SATAKUNNAN AMMATTIKORKEAKOULU

Antti Haavisto

TEST MANAGEMENT TOOL-SOVELLUS PROJEKTI

Liiketalous ja tietojenkäsittely Huittinen
Tietojenkäsittely
Tietoverkkopalveluiden suuntautumisvaihtoehto
2008

TEST MANAGEMENT TOOL-SOVELLUS PROJEKTI

Haavisto Antti
Satakunnan ammattikorkeakoulu
Liiketalous ja tietojenkäsittely, Huittinen
Tietojenkäsittelyn koulutusohjelma
Tietoverkkopalvelujen suuntautumisvaihtoehto
Joulukuu 2008
Pekka Kuisma
UDK: 004.41
Sivumäärä: 51

Asiasanat: Testaus, Scrum, ohjelmistokehitys

Tämän opinnäytetyön tarkoituksena oli luoda testausdokumenttien hallintasovellus, joka tulisi palvelemaan Descom Oy:n tarpeita. Sovellus kehitettiin Lotus Notes –ohjelmistokehitysalustalle ja se on käytössä asiakkaan sisäisessä verkossa.

Descom Oy on Jyväskylässä sijaitseva, vuonna 1997 perustettu ohjelmistoyritys, jossa työskentelee noin 60 asiantuntijaa. Yrityksen ydinosaamisalueena on portaaliratkaisut IBM Websphere Portal –alustalla, Lotus Notes/Domino –ratkaisut, asiakastukipalvelut kahdelle edellä mainituille ja Java EE –pohjaisille sovelluksille ja IBM ohjelmistolisenssipalvelut.

Opinnäytetyön teoriaosa käsittelee kahta erilaista ohjelmistokehitysmenetelmää; struktuurista ohjelmistokehitysmenetelmää ja ketteriä menetelmiä. Teoriaosassa vertaillaan näitä kahta menetelmää ja syvennytään yhteen ketterään menetelmään; Scrumiin.

Opinnäytetyö oli projektiluonteinen, siinä luotiin toimiva sovellus. Työn empriisisessä osassa selvennetään Scrumia näyttämällä miten sitä olisi käytetty tässä projektissa. Empiirisessä osassa käsitellään myös tehtyä sovellusta ja esitellään siinä tehtävien prosessien kuvaus ja tietokantakuvaus. Työ huipentuu esimerkkiprosessiin, jossa kuvataan kohta kohdalta sovelluksen eri osa-alueet.

TEST MANAGEMENT TOOL - PROJECT

Haavisto Antti
Satakunta University of Applied Sciences
The Faculty of Business and Culture Huittinen
Degree Programme in Business and Administration
Information Network Services
December 2008
Pekka Kuisma
UDC: 004.41
Number of pages: 51

Key words: testing, Scrum, software development

The purpose of this bachelor´s thesis was to create a test document manager. This software would serve Descom Inc.'s needs. The software was developed in Lotus Notes application and it is in use in the client's internal network.

The client, Descom Inc. locates in Jyväskylä. The software company was established in 1997 and there are about 60 experts. The core competence of the company is portal solutions in IBM Websphere Portal, Lotus Notes/Domino –solutions, client support for the two above-mentioned ones and Java EE applications and IBM licensing services.

The theory section of the study contains two software development methods; structured development method and agile methods. There is a comparison of these methods and then is focused in one of the agile methods; Scrum.

The study was project-minded, the purpose was to create functional software. In the empiric section of the study clarified Scrum by showing how it would be used in this project. It also proposes the program itself, the processes which would be done in the program and the program´s database schema. The study culminates in example process, in which all parts of the program are shown step by step.

SISÄLLYS

SYMBOLI- JA TERMILUETTELO 6

1 JOHDANTO 7

2 OPINNÄYTETYÖSUUNNITELMA 8

2.1 Ongelma 8
2.2 Rajaus 8
2.2 Materiaalin kerääminen 8

3 OHJELMISTOKEHITYSMENETELMÄT 9

3.1 Vesiputousmalli 9
3.1.1 Vesiputousmallin hyödyt 11
3.1.2 Vesiputousmallin haitat 11
3.2 Ketterät ohjelmistokehitysmenetelmät (Agile-methods) 12
3.2.1 Ketterien menetelmien hyödyt 14
3.2.2 Ketterien menetelmien haitat 14

4 SCRUM 15

4.1 Scrum roolit 17
4.2 Scrum tuotokset 18
4.3 Scrum tilaisuudet 19

5 OHJELMISTOTESTAUS 21

5.1 Perinteinen ohjelmistotestausmenetelmä 21
5.2 Ohjelmistotestaus Scrumissa 22

6 TEST MANAGEMENT TOOL –PROJEKTI SOVELLETTUNA SCRUM-MENETELMÄÄN 24

6.1 Roolit TMT -projektissa 24
6.2 Tuotokset TMT –projektissa 24
6.2.1 Tuotteen työlista 25
6.2.2 Sprintin työlista 26
6.2.3 Jäljellä olevan työmäärän kaavio 28

6.3 Scrum tilaisuudet TMT-projektissa 29
6.3.1 Sprintin suunnittelukokous 30
6.3.2 Sprintin päivittäiset tapaamiset 30
6.3.3 Sprintin katselmustilaisuus 30

7 TEST MANAGEMENT TOOL 31

7.1 Prosessin kuvaus 31
7.2 Tietokantakuvaus 33
7.3 Sovelluksen osa-alueet 34
7.3.1 Testisuunnitelma 34
7.3.2 Testitapaus 35
7.3.3 Testiajo 36
7.3.4 Testitapausajo 37
7.3.5 Valikkoasetus 38
7.3.6 Otsikko 38
7.3.7 Ulkoiset osa-alueet 38
7.4 Esimerkkejä sovelluksen käytöstä 39
7.4.1 Testisuunnitelma-lomake 40
7.4.2 Testitapaus-lomake 42
7.4.3 Testitapausajo-lomake 44
7.4.4 Testiajo-lomake 44
7.2.5 Asetukset-lomakkeet 45

8 YHTEENVETO 47

9 LOPPUTULOKSEN ARVIOINTI 47

LÄHDELUETTELO 48

SYMBOLI- JA TERMILUETTELO

Vaihejakomalli	Malli, jolla kuvataan ohjelmiston kehitystyötä tai koko elinkaarta ja niiden vaiheita.
ROI	Pääoman tuottoaste investoinnissa. (Return of investment).
Lotus Notes	IBM:n kehittämä ja ylläpitämä työryhmä- ja vistintäohjelmisto.
JIRA	Descom Oy:ssä käytettävä, Atlassian Software System:n ketterä tehtävänhallintajärjestelmä.

1 JOHDANTO

Kipinän tämän opinnäytetyön tekemiseen sain toukokuussa 2008, jolloin kävimme yritysvierailulla Jyväskylässä sijaitsevassa Descom Oy:ssä, joka tuottaa portaali-, internet- ja intranetpalveluja Lotus Notes/Websphere alustalle. Mukana olivat lehtori Pekka Kuisma, Lotus Notes-asiantuntija Antti Suonpää ja luokkatoverini Niko Roukkio.

Yritysvierailulla tuli ilmi, että heillä olisi tarvetta Lotus Notes –asiantuntijoille. Tästä alkoi sähköpostiviestittely, joka johti työhaastatteluun ja lopulta opinnäytetyön tekemiseen. Kerran aiheen vaihduttua päädyimme projektiluontoiseen ratkaisuun – Testauksenhallintasovelluksen rakentamiseen Descom Oy:n sisäiseen verkkoon.

Työn teoriaosassa arvioidaan ja vertaillaan kahta eri ohjelmistokehitysmenetelmää; struktuurista ohjelmistokehitysmenetelmään liittyvää vesiputousmallia ja ketteriä menetelmiä. Molemmista menetelmistä kaivetaan hyviä ja huonoja puolia esiin. Vertailun jälkeen syvennytään yhteen ketterään menetelmään, Scrumiin, josta esitellään eri osa-alueet.

Empiiriaosassa esitellään miten projekti olisi taipunut Scrum-menetelmään, näytetään siihen liittyviä dokumentteja ja taulukoita ja kerrotaan miten tapaamiset sujuivat. Loppusivuilla nähdään mistä osista tehty sovellus koostuu ja miten ne ovat yhteydessä toisiinsa.

Liite 1. näyttää sovellukselle asetetut vaatimukset.

2 OPINNÄYTETYÖSUUNNITELMA

2.1 Ongelma

Työn teoriaosassa tutkitaan eri ohjelmistokehitysmenetelmiä; struktuurista kehitysmenetelmää ja sitä kuvastavaa vesiputousmallia verrataan ketteriin menetelmiin, yksityiskohtaisemmin Scrum-menetelmään.

Työn empiirisessä osassa tehdään työn tilaajan, Descom Oy:n testauksiin liittyvien dokumenttien hallinta –sovellus joka käyttää nimeä Test Management Tool (TMT). Yrityksellä on tällä hetkellä työkalu, joihin he eivät ole täysin tyytyväisiä, joten he haluaisivat teettää itselleen juuri sopivan sovelluksen näiden hallintaan.

Projekti esitetään Scrum-projektin muodossa ja sovelluksesta dokumentoidaan tietokanta- ja testausprosessien kuvaukset.

2.2 Rajaus

Opinnäytetyössäni suunnittelen, luon ja testaan testausdokumenttien hallintasovelluksen (TMT). Teoriaosassa tutkin struktuurista ohjelmistokehitysmenelmää ja ketteriä ohjelmistokehitysmenetelmiä, joista pääpaino on Scrum-menetelmässä.

Ulkopuolelle on rajattu sovelluksen rakentamisessa käytetyt yksityiskohdat.

2.2 Materiaalin kerääminen

Käytetty materiaali koostuu pääosin internet-lähteistä; verkkoartikkeleista ja verkkokirjoista. Materiaalia olen kerännyt monista eri lähteistä tarkoituksena laaja näkemys käsitellyistä asioista. Suurin osa maateriaalista on englanninkielistä, tämä johtuu luonnollisesti siitä, että se on yleisesti käytetty kieli IT-alalla.

3 OHJELMISTOKEHITYSMENETELMÄT

Ohjelmistokehitysmenetelmiä ja niitä kuvaavia vaihejakomalleja on useita; evo-malli, ketterät menetelmät, kuten inkrementaalinen mallinnus, kollaboratiivinen-malli (Collaborative software development model) ja useita muita. (Benediktsson O., Dalcher D. ja Thorbergsson H. 2006. 1)

Ohjelmistotuotannossa on kuitenkin kaksi vedenjakajaa; struktuuriset menetelmät (vesiputousmalli) ja ketterät ohjelmistokehitysmenetelmät, joita käyn tässä työssäni tarkemmin läpi.

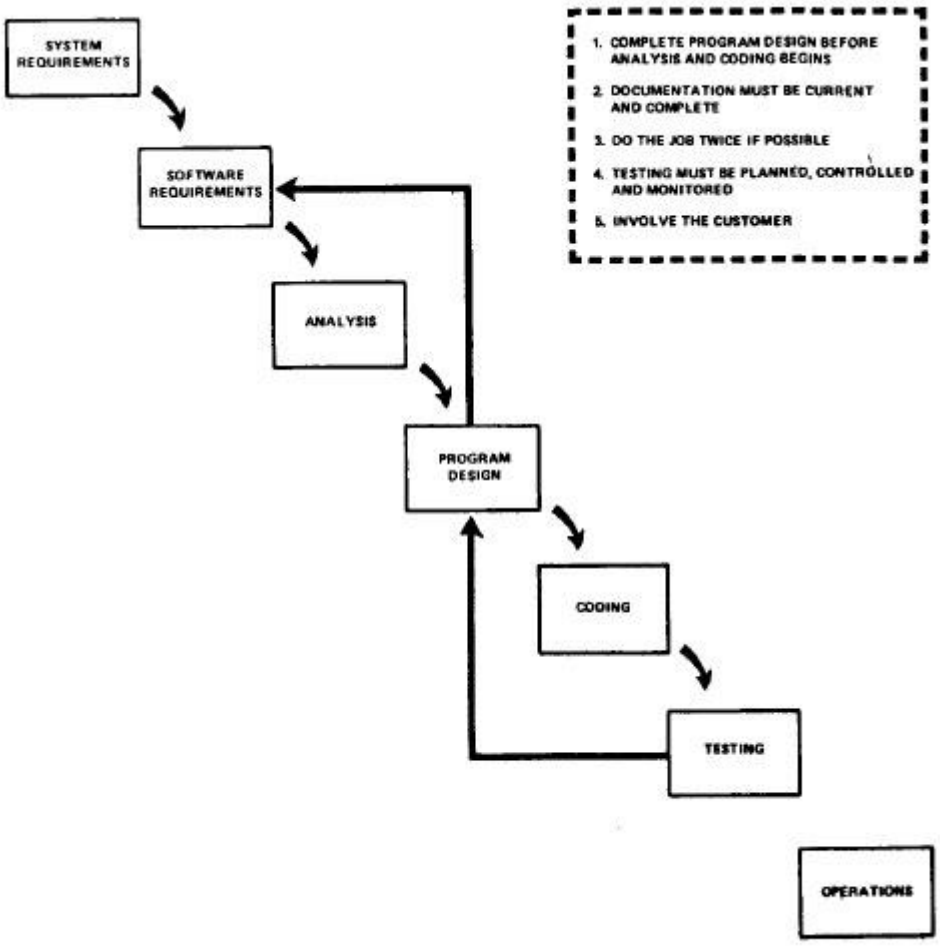
3.1 Vesiputousmalli

Struktuurisesta ohjelmistokehitysmenetelmästä puhuttaessa, puhutaan usein samalla myös vesiputousmallista. Vesiputousmalli on vaihejakomalli, joka esittää tietoteknisen sovelluksen elinkaarta. Vesiputousmalli julkaistiin ensi kertaa Tohtori Winston W. Roycen artikkelissa ”Managing the development of large software systems” vuonna 1970.

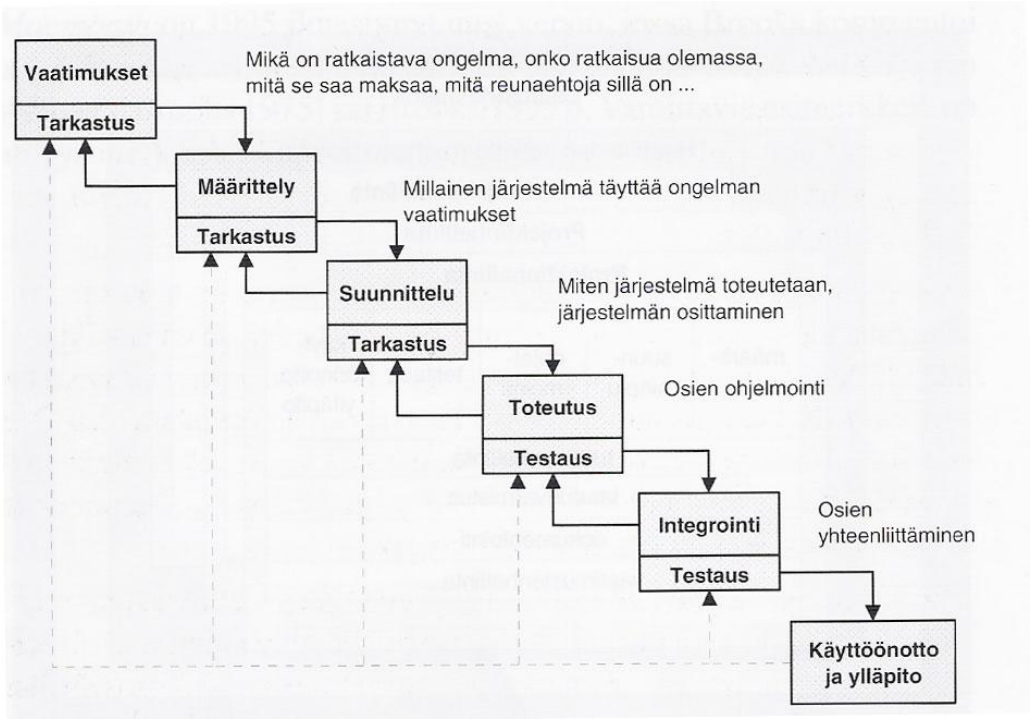
Nykyään vesiputousmallista on eri versioita, mutta suurimmassa osassa malleista on esitelty ainakin seuraavat kohdat: määrittely, suunnittelu, toteutus, integrointi, käyttöönotto ja ylläpito. Laadunvarmistamis-tilaisuuksia eli projektin katselmuksia ja kokouksia pidetään yleensä näiden vaiheiden jälkeen. (Haikala ja Märijärvi, 2004, 36-38)

Vesiputous-sana viittaa siihen, että vaiheet virtaavat alaspäin kuin vesi ja edeltävä vaihe tulee saada valmiiksi ennen kun aletaan seuraavaa, alempana olevaa vaihetta työstämään. Tässä on kuitenkin paradoksi, koska oikeassa joissain tapauksissa, ohjelmistokehitysympäristössä pitää päästä takaisin edelliseen vaiheeseen muuttamaan sen sisältöä sopivaksi tai päivittämään sen sisältöä. Tätä varten on lisätty nuolet myös ylöspäin kuvion 2 esittämällä tavalla (Huttunen, 2006, 8)

Vertailun vuoksi kuvio 1 esittää W. W. Roycen alkuperäistä vesiputousmallia ja kuvio 2 esittää nykyaikaista mallia.



