histogrammina sekä puukarttana. Eri näkymät tarjoavat eri tapoja rajoittaa tuotettavan kaavion sisältöä. Joissakin tuloksia voidaan tarkentaa esimerkiksi projektin laskutettavuuden mukaan, kun taas toisissa esimerkiksi henkilön tai tiimin mukaan.

5.2 Testauksen suunnittelu

Ajanhallinta- ja raportointityökalulle ei ole aiemmin suoritettu testausta muuten kuin kehittäjien itsensä toimesta. Seuraavaksi suoritettava testaus on alfatestausta, jonka tarkoituksena on käyttää tuotettua raportointityökalua sen loppukäyttäjän tavoin.

Testauksesta laadittiin kirjallinen suunnitelma, josta ilmenee testitapaukset kullekin testattavalle osiolle. Kussakin testitapauksessa määriteltiin, mitä osiota testataan ja miten, sekä millaisia tuloksia ohjelmalta odotetaan kyseistä osiota testatessa. Tes-

taus päädyttiin suorittamaan Google Chrome -selainta käyttäen, mutta tuloksia verrattaisiin myös muilla selaimilla – Internet Explorerilla ja Mozilla Firefoxilla – suoritettujen testien tuloksiin. Kullekin osiolle määriteltiin kolme testitapausta, joista ensimmäinen käsitteli kaavion luontia ja näkymää, toinen virhetilanteita ja kolmas datan tallentamista tietokoneelle. Ensimmäisessä testitapauksessa siis tarkastellaan, luoko ohjelma halutun kaavion, sekä näkyykö ja toimiiko se tarkoitetulla tavalla. Toisessa testitapauksessa ohjelmalle on tarkoitus luoda jonkinlainen virhetilanne, jotta nähtäisiin miten se reagoi virheisiin ja miten se niistä käyttäjälle ilmoittaa. Kolmannessa testataan, toimiiko kaavioiden tallentaminen koneelle ja saadaanko ne tallennuksen jälkeen vielä auki ohjelman ulkopuolelta. Kunkin testitapauksen määrittelemää osiota testataan vähintään viisi kertaa ja erilaisia kriteereitä käyttäen.

5.2.1 Testitapaus 1: Kaavioiden tulostus

Päivittäisten työtuntien näkymässä menetellään valitsemalla projekti ja aikaväli, jolla tunteja halutaan tarkastella. Odotettavissa on, että ohjelma luo kriteereiden mukaisen viivakaavion, jota voidaan jälkikäteen tarkentaa työtuntien tyypin mukaan. Työtuntien tyypiksi voidaan valita kokonaissumma tai niitä voidaan tarkastella sen mukaan, ovatko ne laskutettavissa asiakkaalta.

Kumulatiivisesti esitettävien päivittäisten työtuntien näkymässä applikaation tulisi toimia muuten samoin kuin ensimmäisessä näkymässä, mutta viivakaaviossa esitettävien työtuntien tulisi näkyä kumulatiivisesti eli kasautuvasti.

Kolmannella välilehdellä saatavilla olevassa lämpökarttanäkymässä käyttäjä voi päivämäärävalintojen lisäksi valita, tarkastellaanko koko yrityksen työtunteja, vai tehdäänkö rajaus henkilön, segmentin tai tiimin mukaan. Applikaation tulisi luoda näkymään lämpökarttakaavio käytettyjen kriteereiden mukaan.

Seuraavassa näkymässä applikaation tulisi luoda sankey-diagrammi sen mukaan, minkä asiakkaan, projektin ja aikavälin käyttäjä on valinnut. Diagrammista tulisi olla nähtävissä erikseen myös asiakkaalta laskutettavat ja ei-laskutettavat työtunnit.

Histogramminäkymässä voidaan tulostaa histogrammi valitun projektin ja päivämäärien mukaan. Applikaation tulisi luoda kaavio, josta ilmenee tehdyt työtunnit kyseisen projektin parissa, kyseisen aikavälin aikana. Jälkikäteen tuloksia kuuluisi saada tarkennettua laskutettavuuden mukaan.

