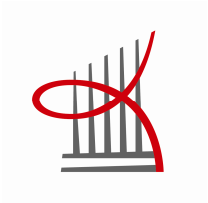


JÄRJESTELMÄINTEGRAATIOIDEN TESTAAMINEN

Case: Solteq Oyj

Korpinen Katja

Opinnäytetyö
Marraskuu 2011
Ylempi AMK
Tietojärjestelmäosaaminen
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

SISÄLLYS

1 JOHDANTO 6

2 TESTAUS YLEENSÄ 8

2.1 Mitä testaus on? 8

2.2 Testauksen tarkoitus, mitä testauksella tavoitellaan 10

2.3 Testauksen periaatteet & onnistunut testaus 13

2.4 Testaaminen käytännössä 16

2.5 Testaustyypit 23

2.5.1 Yksikkötestaus (unit testing) 23

2.5.2 Integraatiotestaus 25

2.5.3 Järjestelmätestaus (system testing) 28

2.5.4 Hyväksyntätestaus (acceptance testing) 30

3 JÄRJESTELMÄINTEGRAATIOT 31

3.1 Mitä järjestelmäintegraatiot ovat? 31

3.2 Miksi järjestelmäintegraatiot ovat tarpeellisia? 33

3.3 Integraatioratkaisuiden perusarkkitehtuuri 36

3.4 Integraatioratkaisuiden haasteet 39

4 JÄRJESTELMÄINTEGRAATIOIDEN TESTAAMINEN 42

4.1 Mitä järjestelmäintegraatiotestaaminen on? 42

4.2 Järjestelmäintegraatioiden testaamisen haasteet 43

4.3 Näkökulmia järjestelmäintegraatiotestaukseen 45

4.3.1 Järjestelmäintegraatioiden arkkitehtuuri 45

4.3.2 Järjestelmäintegraatioiden haasteet 47

4.3.3 Muut testaustyypit 48

4.3.4 Liiketoiminnan näkökulma 51

5 CASE: JÄRJESTELMÄINTEGRAATIOIDEN TESTAAMINEN SOLTEQ OYJ:SSÄ 52

5.1 Taustaa ja tavoite 52

5.2 Tutkimusstrategia eli keinot tavoitteen saavuttamiseksi 56

5.3 Tutkimustulos eli ohjeistus järjestelmäintegraatioiden testaamiseksi Solteqin myymäläjärjestelmäympäristössä 57
5.3.1 Sanomakeskeisen testaamisen menetelmät 58
5.3.2 Sisältökeskeisen testaaminen menetelmät 59
6 JOHTOPÄÄTÖKSET 62
LÄHTEET 64
LIITTEET 65

TIIVISTELMÄ

Tampereen ammattikorkeakoulu

Ylempi AMK

Tietojärjestelmäosaamisen koulutusohjelma

Katja Korpinen: Järjestelmäintegraatioiden testaaminen, Case: Solteq Oyj

Opinnäytetyö 64 s., liitteet 11 s.

Marraskuu 2011

Tässä opinnäytetyössä tutustutaan testauksen ja järjestelmäintegraatioiden teoriaan ja pohditaan muun muassa niiden valossa, miten järjestelmäintegraatioita tulisi testata. Teoriaosuuden lisäksi työ sisältää case-tutkimuksen, jonka tavoitteena on kehittää yhtenäinen testauskäytäntö Solteq Oyj:lle järjestelmäintegraatioiden testaamiseksi myymälä-järjestelmäympäristöissä.

Testaaminen on tärkeä osa ohjelmistotuotantoa. Testaamiseen ei kuulu pelkästään manuaalinen testien suorittaminen vaan koko testauksen elinkaaren vaiheet muun muassa suunnittelu-, valmistelu ja arviointi. Testauksella pyritään parantamaan tuotteen laatua. Testauksen avulla varmistetaan myös, että tuote vastaa tehtyjä määrityksiä ja sopii tarkoitukseensa.

Järjestelmäintegraatiot määritellään toimintamalleiksi ja tekniikoiksi, joiden avulla voidaan saattaa vähintään kaksi eri toiminnallisuutta tarjoavaa tietojärjestelmää jakamaan informaatiota keskenään. Järjestelmäintegraatioiden avulla pyritään siihen, että päätöksentekoon tarvittava tieto on luotettavammin ja nopeammin liiketoimintaprosessin käytettävissä. Järjestelmäintegraatioratkaisu voi kuitenkin heikosti toteutettuna muodostua yrityksessä niin sanotuksi Single point of failure –pisteeksi. Jos tämä yksi piste vikaantuu, voi yrityksen kaikki liiketoimintaprosessien kannalta oleellinen informaatio lakata kulkemasta ja pahimmillaan vaarantaa yrityksen toiminnan. Testaaminen on siis erityisen tärkeää järjestelmäintegraatioissa.

Järjestelmäintegraatioiden testaamisen teoriassa on esitelty neljä eri lähtökohtaa järjestelmäintegraatioiden testaamiseksi; järjestelmäintegraation haasteet, järjestelmäintegraatioiden arkkitehtuuri, liiketoimintaprosessit ja muiden testaustyyppien käyttämät keinot testauksessa. Järjestelmäintegraatioiden jokainen toteutus on kuitenkin niin erilainen, että yhtenäistä testauskäytäntöä niiden testaamiseksi on mahdoton antaa; käytettävä testausmenetelmä tuleekin aina loppuen lopuksi valita tapauskohtaisesti.

Avainsanat: järjestelmäintegraatio testaaminen

ABSTRACT

Tampere University of Applied Science

Master of Business Administration

Degree Programme in Information Technology

Katja Korpinen: Systems Integration Testing, Case: Solteq Ltd.

Master's Thesis 64 pages, appendices 11 pages

November 2011

In this Master's Thesis we learn about the theory of testing and systems integration and discuss based on them and among other things how systems integrations should be tested. In addition to the theory part this Master's Thesis includes a case study which goal is to develop a coherent testing policy for Solteq Ltd. used in systems integration testing in Point of Sales systems.

Testing is an important part of software production. Testing is more than manual test execution. It covers the entire testing life cycle including planning, preparation and assessment. The aim of testing is to improve the quality of product. Testing is done also to make sure that the product meets the specification made and that the product fits for its purpose.

Systems Integration is defined as processes or techniques which can be used to bring together at least two different systems. The purpose of this is to share information between those systems so that the information needed for business processes would be available faster and that the information would be more reliable. Systems Integration Solution can, however, poorly implemented, become so called single point of failure. If this single point, that is Systems Integration Solution, fails, can it cause that all the information needed in business processes is not available and at worst this can endanger the whole business. Testing is therefore very important with Systems Integrations.

In Systems Integration testing theory is presented four different point of view for Systems Integration testing; challenges in Systems Integrations, the architecture of Systems Integrations, business processes and methods used for testing with other testing types for example systems testing and integration testing. The implementation of Systems Integrations is always so different that it is impossible to have standard testing method for Systems Integration testing. The used testing method must therefore choose case by case basis.

Keywords: Systems Integration Testing

1 JOHDANTO

Testaamisella pyritään vähentämään tuotteessa olevia virheitä ja varmistamaan, että tuote vastaa tehtyjä määrityksiä ja sopii tarkoitukseensa. Lyhyesti sanottuna testaamisella pyritään varmistamaan tuotteen laadukkuus. Olemme viime aikoina saaneet lukea lehdistä, mitä kaikkea voi tapahtua, jos järjestelmää ei testata kunnolla. Testaamaton järjestelmä voi vaarantaa potilasturvallisuuden, kun potilaalle saatetaan antaa vahingossa väärää lääkettä (Jokinen, Juha-Veli 2011, A04) tai tavalliselta kuluttajalta veloitetaan maksu tililtä ilman vastinetta (VR Maksaa hyvitystä puolitoistakertaisena 2011, A07).

Integraatioratkaisuiden merkitys on kasvanut huomattavasti yrityksissä. Integraatioratkaisuiden avulla pyritään saavuttamaan säästöjä ja joustavuutta, kun manuaalisen työn määrä vähenee ja sitä kautta myös tehtyjen virheiden. Integraatioratkaisut palvelevat monien yrityksien liiketoimintaketjujen prosesseja eli integraatioratkaisut automatisoivat pitkiäkin toimintoketjuja. Integraatioratkaisut ovat sikäli kriittisiä yrityksien toimintojen kannalta, että mikäli ne eivät toimi aiheuttaa se suuria ongelmia, kun tieto ei siirry yrityksessä.

Tässä opinnäytetyössä yhdistetään ohjelmistojen testaaminen ja järjestelmäintegraatiot. Molemmat ovat jo itsessään tärkeitä osa-alueita nykypäivän ohjelmistotuotannossa ja liiketoiminnassa, mutta yhdistettynä vieläkin tärkeämpiä. Integraatioratkaisun toimivuus on kriittisen tärkeää yritykselle. Miten muuten kuin testaamalla voidaan parantaa ratkaisun toimivuutta ja luotettavuutta ja ylipäänsä saada selville sen mahdolliset ongelmat ennen tuotantokäyttöä.

Tämän opinnäytetyön tavoitteena on tutustua testauksen ja järjestelmäintegraatioiden teoriaan sekä erityisesti järjestelmäintegraatioiden testauksen teoriaan. Tähän opinnäytetyöhön kuuluu myös case-tutkimus, jossa tutustutaan Solteq Oyj:n järjestelmäintegraatioiden testaamiseen myymäläjärjestelmäympäristöissä. Case-tutkimuksen osalta opinnäytetyön tavoitteena on selvittää järjestelmäintegraatioiden testauksen nykytila Solteq Oyj:ssä ja luoda Solteq Oyj:lle yhtenäinen käytäntö ja toimintamalli järjestelmäintegraatioiden testaamiseksi ja dokumentoida se.

Case-tutkimus pohjaa laadulliseen tutkimukseen. Tutkimusmenetelminä ovat tapaus- ja toiminnallinen tutkimus sekä bechmarking-tutkimus. Tutkimusmetodeina on käytetty havainnointia, haastatteluja ja tekstianalyysia.

Johdannon jälkeen opinnäytetyön kolme seuraavaa lukua muodostavat opinnäytetyön teoriaosuuden. Luvussa kaksi on käyty läpi testauksen yleistä teoriaa. Luvussa on muuan muassa määritelty mitä testaus yleisesti ottaen tarkoittaa, miten sitä tehdään ja mihin sillä pyritään. Luvussa kaksi on myös käyty läpi erilaisia testaustyyppejä. Luvussa kolme käsitellään järjestelmäintegraatioiden yleistä teoriaa. Luvussa on käyty läpi muun muassa järjestelmäintegraatioratkaisuiden arkkitehtuuria ja järjestelmäintegraatioiden haasteita. Luvussa neljä keskitytään järjestelmäintegraatioiden testaamiseen. Luvussa on yhdistelty kahden edellisen luvun teoriaa ja niiden perusteella esitelty neljä eri lähestymistapaa järjestelmäintegraatioiden testaamiseksi.

Teoriaosuuden jälkeen luvussa viisi on esitelty case-tutkimus, jonka tavoitteena on siis selvittää järjestelmäintegraatioiden testauksen nykytila Solteq Oyj:ssä ja luoda Solteq Oyj:lle yhtenäinen käytäntö ja toimintamalli järjestelmäintegraatioiden testaamiseksi. Luvussa viisi on ensin esitelty tutkimuksen tausta ja tavoite ja sen jälkeen käytetty tutkimusstrategia ja selvitetty tutkimuksen etenemistä. Lopuksi luvussa viisi esitellään tutkimustulos eli järjestelmäintegraatioiden testaamiseksi luotu toimintamalli.