Kuvio 1. W. W. Roycen vesiputousmalli (Lainattu: Royce, W. 1970. Managing the development of large software systems. 338)



Kuvio 2. Nykyaikainen vesiputousmalli (Lainattu: Haikala ja Märijärvi, 2004, 36)

3.1.1 Vesiputousmallin hyödyt

Vesiputousmallin hyöty on se, että sitä käyttämällä saadaan aikaan luotettavia, vakaita ja juuri niille vaadittuja toimintoja suorittavia sovelluksia. Vesiputousmalli on käytännöllinen varsinkin staattisissa sovelluksissa, joiden päivitys-syklit ovat harvempia. Myös, hyvin kokeneen ja osaavan sovelluskehitystiimin käsissä pystytään arvioimaan sovelluksen kuluva aika ja raha suhteellisen tarkasti sen jälkeen kun määritys on nähty.(Hentzen. 2002. 24)

Vesiputousmallia käyttämällä dokumentointi pysyy kunnossa ja kattavana, koska jokaisen vaiheen lopetuksessa tarkastetaan, että siihen vaiheeseen kuuluva dokumentointi on kunnossa. Kun dokumentointia on kirjoitettu, on projektille konkreettista todistusta ja vesiputousmallissa on luontaista tärkeä osa projektia, eli testaus.(Landay. 2001)

3.1.2 Vesiputousmallin haitat

Vesiputousmallia on moitittu mm. sen jäykkyydestä ja vanhanaikaisuudesta. Yksi haitta on, että sen läpikäyminen vie paljon aikaa ja suurin osa ajasta käytetään työhön, jonka tuloksia ei näy. Toinen haitta on että malli ei ole kovin joustava. Esim. jos tuotantovaiheessa asiakkaan vaatimukset vaihtuvat, vaatimuksia ei voi luoda lennossa sovellukseen, vaan on noustava pykälän ylöspäin ja muokattava edellisen vaiheen dokumentoinnin sisältöä. Tämä vie aikaa – ja rahaa. (Hentzen. 2002. 24)

Jotkin lähteet väittävät jopa, että vesiputousmallia ei oikeasti ole olemassa ja sen kirjoittaja W. W. Royce antaa artikkelissaan esimerkin huonosta mallista; vesiputousmallin. Se on kuitenkin noussut kuuluisimmaksi malliksi monien siteerauksien ja lainauksien ansiosta, eivätkä ihmiset ole jaksaneet lukea itse Roycen artikkelia paria sivua pidemmälle. Vesiputousmalli tuli Yhdysvaltain armeijan standardi (DoD-2167) ja vähän tämän jälkeen NATO-maat tarvitsivat ohjelmistosuunnittelu-strandardin ja hekin päätyivät vesiputousmalliin. Tästä voi johtua, että sitä käytetään niin paljon; ’jos armeija käyttää sitä, sen on pakko olla hyvä’. (Toikkanen, T. 2005). Jätän vesiputousmallin oikeellisuuden kuitenkin lukijan päätettäväksi.

Toinen lähde väittää taas, ettei ole nähnyt yhtään onnistunutta projektia, joka olisi tiukasti noudattanut vesiputousmallia. (Weisert, C. 2003)

3.2 Ketterät ohjelmistokehitysmenetelmät (Agile-methods)

Ketteriä menetelmiä on esiintynyt jossain muodossa vuosikymmeniä, mutta termi ”Agile methods” lanseerattiin vuonna 2001, kun 17 silloin kutsutun kevyiden menetelmien asiantuntijaa ja kannattajaa kokoontuivat. Kokouksessa päätettiin seuraavat asiat:

- Tarvitaan menetelmiä, jotka on suunniteltu vastaamaan muutoksiin ohjelmistoprojektin aikana.
- Kevyt –sana (engl. light) vaihdettiin ketterä –sanaan (engl. agile), koska ne joilla on monia ohjelmoijia tai ne jotka ohjelmoivat turvallisuuskriittisiä ohjelmistoja eivät hyväksy sanaa kevyt, mutta tarvitsevat projektiinsa ketteryyttä

- Ketterä julkaisu (engl. Agile Manifesto) luotiin. Se sisältää vapaasti englannista käännettynä seuraavan tekstin:
- ”Selvitämme parempia tapoja kehittää ohjelmaa tekemällä sitä itse ja auttamalla muita tekemään sen. Tämän työn tuloksena olemme päätyneet arvostamaan:
    - **Yksilöitä ja vuorovaikutusta** enemmän kuin prosesseja ja työkaluja
    - **Toimivaa sovellusta** enemmän kuin täydellistä dokumentaatiota
    - **Yhteistyötä asiakkaiden kanssa** enemmän kuin sopimusneuvotteluita
    - **Muutokseen vastaamista** enemmän kuin suunnitelman noudattamista

  Vaikka teksti oikealla puolella on arvokasta, arvostamme enemmän asioita vasemmalla puolella.” (Lainattu ja vapaasti suomennettu: www.agilemanifesto.org)
- 12 yksityiskohtaisempaa ja konkreettiseen viittaava ketterää periaatetta julkaistiin.

Kokouksesta lähtien ketterät menetelmät ovat olleet suosittu puheenaihe ohjelmistotuotannon ammattilaisten keskuudessa. Ketteriä menetelmiä kehittää ja ylläpitää järjestö nimeltään Agile Alliance. (Beck, K. & Paulk, M. 2004. 3-4).

Ketteriä ohjelmistokehitysmenetelmissä on myös valinnanvaraa. Niitä on kymmenkunta, esimerkiksi eXtreme Programming, Crystal Methods, Lean Development, Adaptive Software Development (ASD), Dynamic System Development Method (DSDM) ja Feature Driven Development (FDD). Kaikissa on joitain yhteisiä piirteitä, kuten se että projekti tehdään pienissä iteraatioissa, eli pieni osa sovellusta kerrallaan. Myös asiakkaan ja tuotantotiimien tiivis yhteistyö ja se, että kaikki ketterät menetelmät noudattavat Agile Manifestoa, on yhteistä jokaisessa menetelmässä. (Huttunen, 2006, 14-15)

Luvussa 3.3 tarkistelen syvemmin Scrum-menetelmää, koska se on lähinnä Descom Oy:n toimintaperiaatetta.

3.2.1 Ketterien menetelmien hyödyt

Miksi yhä useammat ohjelmistotalot ovat siirtyneet ketteriin menetelmiin tai ovat kiinnostuneita niistä? Tai miksi minä kirjoitan siitä työssäni? Seuraavat kohdat vastaavat, että miksi.

Nopea oppiminen. Tämä on ehkä tärkein ketterien menetelmien käyttämisestä saatu hyöty, koska siinä saa nopeammin palautetta asiakkaalta, että meneekö homma heidän mielestä oikeaan suuntaan. Nopeaa oppimista tapahtuu myös asioiden tekemisessä yhä tehokkaammin; jokaisen iteraation jälkeen tiimi voi punnita, miten edellinen iteraatio meni ja jatketaanko samalla tavalla – vai vaihdetaanko työskentelytapaa.

Nopeampi sijoitetun pääoman takaisinsaaminen (ROI). Ensimmäisestä iteraatiosta lähtien jokainen iteraatio tuottaa arvoa, kun taas vesiputousmallissa arvoa saadaan vasta valmiista sovelluksesta.

Lisääntynyt hallinta. Hänellä, kenellä on priorisointivastuu yrityksessä, on vapaat kädet ohjata tiimejä tekemään eri iteraatioita. Tämä on hyöty jos projektin johtajalla on taitoa pitää langat käsissään.

Muutokseen vastaaminen. Ketterän menetelmän perusajatus on luotu tämän pohjalle. Asiakkaiden haluamiin muutoksiin voidaan vastata joka iteraatiossa, eikä sitä varten tarvitse tehdä paljoa työtä, kuten vesiputousmallissa.

3.2.2 Ketterien menetelmien haitat

Ketterissä menetelmissä ja niiden omaksumisessa on kriittisiä kohtia, joiden pieleen menolla varmistetaan projektin epäonnistuminen. Uhkia projektin menestymiselle ovat:

- Käyttäjien aktiivinen osallistumien ja tiivis yhteistyö asiakkaan ja tuottajan välillä. Jos yhteistyö ei ole tarpeeksi tiivistä, tulos ei välttämättä ole odotuksien kal-

tainen. Tiivis yhteistyö vie asiakkaalta resursseja, kun heidän edustajansa istuu kokouksissa ja katselmuksissa.

- Vaatimukset kehittyvät ja niitä tulee lisää kehittämisprojektin aikana. Tämän takia vaarana on, että projektista tulee ikuinen. Tämä voidaan välttää sopimusteitse; tehdään sopimus perus-sovelluksesta, jonka asiakas maksaa sen saatuaan. Jos taas halutaan tehdä - ja tehdäänkin – muutoksia niistä sovitaan ja laskutetaan erikseen.(Koch, A. 2004. 16-17)
- Vaatimukset ovat hädin tuskin tyhjentäviä ja jotkut vaatimukset eivät ole lopputuotteelle keskeisiä. Koska keskusteluja käydään palavereissa, ei vaatimusten dokumentointi ole aina riittävää tai tarkkaa. Tämä taas tuottaa ongelmia projektiin myöhemmin tulevien ymmärtämistä lopputuotteesta ja sen toiminnoista.
- Ainainen testaaminen, joka tarkoittaa että testaajan täytyy olla projektissa lähes koko ajan mukana. Tämä taas tarkoittaa isompia kustannuksia.
- Tuotteen ominaisuuden hyväksyttäminen asiakkaalla. Tämä voi olla asiakkaalle rasittavaa, koska heidän tulee varata resursseja sen tarkastamiseen ja hyväksymiseen.(Agile Software Development, 2007)

Siirryttäessä ketteriin menetelmiin, vastaan voi tulla myös tuttu uhka; ihmisten vastarinta uutta asiaa vastaan. Esimerkiksi Scrum-tiimissä projektipäälliköstä tulee Scrum-mestari (ScrumMaster), joka projektipäällikkönä tehtävien antamisen ja tuloksien tarkastamisen sijaan alkaa Scrum-mestarin roolissa raivaamaan esteitä tiimin tieltä ja helpottamaan tiimiläisten työskentelyä. (Heidema, J. 2008)

4 SCRUM

Scrum-sana tulee rugby-pelistä. Rugbyssä se viittaa kuvioon, jossa molemmat joukkueet ovat pallon tullessa takaisin peliin. Se on yksi ketteristä ohjelmistokehitysmenetelmistä. Vaikka sitä ei ole ensisijaisesti kehitetty ohjemistoja varten, vaan sitä voidaan käyttää eri tekniikoissa ja eri projekteissa, myös ohjelmistokehitysprojektissa. Nykyiseen olomuotoonsa se kasvoi Japanissa 80-luvun puolivälissä. (Koch, A. 2004. 257) Äskeisen lauseen todenmukaisuutta voidaan kuitenkin epäröidä; Scrum Alliancen mukaan Scrum-menetelmän on luonut Jeff Sutherland vuonna 1993. Menetelmä julkaistiin

maailmanlaajuisesti vuonna 1995, jolloin Ken Schwaber julkaisi ensimmäisen artikkelinsa(Scrum Alliance, Inc. 2008a.).

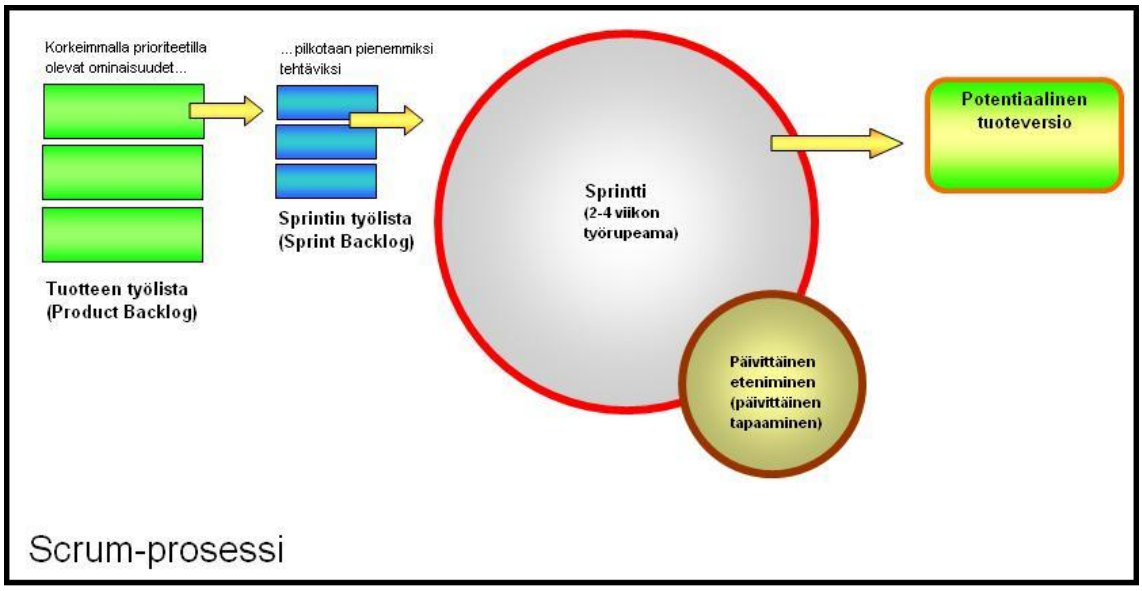
Scrumia käytetään muiden Agile-menetelmien lisäksi mm. Euroopan parhaimmaksi yritykseksi valitussa suomalaisessa Reaktor Innovation Oy:ssä. (Lindström, Jukka. 2006 & Great Place to Work Institute. 2008.). Scrum on maailmalla yksi tunnetuimmista ketteristä menetelmistä(Kuha, J. 2008. 19).

Scrum antaa toimintakehyksen sovelluksen tuotannolle. Se ei puutu liikaa siihen, mitä työkaluja missäkin kohdassa tulisi käyttää, eikä siihen miten sovellusta kehitetään. Menetelmän perussanoma on, että sovelluskehitysprosessia tulisi kehittää ja tehostaa koko ajan. Tämä taas tarkoittaa tuotantoryhmien ja niissä olevien yksilöiden jatkuvaa oman työn tehokkuuden tarkkailua ja kehittämistä. (Kuha, J. 2008. 19).

Scrum-projekti jäsentyy pieniin iteraatioihin (sykleihin), joita Scrumissa kutsutaan sprinteiksi (ts. kiri, engl. sprint). Sprintit ovat lyhytkestoisia, noin 2 – 4 viikkoa kestäviä prosesseja joissa jokainen Scrum-tiimi ottaa asiakkaan vaatimuksia priorisoidulta listalta ja tekee vaaditut ominaisuudet ja toiminnot sprintin aikana. (Scrum Alliance, Inc. 2008a.) Sprintissä tehdään myös ominaisuuksiin tai toimintoihin liittyvät dokumentit ja ne testataan, toisin sanoen; ominaisuudet tehdään sprintissä kokonaan valmiiksi. Tämä on tärkeää, koska se estää vesiputousmallissakin esiintyviä ongelmia, joissa jotkin toiminnot jätetään roikkumaan keskeneräisiksi koko projektin ajaksi. (Kuha, J. 2008. 19).

Jokaisen sprintin jälkeen valmistuu potentiaalinen tuoteversio.(Scrum Alliance, Inc. 2008a.)

Scrum-toimintatavassa on kolme roolia, kolme tilaisuutta ja kolme tuotosta. (Scrum Alliance, Inc. 2008a.)