Viimeisessä näkymässä valitaan asiakas ja päivämäärät, joiden välillä tehdyistä työtunneista applikaation kuuluisi luoda kaiken kattava puukartta. Puukartasta tulisi nähdä hierarkkisesti kaikki kriteereiden mukainen data, kuten työtunnit ja laskutettavuus kyseisessä projektissa.

5.2.2 Testitapaus 2: Virhetilanteen luominen

Päivittäisten työtuntien näkymässä voidaan yrittää luoda virhetilanne esimerkiksi valitsemalla käsittelyyn projekti, jonka parissa ei ole valittuna aikana työskennelty. Lisäksi voidaan selvittää, miten applikaatio reagoi, jos sille ei anna kaikkia vaadittavia kriteereitä kaavion luomiseen. Näitä tullaan testaamaan käyttäen eri projekti- ja päivämäärävalintoja, jotta nähdään mahdolliset eroavaisuudet eri projektien käytön välillä.

Kumulatiivisesti esitettävien päivittäisten työtuntien näkymässä virhetilanteet luodaan samalla lailla kuin päivittäisten työtuntienkin näkymässä, sillä näissä on valittavissa samat kriteerit, vaikka saatava kaavio poikkeaakin toisesta.

Lämpökarttanäkymässä voidaan yrittää saada virheilmoituksia jättämällä valitsematta eri kriteereitä. Näin voidaan selvittää, mitkä kentät ovat pakollisia, missä tilanteissa virheilmoituksia saadaan ja millaisia tuloksia ylipäätään saadaan. Odotettavissa olisi saada jotain virheilmoituksia ainakin riittämättömistä kriteerien käytöistä.

Sankey-diagrammeja tulostavassa osiossa kokeillaan myös syöttää vajaata informaatiota applikaatiolle siitä, mitä sen haluttaisiin tulostavan. Tätä voidaan testata valitsemalla yksi kriteeri kerrallaan, jättäen muut kentät koskemattomiksi. Vajaista kriteerivalinnoista tulisi saada jonkinnäköistä virheilmoitusta.

Histogrammeja tulostavassa näkymässä menetellään samoin kuin kahdessa edellisessäkin. Applikaatiolle annetaan riittämättömästi kriteereitä esimerkiksi valitsemalla ainoastaan yksi kriteeri. Tarkoituksena on selvittää, millaisia virheilmoituksia näin menetellen saadaan. Ohjelman tulisi ilmoittaa virheestä, kun käytettäviä kriteereitä ei ole valittu tarpeeksi.

Myös puukarttanäkymässä voidaan hakea virhetilanteita vajaasti käytetyillä kriteereillä. Sen tuloksissa on kuitenkin odotettavissa poikkeuksia, sillä se näyttää laajemmin kaiken datan, kuin muiden näkymien kaaviot.

5.2.3 Testitapaus 3: Datan tallennus tietokoneelle

Kolmannessa testitapauksessa testataan datan tallennusta tietokoneelle kaikissa näkymissä, joissa kyseinen toiminto on mahdollistettu. Näihin kuuluvat kaikki muut, paitsi sankey- ja puukarttakaavioita tarjoavat näkymät. Testauksessa selvitetään, toimiiko sekä Excel- että PDF-tiedostojen tallennus normaalisti painikkeiden avulla ja avautuuko tallennettu kaavio tietokoneen kautta odotetusti. Lisäksi testataan, miten tallennus onnistuu eri kriteerivalinnoilla ja mitä tapahtuu, kun yritetään tallentaa kaaviota, jossa ei ole lainkaan dataa.

5.3 Testauksen toteutus ja tulokset

Testaus toteutettiin vaihe vaiheelta kirjallisen suunnitelman mukaan ja tulokset kirjattiin samaan raporttiin. Raporttiin lisättiin myös kuvakaappauksia saaduista virheilmoituksista. Testauksiin valittiin viisi tiettyä projektia, joita käytettiin jokaisessa projekti-kriteerin mahdollistavassa osiossa. Samoja projekteja käytettäessä voitiin helpommin nähdä, ovatko kunkin näkymän tuottamien kaavioiden arvot oikein. Raportin liitteeksi lisättiin Excel-tiedosto, josta voidaan visuaalisesti nähdä tallennusosion testauksen tuloksia kussakin näkymässä.