2 TESTAUS YLEENSÄ

2.1 Mitä testaus on?

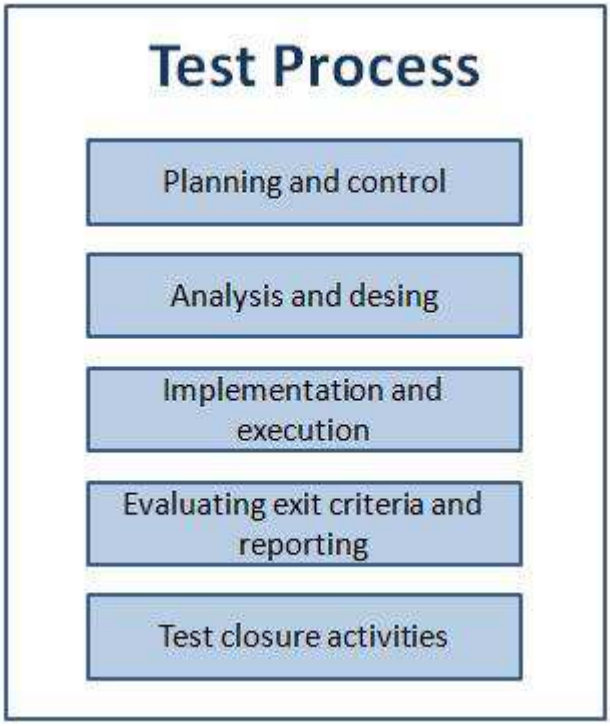
Testaaminen on tärkeä osa ohjelmistotuotantoa. Oman kokemukseni mukaan testaamisella pyritään tuotteen laadukkuuteen ja vähentämään tuotteen virheitä. Näiden avulla pyritään parantamaan asiakastyytyväisyyttä ja yrityksen taloudellista asemaa. Mutta mitä testaaminen oikeastaan tarkoittaa ja mitä kaikkea siihen kuuluu? Seuraavista kappaleista esitellään pari määritelmää siitä, mitä testaaminen oikeastaan on.

Cem Kanerin internet-sivuilta löytyvän materiaalin Explotory testing (2006, 11) mukaan testauksen määritelmä kuuluu seuraavasti: “Testing is an empirical technical investigation of the product / system / artifact / service under test conducted to provide stakeholders with information about the quality”. Canerin mukaan testaus on siis empiiristä tuotteen tutkimista, jonka tarkoituksena on tuottaa omistajille tietoa tuotteen laadusta.

ISTQB (international software testing qualifications board) sen sijaan määrittelee testauksen laajemmin. ISTQB:n internet-sivuilta löytyvän Standard glossary of terms used in software testing –sanaston (2010, 45) mukaan testauksen määritelmä kuuluu seuraavasti: ”The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.” ISTQB:n määritelmän mukaan testaukseen ei kuulu pelkästään manuaalinen testien suorittaminen vaan koko testauksen elinkaaren vaiheet muun muassa suunnittelu-, valmistelu ja arviointi-vaiheet. Lisäksi testauksella pyritään varmistamaan, että tuote vastaa tehtyjä määrityksiä ja sopii tarkoitukseensa.

Nämä kaksi määritelmää vahvistivat oman kokemukseni; testauksella tosiaan pyritään laatuun ja virheiden vähentämiseen. Määritelmissä otettiin myös kantaa siihen, että testatessa pitää tulosta verrata määrityksiin ja miettiä sitä, sopiiko tuote tarkoitukseensa. Määrityksistä kävi myös ilmi testauksen eri vaiheita, testaus ei siis ole pelkkää manuaalista testien suorittamista, vaan paljon muutakin. ISTQB:n Foundation level syllabus–

tutkintosäännöksessä (2010, 15-16) on tarkemmin kuvattu testauksen eri vaiheita. Testauksen vaiheet on havainnollistettu kuviossa 1.



KUVIO 1 Testauksen vaiheet (mukaellen ISTQB Foundation level syllabus-tutkintosäännös 2010, 15-16)

Ensimmäisenä kuviossa on Planning ja Control –vaihe eli testauksen suunnittelu ja valvonta. Testauksen suunnittelu -vaiheeseen kuuluu testauksen tavoitteiden määrittely sekä testaustehtävien määrittely. Testauksen valvonta -vaiheen tehtävänä on taas verrata läpi testauksen ajan tehtyjä suunnitelmia toteutukseen ja tarvittaessa ryhtyä toimenpiteisiin, mikäli toteutus ei vastaa suunniteltua. (ISTQB Foundation level Syllabus 2010, 15.)

Analysis and design –vaihe eli analysointi ja suunnittelu muuttaa suunnittelu-vaiheen tavoitteet konkreettisiksi testitapauksiksi. Tässä vaiheessa siis muun muassa tunnistetaan ylemmän tason testitapaukset, tarvittava testiaineisto ja testiympäristö. (ISTQB Foundation level Syllabus 2010, 15.)

Implementation and execution–vaiheessa eli toteutus- ja suoritus-vaiheessa aloitetaan varsinainen testaaminen. Testitapaukset viimeistellään ja järjestellään järkeviksi kokonaisuuksiksi, testiympäristö pystytetään ja aletaan suorittaa testitapauksia. Testien tu-

lokset kirjataan ylös ja niitä verrataan odotettuihin tuloksiin. Mahdollisten korjauksien jälkeen suoritetaan uusia testikierroksia, kunnes suunnittelu-vaiheen mittarit saadaan täytettyä. (ISTQB Foundation level Syllabus 2010, 16.)

Evaluating exit criteria and reporting –vaiheessa arvioidaan testin tuloksia suunnittelu-vaiheessa määriteltyihin lopetuskriteereihin nähden ja päätetään tarvitaanko lisää testejä vai pitääkö lopetuskriteereitä muuttaa. Testauksen jälkeen kirjoitetaan yhteenvetoraportti eri osapuolia varten. (ISTQB Foundation level Syllabus 2010, 16.)

Test closure activities –vaiheessa kerätään kokemuksia tehdyistä toimenpiteistä ja tarkastetaan läpi vielä kerran aikaansaadut tulokset verrattuna suunnitelmiin. Lisäksi tässä päättämis-vaiheessa dokumentoidaan järjestelmän hyväksyminen. (ISTQB Foundation level Syllabus 2010, 16.)

2.2 Testauksen tarkoitus, mitä testauksella tavoitellaan

Edellisen luvun mukaan testaus on siis pitkäjänteistä, järjestelmällistä ja vaiheissa etenevää työtä, jonka tarkoituksena on kaikin tavoin varmistaa tuotteen laadukkuus, niin virheiden määrän pienuutena kuin tuotteen soveltuvuutena käyttötarkoitukseensa. Tarkastellaan seuraavaksi vielä tarkemmin testauksen tarkoitusta. Seuraavissa kappaleissa on siihen parinkin eri kirjoittajan mielipide.

Testauksella on useita tarkoituksia. Schaeferin (2008, 41) mielipide testauksen tarkoituksesta mukailee paljolti sitä, mikä kävi ilmi ISTQB:n ja Kem Canerin materiaaleista, joita käsiteltiin edellisessä luvussa. Schaeferin (2008, 41) mukaan testauksella on kolme tarkoitusta:

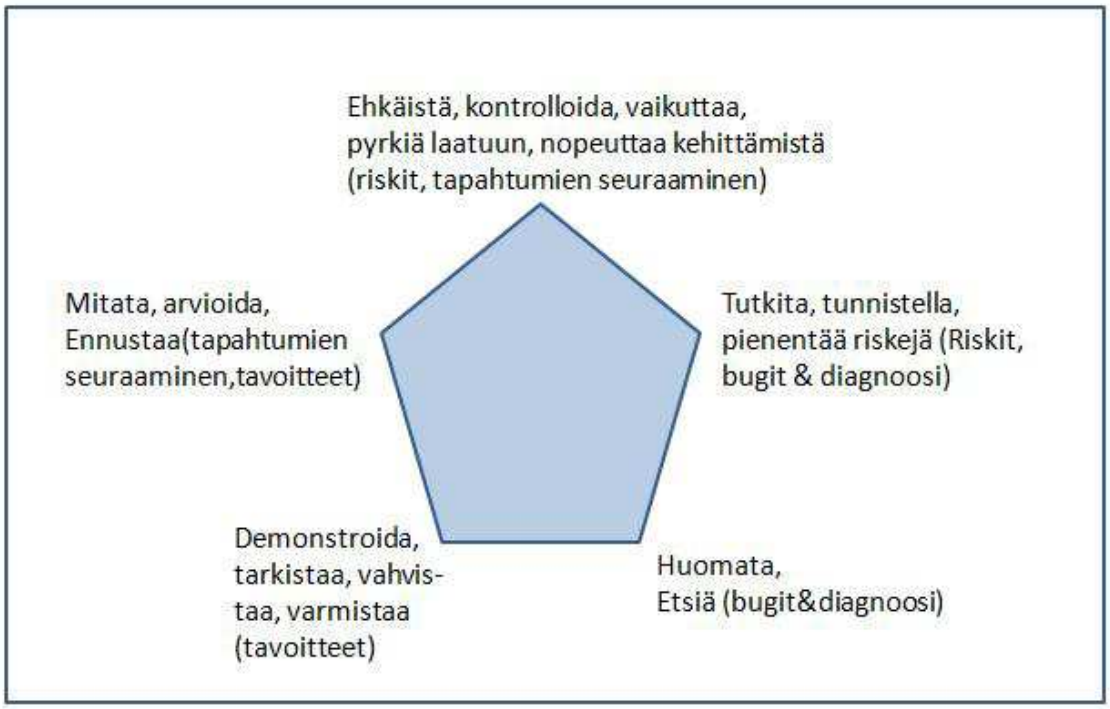
- tuotteen laadun mittaaminen
- luottamuksen luominen tuotteeseen ja
- tuotteen käyttämisen riskien lieventäminen

Laadun mittaamisessa Schaefer (2008, 41) korostaa sitä, että testaajan tulee tietää, mitkä laatu-kriteerit ovat toisia tärkeämpiä ja millä tarkkuudella mittaaminen tulee suorittaa.

Schaeferin (2008, 41) mukaan luottamus tuotteeseen luodaan parhaiten, kun etsitään sen puutteita. Puutteita voidaan löytää tekemällä enemmän testitapauksia tai tarkastelemalla tuloksia ja analysoimalla yksityiskohtia tarkemmin. Jokaisen testaajan tulisi myös tietää, että puutteet esiintyvät yleensä ryhmittäin eli esimerkiksi ohjelmiston tietty kriittinen tai monimutkainen moduuli voi olla usean puutteen takana. Testaajan tulee myös kiinnittää huomiota laatu-indikaattoreihin eli esimerkiksi siihen, onko käytetty uutta teknologiaa tai mikä toiminto ohjelmistossa on erityisen monimutkainen tai minne on tehty muutoksia; testaajan tulisi siis tietää, mistä etsiä puutteita.