Kuvio 3. Scrum-prosessi

4.1 Scrum roolit

Scrum Mestari (ScrumMaster) on vastuussa Scrumin menestymisestä. Vaikka Scrumissa tämä on uudenlainen rooli verrattuna muihin menetelmiin, siinä toimii ennestään projektinjohtajana tai tiimin vetäjänä ollut henkilö. (Koch, A. 2004. 257). Scrum Mestarin tehtäviin lukeutuu seuraavat asiat:

- Varmistaa, että Scrumin periaatteita noudatetaan ja prosessissa aikaansaadaan arvoa tuottavia asioita, kuten ominaisuuksia ja toimintoja. (Koch, A. 2004. 257).
- Poistaa Scrum-tiimin raportoimia esteitä ja helpottaa heidän työtään kaikilla mahdollisilla tavoilla, jotta he voivat työskennellä optimaalisella tavalla.
- Aikaansaada tiivis yhteistyö kaikkien roolien kesken
- Suojata Scrum-tiimiä ulkoisilta häiriöiltä (Scrum Alliance, Inc. 2008b.)

Scrum Mestarin täytyy myös vetää päivittäisiä Scrum-tapaamisia. Tätä varten hänen täytyy tietää mitä tehtäviä ollaan tekemässä, mitkä tehtävät ovat valmiina ja mitä tehtäviä – uusia vaatimuksia – on löytynyt. Mestarin tulee kaivaa esille riippuvuudet ja esteet Scrumin toimivuudelle, jotka hänen pitää prorisoida ja selvittää niiden syyt. Esteiden korjaamisesta tulee tehdä suunnitelma, niiden purkaminen voidaan tehdä tiimin sisällä tai tiimien kesken. Johdon kanssa täytyy tehdä yhteistyötä silloin kun este tulee yhtiön sisäisistä ongelmista.

Yksilöiden ongelmat ovat myös tärkeä huomio, jota Scrum-mestarin täytyy tarkkailla ja tehdä siihen liittyen yhteistyötä henkilöstöhallinnon ja johdon kanssa. James Coplien, joka on toiminut Scrum Mestarina yli 200 tapauksessa sanoo, että 50% tuottavuuden tappiosta johtuu yksilöiden omista ongelmista. (Scrum Alliance, Inc. 2008b.)

Scrum-tiimissä ei ole erikseen koodareita, suunnittelijoita tai johtajia. Vaan kaikki ovat asiansa tarpeeksi hyvin osaavia tiimiläisiä, joilla on yhteinen päämäärä; saada annetut tehtävät valmiiksi (Poimala S.). Lähes kaikissa ohjemistotuotannon malleissa käytetään tiimejä. Mutta Scrum eroaa muista siksi että, tiimillä on kaikki valta tehdä mitä tahansa he tarpeelliseksi näkevät saavuttaakseen päämääränsä, jopa lopettaa sprintin. Tämänlainen vapaus päättää asioista on sekä suuri muutos organisaatiossa, kun siirrytään käyttämään Scrumia, (Koch, A. 2004. 258) se antaa myös vastuun tuotteen onnistumisesta tiimeille, ei yksilöille. (Schuh. 2004. 24).

Tuotteen omistaja (Product owner) voi olla tuotepäällikkönä toimiva henkilö, asiakasprojekteissa tuotteen omistaja on yleensä asiakkaan tekninen projektipäällikkö tai asiakkaan yhteyshenkilö (Poimala S.). Hän on viimekädessä vastuussa tuotteen ominaisuuksista ja toiminnoista, julkaisupäivämäärästä, ominaisuuksien priorisoinnista ja työn hyväksymisestä tai hylkäämisestä (Scrum Alliance, Inc. 2008b.). Tuotteen omistajan on siis oltava kykeneväinen valitsemaan tuotteen työlistaan vain arvoa tuottavat toiminnot ja priorisoitava nämä arvon mukaan. Nämä toiminnot teettävät työtä, ja työ tarkoittaa rahanmenoa, siksi hänellä on oltava myös tarpeeksi valtaa tehdä päätöksiä jotka koskevat budjettia saattaakseen toiminnot valmiiksi sovittuun aikaan mennessä. Siksi isommissa projekteissa onkin parempi turvautua yhden henkilön sijasta tiimiin, joka toimii tuotteen omistajana (Gravendeel, E & Ramos, C. 2008)

Tuotteen omistajan hommiin kuuluu myös pitää ensimmäiset kolme päivittäistä Scrum-tilaisuutta.

4.2 Scrum tuotokset

Kolme Scrumissa syntyvää tuotosta ovat: Tuotteen työlista (product backlog), sprintin työlista (sprint backlog) ja jäljellä olevan työmäärän kaavio (burndown chart) (Scrum Alliance, Inc. 2008c).

Jotta Scrum pääsee alkamaan, tuotteen omistajalla pitää olla visio tuotteesta jossa on asiakkaan keskeisin vaatimuskehys. Tämän lisäksi hänellä pitää olla liiketoimintasuunnitelma jossa on tiedot arvioidusta tulovirrasta tiettynä aikavälinä ja investoinnin panoksesta.

Näistä visioista tuotteen omistaja tekee listan, jossa on listattu vaatimukset järjestettynä niiden arvokkuuteen asiakkaan kantilta katsottuna. Tätä listaa sanotaan tuotteen työlistaksi (Scrum Alliance, Inc. 2008d). Työlistalla ensimmäisinä olevat (tärkeimmät-) vaatimukset pitää purkaa yksityiskohtaisiin, pieniin osiin, jotta ne pystyttäisiin tuottamaan, testaamaan ja toimittamaan ensimmäisen sprintin jälkeen. Noin kymmenen päivää/osa on tärkeimmille osille sopiva pituus (Scrum Alliance, Inc. 2008c).

Tuotteen työlistasta vaatimukset, jotka halutaan/voidaan toteuttaa seuraavassa sprintissä, jaotellaan eri tehtäviin ja tehtävät puretaan vielä pieniin paloihin (kesto noin kaksi työpäivää), jotka jaetaan tiimien kesken tehtäviksi. Tätä listaa kutsutaan sprintin työlistaksi.

Jäljellä olevan työmäärän kaavio (Burndown Chart) näyttää kumulatiivisesti olevan työmäärän ajan sprintissä ja koko projektissa. Sprintin kokonaisaika saadaan sprintin työlistasta ja projektin kokonaisaika saadaan tuotteen työlistasta. Kun tehtäviä valmistuu sprintissä, Scrum-mestari päivittää sprintin kaaviota. Kaaviossa käyrä voi pomppia ylös-alas, mutta suunnan pitäisi olla laskeva. (Scrum Alliance, Inc. 2008c). Sprintin tullessa valmiiksi reaaliaikaista kaaviota verrataan suunniteltuun kaavioon, siten saadaan selville tehtävien oikea kesto ja voidaan lisätä tai poistaa tehtäviä seuraavasta sprintistä (Gavaldo, Eric. 2008.)

4.3 Scrum tilaisuudet

Scrumin kolme tilaisuutta ovat sprintin suunnittelukokous (sprint planning meeting), päivittäinen Scrum tapaaminen (daily Scrum meeting) ja sprintin katselmuskokous (sprint review meeting).

Äsken läpi käydyt tuotokset tehdään ja niitä ylläpidetään tilaisuuksissa. Ensimmäinen tilaisuus on sprintin suunnittelukokous. Sitä ryhdytään valmistelemaan, silloin kun tuotteen työlistaan ollaan saatu määriteltyä ja priorisoitua tarpeeksi tehtäviä ensimmäiseen 30 päivän sprinttiin. Tuotteen omistaja esittelee kokouksessa tuotteen työlistan, etenemissuunnitelman (roadplan) ja julkistussuunnitelman (release plan). Tuotteen työlistasta tiimit valitsevat niin monta korkean prioriteetin ominaisuutta tehtäväkseen seuraavalla sprintillä, kuin he suunnittelevat ehtivänsä tehdä. Alemmalla prioriteetilla olevat ominaisuudet jätetään odottamaan seuraavaa sprinttiä. Tiimin pitää valitessaan ottaa huomioon sen koko, tämänhetkinen tuottavuus ja käytettävissä oleva tuntimäärä. Valitsemisen jälkeen tiimit pilkkovat Scrum mestarin avulla ominaisuudet pienempiin palasiin, eli tehtäviin. Tehtävät lisätään sprintin työlistaan. (Scrum Alliance, Inc. 2008d.)

Sprintin suunnittelukokouksessa sovitaan myös sprintin päämäärästä (sprint goal). Siinä määritellään muutamalla lauseella mitä tämän sprintin aikana tullaan saavuttamaan. Suunnittelukokouksessa tehtyä päämäärä dokumenttia verrataan saavutettuihin päämääriin sitten sprintin katselmuksessa. (Mountain Goat Software. 2008a.)

Suunnittelukokouksen jälkeen aletaan työstämään sprinttiä. Sprintissä pidetään päivittäin Scrum tapaamisia. Päivittäinen kokous on noin 15 minuuttia pitkä ja sitä johtaa Scrum mestari. Siihen voi osallistua kuka haluaa, mutta puhua saavat vain Srcrum tiimissä olevat henkilöt. Kukin tiimiläinen vastaa kysymyksiin; mitä tein eilen, mitä tein tänään ja mitä esteitä minulla tuli vastaan. Päivittäisen kokouksen tarkoituksena on luoda kokonaiskuva projektin edistymisestä ja päivittää sprintin työlistaa ja jäljellä olevan työmäärän kaaviota reaaliaikaiseksi. (Scrum Alliance, Inc. 2008d.)

Sprintin loputtua pidetään sprintin katselmuskokous. Kokous on noin neljän tunnin pituinen, kaksiosainen tapahtuma johon osallistuvat scrum rooleissa olevien henkilöiden lisäksi asiakkaan yhdyshenkilöitä (Scrum Alliance, Inc. 2008d.). Ensimmäisessä puolikkaassa scrum tiimi esittelee työnsä tulokset sprintin aikana yleensä demoamalla tehtyjä ominaisuuksia. Tätä vaihetta on pyrittävä pitämään mahdollisimman epävirallisena, joten tapaamisen valmistautumiseen ei kannata käyttää enempää kahta tuntia ja powerpoint-esityksiä ei kannata tehdä. Sprintin katselmuskokouksessa verrataan sprintille annettua tavoitetta realistiseen saavutukseen.(Mountain Goat Software. 2008b.)

Toisessa puolikkaassa tiimit arvioivat scrum mestarin johdolla sitä miten he työskentelivät yhdessä; mikä siinä oli mukavaa ja mitä työskentelytapaa pitäisi käyttää uudelleen seuraavassa sprintissä. Ja mitä työtapoja pitäisi kehittää seuraavassa sprintissä.

Sprintti on nyt loppu ja voidaan aloittaa taas seuraava sprintti suunnitelmakokouksella. Tätä rataa jatketaan, kunnes tarpeeksi monta ominaisuutta on saatu valmiiksi ja voidaan julkaista tuote – ohjelman ensimmäinen versio. (Scrum Alliance, Inc. 2008d)

5 OHJELMISTOTESTAUS

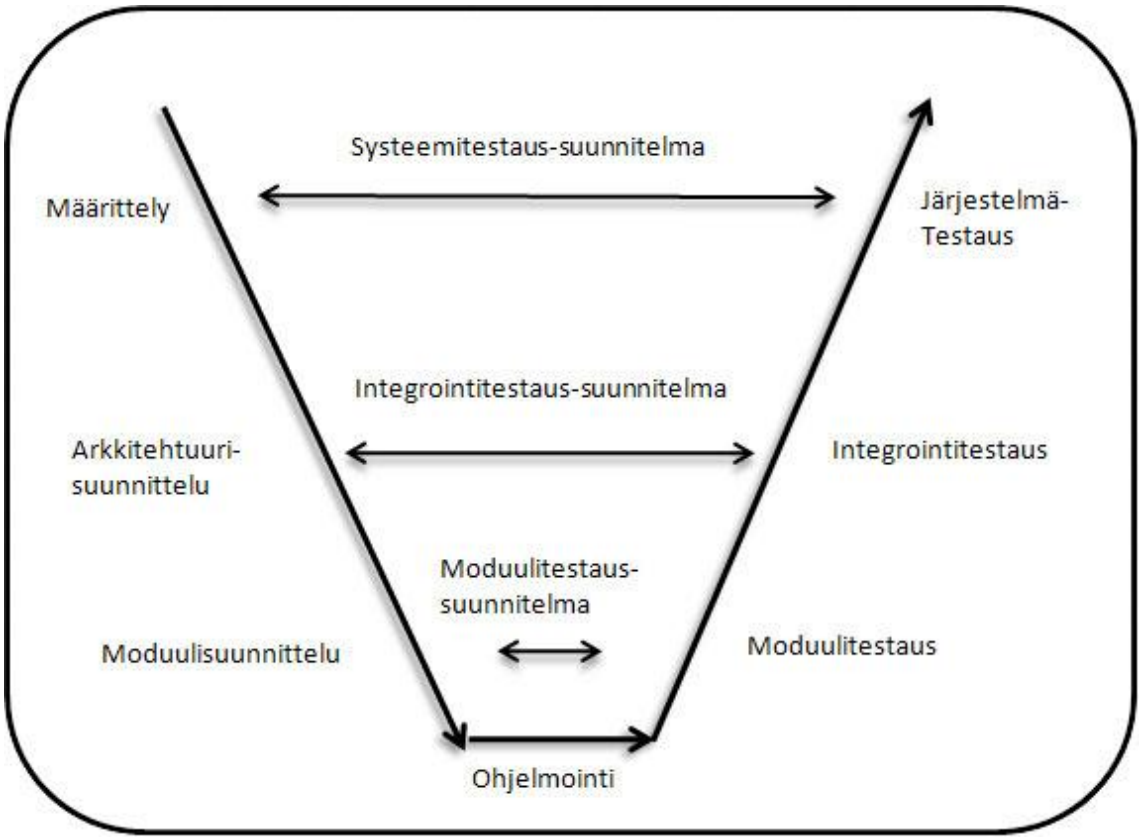
Kun varsinainen opinnäytetyö koskee testausdokumentin hallintaohjelmaa tulen seuraavaksi avaamaan hieman testauksen maailmaa sekä perinteisessä ohjelmistotestausmenetelmässä, johon ko. sovellus tehdään, että Scrumissa.’

Testaus sanana tarkoittaa jonkinnäköistä kokeilemista, mutta ohjelmistotuotannossa tarkempi kuvaus on virheiden tarkoituksellista etsimistä. Virheet tarkoittavat tässä yhteydessä asioita, jotka eivät ole niinkuin ne on määrittelyssä sanottu. (Haikala ja Märijärvi, 2004, 284-287)

Olkoot ohjelmistokehitysmenetelmä mikä tahansa, testaus on aina tärkeää. Siihen kuluu tutkimusten mukaan noin 50 % ohjelmistotuotantoprojektin resursseista (SwBusiness. 2007).

5.1 Perinteinen ohjelmistotestausmenetelmä

Struktuurisessa ohjelmistokehitysmenetelmässä testaus tapahtuu yleensä vasta silloin kun tuote on valmis. Testisuunnittelmat tehdään yleensä määrittelyn ja suunnittelun aikana. Testauksen eri vaiheiden kulkua ja testaus-suunnitelmien tekoa kuvaa V-Malli, jota kuvio 4. esittää.



Kuvio 4. V-Malli

Testausta varten perustetaan testausryhmä, joka koostuu ammatti-ohjelmistotestaajista, yrityksen sisäisistä tai ulkoisista työntekijöistä (Van Roosmalen, Ralph. 2007.). Testaus sovelluksen ohjelmoijien kanssa voi olla hankalaa; ohjelmoijat voivat alitajuntaisesti olla etsimättä virheitä ja yrittää todistaa ohjelman toimivuus (Haikala ja Märijärvi, 2004, 284).

5.2 Ohjelmistotestaus Scrumissa

Perinteinen ohjelmistotestaus ei sovi Scrum-projektiin, koska siinä testaus yleensä alkaa vasta kun sovellus on valmis. Scrumin sprintissä ei ole aikaa aloittaa testaamista sovelluksen, tai se osan ollessa valmis, vaan sitä pitää tehdä sovellusta kehitettäessä. Tämän takia olisi hyvä, että ammatti-testaaja olisi mukana Scrum-tiimissä. (Van Roosmalen, Ralph. 2007.)