5.3.1 Viivakaavionäkymät

Molemmissa viivakaavionäkymissä ohjelma tuottaa viivakaavion käytettyjen kriteereiden mukaan odotetusti. Kun kaavion arvoja tarkentaa laskutettavuuden mukaan, kaavio muuttuu kyseisesti.

Kun ensimmäisessä näkymässä (Kuva 7) tarkasteluun valitsee ainoastaan projektin ilman päivämäärävalintoja, sivu näyttää lataavan jonkun aikaa, kuitenkin tuottaen viimein kaavion virheilmoituksen sijaan. Kun taas päivä on valittuna ja projekti ei, ohjelma tuottaa viivakaavion jokaisesta projektista, jolle löytyy työtunteja kyseisenä aikana. Kun jättää kaikki kohdat valitsematta, sivu näyttää lataavan hetken, kuitenkin tuottamatta yhtään kaavioita. Virheilmoitus saadaan vain valittaessa projekti, jolle ei työtunteja ole valittuna aikana. Tätä testattaessa käytettiin sattumanvaraisesti valittuja projekteja ja päivämäärävalinnoissa käytettiin sekä pikavalintaa että kustomoituja päivämäärävalintoja.

Kuva 7. Päivittäisten työtuntien viivakaavionäkymä

Päivittäisiä työtunteja kumulatiivisesti esittävässä näkymässä tulokset poikkeavat hieman ensimmäisen näkymän tuloksista (Kuva 8). Kun tässä tarkasteluun valitaan ainoastaan projekti, mutta ei päivää, ilmaantuu sovellusvirheilmoitus. Samainen virheilmoitus ilmaantuu myös, kun jätetään käyttämättä mitään kriteereitä. Valittaessa päivämäärät ilman projektivalintaa ohjelma tuottaa kaavion, josta löytyy jokainen projekti, jolle työtunteja löytyy valittuna aikana. Käyttäessä projektia, jolle työtunteja ei ole valittuina päivinä, ohjelma ilmoittaa, että dataa ei löytynyt.

Kuva 8. Viivakaavionäkymä kumulatiivisesti esitettävistä työtunneista

Ensimmäisessä näkymässä kaavion tallennus tapahtuu odotetusti. Kun valitaan projekti ja päivämäärä, jolle kyseisellä projektilla ei ole työtunteja, sekä painetaan tallennuspainiketta, saadaan virheilmoitus. Sama virheilmoitus tulee riippumatta siitä, painetaanko kaavion tulostuspainiketta ensin vai ei. Painikkeen klikkausta ei myöskään vaadita päivämäärävalinnan muutoksissa, vaan arvo päivittyy kaavioon suoraan. Päivitetty kaavio saadaan tallennettua tietokoneelle, vaikka se ei olisi päivittynyt vielä itse näkymään.

Päivittäisiä työtunteja kumulatiivisesti esittävässä näkymässä saadaan myös virheilmoitus, mikäli valitulle projektille ei ole työtunteja valittuna aikana. Kun dataa ei ole, sitä ei myöskään voida tallentaa Excel-tiedostona tietokoneelle. Kuitenkin yritettäessä tallentaa kaaviota PDF-tiedostona, tyhjä kaavio tallentuu tietokoneelle. Tallenna Excel-tiedostona -painike ei toimi tässä näkymässä lainkaan, lukuun ottamatta puutteellisina tulostuneita kaavioita.

5.3.2 Lämpökarttanäkymä

Lämpökarttanäkymässä ohjelma tuottaa lämpökartan käytettyjen kriteereiden mukaisesti. Kun valitulle henkilölle, segmentille tai tiimille ei löydy dataa valittujen päivämäärien ajalle, ohjelma antaa siitä virheilmoituksen. Henkilöitä, segmenttejä ja tiimejä valittiin 2–3 kappaletta, joita käytettiin tämän osion testauksessa. Nämä valittiin listasta sattumanvaraisesti ja kaikki testattiin käyttäen eri päivämäärävalintoja.