Kolmas Schaeferin (2008, 42) testauksen tarkoituksista on riskien mittaaminen. Tuotteen riskiä voidaan selvittää esimerkiksi käyttäjäprofilointien avulla ja arvioimalla mahdollista vahinkoa. Käyttäjäprofiloinnissa listataan ylös tuotteen erilaisia käyttäjiä, heidän tuotteen käyttötarkoitus ja se, kuinka usein he tuotetta käyttävät sekä se miten nämä roolit ja käyttötarkoitukset muuttuvat ajan myötä. Käyttäjiä ajatellen ei pidä unohtaa käyttäjien antamia vääriä syötteitä ja väärinymmärryksiä tuotteen käytön suhteen. Mahdollisen vahingon arvioiminen voi olla hankalaa. Testaajan tulisi arvioida aina; mikä on pahinta, mitä voi sattua, käytettäessä esimerkiksi tätä ohjelman toimintoa.

Zimmererin (2010, 7) mukaan testauksella on viisi eri ulottuvuutta (katso kuvio 2). Zimmerin (2010, 7) mukaan näiden ulottuvuuksien avulla voi kuvata testauksen tarkoitusta, motivaatiota, toimintoja ja arvoja. Kuten Schaefferin (2008, 41) mallissa, myös Zimmererin viiden ulottuvuuden mallissa otetaan huomioon riskit, laatu ja virheiden etsiminen.



KUVIO 2 Testauksen viisi ulottuvuutta (mukaellen Shaeffer 2010, 7)

Zimmererin (2010, 7) mukaan muun muassa demonstroimalla ja varmistamisella pyritään siihen, että tuote toimii määrittelyiden mukaisesti. Samalla pystytään myös vahvistamaan, että ollaan tekemässä oikeaa asiaa oikein eli saavutetaan määritellyt tavoitteet.

Zimmererin (2010, 7) toisen ja kolmannen ulottuvuuden voi nivoa melko tiiviisti yhteen. Näiden ulottuvuuksien tarkoituksena on määritellä tuotteen riskit ja pienentää niitä. Riskejä voidaan pienentää muun muassa siten, että testauksen avulla etsitään tuotteen virheitä mahdollisimman aikaisessa vaiheessa ja sitten kartoitetaan tuotteen toimintaa testauksen alla.

Neljännen ja viidennen ulottuvuuden avulla pystytään seuraamaan tapahtumia ja valmistamaan tuotetta tulevaisuuden kehittämiseen. Testauksen aikana kerätään tietoa tuotteesta ja mitataan tuotteen suorituskykyä, luotettavuutta ja saatavuutta. Näiden tietojen avulla saatua tietoa hyödynnetään tuotteen kehittämisessä, tuotteistamisessa, tuotteeseen tehtävissä muutoksissa nyt ja tulevaisuudessa. (Zimmerer 2010, 8.)

Kuten on jo pariin kertaan todettu, niin testauksen tarkoituksena on löytää tuotteen virheitä ja parantaa tuotteen laatua. Tässä luvussa esiteltiin myös riskiajattelua; tuotteen virheitä voi löytää tunnistamalla tuotteen käytön riskejä esimerkiksi eri käyttäjien näkö-

kulmista. Miten näihin tavoitteisiin sitten päästään? Miten testausta tulisi suorittaa käytännössä? Seuraavassa luvussa on otettu kantaa tähän asiaan.

2.3 Testauksen periaatteet & onnistunut testaus

Jokaisessa yrityksessä yritetään tehdä mahdollisimman laadukkaita sovelluksia tiukkojen aikarajojen puitteissa. Maes ja Mertens (2008, 52-53) ovat luoneet kymmenen kohdan listan, jonka avulla testausprosessia voi parantaa ja siten myös testattavien sovelluksien laatua.

Ensimmäisenä Maes ja Mertens (2008, 52) mainitsevat listassaan, että testaus tulisi aloittaa ajoissa ja mahdollisimman valmistautuneena. Tämä ei tarkoita pelkästään sitä, että valmiin tuotteen testaamiseen tulisi varata riittävästi aikaa vaan myös sitä, että testaajien pitäisi päästä mukaan jo projektin alussa määrittely- ja suunnitteluvaiheisiin. Heti alusta asti tulisi siis testaajien tarkastella ja käydä läpi kaikkia luotavia dokumentaatioita. Testaajien ei kannata jäädä odottamaan valmista ohjelmakoodia, vaan aloittaa testaaminen heti katselmoimalla luotuja dokumentteja. Aikainen aloittaminen on tärkeää sillä Meas & Mertensin (2008, 52) mukaan hyvin läpikäyty dokumentaatio vähentää seuraaviin vaiheisiin siirtyviä virheitä 65 prosenttia ja vikojen korjaaminen suunnitteluvaiheessa on keskimäärin sata kertaa halvempaa kuin tuotantovaiheessa. (Maes & Mertens 2008, 52.)

Tamres (2002, 25-27) selventää vielä lisää testaajan roolia ja testaajan mahdollisuuksia puuttua aikaisin tilanteeseen. Jo ennen toteutusvaihetta testaajat voivat antaa arvokasta informaatiota tuotteesta jo pelkästään katselmoimalla tuotteen määrittelydokumentaatiota ja suunnittelemalla testausta sen perusteella. Testitapauksien suunnittelu olemassa olevan testidokumentaation perusteella saattaa paljastaa puutteita, epäjohdonmukaisuuksia ja virheitä tuotteesta. Näiden puutteiden huomaaminen jo määrittelyvaiheessa säästää huomattavasti kustannuksia, kun yllä oli jo todettu. Lisäksi niiden avulla saadaan tarkennettua olemassa olevaa suunnitelmaa tuotteen toteuttamiseksi, mikä voi taas ennaltaehkäistä esimerkiksi tuotteen jatkokehittämisen ja integroimisen ongelmia. Lisäksi testaajat saattavat huomata, että tuote ei palvele lainkaan asiakkaan tarvetta tai että ollaan kehittämässä kokonaan väärää tuotetta.

Omasta kokemuksestani voin sanoa, että kun alkaa tuotteen valmistumisen kynnyksellä tehdä testisuunnitelmaa määrittelydokumentaation pohjalta, huomaa usein, että se ei tarjoa kaikkea tarvittavaa tietoa testisuunnitelman laatimiseksi. Tällöin tarvitsee alkaa kysellä lisäinformaatiota tuotteesta ja sen ominaisuuksista ja sovituista toiminnallisuuksista muun muassa projektipäälliköiltä ja tuotannosta. Kaikki tämä vie arvokasta aikaa testaukselta.

Seuraavana Maes ja Mertens (2008, 52) esittävät kymmenen kohdan listassaan, että universaalia testaus menetelmää ei ole, joka kävisi kaikkien mahdollisten sovelluksien testaamiseen tai sopisi kaikille erityyppisille yrityksille. Testaajien tulee siis kiinnittää huomiota kehitettävän sovelluksen erityispiirteisiin ja käytettävä testausmenetelmä tulisi valita sen mukaan. (Maes & Mertens 2008, 52.)

Koko sovelluksen kaikkien mahdollisten toimintojen testaaminen on mahdotonta. Tämän vuoksi testaajan tulee priorisoida testaamistaan ja riskianalyysin perusteella määritellä sovelluksen riskit ja yleisemmin käytetyt toiminnot ja kiinnittää testauksessa erityisesti huomiota niihin. (Maes & Mertens 2008, 52-53.)

Ennen kuin aletaan testata, tulisi määritellä niin sanotut lopetuskriteerit eli millä ehdoilla testaaminen lopetetaan. Yleensä testaaminen kannattaa lopettaa, kun seuraavaan virheen löytämiseen kuluu liikaa aikaa ja rahaa. Testaamista ei siis kannata jatkaa loputtomiin. (Maes & Mertens 2008, 53.)

Testauksessa käytetyt testitapaukset kannattaa säilyttää seuraavaa testauskierrosta tai seuraavan version testaamista varten. Pyörää ei kannata keksiä joka kerta uudelleen. Vanhojen testitapauksien käyttäminen seuraavissa testauksissa saattaa säästää jopa 50 prosenttia kokonaistestauskustannuksissa. (Maes & Mertens 2008, 53.) Samoja testitapauksia ei kuitenkaan voi käyttää loputtomiin. ISTQB:n Foundation level syllabus–tutkintosäännöksessä (2010, 14) on mainittu niin sanottu hyönteismyrkky-paradoksi, jonka mukaan tietty testitapauksien joukko ei enää löydä uusia vikoja tietyn ajan jälkeen. Käytettyjä testitapauksia tuleekin siis päivittää ja käydä läpi tietyin aikavälein. (ISTQB:n Foundation level syllabus–tutkintosäännös 2010, 14.)

Henkilöiden, jotka testaavat sovellusta, tulisi pyrkiä olemaan mahdollisimman objektiivisia testaukseen yleensä ja testattavaan sovellukseen nähden. Testaajien tulisi siis mitata työnsä tehokkuutta erilaisin mittarein. Esimerkiksi kuinka monta virhettä sovelluksesta löytyy ensimmäisen kuukauden käytön jälkeen? Objektiivisuus vaatii yleensä myös sitä, että henkilö ei voi kuulua kovin tiivisti projektiryhmään, sillä aikataululliset ja kustannukselliset tavoitteet ajavat monesti yli testauksen. Testaajien olisikin hyvä kuulua omaan organisaatioonsa projektiryhmän ulkopuolella. Tästä huolimatta testaajien tulee kommunikoida projektiryhmän, loppukäyttäjien, tuotannon ja johtoportaan kanssa. (Maes & Mertens 2008, 52-53.)

Myös ISTQB:n Foundation level syllabus–tutkintosäännöksessä (2010, 14) on käyty läpi yleisiä testauksen periaatteita. Testauksen aikainen aloittaminen, testauksen tilannesidonnaisuus ja täydellisen testauksen mahdottomuus ovat samoja periaatteita kuin mitkä Maes & Mertens (2008, 52-53) ovat myös todenneet.

Näiden lisäksi ISTQB:n Foundation level syllabus–tutkintosäännöksessä (2010, 14) on tuotu esiin näkemys, jonka mukaan testauksella voidaan osoittaa virheiden olemassaolo, mutta ei niiden puuttumista. Testauksen avulla voidaan siis pienentää virheiden todennäköisyyttä, mutta vaikka testauksen avulla ei sovelluksesta löydettäisi yhtään virhettä, ei tämä kuitenkaan tarkoita sitä, että sovellus olisi virheetön. (ISTQB:n Foundation level syllabus–tutkintosäännös 2010, 14.)

Maes & Mertens (2008, 52-53) mainitsivat, että testausta tulee priorisoida. Yksi keino priorisointiin käy ilmi ISTQB:n Foundation level syllabus–tutkintosäännöksestä (2010, 14). Virheillä on tapana kasaantua tiettyihin moduuleihin. Yleensä joku tietty, yleensä monimutkainen, osa ohjelmistosta on se, mistä suurimmat osat virheistä löytyvät ja joka samalla aiheuttaa virheitä muihinkin toimintoihin. (ISTQB:n Foundation level syllabus–tutkintosäännös 2010, 14.) Priorisoinnissa kannattaa käyttää apuna tuotannon asiantuntemusta; tuotetta ohjelmoinut asiantuntija osaa antaa testaajalle vinkkejä siitä, mihin toimintoihin testaus kannattaa keskittää. Kuten pari kappaletta aiemmin tulikin jo mainittua, testaajan tulee siis tehdä tiivistä yhteistyötä muun muassa tuotannon kanssa.