Lähtökohta on, että Scrum-prosessissa ohjelmistotestaus tehdään sprintin aikana; silloin kun ohjelmaa ollaan tekemässä (Kuha, J. 2008. 21). Scrum ei sisällä minkäänlaista oh-

jeistusta ohjelmiston testauksesta ja perinteistä ohjelmistotestausmenetelmää ei voi suoraan Scrumissa käyttää. Joten on yleistä, että ohjeet testaukseen lainataan toisesta ketterästä menetelmästä; XP:sta (eXtreme Programming). XP-prosessi ohjeistaa testaamisen, mutta sekään ei ole ohjekirja testaajalle. (Van Roosmalen, Ralph. 2007.)

6 TEST MANAGEMENT TOOL –PROJEKTI SOVELLETTUNA SCRUM-MENETELMÄÄN

Vaikka Descom Oy käyttää joissain sen projekteissa Scrum-menetelmää, sitä ei varsinaisesti käytetty tässä projektissa vaikka viitteitä menetelmään olikin. Se käytiin läpi siten, että palavereissa näytin mitä asioita olen saanut valmiiksi ja Descom Oy:n yhteyshenkilöt hyväksyivät ne tai antoivat korjauskehotteita. Näistä kehotteista ja palaverissa ilmenneistä virheistä tein korjauslistan. Tämän jälkeen suunniteltiin, mitä teen ensi palaveriin mennessä valmiiksi.

Scrumissa - kuten muissakin ketterissä menetelmissä on kuitenkin se hyvä piirre, että sääntöjä ei ole ns. kiveen hakattu ja niitä voidaan soveltaa. Tässä luvussa käyn läpi miten tämä projekti oltaisiin suoritettu Scrum-menetelmällä, miten Scrumin roolit olisivat jakautuneet tässä projektissa, miten tapaamisia sovellettaisiin tähän projektiin ja miten Scrumin eri dokumentteja sovellettaisiin tähän projektiin.

6.1 Roolit TMT -projektissa

Projekti on yhden opiskelijan tekemä opinnäytetyö. Tämä rajaa rooleja, mutta niitä voidaan silti soveltaa projektiin seuraavalla tavalla

Tuotteen omistaja: Timo Pyykkö ja Jussi Korhonen / Descom Oy
Scrum-mestari: Antti Haavisto
Scrum-tiimi: Antti Haavisto

6.2 Tuotokset TMT –projektissa

Projektin alkaessa tuotteen omistajalla oli visio tehtävästä sovelluksesta. Visio oli, kuten kannattaakin olla, kirjoitettu paperille määritelmän muotoon. Määrittely löytyy liitteestä 1.

Määrittelystä tehtiin tuotteen työlista.

6.2.1 Tuotteen työlista

Taulukko 1. esittää tuotteen työlistaa. Tuotteen työlistassa on listattu työlistan asioita (PBI, Product Backlog Item) lihavoituna, asiat ollaan pilkottu yhteen tai useampaan asian tuottamaan ponnisteluun (Product Backlog Item Effort). Yksittäiset ponnistelujen arvoitu kesto merkitään tunteina. (Szalvaym, Victor. 2007)

Taulukko 1. Tuotteen työlista

Prioriteetti	Asia #	Kuvaus	Arvioitu aika tunteina	Tekijä
Korkein				
		Testplan –ympäristö		
	1	Testplan –lomake	20	A.H
	2	Testplan –näkymät	6	A.H
		Testcase –ympäristö		
	3	Testcase –lomake	24	A.H
	4	Testcase –näkymät	6	A.H
		Ominaisuuksien muokkausympäristö		
	5	Lomake valintalistojen vaihtoehtojen muokkaamiseen	17	A.H
	6	Lomake testplan-kentän otsikoiden muokkaamiseen	18	A.H
	7	Lomake ulkoisten tietokantojen polkujen muokkaamiseen	14	A.H
	8	Ominaisuuksille näkymä	5	A.H
	9	Käyttöliittymä	10	A.H
Korkea				
		Testrun –ympäristö		
	10	Testrun –lomake	8	A.H
	11	Testrun –näkymät	6	A.H
		Testcaserun –ympäristö		
	12	Testcaserun –lomake	15	A.H
	13	Testcaserun –näkymät	6	A.H
		Muut asiat		
	14	Yhteiset metatiedot	4	A.H
Matala				
	15	Raporttiexportit	8	A.H
	16	Project Home	20	A.H
	17	Testauksen aikataulun suunnittelu ja seuranta sekä muistutukset	23	A.H
	18	Raporttiexportit	8	A.H
	19	Dashboard	20	A.H
		YHT.	238	**tuntia**

Tuotteen työlistasta korkeimmalla prioriteetilla olevat asiat muutetaan tehtäviksi ensimmäiseen sprinttiin.

6.2.2 Sprintin työlista

Sprintin työlista on oikein käytettynä tehokas työkalu, koska se ohjaa ja kannustaa tiimiläisiä suorittamaan sille otetut tehtävät. Siitä tiimiläiset näkevät helposti mitä tehtäviä on suorittamatta ja kukin voi yksilöllisesti ottaa sieltä tehtäviä suoritukseen kun on esimerkiksi saanut oman työnsä päätökseen. Sen avulla pystytään myös keskittymään yhteen tiettyyn asiaan kerrallaan; siihen on arvioitu kuinka paljon aikaa käytetään eri päivinä kunkin tehtävän tekemiseen Jotta sprintin työlista olisi tehokas, tulisi tiimin kaikkien henkilöiden osallistua sen käyttämiseen, muokkaamiseen ja noudattamiseen. Työlista on myös tiimin sisäinen työkalu, joten sitä ei tarvitse esitellä muille prosessin jäsenille; esimerkiksi tuotteen omistajalle. (Agilecollab. 2008.)

Ensimmäisessä sprintin suunnittelukokouksessa valittiin sprintissä tehtäväksi korkeimmalla olevat Testplan-ympäristön tekemisen ja Testcase-ympäristön tekemisen. Tuli kuitenkin ilmi, että ominaisuuksien muokkausympäristö pitää saada valmiiksi, jotta edellä mainittuja ympäristöjen toimivuutta voidaan testata; sitä varten otettiin sprinttiin myös ominaisuuksien muokkausympäristön tekeminen. Suunnittelukokouksessa päätettiin myös sprintin pituudeksi kaksi viikkoa, eli 14 päivää.

Sprintin työlista voi olla lähes mikä tahansa dokumentti; excel-taulukko, tekstiasiakirja, internet-dokumentti.

Internetistä löytyy hyviä esimerkkejä työlistasta. Esittelen tässä Artem Archencon tekemä yksinkertainen, aloittelijoille sopiva malli muokattuna TMT-projektiin.

Taulukko 2. Sprintin työlista

Sprint 1. Testplan- ja Testcase-ympäristön tekeminen

Asia#	Tehtävä	Päiviä sprintissä / Työtä jäljellä (tunteina)														
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		105	97	86	82	68	55	45	38	32	27	20	16	12	10	6
5	**Lomake valintalistojen vaihtoehtojen muokkaamiseen**															

	Suunnittele ja luo kentät	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Luo tallenna-,tallenna ja sulje- ja sulje-napit	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Synkronointi testisuunnitelma-lomakkeen kanssa	11	14	11	10	5	0	0	0	0	0	0	0	0	0	0
6	**Lomake testplan-kentän otsikoiden muokkaamiseen**															
	Suunnittele ja luo kentät	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	Luo tallenna-,tallenna ja sulje- ja sulje-napit	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Synkronointi testisuunnitelma-lomakkeen kanssa	12	11	10	8	4	2	2	12	11	10	5	0	0	0	0
7	**Lomake ulkoisten tietokantojen polkujen muokkaa-miseen**															
	Suunnittele ja luo kentät	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
	Luo tallenna-,tallenna ja sulje- ja sulje-napit	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Synkronointi testisuunnitelma-lomakkeen kanssa	11	10	9	9	8	6	4	0	0	0	0	0	0	0	0
1	**Testplan-lomake**															
	Luo tallenna-,tallenna ja sulje- ja sulje-napit	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	Suunnittele ja luo kentät	8	8	8	8	6	6	6	4	2	1	0	0	0	0	0
	Muokkaa näkyvyydet lomakkeella	4	4	4	4	4	4	4	1	0	0	0	0	0	0	0
	Luo toiminto, jossa käyttäjä pystyy lisäämään ja liittä-mään tapauksen suunnitelman alle	2	2	2	2	2	2	2	2	2	2	0	0	0	0	0
	Luo toiminto, jossa käyttäjä pystyy liittämään jo olemas-sa olevan tapauksen suunnitelman alle	5	5	5	5	5	5	5	5	5	5	11	12	10	9	6
2	**Testplan-näkymät**															
	Luo näkymät, siten kun ne on määrittelyssä toivottu	6	6	6	6	6	6	6	6	6	6	4	4	2	1	0
3	**Testcase-lomake**															
	Luo tallenna-,tallenna ja sulje- ja sulje-napit	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
	Suunnittele ja luo kentät	5	5	5	5	4	3	2	0	0	0	0	0	0	0	0
	Muokkaa kenttien näkyvyyksiä lukutilassa	7	7	7	7	6	4	3	0	0	0	0	0	0	0	0
	Tee steppien lisääminen siten, että niitä voi jouhevasti lisätä	10	10	10	10	10	9	9	8	6	3	0	0	0	0	0
	Luo testiajonappi, johon tehdään toiminto testiajoympä-ristön yhteydessä	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4	**Testcase-näkymät**															
	Luo näkymät, siten kun ne on määrittelyssä toivottu	6	6	6	6	6	6	1	0	0	0	0	0	0	0	0

Sprintin työlista tehdään siirtämällä tuotteen työlistalta ominaisuuksia ja toimintoja ja purkamalla ne pienempiin tehtäviin. Tehtävien kesto arvioidaan tämän jälkeen tunteina. Sprintin työlistaa päivitetään joka päivä päivittäisessä Scrum tapaamisessa. Työn alla olevien tehtävien kestoa arvioidaan jokaisessa tapaamisessa.

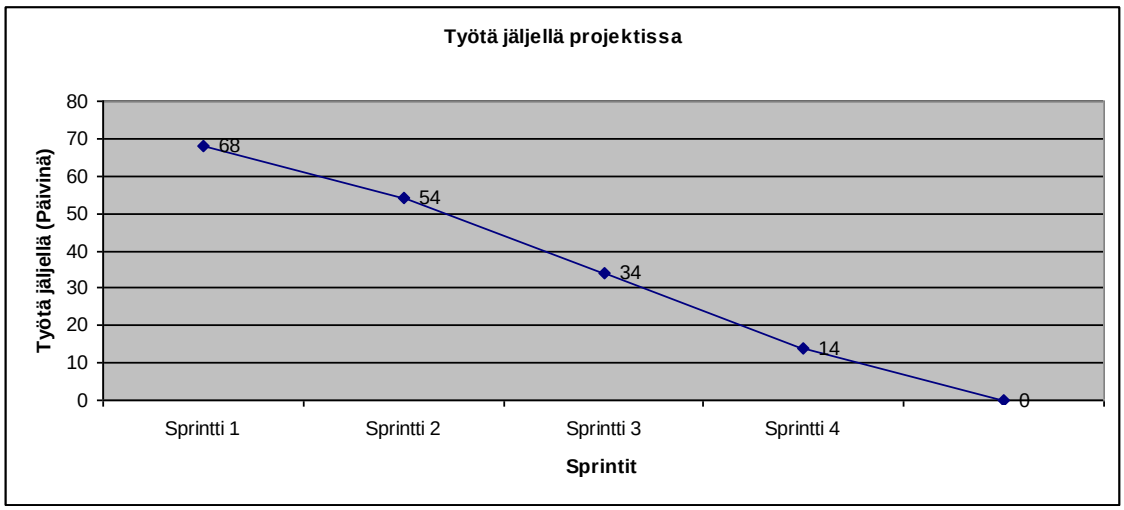
Taulukko 2. esittää projektin ensimmäisen sprintin työlistaa sprintin ollessa valmis. Taulukosta huomataan ettei tehtävää ole aloitettu, kun jäljellä oleva tuntimäärä ei ole muuttunut. Joissain tehtävissä jäljellä olevan ajan huomataan olleen ali-arvioitu niiden vaativuuteen verrattuna, tällainen on tapahtunut esimerkiksi tehtävässä ”Luo toiminto, jossa käyttäjä pystyy liittämään jo olemassa olevan tapauksen suunnitelman alle”. Tehtävä on aloitettu sprintin 10. päivänä, sen jälkeen kun Testcase-ympäristö ollaan saatu lähes valmiiksi. Tehtävä on kuitenkin yllättänyt vaativuudellaan ja sen arvioitua kestämisaikaa on pidennetty. Siksi tehtävä on jäänyt sprintissä vähän kesken ja se tehdään loppuun mahdollisesti ensi sprintin aikana.

6.2.3 Jäljellä olevan työmäärän kaavio

Nimensä mukaisesti jäljellä olevan työmäärän kaavio (Burndown Chart) näyttää jäljellä olevan tuntimäärän sprintissä (Szalvaym, Victor. 2007). Tuotteen työmäärän kaaviossa voi arvona olla haluttujen toimintojen tai työpäivien määrä. Scrum tiimit päivittävät tuotteen kaaviota sprintin lopulla. (Mountain Goat Software. 2008c.) Sprintin kaaviota päivittäisissä scrum tapaamisissa.

Kaaviokin voi olla lähes millä tahansa tehty, mutta internetistä löytyvistä tuotteen- ja sprintin työlistojen pohjissa on yleensä mukana excel-kaavio, joka päivittyy automaattisesti kun arvoja taulukossa muutetaan.

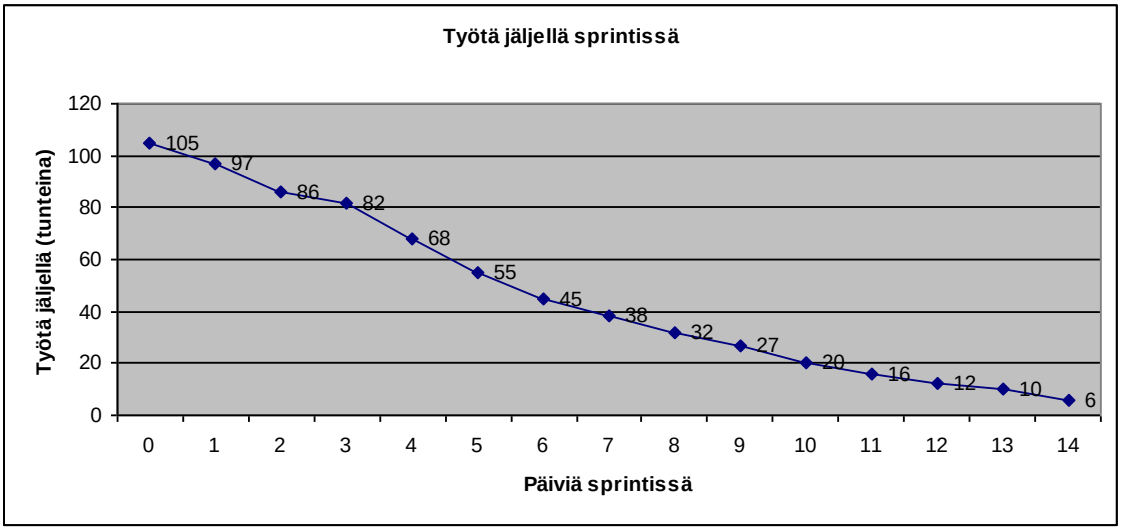
Kuvio 5. esittää nykyisestä tuotteen työlistasta tehtyä kaaviota. Sprinttejä oli ensimmäisessä sprintin suunnittelukokouksessa vain kolme, mutta ensimmäisen sprintin jälkeen lisättiin yksi sprintti, koska vaaditut toiminnot olivat vaativampia, mihin oltiin varauduttu.



Kuvio 5. Projektissa jäljellä olevan työmäärän kaavio

Kuvio 6. esittää jäljellä olevaa tuntimäärän kaaviota TMT-projektin ensimmäisessä sprintissä. Linjan tulisi pomppia ylös-alas, mutta suhdanteen tulisi olla alaspäin. Käyrän pomppiminen viittaa siihen, että tiimin jäsenet ovat raportoineet päivittäin realistisesti

siitä mitä he arvioivat olevan kunkin tehtävän jäljellä oleva aika. (Szalvaym, Victor. 2007)



Kuvio 6. Sprintissä jäljellä olevan työmäärän kaavio

Niin kuin kaaviosta huomaa, kaavion käyrä ei poukkoile ylös-alas tässä sprintissä. Tämä siksi, koska tässä projektissa oli vain yksi tiimiläinen ja tehtäviä tehtiin yleensä yksi kerrallaan. Kaaviosta huomaa myös, että sprintti ei saavuttanut yhtä tavoitettaan; 0 tuntia työtä jäljellä sprintin lopussa.