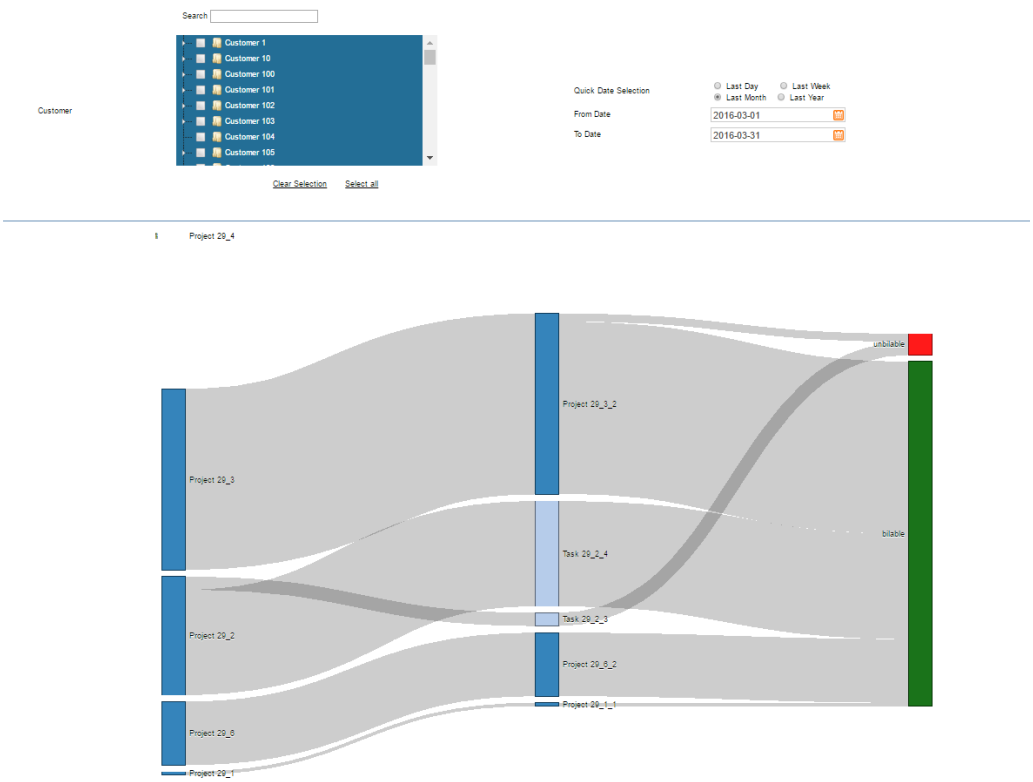
Tässä näkymässä ohjelma tuottaa lähes aina jonkun lämpökartan, vaikka annetut kriteerit olisivatkin riittämättömät. Kun tarkasteluun valitaan kategoriaksi koko yritys, henkilö, segmentti tai tiimi, mutta ei tarkennusta sille, virheilmoituksia ei tule, vaan ohjelma tuottaa aina saman lämpökartan vuosilta 2013–2015. Sama lämpökartta näkyy myös, kun jätetään valitsematta mitään kriteereitä. Virheilmoitus saadaan valittaessa kategoriaksi segmentti ja sille listasta tarkasteltava kohde. Kun sivua ladataan liian kauan palvelimen vastaamatta, saadaan sovellusvirheilmoitus. Joissain tilanteissa valittaessa kategoria ja sen tarkempi kohde, ohjelma tuottaa lämpökartan vuodesta 2013 nykypäivään asti valittujen kriteereiden mukaan.

Tallennuksen suhteen kaikilla kriteereillä tulokset ovat samat tässä näkymässä. Kaikki saatu data voidaan tallentaa sekä Excel- että PDF-tiedostona, vaikka mitään tallennettavaa ei olisikaan. Kuitenkin kun tiedostot avataan tietokoneella, ne eivät sisällä juuri mitään. PDF-tiedostot ovat kokonaan tyhjiä ja Excel-tiedostoa avattaessa saadaan ilmoitus mahdollisesti korruptoituneesta tiedostosta.