Testaajan tulee siis hallita useita osa-alueita. Testaajaan tulee hyväksyä se, että kaikkia virheitä ei voi löytää ja hänen tulee tietää koska testaaminen kannattaa lopettaa; testaaja

ei voi siis pyrkiä täydellisyyteen. Testaaminen on myös hyvin järjestelmällistä ja usein pitkää pinnaa vaativaa työtä joskus jopa yksitoikkoista, kun toistettavien testitapauksien määrä on voi olla laajakin. Testaajan tulee olla myös sosiaalinen, sillä hänen tulee tehdä yhteistyötä monen eri organisaation kanssa yrityksessä. Toisaalta testaaja toimii puun ja kuoren välissä; häntä painaa projektin aikataulu ja toisaalta tuotteen laatuvaatimukset. Monesti testaajaan ei suhtauduta esimerkiksi tuotannon puolelta kovin lämpimästikään; moni tekijä voi ottaa raskaastikin sen, kun joku tulee osoittamaan, mitä virheitä hän on tehnyt työssään.

2.4 Testaaminen käytännössä

Edellisessä luvussa on käyty läpi testauksen periaatteita, jotka takaavat onnistuneen testauksen. Kuten on jo käynyt ilmi, niin testausta toteutetaan käytännössä testitapauksin, jotka pitäisi suunnitella kyseistä tuotetta silmälläpitäen esimerkiksi määrittelydokumentaatioon nojautuen. Koska aika on aina rajallista, ei testitapauksia saa olla liikaa, vaan niitä pitää priorisoida. Miten näitä testitapauksia sitten luodaan ja millaisia niiden pitäisi olla? Ja miten niitä priorisoidaan? Sitä käydään läpi seuraavassa luvussa.

Ennen testauksen aloittamista, testaajan on tunnettava tuote, jota hän on aloittamassa testaamaan. Tässä testaaja voi käyttää apuna tuotteesta olemassa olevia dokumentaatiota esimerkiksi käyttöohjeita ja määrittelydokumentteja. Testaaja voi myös tutustua tuotteeseen alkamalla käyttää sitä ja kokeilemalla mitä mikin toiminto tekee. Myös joku toinen henkilö, esimerkiksi tuotteen suunnittelija, voi demonstroida tuotetta testaajalle. (Tamres, 2002 7.)

Ennen varsinaisen testaamisen aloittamista on asennettava testiympäristö, jossa testaus suoritetaan. Testiympäristön luominen edellyttää muun muassa tarvittavien laitteiden hankkimista ja asentamista sekä tarvittavien sovelluksien asentamista ja konfigurointien tekemistä kyseiseen laiteympäristöön. Lisäksi pitää ottaa huomioon, että testiympäristössä pitää olla tarvittava data olemassa testien tekemistä varten. Esimerkiksi kassaympäristössä pitää olla tuote ja asiakashinnat. Tarvittaessa asennetaan myös testausta automatisoivia työkaluja testiympäristöön. (Tamres, 2002 8; 219.) Testiympäristössä on otettava vielä huomioon se, että testiympäristöksi ei käy mikä tahansa laite tai muu ym-

päristö. Testiympäristön tulee simuloida mahdollisimman tarkasti asiakkaan käytössä olevaa ympäristöä. Testausta tulee siis suorittaa esimerkiksi samoin oheislaittein tai samassa käyttöjärjestelmäympäristössä kuin mitä asiakaskin tulee käyttämään.

Ennen kuin käydään läpi sitä, miten testitapauksia pitäisi luoda ja millaisia niiden pitäisi olla, on syytä selventää, mitä testitapaus on. Testitapauksessa kuvataan tuotteelle annettava syöte ja odotettu tulos, miten tuotteen pitäisi annettuun syötteeseen reagoida. Kun testaaja suorittaa kyseisen testitapauksen eli on antanut tuotteelle testitapauksessa kuvatun syötteen, hän kirjaa testitapaukseen testauksen tuloksen. Testitapauksessa voi olla syötteen, kuvauksen ja tuloksen lisäksi tietoa myös dokumentaatiosta, johon testitapaus viittaa, sekä testiympäristöön liittyviä tietoja esimerkiksi tuotteen ja laitteiston, jolla testi on suoritettu, versionumerot. Lisäksi testitapaukset yleensä erotetaan toisistaan identifioivalla tunnuksella. (Tamres, 2002 59.) Testitapaukset eivät ole kovin laajoja, vaan niillä pyritään aina testaamaan tuotteen jotain tiettyä toiminnallisuutta tai ominaisuutta tietyillä muuttujilla. Esimerkiksi tekstinkäsittelyohjelmassa testitapaus voisi olla tekstin fontin koon muuttaminen.

Kuten aikaisemmin on jo useaan kertaan mainittu, niin testitapauksien suunnittelun pohjana toimii usein määrittelydokumentaatio. Määrittelydokumentaatiosta poimitaan tuotteen käyttötilanteita ja käyttömahdollisuuksia, joita voidaan lähteä jalostamaan testitapauksiksi. Määrittelydokumentaatiosta pitäisi käydä myös ilmi tuotteelle annettavat syötteet ja tulokset, joita tarvitaan testitapauksien suunnittelussa. (Tamres 2002, 27-28.) Mikäli tuotteen suunnittelussa on käytetty prosessien mallintamista, myös käyttötapauksia voidaan käyttää testitapauksien suunnittelun pohjana (Tamres, 2002 55).

Kokemukseni myötä testaaja oppii tunnistamaan eri testitapauksia helpostikin määrittelydokumentaatiosta tai kokeilemalla käyttää testattavaa tuotetta. Haasteellista on löytää ne kaikki eri skenaariot, mitä voi tapahtua ja kaikki eri toiminnot ja syötteet, jotka vaikuttavat testitapaukseen. Ja mistä tietää, milloin testitapauksia on tarpeeksi ja kattavatko ne kaikki tarvittavat toiminnot ja eri käyttötapaukset? Seuraavissa kappaleissa on vastauksia näihin kysymyksiin.

Oleellista testitapauksien suunnittelussa on tuntea testauksen eri kategoriat. Näiden avulla saadaan testitapauksiin tuotteen eri käyttötilanteet. Käytännössä nämä eri katego-

riat vaikuttavat testitapauksien syötteisiin ja siihen, että jokaisesta käyttötilanteesta pitäisi luoda useita käyttötapauksia eri syötteillä, jolloin tulee testattua tuotteen toiminnallisuutta kattavammin. Testikategorioita ovat (Tamres 2002, 34):

- syötettä ei anneta (no data provided)
- suorita kahteen kertaan (do it twice)
- oikeanmuotoinen data (valid data)
- vääränmuotoinen data (invalid data)
- keskeytä (abort)
- virtakatko (power loss)
- järjestelmän rasittaminen (stressing the system)
- suorituskyky (performance)

Tämä listaus eli testauksen eri kategoriat, tarkoittaa käytännössä sitä, että testaamiseen saadaan aikaan erityyppisiä testitapauksia. Esimerkiksi tuotteen tietty ominaisuus tai toiminnallisuus tulisi aina testata oikeanmuotoisilla syötteillä (valid data), mutta lisäksi väärillä syötteillä (invalid data) esimerkiksi antamalla numeerista tietoa luku-kenttään tai antamalla negatiivisen luvun kenttään, johon oletetaan positiivista lukua. Myös nolla-arvot, desimaalit, välilyönnit ynnä muut sellaiset tulee ottaa testitapauksia suunnitellessa huomioon. Sovelluksen tulisi myös selvitä tilanteesta, jossa arvoa ei anneta lainkaan (no data provided) tai arvo annetaan kahdesti (do it twice) esimerkiksi lähettämällä joku tiedosto kahteen kertaan tai painamalla enter-painiketta kahteen kertaan. Testattavan tuotteen ominaisuuksia pitää testata myös muista näkökulmista. Mitä tapahtuu jos käyttäjä keskeyttää toiminnon tai vaadittu online-yhteys katkeaa (abort)? Mitä jos virransyöttö häiriintyy tai katkeaa (power loss), häviävätkö kaikki tallennetut tiedot? Kuinka paljon tapahtumia järjestelmällä pystyy suorittamaan tai mitä tapahtuu jos muisti, kovalevy tai prosessoriteho käy vähiin (stressing the system, performance)? (Tamres, 2002 34-39.)

Kun määrittelydokumentaatiosta löydetyt syötteet on yhdistetty testauksen kategorioihin, saattaa testitapauksia olla kasassa isokin määrä. Erityisesti jos testattava tuote sisältää paljon kenttiä, joihin pitäisi syöttää tietoa. Miten tätä suurta testitapausmäärää kannattaisi hallita tai miten testitapauksien todellisesta määrästä saa käsityksen? Testaajan omassa muistissa testitapauksia ei kannata säilöä, vaan testitapaukset ja varsinkin niiden

tulokset on saatava dokumentoitua. Testitapauksien kuvaukset ovat myös siksi tärkeitä, että testauksessa havaittu puute on voitava toistaa. Lisäksi dokumentoinnista on se hyöty, että testitapauksia voi käyttää uudelleen esimerkiksi seuraavan version testaamisessa.

Suunnitteluvaiheessa kaikki testitapaukset pitää saada dokumentoitua, jotta niiden priorisointia ja rajausta voi alkaa tehdä. Testitapauksia voi siis tulla niin paljon, että kaikkia voi millään käydä läpi vaadittujen aikarajojen puitteissa. Dokumentointitavan, jolla testitapaukset esitetään, pitäisi olla selkeä ja helposti ymmärrettävä. Tamres (2002, 73) ehdottaa yhdeksi ratkaisumalliksi taulukoita. Taulukossa (katso taulukko 1) kuvataan vasemmassa reunassa testitapauksen lähtökohta eli esimerkiksi tuotteen tietty toiminto tai kenttä ja sarakkeissa annetut syötteet, jotka on siis löydetty esimerkiksi määrittelydokumentaatiosta ja joita on etsitty lisää testauksien eri kategorioiden avulla. Suunnittelu-vaiheessa taulukoiden avulla saa kuvan testitapauksien todellisesta lukumäärästä.