6.3 Scrum tilaisuudet TMT-projektissa

Kaikki projektin kokoukset käytiin puhelimitse, apuvälineenä käytettiin Lotus Notes Sametime-ohjelmaa. Tämä osoittautui hyväksi tavaksi, koska Lotus Notes Sametime toimii lähes jokaisessa selaimessa ja siksi kokoukseen voi osallistua lähes mistä vain, missä on internet-yhteydellä varustettu tietokone. Ainoa parempi tapa kokouksille olisi ollut varsinainen tapaaminen, mutta tämä vaihtoehto ei ollut mahdollista pitkän välimatkan takia.

Puhelun äänittäminen osoittautui hyväksi tavaksi, kun puhelinpalavereita pidetään. Tällöin muistiinpanoja ei tarvitse tehdä palaverin aikana vaan pystyy keskittymään palaveriin, esimerkiksi potenttiaalisen tuotteen esittelyyn.

6.3.1 Sprintin suunnittelukokous

Suunnittelukokouksessa tuotteen omistaja esitteli vaatimusmäärittelyn. Vaatimusmäärittelystä tulee ilmi myös projektin tavoite. Vaatimusmäärittely käytiin kohta kohdalta läpi ja Scrum-tiimi (eli minä) kysyin kysymyksiä yleensä riippuen heidän näkökantaansa kyseessä olevista asioista. Ensimmäisen sprintin suunnittelukokous oli kokouksista pisin; se kesti noin 2½ tuntia. Tämän jälkeen Scrum-tiiminä pidin itsenäisen aivoriihen, jossa tein muistiinpanojeni pohjalta tuotteen työlistan ja ensimmäisen sprintin työlistan (Taulukko 1. ja 2.).

6.3.2 Sprintin päivittäiset tapaamiset

Tässä projektissa varsinaisia päivittäisiä kokouksia ei ollut. Kuitenkin pyrin päivittäin vastaamaan kysymyksiin mitä tein eilen? mitä tein tänään? ja mitä esteitä minulla tuli vastaan? Esteistä osan pystyin karsimaan itse ja osaan esteistä kysyin ratkaisua Descom Oy:n puolelta.

6.3.3 Sprintin katselmustilaisuus

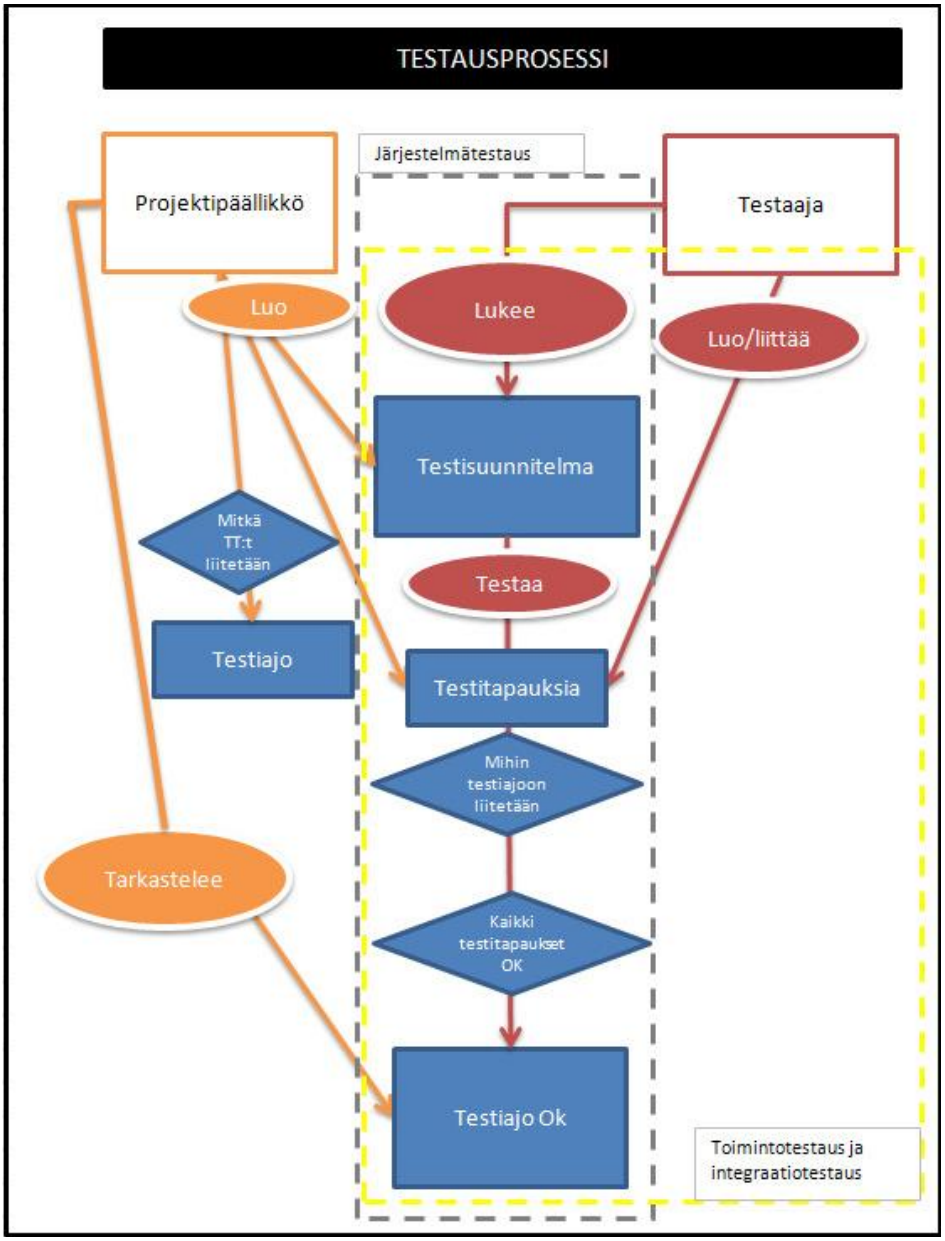
Sprintin ollessa lähes valmiina, sovimme ajan sprintin katselmustilaisuuteen. Näissä tilaisuuksissa esittelin sovellukseen tehdyt muutokset. Esityksen aikana ja sen jälkeen tuotteen omistaja antoi korjauskehoitteita; useimmat korjauskehoitteet johtuivat mielestäni siitä etten aivan selvästi ymmärtänyt mitä halutulta toiminnolta halutaan. Tämä johtui suurimmalta osin siitä, ettei palaveri ollut varsinainen tapaaminen, jossa havaintovälineitä on enemmän.

Suoraan katselmustilaisuuden jälkeen tässä projektissa pidettiin yleensä seuraavan sprintin suunnittelukokous. Siinä kirjattiin äskeisen sprintin korjaus- ja muutoskehoitukset sprinttiin suurimmalla prioriteetilla ja päätettiin sprintissä tehtävät kokonaisuudet.

7 TEST MANAGEMENT TOOL

7.1 Prosessin kuvaus

Sovellus tukee ainakin kolmea eri testaustyyppiä; moduulitestausta, integraatiotestausta ja järjestelmätestausta. Kuvio 15. esittää yleisesti sovelluksen käyttäjien prosesseja.



Kuvio 7. Testausprosessi

Vaikka sovelluksessa ei ole rooleja, testausprosessin selvennöksen vuoksi kaksi roolia; projektipäällikkö ja testaaja (voi käsittää myös kehittäjänä).

Moduulitestauksessa, jossa tarkoituksena testata moduulin toimintaa ja integraatiotestauksessa, jossa tarkoituksena testata moduuleiden yhteensopivuutta prosessi kulkee seuraavasti:

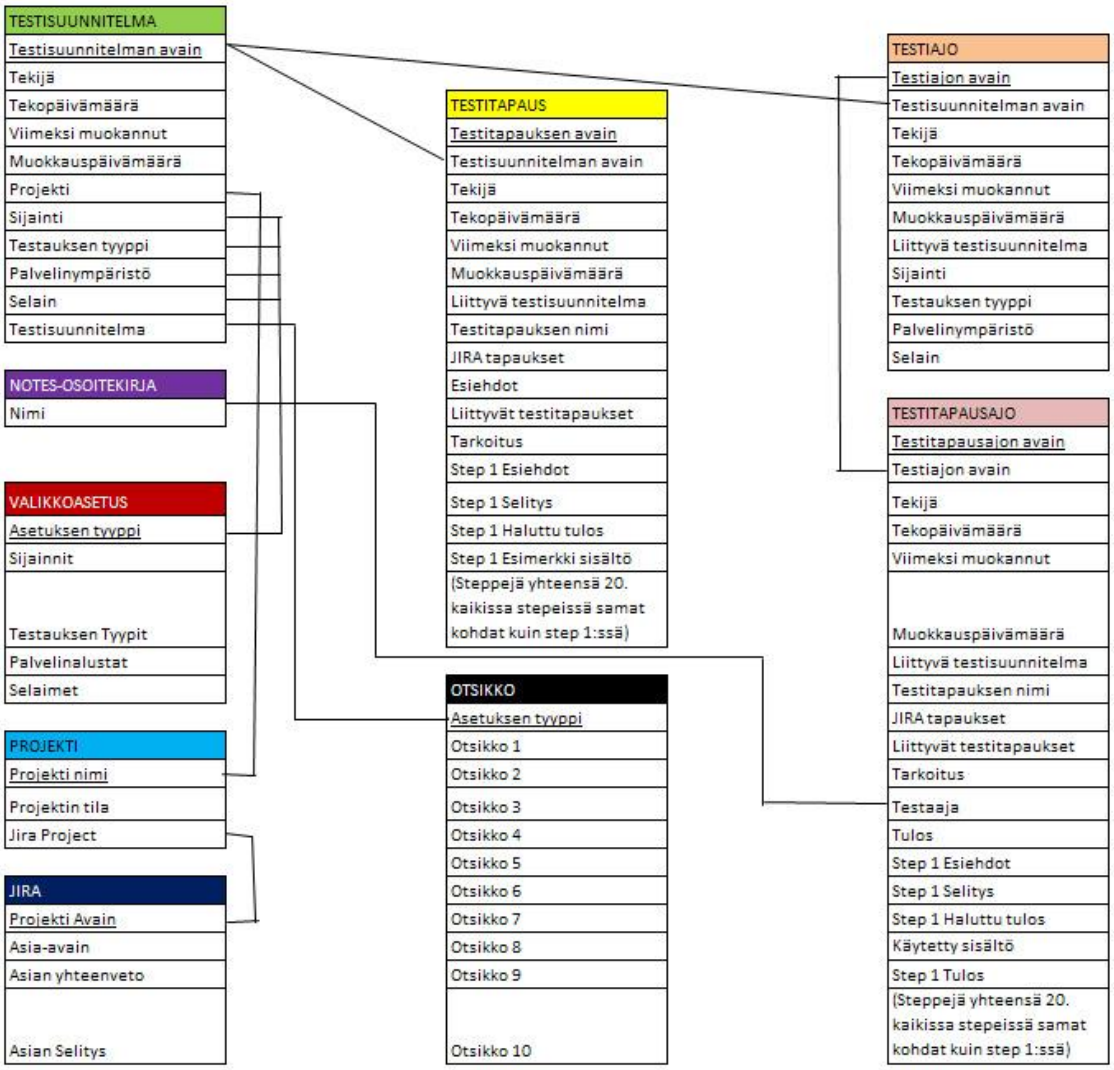
- Projektipäällikkö luo testisuunnitelman.
- Projektipäällikkö luo testiajon, jossa ei ole yhtään testitapausta
- Testaajat (kehittäjät) luovat tekemistään moduuleista tai niiden yhteensopivuuksista testitapauksia suunnitelman alle
- Testaajat (kehittäjät) testaavat tapauksia ja samalla liittävät ne projektipäällikön tekemään tyhjään testiajoon.

Järjestelmätestauksessa, jossa tarkoituksena testata koko sovelluksen toiminta prosessi kulkee seuraavasti:

- Projektipäällikkö luo testisuunnitelman.
- Projektipäällikkö ja testaajat luovat tai liittävät testitapauksia suunnitelman alle
- Projektipäällikkö luo testiajon johon hän liittää kaikki testitapaukset
- Testaajat testaavat testiajon alla olevat testitapaukset

7.2 Tietokantakuvaus

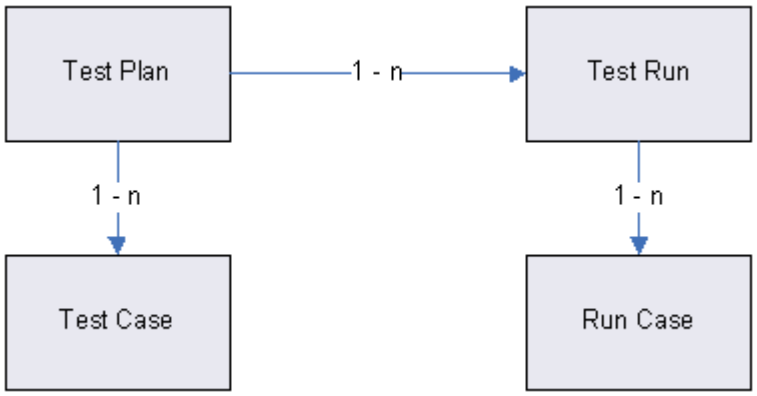
Test Management Tool –sovelluksessa käytettiin neljää päätietotaulua, jotka sisältävät testaamiseen liittyvät tiedo, kahta asetustietotaulua, jotka sisältävät asetuksien vaihtoehtoja ja kolmea ulkoista tietokantataulua, jotka ovat yhteydessä yrityksen eri sovelluksiin. Kuvio 8. esittää sovelluksen tietokantakuvauksen.



Kuvio 8. Tietokantakuvaus

7.3 Sovelluksen osa-alueet

Sovellus sisältää neljä pääosa-aluetta ja neljä apu-osa-aluetta joista kaksi osa-aluetta (JIRA ja PROJEKTI) ovat eri tietokannoissa. Pääosa-alueiden hierarkiaa esittää kuvio 8.



Kuvio 9. Osa-alueiden hierarkia (Lainattu: Jussi Korhonen. 2008)

Testisuunnitelma (Test plan) on ylin pääosa-alue. Projektista tehdään yksi testisuunnitelma joka testausta kohden. Testisuunnitelman alle tehdään testitapauksia (Test case) ja sen pohjalta tehdään testiajoja (Test run). Testiajojen alle tehdään testitapausajoja, jotka sitten testaajan toimesta ajetaan eli käydään tapauksia läpi ja annetaan arvio siitä, onko tapaus hylätty vai hyväksytty.

7.3.1 Testisuunnitelma

Testisuunnitelma on dokumentti, jonka projektipäällikkö luo prosessin aluksi. Dokumentti on tärkein, koska mitään muuta pääosa-alueen dokumenttia ei voida luoda, ennen kuin testisuunnitelma on luotu. Testisuunnitelman alle voidaan luoda uusi testitapaus tai sinne voidaan kopioida jo valmiina oleva testitapaus. Testisuunnitelman pohjalta tehdään myös testiajo. Kuvio 10. esittää testisuunnitelman tietokenttiä.

Testisuunnitelmaan haetaan seuraavat tiedot eri osa-alueista:

- Projekti
    - Projekti-tietokannasta

- Sijainti, testauksen tyyppi, palvelinympäristö ja selain
    - Valikkoasetus-taulusta
- Testisuunnitelma-kentän väliotsikot
    - Otsikko-taulusta

TESTISUUNNITELMA
Testisuunnitelman avain
Tekijä
Tekopäivämäärä
Viimeksi muokannut
Muokkauspäivämäärä
Projekti
Sijainti
Testauksen tyyppi
Palvelinympäristö
Selain
Testisuunnitelma

Kuvio 10. Testisuunnitelma

7.3.2 Testitapaus

Testitapaus-dokumentteja tehdään testisuunnitelman alle. Ne ovat yhteydessä suunnitelmaan testisuunnitelman avaimen avulla. Kenttä liittyvä testisuunnitelma näyttää liitoksissa olevan testisuunnitelman projektin ja testin tyypin. JIRA-tapaukset ovat ko. projektiin kuuluvia JIRA-tapauksia, jotka haetaan JIRA-tietokannasta. Liittyvät testitapaukset –kentässä käyttäjä saa valita liittyykö tämä testitapaus johonkin toiseen testisuunnitelman alla olevaan testitapaukseen.