5.3.3 Sankey-näkymä

Ohjelma tuottaa odotetusti käytettyjen kriteereiden mukaisen sankey-diagrammin, jossa on liikuteltavia osia kaavion tarkempaan tutkailuun (Kuva 9). Nämä osat kuvastavat asiakasta, projektia sekä laskutettavien ja ei-laskutettavien työtuntien jakoa. Pidettäessä hiiren kursoria jonkun osan päällä kaavio näyttää työtuntien määrät kyseisen osan suhteen. Kun valitulle projektille ei löydy dataa valittujen päivämäärien sisällä, ohjelma tuottaa tyhjän kaavion virheilmoituksen sijaan.

Tässä näkymässä kaavion tulostuksen painiketta ei voida klikata, mikäli mitään ei ole valittuna tai vain päivämäärä on valittu. Kun ainoastaan asiakas on valittuna, kaaviossa näkyy kaikki valitulle asiakkaalle kuuluvat projektit. Muissa tapauksissa ohjelma tuottaa aina jonkinlaisen kaavion tai tyhjän sellaisen, mutta virheilmoituksia ei saada. Kuitenkin erään tietyn projektin kohdalla ohjelma tuottaa virheellisen kaavion, ja erästä toista tiettyä projektia käytettäessä ohjelma kaatuu.



Kuva 9. Sankey-näkymä

5.3.4 Histogramminäkymä

Pylväskaavioita tuottavassa histogramminäkymässä ohjelma antaa virheilmoituksen, jonka jälkeen se vasta tuottaa kaavion. Sama virheilmoitus näkyy myös valittaessa tuntityyppiä. Mikäli valittu projekti ei sisällä dataa kyseisinä päivämäärinä, ohjelma tuottaa tyhjän kaavion.

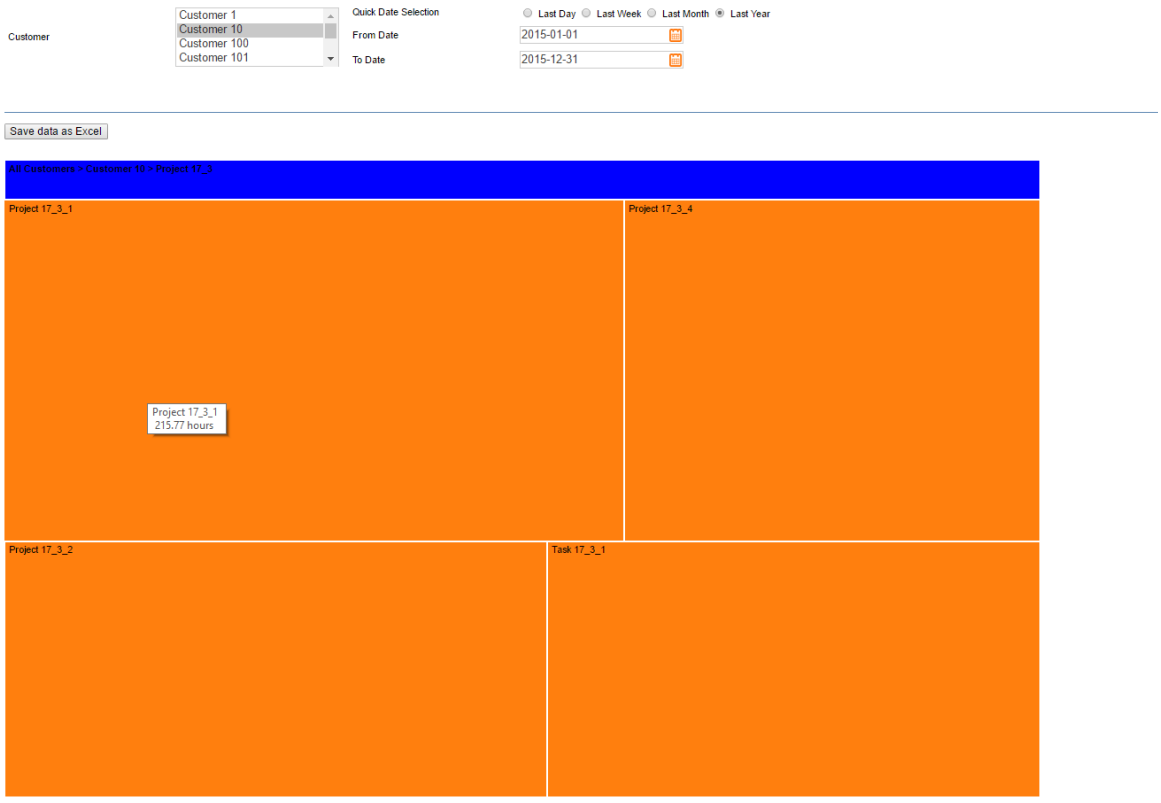
Kun tarkasteluun valitaan projekti ilman muita kriteereitä, saadaan virheilmoitus. Valittaessa ainoastaan päivämäärä ohjelma luo pitkän histogrammin, joka sisältää kaikki projektit, joista dataa löytyy kyseisinä päivinä. Tarkentamalla kaaviota tuntityypin mukaan eli valitsemalla laskutettavat tai ei-laskutettavat työtunnit, kaavio muuttuu siten, että valitun tuntityypin mukaiset projektit näkyvät kaavion yläosassa. Pylväiden värit indikoivat laskutettavia ja ei-laskutettavia tunteja.