TAULUKKO 1 Testitapauksia (mukaellen Tamres 2002, 77)

	Input 1	**Input 2**	**Input 3**	**Input 4**	**Input 5**
Prompt 1	Test case 11	Test case 12	Test case 13	Test case 14	Test case 15
Prompt 2	Test case 21	Test case 22	Test case 23	Test case 24	Test case 25
Prompt 3	Test case 31	Test case 32	Test case 33	Test case 34	Test case 35
Prompt 4	Test case 41	Test case 42	Test case 43	Test case 44	Test case 45
Prompt 5	Test case 51	Test case 52	Test case 53	Test case 54	Test case 55
Prompt 6	Test case 61	Test case 62	Test case 63	Test case 64	Test case 65

Miten suunnitteluvaiheen testitapauksien määrää voi sitten alkaa rajata tai priorisoida? Testitapauksien rajaus ja priorisointi on tärkeää, sillä testaukselle on yleensä varattu vain rajattu aika ja tuossa ajassa tulisi saada testaus suoritettua mahdollisimman kattavasti. Tamres (2002, 194) esittelee teoksessaan neljä keinoa testitapauksien määrän vähentämiseksi;

- priorisointi kategorioittain (priority category scheme)
- riskianalyysi (risk analysis)
- ongelma-alueiden tunnistaminen haastattelemalla (interviewing to identify problem areas)
- yhdistelmä edellisistä (combination schemes)

Testitapauksien priorisointi on aina kompromissi; testaaja valitsee suoritettavat testitapaukset niiden kustannuksella, joita ei suoriteta. Priorisoinnissa testaaja joutuu pohtimaan sitä, mitkä tapaukset on testattava ja toisaalta, mitkä ovat mahdolliset seuraukset, jos testaaminen jätetään tekemättä. Seurauksien arviointiin on hyvä ottaa mukaan myös yrityksen tai projektin johdon mielipide. Helpoin tapa priorisoida testitapauksia on verrata niitä keskenään ja jakaa ne sen mukaan kolmeen kategoriaan;

- testaus on välttämätöntä suorittaa (this test must be executed)
- jos aikataulu sallii, testataan (if time permits, execute this test)
- testaus voidaan jättää suorittamatta (if this test is not executed, the team won't be upset)

Kun testitapaukset on jaettu näihin kategorioihin, arvioidaan, paljonko aikaa kunkin kategorian suorittamiseen eli testaamiseen menee. Jos aika riittää hyvin ensimmäisen kategorian testien suorittamiseen, ongelma on ratkennut. Mikäli ei, tarvitsee näitä kategorioita alkaa rajaamaan ja priorisoimaan tarkemmin. (Tamres 2002, 194.195.)

Riskianalyysissä listataan ensin mahdolliset ongelmat ja arvioidaan sen jälkeen jokaisen ongelman toteutumismahdollisuus ja haitallisuus toteutuessaan yksinkertaisella asteikolla esimerkiksi yhdestä kymmeneen. Riskiluku saadaan laskettua näistä kahdesta arvosta yksinkertaisella kertolaskulla. Kun riskiluvut on saatu laskettua, järjestetään ongelmat niiden mukaiseen järjestykseen. Luonnollisesti ne ongelmat, joissa on suurin riskiluku, ovat niitä, joiden testaukseen kannattaa panostaa. Riskianalyysin avulla testaaja voi valita testitapauksista ne, jotka liittyvät kyseisiin ongelmiin. Tällainen riskianalyysi on helppo tehdä taulukkomuotoisesti. Myös nelikenttä-analyysia, jossa toteutumismahdollisuus on toisella akselilla ja haitallisuus toisella, voi käyttää, jos haluaa antaa enemmän painoarvoa tekijöille erikseen. (Tamres 2002, 196-200).

Kuten testauksen periaatteissa kävi jo ilmi, haastattelemalla tuotteen parissa työskennelleitä henkilöitä saadaan tietoa testausta varten. Tässä tapauksessa haastattelemalla saadaan kerättyä tietoa testitapauksien priorisointiin. Testaaja on yleensä tietämätön tuotteen suunnittelussa ja kehittämisessä tehdyistä ratkaisuista, jotka voivat vaikuttaa tuotteen testaamiseen. Kehittäjiltä kannattaakin kysellä, mitkä tuotteen kohdat ovat sellaisia

mitä kannattaisi testata. Oliko jonkin tietyn ominaisuuden toteuttaminen erityisen hankalaa tai käytettiinkö jotain uutta teknologiaa? Tai oliko esimerkiksi määrittelyissä epäjohdonmukaisuuksia jonkin toiminnon osalta? Asiakkaiden näkökulmakin kannattaa ottaa tässä myös huomioon. Mitkä tuotteen ominaisuuksista ovat erityisen tärkeitä asiakkaille? Onko tuotteen ominaisuuksista joku sellainen, että jos se ei toimi, vaarantaa se asiakkaan liiketoiminnan kokonaan? Myös tuotteen parissa työskennelleiden henkilöiden osaaminen ja motivaatio kannattaa ottaa huomioon, vaikka sen selvittäminen voikin olla vähän hankalaa. Jos tunnetusti joku koodaaja tekee erittäin laadukasta työtä, kannattaa hänen toteuttamiensa moduulien testaamiseen käyttää ehkä vähemmän aikaa, kuin henkilön, jonka toteutuksesta on ennenkin löytynyt puutteita tai jonka motivaatio on ollut alhainen. (Tamres 2002, 201-207.)

Kun tuotteen eri moduuleita eli esimerkiksi toiminnallisuuksia aletaan yhdistää ja suoritetaan niin sanottua integraatiotestausta, täytyy pari uuttaa näkökulmaa testauksessa ottaa huomioon. Tässä kohtaa on tärkeä miettiä sitä, mitkä yhdistelmät ovat todennäköisiä ja mitkä eivät eli minkä moduuleiden yhteistoimintaa on tärkeää testata ja minkä ei. (Tamres 2002, 207.)

Toteutusvaiheessa taulukoita voi ja kannattaa myös käyttää eli tällöin taulukon soluun kuvataan tarkempi suoritustieto testitapauksesta (katso taulukko 2). Testitapauksien suorituksen tuloksen voi myös kirjata vastaavaan taulukkoon, tällöin on hyvä käyttää lisänä kolmatta taulukkoa, jonka avulla suoritusohjeet ja tulokset linkitetään toisiinsa. (Tamres 2002, 80; 83-87.)

TAULUKKO 2 Testitapauksia (mukaellen Tamres 2002, 80)

	a	b	Shift + a	Shift + b	Enter
Username field	add a	add b	add A	add B	Process username/password
password field	add *	add *	add *	add *	Process username/password
OK Button	no change	no change	no change	no change	Process username/password
Cancel Button	no change	no change	no change	no change	exit window

Testitapauksen tiedot ja tulokset voi myös esittää yhdessä ja samassa taulukossa (katso taulukko 3). Tällöin yksi testitapaus on yhdellä rivillä ja eri sarakkeissa annetaan syöttö-

ja tulostiedot. (Tamres 2002, 93). Tällaisessa taulukossa testitapauksien luettavuus paranee, mutta jos testitapauksia todella suuri määrä, tulee taulukosta melko massiivinen.

TAULUKKO 3 Testisuunnitelma (mukaellen Tamres 2002, 93)

	Initial state					
Test case id	**username**	**password**	**cursor location**	**Input**	**Expected results**	**Actual result**
tc-401	(empty)	(empty)	Place cursor in username field	a	'a' appears in username field	
tc-402	m	(empty)	place cursor after 'm' in username field	b	'mb' appears in username filed	
tc-403	jasmine	(empty)	place cursof after 'j' in username field	a	'jasmine' appears in username field	
tc-404	joelouis	(empty)	place cursor at end of username field	2	ignore keystoke (maximum number of characters exceeded): no change	

Taulukon voi myös suunnitella kattamaan testauksen uudet testikierrokset, kun samoja ominaisuuksia testataan tehtyjen korjauksien jälkeen samoilla testitapauksilla. Kuviossa 3 on taulukko, jossa on varattu samaan taulukkoon tilaa sekä testitapauksille että testin tuloksille, mutta myös toiselle testikierrokselle. Testitapaukset on selvitetty sanallisesti ja taulukon sarakkeisiin G-K on varattu tilaa ensimmäisen testikierroksen tuloksille. Taulukkoon on varattu myös tilaa seuraavalle testikierrokselle (katso taulukon sarakkeet M-O).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Kuitin uudelleentulostus**														
2															
3	**Testitapauksen numero ja kuvaus**				**Odotettu tulos**		**Saatu tulos, testikierros 1**				**Tuotannon vastaus**		**Saatu tulos, testikierros 2**		
4							**Kuitti**	**Tulos**	**Kommentit**				**Kuitti**	**Tulos**	**Kommentit**
5	**1**	**Kuitin uudelleen tulostus, heti**													
6			Sisäänkirjaudu kassaan, lue tuote, paina loppusummaa, päätä kuitti käteiseen												
7			Paina "Kuitin uudelleentulostus-painiketta"		Juuri tehty myyntikuitti tulostuu uudelleen. Uudelleentulostuneella kuitilla näkyy merkintä siitä, että kuitti on kopio ja lisäksi tieto siitä minä päivänä, mihin kellonaikaan ja kuka (kassanhoitaja) kuitin on uudelleentulostanut		599	HUOMIO	Kuittikopio ei tulostunut, vaan kassa antoi virheilmoituksen.		Korjattu		99	OK	Kuittikopio tulostui ok
8															
9															
10	**2**	**Kuitin uudelleen tulostus, heti**													
11			Sisäänkirjaudu kassaan												
12			Paina "Kuitin uudelleentulostus-painiketta"		Sisäänkirjauskuitti tulostuu uudelleen. Uudelleentulostuneella kuitilla näkyy merkintä siitä, että kuitti on kopio ja lisäksi tieto siitä minä päivänä, mihin kellonaikaan ja kuka (kassanhoitaja) kuitin on uudelleentulostanut		601	OK	Kuittikopio tulostui ok				101	OK	Kuittikopio tulostui ok
13															
14															

KUVIO 3 Testitapaukset ja niiden tulokset

2.5 Testaustyypit

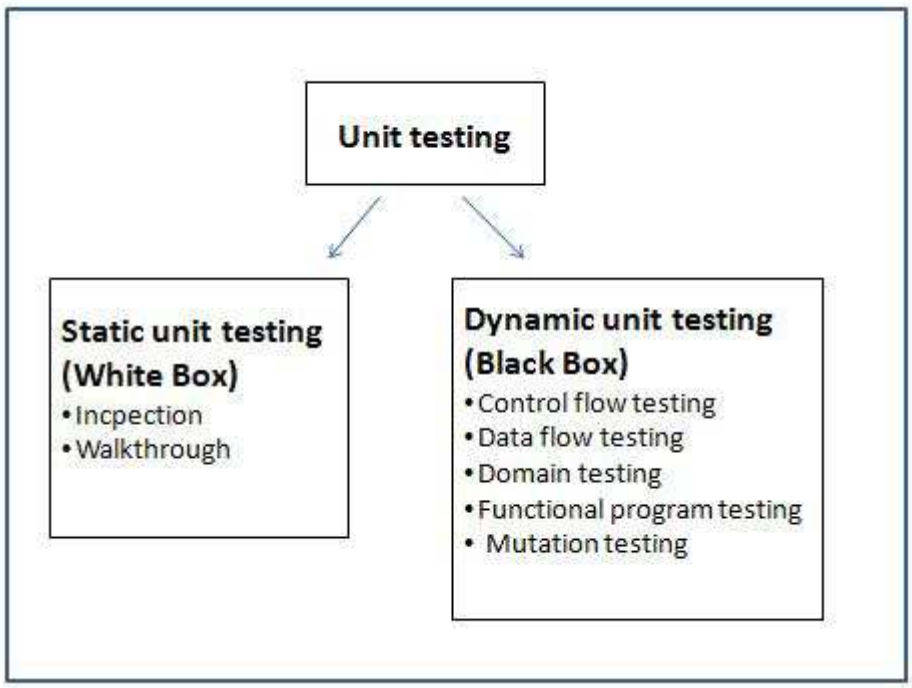
Testausta voi ja pitää suorittaa järjestelmän kehittämisen eri vaiheissa, jolloin testaamisella on aina myös eri tarkoitus ja päämäärä. Testausta suoritetaan heti aluksi, kun järjestelmän yksittäisiä toimintoja kehitetään. Testausta ei pidä myöskään unohtaa, kun kehitettyjä ominaisuuksia aletaan nivoa toimivaksi kokonaisuudeksi.