Määrittelyssä mainittiin, että testitapausten kohdat tulisi antaa kohta kohdalta, stepeittäin. Steppejä on yhteensä 20. Testitapaus voidaan ajaa, jolloin testitapauksen tiedot kopioidaan testitapausajo -osa-alueeseen. Kuvio 11. esittää testitapauksen tietokenttiä.

TESTITAPAUS
Testitapauksen avain
Testisuunnitelman avain
Tekijä
Tekopäivämäärä
Viimeksi muokannut
Muokkauspäivämäärä

Liittyvä testisuunnitelma
Testitapauksen nimi
JIRA tapaukset
Esiehdot
Liittyvät testitapaukset
Tarkoitus
Step 1 Esiehdot
Step 1 Selitys
Step 1 Haluttu tulos
Step 1 Esimerkki sisältö
(Steppejä yhteensä 20. kaikissa stepeissä samat kohdat kuin step 1:ssä)

Kuvio 11. Testitapaus

7.3.3 Testiajo

Testiajo-dokumentilla määritellään kyseisen testiajon:

- Palvelinympäristö
    - Vaihtoehdot ovat määritelty testisuunnitelman palvelinympäristö-kohtaan
- Selain
    - Vaihtoehdot ovat määritelty testisuunnitelman selain-kohtaan

Käyttäjä voi valita vain yhden palvelinympäristön sekä selaimen. Tämä testiajodokumentti kertoo testaajalle millä palvelinalustalla ja selaimella hänen tulisi testi suorittaa. Dokumentilta pääsee katsomaan pohjana olevaa testisuunnitelmaa, jossa on kerrottu enemmän suunnitelmasta. Testiajo-dokumentin alle voidaan tuoda testitapauksia silloin kun testiajo-dokumentti luodaan tai silloin kun testitapaus ajetaan. Kuvio 12. esittää testiajon tietokenttiä

TESTIAJO
Testiajon avain
Testisuunnitelman avain
Tekijä
Tekopäivämäärä
Viimeksi muokannut
Muokkauspäivämäärä
Liittyvä testisuunnitelma
Sijainti
Testauksen tyyppi

Palvelinympäristö
Selain

Kuvio 12. Testiajo

7.3.4 Testitapausajo

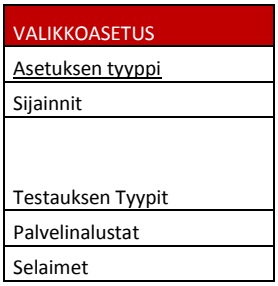
Testitapausajo luodaan joko silloin kun testitapaus ajetaan tai silloin kun testiajon alle tuodaan testitapauksia. Se on lähes identtinen kopio testitapauksesta. Käyttäjä ei kuitenkaan pysty määrittelemään kuin testaajan (haetaan Lotus Notesin osoitekirjasta), steppien esimerkkisisällön/käytetyn sisällön ja steppien tuloksen (Hylätty/Hyväksytty). Kun käyttäjä tallentaa dokumentin, lasketaan mitkä ovat steppien tulokset. Jos kaikki tulokset ovat hyväksyttyjä tulee koko testitapausajon tilaksi hyväksytty. Kuvio 13. esittää testitapausajon tietokenttiä.

TESTITAPAUSAJO
Testitapausajon avain
Testiajon avain
Tekijä
Tekopäivämäärä
Viimeksi muokannut
Muokkauspäivämäärä
Liittyvä testisuunnitelma
Testitapauksen nimi
JIRA tapaukset
Liittyvät testitapaukset
Tarkoitus
Testaaja
Tulos
Step 1 Esiehdot
Step 1 Selitys
Step 1 Haluttu tulos
Step 1 Esimerkki sisältö / Käytetty sisältö
Step 1 Tulos
(Steppejä yhteensä 20. kaikissa stepeissä samat kohdat kuin step 1:ssä)

Kuvio 13. Testitapausajo

7.3.5 Valikkoasetus

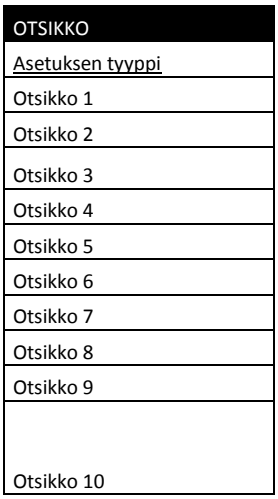
Valikkoasetus –dokumentilla määritellään vaihtoehdot testisuunnitelmalla oleviin sijainti-, testauksen tyyppi-, palvelinalusta- ja selain-kenttiin. Käyttäjä voi määrittellä kunkin kentän vaihtoehdot kirjoittamalla eri vaihtoehtoja rivivälein eroteltuna. Kuvio 14. esittää valikkoasetuksen tietokenttiä



Kuvio 14. Valikkoasetus

7.3.6 Otsikko

Otsikko –dokumentilla määritellään testisuunnitelmalla olevan testisuunnitelma-kentän väliotsikot. Kuvio 15. esittää otsikko-dokumentin tietokenttiä



Kuvio 15. Otsikko

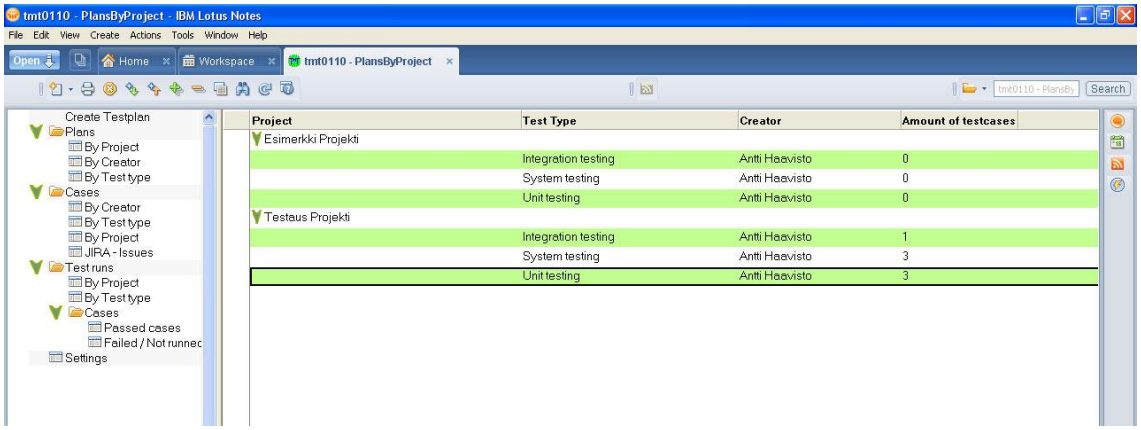
7.3.7 Ulkoiset osa-alueet

Ulkoisia osa-alueita ovat JIRA- ja projekti-tietokannat, jotka ovat demotietokantoja. Todelliset tietokannat ovat Descom Oy:n palvelimilla ja liitetään sovellukseen järjestelmätestauksen yhteydessä. Testaajan nimen hakemisessa käytetään Lotus Notes –ohjelmiston sisäistä osoitekirjaa.

7.4 Esimerkkejä sovelluksen käytöstä

Tässä kappaleessa esitellään kuvia sovelluksen toiminnasta. Aihe etenee kuvion 7. mukaisesti.

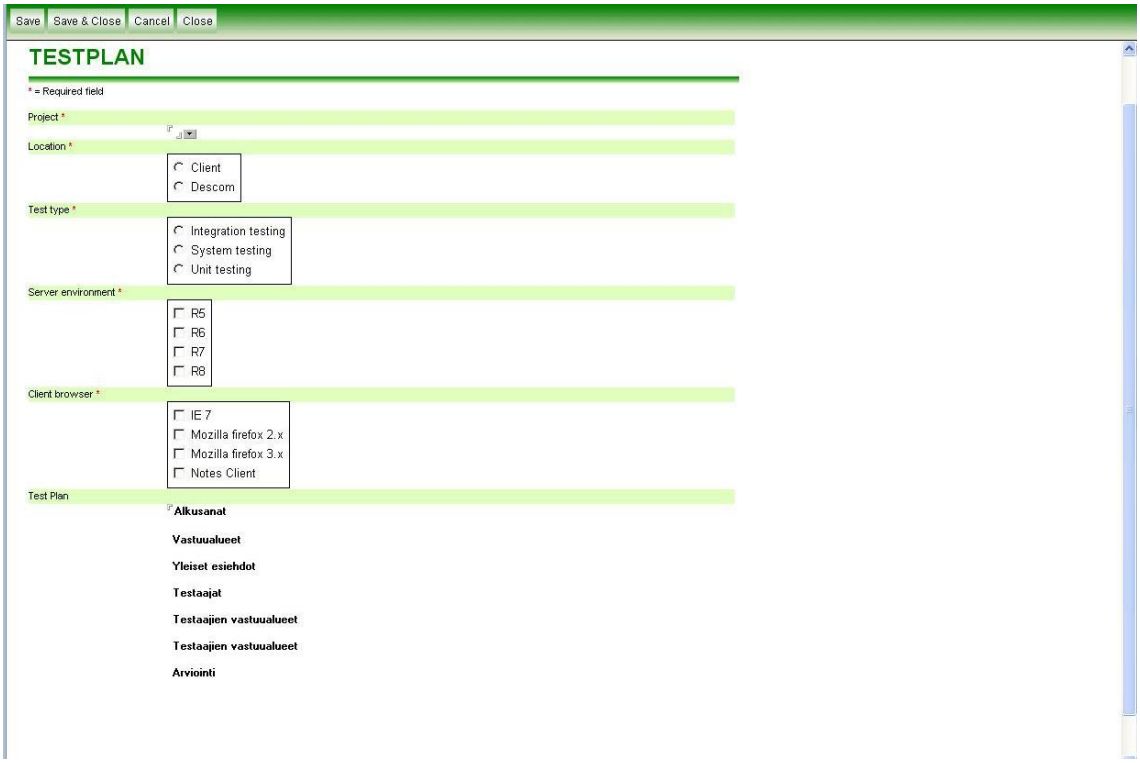
Sovellusta käytetään Lotus Notes-ohjelmiston vakionäkymän avulla.



Kuvio 16. Yleisnäkymä

7.4.1 Testisuunnitelma-lomake

Ensimmäinen testausdokumentti, joka luodaan on testisuunnitelma.



Kuvio 17. Tyhjä testisuunnitelma

Testisuunnitelmaan määritellään:

- Projekti, johon suunnitelma kuuluu,
- sijainti, jossa testaus tehdään,
- testauksen tyyppi,
- palvelinympäristöt, joissa testataan ja
- selain-ohjelmat, joita käyttämällä testataan.

Lisäksi testisuunnitelman luoja voi vapaamuotoisesti kirjoittaa testisuunnitelman.

Testisuunnitelmia tehdään yleensä kolme kappaletta projektia kohti; yksikkötestaus-, integraatiotestaus-, ja systeemitestaus-suunnitelmat.

Testisuunnitelma-lomakkeella, niinkuin kaikissa muissakin lomakkeissa on kirjoitus-tilassa neljä toimintonäppäintä;

- Save (Tallenna),

    - Tallentaa asiakirjan
- Save & Exit(Tallenna ja sulje),
    - Tallentaa asiakirjan ja sulkee sen
- Cancel (Peru)
    - Menee takaisin lukutilaan.
- Close (Sulje)
    - Sulkee asiakirjan. Jos asiakirjaa on muokattu kysyy tallennetaanko muutokset

Kuvio 18. Toimintonäppäimet kirjoitustilassa

Lukutilassa testisuunnitelma-lomakkeella on viisi toimintonäppäintä:

- Add testcase (Luo testitapaus),
    - Avaa tyhjän testitapauslomakkeen
- Add existing testcase (Liitä olemassa oleva testitapaus),
    - Kysyy minkä testitapauksen käyttäjä haluaa lisätä ja avaa valitun testitapauksen kopion.
- Create testrun (Luo testiajo),
    - Kysyy mitä testitapauksia käyttäjä haluaa liittää testiajon alle ja avaa tämän jälkeen testisuunnitelman pohjalta luodun testiajo-asiakirjan.
- Close (Sulje),
- Edit (Muokkaa)
    - Avaa ko. asiakirjan kirjoitustilaan.

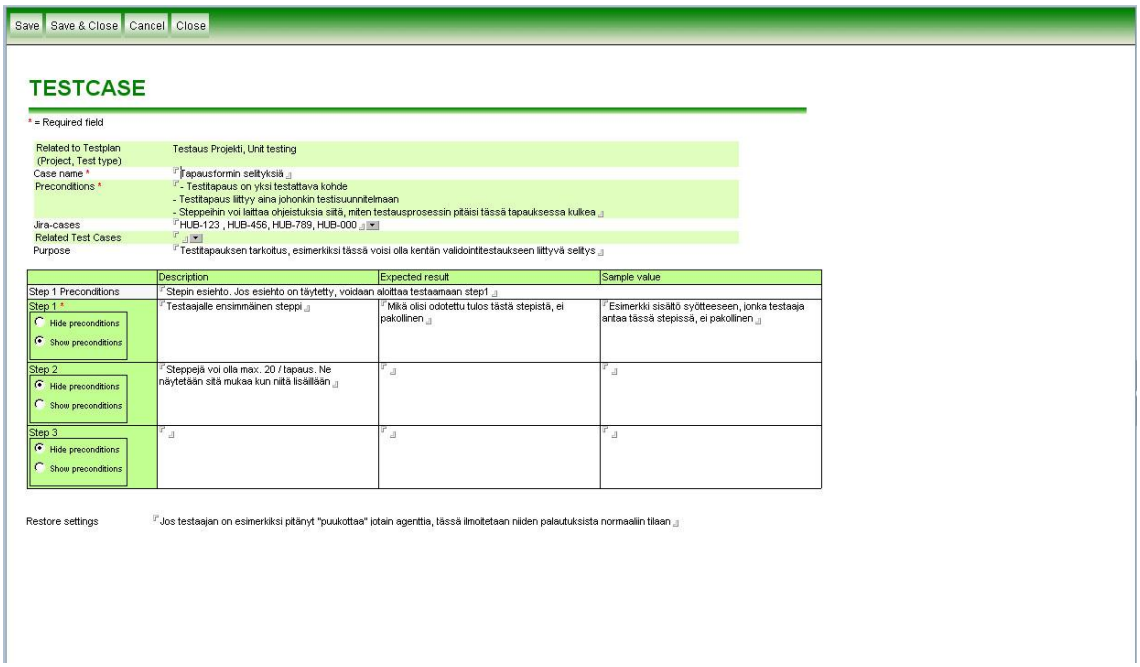
Kuvio 19. Toimintonäppäimet lukutilassa

Kun testisuunnitelma on luotu ja tallennettu, sen alle voidaan lisätä uusia testitapauksia tai liittää valmiina olevia testitapauksia.

Yksikkötestauksessa on myös yleistä, että suunnitelman pohjalta luodaan testiajo-dokumentti, johon sovelluksen kehittäjät voivat liittää tekemiään osa-alueita ja testata nämä. Testiajo-lomake esitellään luvussa 7.4.4

7.4.2 Testitapaus-lomake

Testitapaukselle annetaan nimi, testitapauksen esiehdot, testitapaukseen liittyvät JIRA- ja testitapaukset. Jotta tapauksen tarkoitus tulisi testaajalle tietoon on hyvä myös lisätä testitapauksen tarkoitus.