Kun valitulla projektilla ei valittuina päivinä ole työtunteja ja sitä yritetään tallentaa Excel- tai PDF-tiedostona, saadaan virheilmoitus. Sama virheilmoitus saadaan, mikäli dataa on ja sitä yritetään tallentaa Excel-tiedostona. PDF-muodossa tiedosto saadaan tallennettua tietokoneelle, mutta tuloksena on hieman vajaa kaavio, josta puuttuu muun muassa numeerisia arvoja, sekä muita nimikkeitä.

5.3.5 Puukarttanäkymä

Viimeisessä näkymässä ohjelma tuottaa puukartan, joka näyttää hierarkkisesti kaiken datan, mitä valitun asiakkaan alla on (Kuva 10). Pidettäessä hiiren kursoria tietyn ruudun päällä puukartassa, kaavio näyttää arvoja käytetyistä työtunneista kyseisen osion suhteen. Kaikkia asiakkaita ja projekteja voidaan selailla puukartassa klikkailemalla sen ruutuja.

Valittaessa tarkasteluun ainoastaan asiakas puukartassa näkyvät kaikki projektit kyseiseltä asiakkaalta. Kun taas valitaan vain päivämäärä, kaavio näyttää kaikki asiakkaat, joille dataa löytyy kyseisenä aikana. Mikäli mitään ei valita, puukartassa näkyvät kaikki asiakkaat ja heidän datansa.



Kuva 10. Puukarttanäkymä

6 YHTEENVETO JA POHDINTAA

Yhteenvetona todettakoon, että testaustyö on tärkeä osa ohjelmistotuotantoa. Sillä varmistetaan tuotteen laatu, joka parantaa myös yrityksen menestystä. Testattu ajanhallinta- ja raportointityökalu toimii yleiskatsauksella jo hyvin, mutta lähemmin tarkasteltuna joidenkin näkymien toiminnoissa oli hieman parantamisen varaa. Ohjelmalla on kuitenkin potentiaalia näppäräksi ja helppokäyttöiseksi avuksi muun muassa raportointiin tai työtuntien seurantaan. Käytettävyyden kannalta tarkasteltuna käyttöliittymää oli helppo ymmärtää ja kaikki toiminnot olivat selkeästi käden ulottuvilla. Navigointi oli yksinkertaista ja tuotetut kaaviot selkeästi tarkasteltavissa.