Järjestelmän kehittämisen eri vaiheet ja eteneminen luovat haasteen testaamiselle ja tuovat eri ongelmia kehitettyyn järjestelmään, jotka tulisi testauksen avulla löytää. Testauksen tulee siis vastata järjestelmän eri kehitysvaiheisiin. Seuraavassa käydään läpi testauksen eri tyyppejä ja sitä, miten ne nivoutuvat järjestelmän kehittämiseen ja toimittamiseen ja mihin eri vaiheen testauksella pyritään.

2.5.1 Yksikkötestaus (unit testing)

Yksikkötestaus tarkoittaa pienimmän komponentin testausta järjestelmässä. Pienin komponentti voi olla esimerkiksi olio olio-ohjelmoinnissa tai joku järjestelmän toiminnallisuus. Yksikkötestauksessa varmistutaan siitä, että jokainen yksikkö toimii itsenäisenä keinotekoisessa ympäristössä. (Tamres 2002, 220.) Yksikkötestaus suoritetaan siis ennen kuin komponentit integroidaan toimivaksi kokonaisuudeksi, jotta testauksessa löydetyt virheet on helpompi identifioida tiettyyn komponenttiin ja siten myös helpompi ja nopeampi korjata (Naik & Tripathy 2008, 51).

Yleensä yksikkötestaus jaetaan kahteen eri kategoriaan eli white box ja black box – testaukseen (Tamres 2002, 220). Naik ja Tripathy (2008, 52-53) käyttävät termejä Static testing ja dynamic testing kuten kuviossa 4 on demonstroitu.



KUVIO 4 Unit testing (Tamres 2002, 220; Naik & Tripathy 2008, 52, 54, 64)

White box –testauksessa nojaudutaan tehdyn koodin rakenteisiin ja siinä testataan siis lähinnä koodin toimivuutta ja etsitään siitä virheitä (Tamres 2002, 220). White box –testauksessa suoritetaan siis niin sanottu koodin katselmointi. Katselmointi suoritetaan ryhmässä joko niin, että yksi ryhmän jäsenistä johtaa tilaisuutta ja esittelee ohjelman koodin muille (walkthrough) tai sitten niin, että koodia tutkikaan yhdessä tasavertaisena ryhmänä (inspection). (Naik & Tripathy 2008, 54.)

Black box –testaus johdetaan tehdyistä määrityksistä ja muista vaatimuksista lähtien, jolloin black box –testauksessa löydetään myös loogisia ja toiminnallisia sekä määrittelyistä eroavia virheitä. (Tamres 2002, 220.) Black box –testauksessa lisäksi suoritetaan eli ajetaan varsinainen ohjelmakoodi (Naik & Tripathy 2008, 62).

Yksittäinen komponentti, jonka toimintaa Black Box –testauksessa testataan, on siis osa suurempaa kokonaisuutta, jolloin komponentin toiminta yleensä liittyy toisiin komponentteihin. Tämä tarkoittaa sitä, että testattava komponentti yleensä saa input-arvona tietoa toiselta komponentilta ja kun komponentti on käsitellyt tämän tiedon, ohjaa se sen eteenpäin seuraavalle komponentille, joka taas ottaa tiedon vastaan input-arvona. Unit testing –vaiheessa tämä täytyy ottaa kahdella eri tapaa huomioon. Ensinnäkin kun komponentin toimintaa testataan itsenäisenä, tarvitsee näiden input-arvojen antaminen ky-

seisellä komponentille hoitaa jäljittelemällä. Toisekseen testattava komponentti voi saada erilaisia input-arvoja ja nämä kaikki eri tilanteet tulisi ottaa jollain tapaa testauksessa huomioon, jotta testaaminen olisi kattavaa. (Naik & Tripathy 2008, 62-63.)

Kuviossa 4 on mainittu eri menetelmiä, joita voi käyttää black box –testauksessa. Näiden eri menetelmien avulla voi kartoittaa eri testilanteita ja siten valita testaukseen testitapauksen, jotka todennäköisesti paljastavat komponentin virheet. Control flow testing –menetelmässä kartoitetaan komponentin eri polut piirtämällä prosessikaavion kaltainen kuva komponentin toiminnallisuudesta ja siitä, miten sen suoritus etenee vaihe vaiheelta mahdollisien eri polkujen kautta. Data flow testing –menetelmässä voidaan piirtää myös prosessikaavion kaltainen kuva, mutta tällä kertaa keskitytään komponentissa käytettyihin muuttujiin ja niiden saamiin arvoihin (lukuihin sekä true/false) sekä siihen, että niitä käytetään oikeassa järjestyksessä komponentin suorituksen edetessä. Domain testing –testauksessa keskitytään enemmän siihen, miten komponentille annettu input –arvo vaikuttaa suoritettavan polun valintaan eli miten ohjelman suoritus etenee saadun input –arvon perusteella. Functional program testing –menetelmässä kiinnitetään erityisesti huomiota komponentille annettaviin input-arvoihin ja komponentin suorituksen jälkeen tuloksena syntyviin arvoihin ja yritetään matemaattista ajattelua hyväksikäyttäen valita sellaiset arvot, jotka testaavat hyvin komponentin toiminallisuutta. Mutation testing –menetelmän avulla arvioidaan edellisten menetelmien avulla saatujen testitapauksien laatua eli menetelmää voidaan käyttää apuna, kun valitaan monien testitapauksien joukosta ne, joiden avulla todennäköisimmin saadaan komponentin virheet esille. (Naik & Tripathy 2008, 64, 88-93, 112-119, 135-137, 222-223.)

2.5.2 Integraatiotestaus

Integraatiotestausta suoritetaan siinä vaiheessa, kun itsenäisiä komponentteja aletaan yhdistää toimivaksi sovellukseksi. Integraatiotestauksen avulla varmistutaan siitä, että yhdistetyt komponentit toimivat yhdessä esimerkiksi käyttöliittymätasolla ja että ne kommunikoivat keskenään niin kuin oli tarkoituskin. (Tamres 2002, 221.)

Vaikka jokainen ohjelmiston komponentti on testattu jo unit testing –tasolla, tarvitsee ne testata uudelleen, kun ne integroidaan toimivaksi kokonaisuudeksi. Jo unit testing –

vaiheessa otetaan huomioon komponenttien liittyminen toisiinsa input-arvojen avulla, mutta unit testing –testaus on kuitenkin suoritettu simuloidussa ympäristössä, kuten edellisessä luvussa kävi ilmi. Kun testaus on suoritettu simuloidussa ympäristössä, testaus ei paljasta kaikkia virheitä, joita tulee esille, kun komponentit oikeasti nivotaan yhteen ja suoritetaan testaus todellisessa toimintaympäristössä. (Naik & Tripathy 2008, 158-189.)

Integraatiotestauksessa voi käyttää eri tekniikoita:

- bottop-up
- top-down
- sandwitch
- big-bang

Eri tekniikoiden nimet kuvaavat sitä, missä järjestyksessä sovelluksen komponentteja testataan sovelluksen rakenteeseen nähden. Bottom-up tekniikassa testataan ensin alemman tason komponentit ja top-down tekniikassa päinvastoin. Sandwitch yhdistää nämä kaksi edellistä ja big-bang -tekniikassa testataan kaikki komponentit yhtä aikaa. (Tamres 2002, 221.)

Naik & Tripathy (2008, 164) esittelevät vielä yhden integraatiotestauksessa käytetyn tekniikan;

- incremental

Tässä testikierroksia on useita ja testattavien komponenttien määrä kasvaa jokaisella testikierroksella eli ensin integroidaan vain pari komponenttia ja testataan niiden välinen toiminta. Seuraavalla testikierroksella integroidaan kokonaisuuteen taas pari komponenttia lisää ja testataan jälleen kokonaisuuden toiminta. Eli testattavaa kokonaisuutta laajennetaan kierros kierrokselta ja aina välissä korjataan löydetyt virheet.

Käyttämällä eri tekniikoita pyritään siihen, että integroitu ohjelmisto toimii kokonaisuutena. Aluksi keskitytään siihen, että ohjelmiston rajapinnat toimivat sekä sisäisesti että ulkoisesti. Sisäisillä rajapinnoilla tarkoitetaan ohjelmiston komponenttien kommuni-

koimista keskenään. Ulkoisilla rajapinnoilla taas tarkoitetaan sitä, kun ohjelmiston komponentit kommunikoivat ulkoisen ympäristön kanssa. Rajapintoja testatessa keskitytään erityisesti moduulien keskenään lähettämiin tietoihin muun muassa niiden oikeaan muotoon, kokoon ja määrään. Rajapintojen toiminnan lisäksi keskitytään myös ohjelmiston toiminnallisuuteen. Unit testing –vaiheessa suoritut testitapaukset ajetaan läpi uudelleen ja varmistetaan, että jokainen komponentti toimii edelleen oikein, kun simuloitu ympäristö on korvattu integroidulla kokonaisuudella, jossa komponentit oikeasti kommunikoivat keskenään. (Naik & Tripathy 2008, 182-183.)

Rajapintojen ja toiminnallisuuden yhteydessä keskitytään paljon integroidun kokonaisuuden erillisiin komponentteihin, mutta on tärkeää testata myös koko kokonaisuuden toimivuus. Tämä varmistetaan niin sanotulla end-to-end testauksella. Tässä vaiheessa suoritetaan testitapauksia, joissa otetaan huomioon koko ohjelmisto eli testitapauksia, jotka käyttävät ohjelmiston useita eri komponentteja ja niissä suoritetaan kokonaisia toiminnallisuuksia. Lisäksi on tarpeen testata ohjelmiston toiminnallisuus muiden ohjelmistojen kanssa esimerkiksi verkon yli eli suoritetaan niin sanottu pairwise testing –testaus. (Naik & Tripathy 2008, 183.)

Näiden jälkeen integroidulle ohjelmistolle on syytä suorittaa vielä erilaisia stressi- ja kestävyys –testauksia. Stressi-testauksessa testataan muun muassa ohjelmiston virheen käsittelyä, lokien toimintaa ja tapahtumien käsittelyä eli esimerkiksi sitä, kuinka ohjelmisto antaa ilmoituksia ja varoituksia eri tilanteissa. Yleensä ohjelmistoissa on esimerkiksi asiakaskohtaisia komponentteja, joita kaikki asiakkaat eivät välttämättä käytä. Ohjelmistoa pitäisi myös testata erilaisilla kokoonpano vaihtoehdoilla ja tässä kohtaa erityisesti tulisi kiinnittää huomiota siihen, miten ohjelmisto toimii, kun kokoonpanoja muokataan. Lisäksi on syytä kiinnittää huomiota siihen, miten ohjelmisto toimii, kun muisti tai kovalevyn tila on vähissä tai kun prosessitehoja työasemassa, jossa ohjelmistoa suoritetaan, ei enää riitä. On myös syytä testata, miten paljon yhtäaikaisia toimintoja ohjelmisto pystyy suorittamaan eli miten se toimii kiireessä; jääkö kenties joitain toimintoja tai viestejä kokonaan suorittamatta? Testauksessa tulisi tutkia myös sitä, kuinka pitkään ohjelmistoa voi käyttää, ilman että se pitää uudelleenkäynnistää. (Naik & Tripathy 2008, 183-184.)