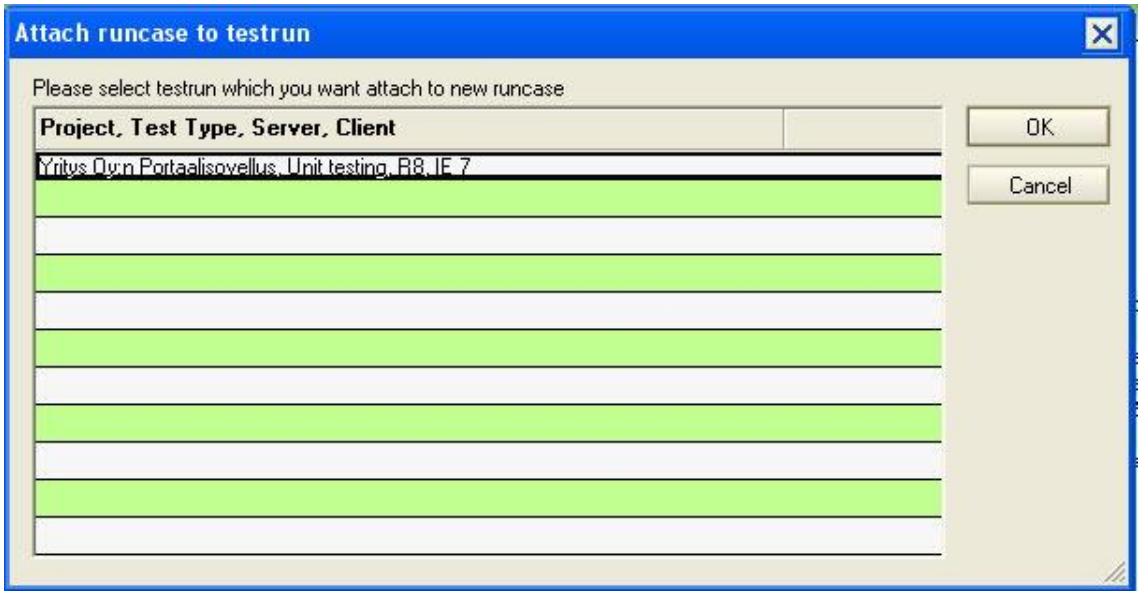
Kuvio 20. Täytetty testitapaus

Yksi sovelluksen vaatimuksista oli, että tapaukset pitää saada kirjattua kohta kohdalta, eli stepeittäin. Tätä varten testitapaus-lomakkeella on mahdollisuus kirjata 20 steppiä, jokaiselle stepille esiehto, kuvaus, haluttu tulos ja esimerkkitulos. Stepit tulevat näkyviin sitä mukaa kun niitä täytetään. Tarkemmin sanottuna; kun poistutaan stepin kuvaus-kentästä, tulee seuraava steppi näkyviin. Testitapaus-lomakkeella on myös palautusohje-kenttä, johon testitapauksen tekijä kirjoittaa mahdolliset ohjeet testausympäristön normalisointiin.

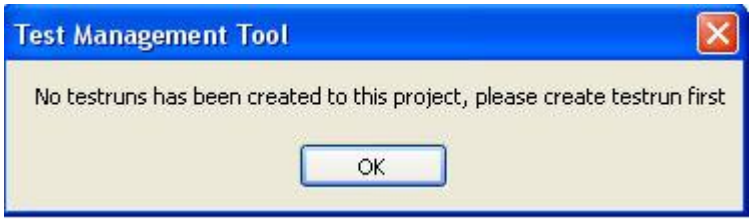
Testitapaus-lomakkeen ollessa lukutilassa toimintonäppäimiä on kolme kappaletta:

- Run (Aja),
    - Kysyy mihin testiajoon ko. testitapaus liitetään ja tämän jälkeen avaa testitapausajo-lomakkeen.
- Close (Sulje),
- Edit (Muokkaa)

Testitapauksia voidaan ajaa, eli testata testitapauksen ohjeistama prosessi ja arvioida toimivatko kaikki kohdat halutulla tavalla. Tämä testauksen kirjaaminen tehdään testitapausajo-lomakkeella. Testitapausajon pitää kuulua johonkin testiajoon, joten sitä kysytään kun käyttäjä painaa Run-näppäintä. Jos testiajoja ei ole luotu ko. projektille, tulee sen tarpeesta ilmoitus ja käyttäjän on tehtävä testiajo-dokumentti ennen kuin pystyy yksittäisen tapauksen testaamaan.



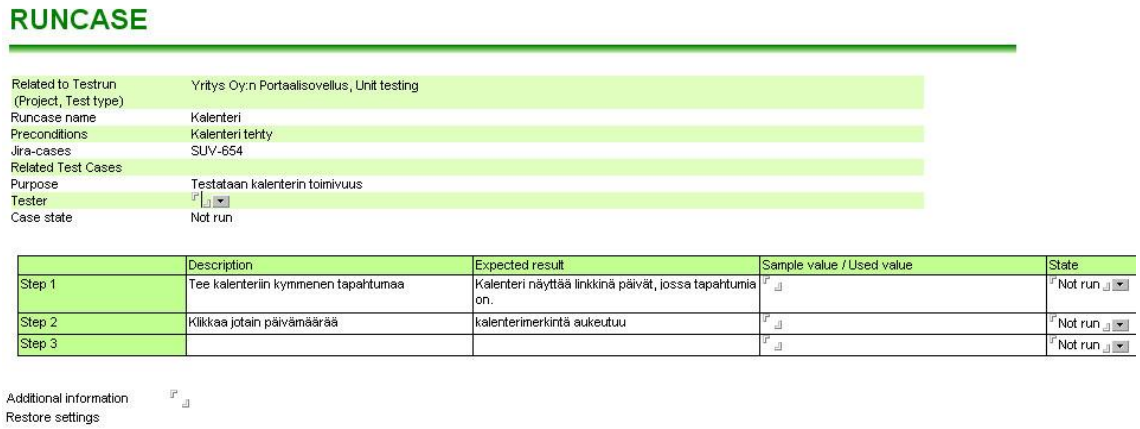
Kuvio 21. Valintaikkuna, jossa käyttäjä voi liittää testitapauksen haluamaansa testiajoon.



Kuvio 22. Virheilmoitus, joka kehoittaa käyttäjää luomaan testiajo-dokumentin ensin.

7.4.3 Testitapausajo-lomake

Kun testiajo-dokumentti on valittu testitapausajo-lomake avautuu. Testaajan pitää määritellä testaajan nimi ja stepeittäin testauksessa käytetty sisältö ja stepin tila. Kun testaaja tallentaa testitapauksen, lasketaan koko testitapausajon tila; jos kaikkien steppien tila on läpäisty, tulee tapauksen tilaksi läpäisty. Muuten tilaksi tulee ei ajettu. Testaaja voi kirjoittaa lisätietoja liittyen tapaukset testaukseen.



RUNCASE

Related to Testrun (Project, Test type)	Yritys Oy:n Portaalisovellus, Unit testing
Runcase name	Kalenteri
Preconditions	Kalenteri tehty
Jira-cases	SUV-654
Related Test Cases	
Purpose	Testataan kalenterin toimivuus
Tester	
Case state	Not run

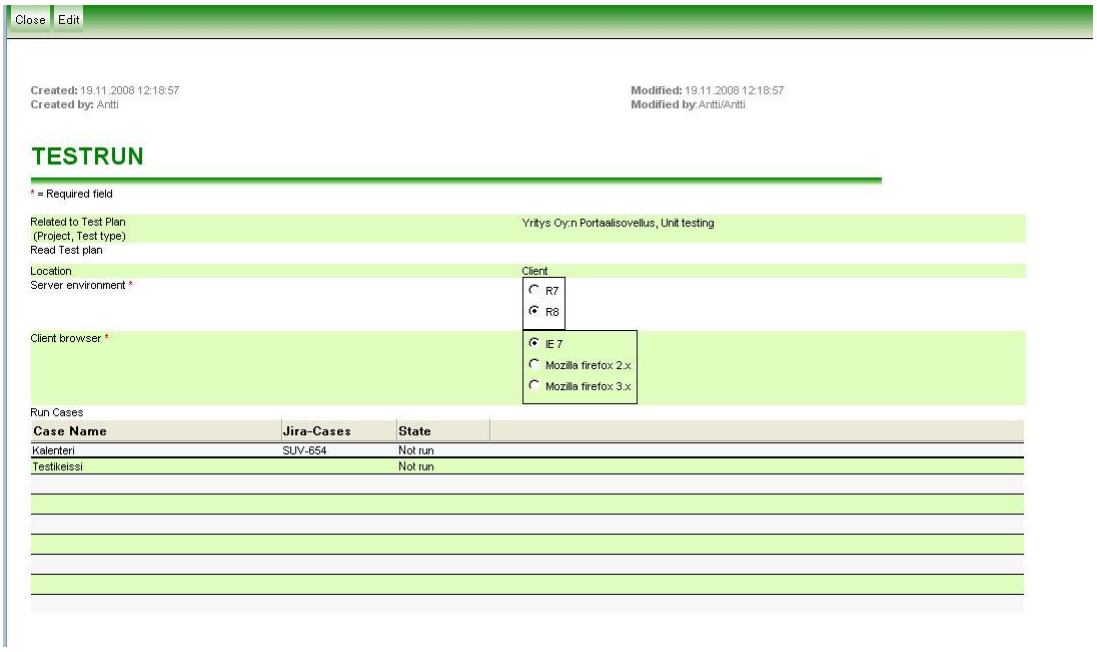
	Description	Expected result	Sample value / Used value	State
Step 1	Tee kalenteriin kymmenen tapahtumaa	Kalenteri näyttää linkkinä päivät, jossa tapahtumia on.		Not run
Step 2	Klikkaa jotain päivämäärää	kalenterimerkintä aukeutuu		Not run
Step 3				Not run

Additional information
Restore settings

Kuvio 23. Testitapausajo-lomake kirjoitustilassa

7.4.4 Testiajo-lomake

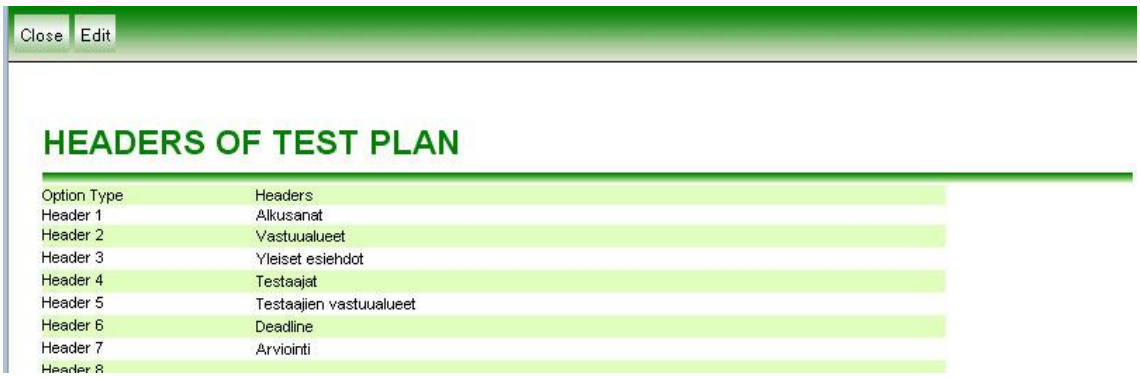
Testiajo-lomakkeelle määritellään missä palvelinalustalla ja millä selaimella testaus suoritetaan. Nämä tiedot ovat määritelty testiajoon liittyvällä testisuunnitelmalla. Lomakkeella näytetään myös testiajoon liittyvät testitapausajot.



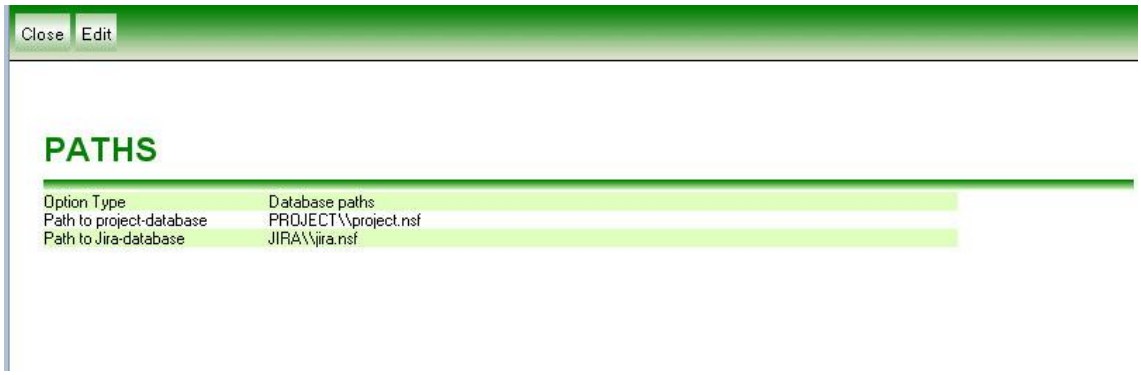
Kuvio 24. Testiajo-lomake

7.2.5 Asetukset-lomakkeet

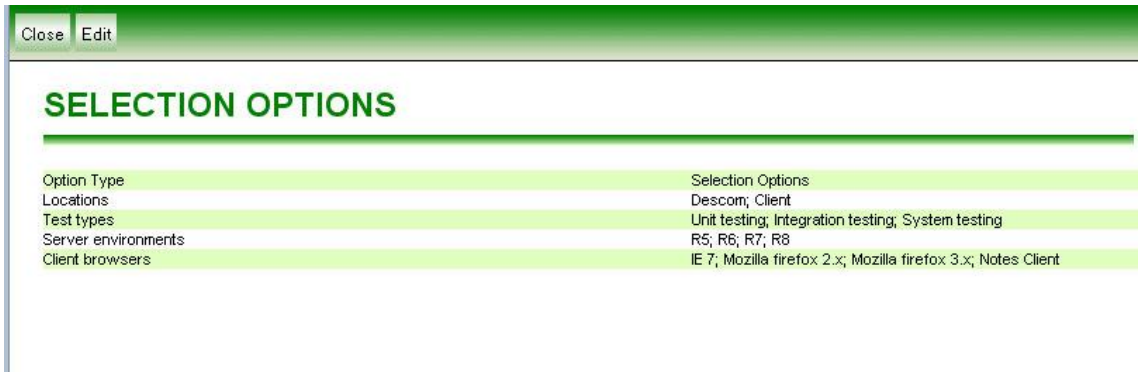
Sovelluksessa on kolme asetuslomaketta; testisuunnitelma-kentän otsikkojen muokkaukseen, ulkoisten tietokantojen polkujen muokkaukseen ja testisuunnitelma-lomakkeen monivalinta-kenttien vaihtoehtojen muokkaukseen.



Kuvio 25. Otsikoiden muokkauslomake



Kuvio 26. Ulkoisten tietokantojen polkujen muokkauslomake



Kuvio 27. Monivalinta-kenttien muokkauslomake

8 YHTEENVETO

Mitä ohjelmistokehitysmenetelmää siis käyttää, kun esimerkiksi aloittaa ohjelmistoja tuottavan yrityksen? Kyse on mielestäni ohjelmisto- tai sen kehitys-projektin suuruudesta. Jos kyse on isosta projektista, jossa on paljon avoimia kysymyksiä, asianhaaroja ja investointia pelissä, tulisi käyttää struktuurista ohjelmistokehitysmenetelmää. Itse Scrumin kehittäjä Ken Schwaber oli kehoittanut pankkia käyttämään struktuurista menetelmää, kun he olivat päivittämässä uutta sovellusta jossa oli paljon rahaa ja selvittämättömiä kysymyksiä. Tämä siksi, koska struktuurisessa menetelmässä käytetään suurin osa ajasta riskien arvioimiseen, määrittämiseen ja suunnitteluun. Jos ko. pankki olisi alkanut viemään projektia eteenpäin käyttämällä Scrumia, olisi projekti mahdollisesti riistäytynyt käsistä ja tuottanut suuret tappiot yritykselle ja kenties sen asiakkaille. (Googletechtalks. 2007.)

Nykypäivänä, kun monia sovelluksia on siirtynyt internetiin ja tämä trendi tuntuu kiihtyvän, Scrum on hyvä vaihtoehto. Se sopii pieniin ja keskisuuriin projekteihin silloin kun tiimeissä on asiantuntevia ihmisiä. Scrumin avulla voidaan vastata asiakkaiden tai markkinoiden vaatimaan muutokseen kivuttomammin kuin struktuurisissa menetelmissä.

9 LOPPUTULOKSEN ARVIOINTI

Test Management Tool:sta syntyi Descom Oy:n tarpeita palvelemaan räätälöity testauksen työkalu. Projektin rajattu aikataulu ja henkilökohtaiset rajatut taidot johtivat siihen, että sovellus ei ole täysin virheetön tässä vaiheessa.

Työn teoriaosa avaa lukijalle perustiedot ketteristä menetelmistä ja etenkin Scrum-menetelmästä. Se ei ole kattava, kaiken kertova opas, mutta se voi toimia yhdessä lähdeluettelon kanssa ponnahduslautana ketterien menetelmien maailmaan.

LÄHDELUETTELO

Agilecollab. 2008. The “WHY” of Sprint Backlog : The Spirit of Sprint Backlog. [verkkoartikkeli], [viitattu 05.11.2008] Saatavissa:
http://www.agilecollab.com/tags/backlog

Agile Software Development. 2007. Disadvantages of Agile Software Development [verkkoartikkeli],
[viitattu 01.10.2008]. Saatavissa:
http://www.agile-software-development.com/2007/09/disadvantages-of-agile-software.html

Artem Archenco. 2008. Video tutorial: Managing your Sprint Backlog with the help of a simple Excel sheet. [verkkovideo],[viitattu 23.10.2008] Saatavissa:
http://agilesoftwaredevelopment.com/videos/scrum/simple-sprint-backlog

Beck, K. & Paulk, M. 2004. Agile Software Development : Evaluating the Methods for Your Organization. Artech House, Incorporated.