Testaustyö onnistui ilman suurempia ongelmia ja toimeksiantajalle saatiin kirjallinen raportti testauksen etenemisestä, testausmenetelmistä ja tuloksista. Näitä tietoja hyödyntäen ohjelmaa voidaan kehittää ja korjailla, sekä tunnistetaan ongelmalliset kohdat ja ominaisuudet. Testaustyöhön kului aikaa, sillä testattavia näkymiä, toimintoja ja testaustapoja oli useita, ja niistä jokainen testattiin useampaan otteeseen tulosten varmentamiseksi. Näin kuitenkin saatiin mahdollisimman perusteellinen testaustyö suoritettua. Sen tuloksia voidaan hyödyntää tulevaisuudessa.

Suoritettu testaustyö oli erittäin mielenkiintoista, mutta samalla myös opettavaista. Siinä oppi, mitä kaikkea oikeastaan vaaditaankaan ennen testauksen suorittamista ja mihin erityisesti täytyy kiinnittää huomiota. Itse testauksen kulku ja siitä raportoiminen tuli tutuksi sekä perehdyttäjän toimesta, että yrityksen testiraporttipohjasta, johon oli selvästi merkitty, mihin kirjataan mitäkin.

Opinnäytetyöraporttia kirjoittaessa pääsi tutustumaan tarkemmin erilaisiin testausmenetelmiin ja siihen, miten testauksen rooli vaihtelee erilaisissa ohjelmistotuotannon malleissa. Tällä saatiin vielä enemmän lisättyä tietoa ja mielenkiintoa testaustyöhön sekä erilaisten testausmenetelmien kokeiluun tulevaisuudessa.

LÄHTEET

James, M. Ei päiväystä. An Empirical Framework For Learning (Not a Methodology). [Verkkosivu]. Scrum Methodology. [Viitattu 22.5.2016]. Saatavana: http://scrummethodology.com/

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. 1. p. Jyväskylä: Docendo Oy.

Lehtimäki, T. 2006. Ohjelmistoprojektit käytännössä. 1.p. Jyväskylä: Readme.fi Oy.

QaQuench. 2016. Unit and white box testing. [Verkkosivu]. QaQuench. [Viitattu 11.10.2016]. Saatavana: http://www.qaquench.com/testing-services/unit-and-white-box-testing/

Rational Software. 1998. Rational Unified Process: Best Practices for Software Development Teams. [PDF-dokumentti]. Rational Software Corporation. [Viitattu 20.4.2016]. Saatavana: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Rongala, A. 9.5.2015. What is Black Box Testing: Advantages and Disadvantages. [Verkkosivu]. Invensis Technologies. [Viitattu 11.10.2016]. Saatavana: https://www.invensis.net/blog/it/black-box-testing-advantages-disadvantages

Schwaber, K. & Sutherland, J. 2014. The Scrum Guide. [PDF-dokumentti]. Scrum.org & ScrumInc. [Viitattu 22.5.2016]. Saatavana: http://www.scrum-guides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100

Techopedia. 2016a. Alpha Test. [Verkkosivu]. Techopedia™. [Viitattu 20.4.2016]. Saatavana: https://www.techopedia.com/definition/5935/alpha-test

Techopedia. 2016b. Beta Test. [Verkkosivu]. Techopedia™. [Viitattu 20.4.2016]. Saatavana: https://www.techopedia.com/definition/27136/beta-test-gaming

Testing Lab. Ei päiväystä. Käytettävyystestaus. [Verkkosivu]. Testing Lab. [Viitattu 21.4.2016]. Saatavana: http://www.testinglab.fi/palvelut/kaytettavyystestaus/

Tolvanen, P. 6.6.2013. Ketteryys haltuun: Scrum pähkinänkuoressa. [Verkkojulkaisu]. Sininen Meteoriitti. [Viitattu 24.5.2016]. Saatavana: http://www.meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/

Tutorials Point. Ei päiväystä. SDLC – Waterfall model. [Verkkosivu]. Tutorials Point. [Viitattu 13.4.2016]. Saatavana: http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Wapice Oy. Ei päiväystä. Tietoa Wapicesta. [Verkkosivu]. Vaasa: Wapice Oy. [Viitattu 11.5.2016]. Saatavana: https://www.wapice.com/fi/tietoa-meista/tietoa-wapicesta