2.5.3 Järjestelmätestaus (system testing)

Järjestelmätestauksessa testataan kokonaisen tuotteen toimivuus verraten sitä projektin ja asiakkaan vaatimuksiin. Järjestelmätestaus on todella laaja kokonaisuus, jossa yritetään ottaa huomioon mahdollisimman monta asiaa ja testata, että tuote toimii kaikista näkökulmista, niin kuin on määritelty.

Tamreksen (2002, 222-223) mukaan järjestelmätestaukseen kuuluu seuraavien osa-alueiden testaaminen. Järjestelmätestauksessa testataan tuotteen ympäristö ja sen toimivuus. Ympäristön toiminnassa otetaan huomioon muun muassa laitteet, käyttöjärjestelmä ja verkkoyhteydet. Tuotteen toimintaa testataan myös eri kokoonpanoissa eli esimerkiksi eri oheislaitteilla ja käyttöjärjestelmillä. Testauksessa tarkistetaan myös tuotteen sisäisien diagnostiikkatyökalujen ja huoltotiedotteiden toiminta. Ennen näiden testaamista tuote on kuitenkin asennettava, joten testaus aloitetaan testaamalla tuotteen asentaminen ja poistaminen sekä päivityksien asentamisen toimivuus, kuten on määritelty. (Tamres 2002, 222-223.)

Järjestelmätestauksessa testataan myös tuotteen käytettävyys ja tarkoituksenmukaisuus. Testauksessa verrataan siis tuotetta tehtyihin määrittelydokumentaatioihin ja varmistetaan, että tuote toimii, kuten on määritelty. Tuotetta testataan myös käytettävyyden näkökulmasta. Myös käyttöohjeet tulee katselmoida. (Tamres 2002, 222-223.)

Järjestelmätestauksessa otetaan kantaa myös tuotteen luotettavuuteen ja toimintaan kuormituksen alla. Erilaisin testausmenetelmin halutaan varmistua siitä, että tuote ensinnäkin toimii normaalin käytön aiheuttamassa rasituksessa tietyn ajanjakson ajan. Tuotetta testataan myös erilaisissa kuormitustilanteissa ja seurataan sitä, kauanko eri toiminnot kestävät, kun tuotetta rasitetaan ja että tuote ylipäänsä pysyy käytettävänä ja käynnissä rasituksen alla. Ääriolosuhteissa testaamalla ei pyritä vain siihen, että saadaan tieto siitä, että tuote pystyy niissä toimimaan, vaan tällä pyritään myös etsimään tuotteen mahdollisia virhetilanteita ja ongelmia tuotteen toiminnassa. Järjestelmätestauksessa testataan myös se, että tuote palautuu ongelmien jälkeen. Eli esimerkiksi laitevian tai tuotteen kaatumisen jälkeen tuote toimii taas normaalisti. (Tamres 2002, 222-223.)

Järjestelmätestauksessa otetaan kantaan myös tuotteen tietoturvallisuuteen. Testauksen avulla halutaan varmistua siitä, että muun muassa vain sallitut henkilöt pääsevät tuotetta käyttämään. Tässä vaiheessa voidaan testata esimerkiksi käyttäjän sisäänkirjautumiseen ja tunnistautumiseen liittyviä asioita.

Naik & Tripathy (2008, 193-216) ovat koostaneet myös listan asioista, joita kuuluu järjestelmätestaukseen. Osittain listassa on samoja asioita kuin Tamreksen edellä esitellyssä listassa. Kuten Tamres myös Naik & Tripathy korostavat järjestelmätestauksessa tuotteen tarkoituksenmukaisuutta, käytettävyyttä ja tuotteen luotettavuutta. Naik & Tripathyn listasta löytyy myös joitain kohtia, joita ei ollut Tamreksen listauksessa.

Naik & Tripathy:n mukaan ennen varsinaisen järjestelmätestauksen aloittamista pitää suorittaa niin sanottu perus-testaus. Perus-testauksessa varmistutaan siitä, että tuote on valmis tarkempaan testaukseen. Tässä vaiheessa käydään läpi ohjelman perustoiminnallisuudet ja varmistetaan, että ne toimivat. Lisäksi testataan tuotteen käynnistyminen ja päivityksien onnistuminen sekä laitteiston toiminta. Naik & Tripathy ottavat järjestelmätestauksessa huomioon myös kolmannet osapuolet eli tuotteen toiminta tulee testata myös kolmannen osapuolten tuotteisiin nähden. (Naik & Tripathy 2008, 193-216.)

Tuotteen korjauksien ja versiopäivityksien jälkeen varmistetaan esimerkiksi jo aikaisemmin suoritettujen testitapauksien avulla, että tehdyt korjauksen ja muutokset eivät ole vaikuttaneet muihin toiminnallisuuksiin. Tätä vaihetta kutsutaan uudelleentestaukseksi (regression testing). Tässä testaustyypissä ei siis oteta kantaan tuotteen uusin ominaisuuksiin vaan testataan ainoastaan, että jo aikaisemmassa versiossa olleet ominaisuudet toimivat edelleen uudessa versiossa. (Naik & Tripathy 2008, 193-216; Tamres 2002, 223.)

Käytännössä uuden version myötä on tehty uusia ominaisuuksia sekä korjauksia vanhoihin toimintoihin, joiden takia ohjelmakoodia on muutettu. Tämä saattaa aiheuttaa muutoksia myös vanhojen toimivien ominaisuuksien toimintaan, jolloin tarvitsee varmistua siitä, että ne edelleen toimivat kuten aikaisemminkin.

2.5.4 Hyväksyntätestaus (acceptance testing)

Hyväksyntä-testauksen avulla varmistetaan, että tuote toimii asiakkaan vaatimusten mukaisesti ja että tuote on valmis tuotantokäyttöön. Tämä vaihe suoritetaan järjestelmätestauksen jälkeen. (Tamres 2002, 223-224.)

Hyväksyntätestaus voidaan jakaa kahteen eri vaiheeseen:

- asiakkaan suorittama hyväksyntätestaus (user acceptance testing)
- kehitysorganisaation suorittama hyväksyntätestaus (business acceptance testing)

User acceptance –testing tarkoittaa asiakkaan suorittamaa hyväksyntätestausta, joka yleensä suoritetaan asiakkaan tiloissa ja monesti vielä asiakkaan itsensä toimesta. Tässä testauksessa asiakas varmistaa, että ohjelmisto täyttää sopimukselliset kriteerit ja asiakkaan vaatimukset. Ennen asiakkaan suorittamaa hyväksyntätestausta kehitysorganisaatio voi itse suorittaa hyväksyntätestauksen eli Business acceptance testing. Tällä kehitysorganisaatio voi yrittää varmistaa, että asiakkaan suorittama hyväksyntätestaus menee läpi. (Naik & Tripathy 2008, 450-451.)

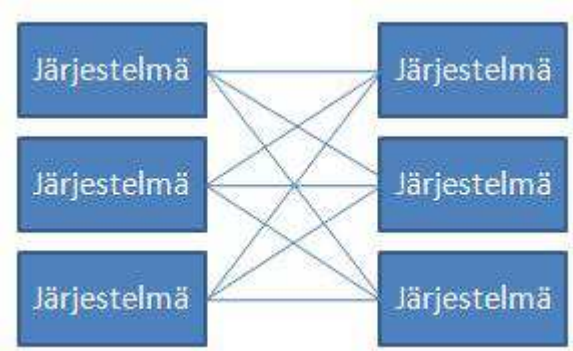
3 JÄRJESTELMÄINTEGRAATIOT

3.1 Mitä järjestelmäintegraatiot ovat?

Järjestelmäintegraatioista käytetään yleensä synonyymiä sovellusintegraatio. Englanninkielinen termi kuuluu systems integration. Riippumatta termistä, Tähtinen (2005, 48) on määritellyt järjestelmäintegraation seuraavasti: ”toimintamalleiksi ja tekniikoiksi, joiden avulla voidaan saattaa vähintään kaksi eri toiminnallisuutta tarjoavaa tietojärjestelmää jakamaan informaatiota siten, että informaation siirto ja muunnokset ovat kontrolloitavissa ja monitoroitavissa yhdestä tai useammasta keskitetystä pisteestä”.

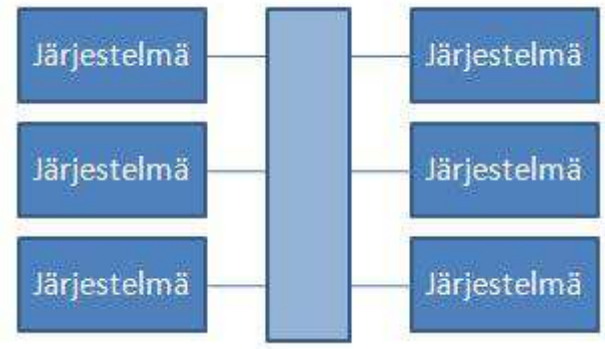
Järjestelmäintegraatiot voidaan jakaa sisäisiin ja ulkoisiin integraatioihin. Sisäisistä integraatioista puhuttaessa tarkoitetaan yleensä tiedon siirtämistä yrityksen sisäisten järjestelmien välillä. Tällöin käytetään yleensä termiä EAI eli Enterprise Application Integration. Nykyään yritykset ovat kuitenkin verkottuneita ja keskittyvät omaan ydinosaamiseen, jolloin käytetään paljon kumppaneita, joten yrityksen seinien ulkopuolellakin on paljon informaatiota, jota yritys tarvitsee. Tällöin yrityksen järjestelmiä pitää integroida yrityksen ulkopuolisten järjestelmien kanssa. Tämäntyyppisestä integraatioratkaisusta käytetään yleensä termiä B2Bi eli business-to-business integration tai B2Bai eli business-to-business application integration. (Tähtinen 2005, 33-34.)

Järjestelmäintegraatioiden toteuttamiseksi on oikeastaan kaksi vaihtoehtoa. Yksinkertaisin ja järjestelmäintegraatioiden historiassa ensimmäinen tapa lähteä toteuttamaan järjestelmäintegraatiota on niin sanottu point-to-point –integraatio. Tässä integroitavien ohjelmien välille rakennetaan linkkejä tarvittaessa (katso kuvio 5), jolloin integroinnin hallinnointi on hajautettua. Point-to-point –integrointi sopii yleensä pienille yrityksille, joissa on tarve integroida vain muutama yksittäinen sovellus. Useiden sovelluksien integrointiin tämä ei ole hyvä ratkaisu, sillä linkkien määrä lisääntyy eksponentiaalisesti ja pian integraatio kasvaakin hallitsemattomaksi. Myös muutoksien tekeminen point-to-point –integraatioon on hankalaa, sillä jos yhden sovelluksen rajapinta muuttuu, tarvitsee muutos tehdä jokaiseen linkkiin, jolla kyseinen sovellus on yhdistetty muiden sovelluksien kanssa, erikseen. (Tähtinen 2005, 65-66 ja 144.)