Benediktsson, O., Dalcher, D. ja Thorbergsson, H. 2006. Comparison of software development life cycles: a multiproject experiment. [verkkoartikkeli],
[viitattu 30.9.2008].Saatavissa:
http://web.ebscohost.com.lillukka.samk.fi/ehost/pdf?vid=4&hid=21&sid=18a25135-a1df-4727-a659-7f6d799e5d0e%40sessionmgr8

Gavaldo, Eric. 2008. Scrum. [verkkoartikkeli],
[viitattu 03.10.2008]. Saatavissa:
http://www.xqual.com/documentation/tutorial_scrum.html

Googletechtalks. 2007. Scrum et al.[verkkovideo],
[viitattu 23.10.2008] Saatavissa:
http://www.youtube.com/watch?v=IyNPeTn8fpo

Gravendeel, E & Ramos, C. 2008. Top 9 Challenges of Adopting Scrum [verkkoartikkeli],[viitattu 07.10.2008]. Saatavissa: http://www.agilejournal.com/content/view/837/111/

Great Place to Work Institute. 2008. Euroopan 100 parasta työpaikkaa 100 Best Workplaces in Europe. . [verkkoartikkeli], [viitattu 07.10.2008]. Saatavissa: http://www.greatplacetowork.fi/best/list-eu.htm

Heidema, J. 2008. Book Commentary by Jim Heidema [verkkoartikkeli], [viitattu 01.10.2008]. Saatavissa: http://www.agileadvice.com/

Hentzen, Whil. 2002. Software Developer's Guide (3rd Edition). Hentzenwerke Publishing, Inc.

Huttunen, Janne. 2006. Diplomityö, Ketterän ohjelmistokehitysmenetelmän määrittely, vertailu ja käyttäjäkysely

Korhonen, Jussi. 2008. Dokumenttien suhteista. [sähköpostiviesti]. Vastaanottaja: antti.haavisto@student.samk.fi. Lähetetty 16.9.2008 klo 16:10. [Viitattu 15.11.2008].

Kuha, J. 2008. Tehokas Java EE-sovellustuotanto. Docendo

Landay, James. 2001.The Software Life Cycle [verkkoartikkeli], [viitattu 26.9.2008].Saatavissa: http://bmrc.berkeley.edu/courseware/cs169/spring01/lectures/lifecycle/tsld011.htm

Lindström, Jukka. 2006. Scrum. [verkkoartikkeli], [viitattu 06.10.2008]. Saatavissa: http://www.ri.fi/web/fi/teknologia-ja-tutkimus/scrum

Mountain Goat Software. 2008a. Sprint Planning Meeting. [verkkoartikkeli],

[viitattu 06.10.2008]. Saatavissa:
http://www.mountaingoatsoftware.com/sprint-planning-meeting

Mountain Goat Software. 2008b. Sprint Review Meeting. [verkkoartikkeli],
[viitattu 09.10.2008]. Saatavissa:
http://www.mountaingoatsoftware.com/sprint-review-meeting

Mountain Goat Software. 2008c. Release Burndown. [verkkoartikkeli],
[viitattu 10.11.2008]. Saatavissa:
http://www.mountaingoatsoftware.com/release-burndown

Poimala, Sami. Scrum - mahdollisuuksien taide. [verkkoartikkeli],
[viitattu 03.10.2008]. Saatavissa:
http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/

Schuh. 2004. Integrating Agile Development in the Real World. Charles River Media.

Scrum Alliance, Inc. 2008a. What Is Scrum? [verkkoartikkeli],
[viitattu 02.10.2008]. Saatavissa:
http://www.scrumalliance.org/pages/what_is_scrum

Scrum Alliance, Inc. 2008b. Scrum Roles [verkkoartikkeli],
[viitattu 02.10.2008]. Saatavissa:
http://www.scrumalliance.org/view/scrum_roles

Scrum Alliance, Inc. 2008c. Scrum Roles. [verkkoartikkeli],
[viitattu 03.10.2008]. Saatavissa:
http://www.scrumalliance.org/view/scrum_artifacts

Scrum Alliance, Inc. 2008d. Scrum Ceremonies. [verkkoartikkeli],
[viitattu 03.10.2008]. Saatavissa:
http://www.scrumalliance.org/view/scrum_ceremonies

SwBusiness. 2007. Väitös: Havaintoja ohjelmistotestauksen käytännöistä.

[verkkoartikkeli],
[viitattu 24.10.2008] Saatavissa:
http://www.swbusiness.fi/portal/research_insights/research_news/?id=20022

Szalvaym, Victor. 2007. Glossary of Scrum Terms [verkkoartikkeli],
[viitattu 10.11.2008]. Saatavissa:
http://www.scrumalliance.org/articles/39#1127

Toikkanen, T. 2005. Don’t draw diagrams of wrong practices - or: Why people still believe in the Waterfall model. [verkkoartikkeli],
[viitattu 01.10.2008]. Saatavissa:
http://tarmo.fi/blog/2005/09/09/dont-draw-diagrams-of-wrong-practices-or-why-people-still-believe-in-the-waterfall-model/

Van Roosmalen, Ralph. 2007. Scrum and Testing. [verkkoartikkeli],
[viitattu 16.10.2008]. Saatavissa:
http://www.scrumalliance.org/resources/270

Weisert, C. 2003.There's no such thing as the Waterfall Approach! (and there never was) [verkkoartikkeli],
[viitattu 01.10.2008]. Saatavissa:
http://www.idinews.com/waterfall.html

1. Johdanto

Dokumentti on lyhyt sanallinen kuvaus opinnäytetyönä kehittettävän Test Management Tool -sovelluksen vaatimuksista ja toiminnallisuudesta.

2. Kehitysympäristö

Sovellus tehdään Notes Domino 8 ympäristöön. Sovellusta käytetään Notes clientillä. Web liittymää ei tässä vaiheessa kehitetä, koska todennäköisesti web-liittymä koodataan myöhemmin suoraan WebSphere portal ympäristöön (Descomin Intranet) soveltuvaksi portletiksi.

3. Osa-alueet

3.1 Testisuunnitelma (Test Plan)

Testisuunnitelma on yksi notes dokumentti, jossa kuvataan perustiedot ko. projektin testauksesta.

Projekti (valintalista)

Testauksen tyyppi (valintalista asetuksista [Toimintotestaus, Järjestelmätestaus])

Testausympäristön sijainti (asiakas vai Descom)

Palvelinympäristö (valintalistaan asetuksista lista domino versioista, R6.5, R7, R8)

Client ympäristö (valintalistaan asetuksista lista clienteistä, R6.5, R7, R8, Firefox 2, IE 7, …)

Testisuunnitelma (RT –kenttä, jossa on asetuksissa määritellyt väliotsikot oletusarvona, esim. kaikille testitapauksille yhteiset esiehdot)

Testitapaukset (embedded view tms., joka listaa ko. suunnitelmaan liitetyt testitapaukset) + nappi testitapausten lisäämiseksi ko. suunnitelman alle.

Testisuunnitelmat testitapauksineen pitää voida kopioida uudelle projektiversiolle. Käyttäjä voi valita, että kaikkia testitapaukset kopioidaan tai sitten tietyt testitapaukset. Näin olemassa olevia testitapauksia voidaan hyödyntää jatkokehitysprojekteissa testauksen suunnittelun pohjana.

3.2 Testitapaus (Test Case)

Testitapaus on yksi notes dokumentti, jossa kuvataan yksittäinen testitapaus. Testitapaus liittyy aina testisuunnitelmaan.

Esiehdot (RT kenttä, jossa on asetuksissa määritellyt väliotsikot oletusarvona, esim. testitapauskohtaiset esiehdot.)

Jira issuet (valintalistasta (multivalue) valitaan jira-issueit, jotka tämä testitapaus kattaa)

Testitapauksen stepit taulukkona. Tämän toteutustapaa pitää pohtia. Yksinkertaisin versio olisi RT kenttä, jossa on oletuksena taulukko ao. headereilla. Toisaalta raporttiexportteja voisi olla helpompi tehdä / muotoilla jos testitapaus koostuu joukosta erillisiä kenttiä.

- Stepin numero | Steppi | Expected result | Sample Value

3.3 Testiajo (Test suite / run)

Testiajolla tarkoitetaan toiminnallisuutta, jossa valitun testisuunnitelman pohjalta luodaan (kopiointi) joukko testitapauksia testaajan suoritettavaksi. Ts. testitapausdokumenteista luodaan uudet dokumentit hiukan eri sisällöllä ja testaaja raportoi testauksen statuksen ko. dokumentille. Näitä dokumentteja käytetään raporttinäkymillä testauksen tulosten arviointiin ja näyttämiseen.

Testitapauskenttien lisäksi näytetään statuspainikkeet ja lisätietoja kenttä sekä ko. runiin liittyvät metatiedot.

Metatiedot:

Testaaja (osoitekirjasta valittu henkilö)

Testaustyyppi (tulee testisuunnitelmasta)

Palvelinympäristö (valintalistaan asetuksista lista domino versioista, R6.5, R7, R8)

Client ympäristö (valintalistaan asetuksista lista clienteistä, R6.5, R7, R8, Firefox 2, IE 7, …)

Taulukossa status on voitava kirjata stepeittäin ja lopuksi **koko** testitapaus saa statuksen pass, fail tai not run.

- Stepin numero | Steppi | Expected result | Result (pass, fail, not run)

Steppien alla näytetään lisäksi tekstikenttä lisätiedoille. Tätä testaaja käyttää kun haluaa välittää jotain lisätietoa / kysymyksiä yms.

Huom! Testiajoissa sama dokumenttihierarkia kuin testisuunnitelmissa ja testitapauksissa. Eli Testiajodokumentti on kopio testisuunnitelmasta ja testiajodokumentin alle tulevat kopiot testitapauksista. Testiajodokumentilla on suunnitelman kenttien lisäksi yllä mainitut metatiedot sekä ko. testiajon status (Not run, In Progress, Complete).

Huom! Testaus pitää voida suorittaa jouhevasti, ilman että testitapauksia pitää kaivella näkymiltä. Siksipä voisi olla esim. niin, että testiajodokumentilla on ”Start test” nappi, joka

avaa järjestyksessä ensimmäisen testitapauksen suoritettavaksi ja kun ko. testitapauksen statuksen on asettanut, voisi valita esim. ”close & open next” napin, joka avaa testitapauksia kunnes ne loppuvat ko. testiajosta.

3.4 Raportointi

Raportointi hoidetaan tässä vaiheessa Notes näkymillä. Näkymiä tarvitaan yleiskäsityksen muodostamiseen, eli esim. projektipäällikkö voi nopeasti katsoa, onko projekti riittävän hyvässä kunnossa asiakkaalle toimitettavaksi. Käytetään tarvittaessa ikoneita tuomaan lisää visuaalista selkeyttä.

3.4.1 Testisuunnitelmanäkymät

Suunnitelmat projekteittain

Suunnitelmat henkilöittäin (kuka luonut)

Suunnitelmat testaustyypeittäin

3.4.2 Testitapausnäkymät

Testitapaukset suunnitelmittain

Testitapaukset henkilöittäin

Testitapaukset testaustyypeittäin (saadaan suunnitelmadokumentilta)

Testitapaukset projekteittain

Issuet projekteittain ja testitapauksittain (eli ne joille on testitapaus ja joille ei ole)

3.4.3 Testiajojen näkymät

- Testiajot projekteittain ja statuksittain
    - Projekti (Categ.) | Status (Categ.) | Testiajo
- Testiajojen testitapaukset statuksittain
    - Projekti (Categ.) | Testiajo (Categ.) | Testitapauksen status (Categ.) | Testitapaus

4. Testauksen suorittamisesta

Järjestelmätestauksessa on luontevaa, että tehdään uusi testiajo, johon valitaan kaikki testitapaukset ja aloitetaan testaus.

Toimintotestauksessa sen sijaan testausta tehdään toteutustyön ohessa. Aina kun vaatimus on valmis, toteuttaja suunnitelee testitapauksen ja suorittaa sen. Tämä aiheuttaa haasteen testiajojen käyttämiselle.

Asia voisi hoitua esim. seuraavasti. Kun projektipäällikkö laatii testisuunnitelman, tekee hän toimintotestausta varten myös perussetin testiajodokumentteja, joihin ei vielä liity yhtään testitapausta. Kun kehittäjä suunnittelee testitapauksen, voisi hän sen jälkeen suorittaa toiminnon 'Suorita testitapaus', joka kysyy käyttäjältä, mihin testiajoon ko. suoritus kytketään. Pop-up ikkunassa näytettäisiin siis lista projektipäällikön laatimista testiajoista ja näin myös 'adhoc' tyyppinen toimintotestaus saataisiin koottua jonkin testiajon sisään ja kokonaisuus olisi siten projektipäällikön arvioitavissa ja seurattavissa. Jos testiajodokumentteja ei löydy, käyttäjää kehotettaisiin kääntymään projektipäällikön puoleen asian hoitamiseksi.

5. Muuta

5.1 Kaikille dokumenteille yhteiset metatiedot

- Henkilö, joka luonut dokumentin
- Henkilö joka muokannut viimeksi dokumenttia
- Viimeisin muokkauspäiväys
- Dokumentin luontipäiväys

6. Liitynnät muihin järjestelmiin

Projektikanta

- projektilistat

Jira Gateway

- jira issuelistat

Saataisko myös projektit suoraan jira gateway kannasta? (Jussi)

6.1 Layout

Antti laatii layoutin omatoimisesti eli kenttien asettelut, formien värimaailmat jne.

7. Avoimia kysymyksiä

7.1 Testitapaukset

- Näytetäänkö testitapauksella jira issueista muuta kuin KEY

- Esim. linkk ko. issueen tai KEY+Summary+Description?

- Pitääkö testitapausten voida versioitua? Pitää pohtia, mutta ainakaan workflow (draft, approved) ei tässä vaiheessa tunnu tarpeeliselta.

7.2 Testiajot

- Pitäskö näyttää liitännäiset bugit ja niiden statukset sekä missä runissa löytynyt?

- Näytetäänkö testitapauksella jira issuet, jotka kirjattu bugeina ko. testitapauksen suorittamisen jälkeen? Näytetäänkö myös issuen status?

7.2.1 Dashboard

Dashboard sivu näyttää koosteen henkilökohtaisesta testaustilanteesta, eli mitkä testitapaukset on ko. henkilölle assignoitu ja mitkä ovat niiden statukset. Tämän sivun (formin?) kautta myös projektipäällikkö näkisi ko. henkilön testaustyökuormaa, eli ei anna liikaa hommia jne.

Esim. lista tapauksista, jotka vielä suorittamatta.

- assignoidaanko testitapaukset? -> test runissa valitaan roolit, eli kuka testaa (tieto kopioidaan runin testitapauksille)

Tämä voisi itseasiassa olla myös näkymä. Jos käytetään Notes Clientin default outlinea eli näkymälistausta, niin siinä voisi olla foldereita, joilla näkymät ryhmitellään sopiviin ryhmiin.

Esim.

My Tests (sopiva joukko näkymiä, joita testaaja käyttää työssään)

- Open Test Runs (Näkymä, jolta näkee ne testiajot, jotka on assignoitu mulle, mutta ovat vielä aloittamatta tai työnalla)

- Complete Test Runs

Projects (sopiva joukko näkymiä, joita projektipäällikkö käyttää työssään

- Xyz

Nämä folderit olisivat sitten luvussa 3.4 speksatut näkymät

Test Plans

Test Cases

Test Runs

7.3 Project home

- Testikattavuuden arviointi, eli yhteenvetoa runien tilanteesta ja tuloksista
- Ko. projektin testeissä löytyneet bugit ja niiden statukset

7.4 Jatkoa (pieni prioteetti)

- Jira integraatio (jira -> test mgm)
- Testauksen aikataulun suunnitteluja seuranta sekä muistutukset?
- Raporttiexportit
  - Testitapaukset word / pdf muotoon asiakkaalle lähetettäväksi
  - Testittulokset word / pdf muotoon asiakkaalle lähetettäväksi

7.5 Käyttäjäroolit

Alkuvaiheessa ei ole tarpeen määritellä käyttäjärooleja ja/tai dokumenttien luku/muokkausoikeuksia. Tämä siksi, että halutaan että käyttäjät voivat ottaa mallia toisten käyttäjien laatimista testitapauksista.