KUVIO 5 Point to Point integraatio (mukaellen Tähtinen 2005, 30)

Seuraava kehitysvaihe järjestelmäintegraatioiden toteuttamiseksi oli keskitetty integraatio eli niin sanottu hub-and-spoke –arkkitehtuuri. Tässä lähestymistavassa sovellusten välisiä tiedonsiirtoja, muunnoksia ynnä muuta voidaan valvoa ja kontrolloida keskitetysti yhdeltä palvelimelta tai työasemalta (katso kuvio 6). Keskitetty integraatio soveltuu monimutkaisiinkin kokonaisuuksiin, sillä tässä linkkien määrä kasvaa vain lineaarisesti, toisin kuin point-to-point –integraatiossa. Keskitetyn integraation vaarana on kuitenkin se, että luodaan ratkaisu, joka on täysin riippuvainen tästä yhdestä pisteestä. Ja mikäli tuo piste vikaantuu, saattaa kaikki tiedonsiirto yrityksessä vaarantua. (Tähtinen 2005, 66-69 ja 143-144.)



KUVIO 6 Keskitetty integraatio (mukaellen Tähtinen 2005, 30)

Seuraava kehitysvaihe järjestelmäintegraatioissa liittyy Service Oriented Architecture (SOA) –malliin eli palvelukeskeiseen arkkitehtuuriin, jolla pyritään informaation jakamiseen järkevästi ja läpinäkyvästä. Palvelukeskeinen arkkitehtuuri on enemmänkin ajattelumalli kuin tekninen toteutustapa ja se on riippumaton laitteista, käyttöjärjestelmistä ja ohjelmointitekniikoista. SOA-mallin avulla on tarkoitus saattaa eri ohjelmistojen tarjoama tietosisältö helposti muiden ohjelmistojen käyttöön. Ohjelmistot tarjoavat sisäl-

tämäänsä tietoa muille ohjelmistoille palveluna standardinmukaisten rajapintojen ja esitystapojen avulla. Esimerkiksi taloushallinnon ohjelmisto voi tarjota palvelun, jolla välitetään kysyjälle kyseisessä järjestelmässä olevia asiakastietoja. Standardoidut rajapinnat ja esitystavat mahdollistavat sen, että käytetyllä teknologialla ei ole merkitystä ja tiedonhaku onnistuu helposti, yksinkertaisesti ja joustavasti. (Tähtinen 2005, 96-97 ja 144-145.)

SOA-malli ei kuitenkaan tarkoita paluuta point-to-point tyyppisiin integraatioratkaisuihin, vaikka se kuulostaakin siltä, että ohjelmisto voisi itse kysyä tietoa toiselta ohjelmistolta. Järjestelmäintegraatio-ratkaisun rooliksi jää edelleen kokonaisuuden kontrollointi ja informaatiosisältöjen yhdenmukaisuus. Järjestelmäintegraatioratkaisu vastaa SOA-mallissa palveluiden rekisteröinnistä, julkaisusta ja palvelupyyntöjen ohjaamisesta ja välittämisestä oikeaan osoitteeseen. Palvelua kysyvän ohjelmiston ei siis tarvitse tietää osoitetta, josta se tiedon saa, vaan järjestelmäintegraatioratkaisu vastaa tästä. Tällaista järjestelmäintegraatioratkaisua kutsutaan Enterprise Service Bus (ESB) –ratkaisuksi. Monimutkaisemmissa tapauksissa järjestelmäintegraario-ratkaisun avulla voidaan rakentaa myös niin sanottuja komposiittisovelluksia, joissa palvelun tarvitsema tieto haetaan, ei vaan yhdestä palvelusta, vaan useasta palvelusta. (Tähtinen 2005, 98-100 ja 144-146.)

3.2 Miksi järjestelmäintegraatiot ovat tarpeellisia?

Miksi yrityksen käyttämiä järjestelmiä sitten pitää integroida keskenään eli saada ne juttelemaan toistensa kanssa ja siirtämään keskenään tietoa? Tähtisen (2005, 23) mukaan järjestelmäintegraatioilla tavoitellaan yleensä kustannussäästöjä, joustavuutta ja raportoinnin ja monitoroinnin kehittämistä.

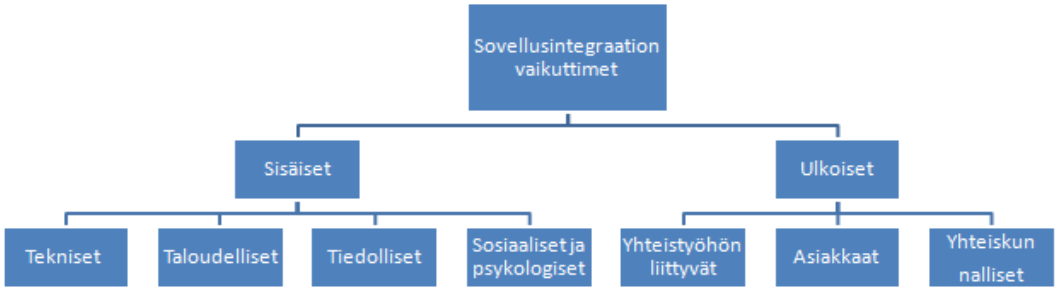
Järjestelmäintegraatio-ratkaisun käyttöönotolla pyritään virtaviivaistamaan ja tehostamaan liiketoimintaprosesseja. Yrityksien liiketoimintaprosesseissa käytetään nykyään paljon erilaista tietoa, kuten myyntilukuja ja –tilastoja. Järjestelmäintegraatioiden avulla pyritään siihen, että päätöksentekoon tarvittava tieto on luotettavammin ja nopeammin liiketoimintaprosessin käytettävissä. Automatisointi, nopeuttaminen ja virheiden vähen-

tyminen johtavat kustannussäästöihin, parantavat asiakastyytyväisyyttä ja sitä myöden kannattavuutta ja yrityksen kilpailukykyä. (Tähtinen 2005, 23-26.)

Joustavuudella Tähtinen tarkoittaa sitä, että integraatioratkaisun avulla yritys voi varautua erilaisiin muutoksiin liiketoiminnassaan. Hyvin suunniteltu integraatioratkaisu tuo joustavuutta esimerkiksi tilanteessa, jossa yrityksen liiketoimintaprosesseissa tai yrityksen organisaatiossa tapahtuu muutoksia. Joustavasti rakennettu integraatioratkaisu vähentää myös riippuvuutta yksittäisistä järjestelmistä ja järjestelmätoimittajista. Tämä kaikki on mahdollista, kun integraatioratkaisu on suunniteltu niin, että järjestelmät eivät ole linkitetty suoraan keskenään vaan integraatioratkaisun avulla, jolloin yhden järjestelmän muuttuminen ei esimerkiksi vaadi muutoksia useassa paikassa. Ja kun kaikki tieto kulkee ja ohjataan integraatioratkaisun kautta, on tietovirtojen suuntia ja aikatauluja helppo muuttaa esimerkiksi liiketoimintaprosessin muutoksen yhteydessä. (Tähtinen 2005, 27-31.)

Järjestelmäintegraatioratkaisu toimii yrityksen tietojärjestelmien linkkinä eli se välittää tietoa järjestelmästä toiseen jatkokäsittelyä varten. Tämän roolin vuoksi järjestelmäintegraatioratkaisu on luonnollinen paikka raportointia ja monitorointia varten. Tästä yhdestä pisteestä saa parhaiten kuvan yrityksen prosessien tilasta. Missä vaiheessa mikin prosessi parhaillaan on? Kuinka kauan sen suoritus kestää? Kuinka usein tietty prosessi aktivoituu esimerkiksi vuorokauden aikana? Tätä kaikkea tietoa voidaan käyttää muun muassa liiketoimintaprosessien analysointia varten tai seurantaa varten luomalla esimerkiksi erilaisia hälytyksiä järjestelmäintegraatioratkaisuun. (Tähtinen 2005, 31-32.)

Siltasen (2004, 30-32) mukaan järjestelmäintegraation valintaan vaikuttavat sekä yrityksen sisäiset että ulkoiset syyt (katso kuvio 7). Yrityksen sisäiset syyt Siltanen on jakanut teknisiin, taloudellisiin, tiedollisiin ja sosiaalisiin sekä psykologisiin syihin. Ulkoiset syyt Siltanen on jakanut yhteistyöhön liittyviin, asiakkaisiin ja yhteiskunnallisiin syihin.



KUVIO 7 Integraation vaikuttimet (mukaellen Siltanen 2004, 32)

Teknisten syiden taustalla on yleensä se, että yrityksissä vuosien mittaan tehty järjestelmäkehitys on ajanut tilanteen siihen, että yrityksellä on käytössä useita erilaisia järjestelmiä. Yksi ongelmakohta on aina paikattu hankkimalla järjestelmä, joka on kyseisen ongelman ratkaissut, mutta hankinta on tehty ilman, että olisi ajateltu sen enempää tulevaisuutta järjestelmien parissa. Tähän ongelmaan on aikaisemmin tarjottu ratkaisuksi ERP-toiminnanohjausjärjestelmiä, mutta edes ne yksinään eivät välttämättä riitä kattamaan kaikkia yrityksen tietojärjestelmätarpeita. ERP-järjestelmän käyttöönoton myötä yrityksessä ei myöskään aina haluta luopua vanhoista järjestelmistä, jos niiden käytöstä kuitenkin saadaan vielä jotain lisäarvoa. Näiden erilaisten teknisten syiden vuoksi yrityksellä on käytössää useita eri järjestelmiä, jotka pitäisi saada jotenkin kommunikoimaan keskenään. (Siltanen 2004, 28-29.)

Tietoon liittyvät syyt liittyvät teknisiin syihin eli juontuvat siitä, että käytössä on useita eri järjestelmiä. Kun käytössä on useita eri järjestelmiä, tieto eriytyy järjestelmäkohtaiseksi. Tämä saattaa aiheuttaa ongelmia, kun samalla tiedolla tai samalla käsitteellä voi olla eri merkityksiä eri järjestelmissä. Tiedoissa voi olla myös päällekkäisyyksiä. Usein tilanne on myös se, että yksikään järjestelmistä ei kata vaadittavaa tietokokonaisuutta täydellisesti, jolloin järjestelmien integraatio on välttämätöntä, jotta yritys pystyisi käyttämään liiketoiminnassa tarvittavia tietoja helposti ja luottaa myös järjestelmästä saataviin tietoihin. (Siltanen 2004, 30.)

Sosiaaliset ja psykologiset syyt liittyvät lähinnä yrityksen työntekijöiden työn mielekkyyteen ja työmotivaatioon. Ilman järjestelmien integraatiota työntekijät joutuvat kopioimaan käsin huomattavan määrän tietoa järjestelmästä toiseen, päivittäin. Tällainen työ

ei ole pitkän päälle kenenkään mielestä mielekästä eikä motivoivaa, saati sitten että se oli työn kustannuksen kannalta mitenkään järkevää. (Siltanen 2004, 30.)

Taloudelliset syyt syntyvät näiden muiden tekijöiden yhteisvaikutuksesta. Monimutkaisen, lukuisia eri järjestelmiä sisältävät tietojärjestelmäarkkitehtuurin ylläpitäminen on kallista. Integraation myötä käsin tehty työ vähenee, mikä vähentää työn kustannuksia ja nopeuttaa tiedon siirtymistä paikasta toiseen. Tiedon nopea siirtyminen ja sitä myötä tiedon käytettävyys ja tiedon eheyden myötä sen luotettavuus lisäävät liiketoiminnallisia mahdollisuuksia tehostaa ja parantaa toimintaa sekä reagoida muutoksiin nopeammin.

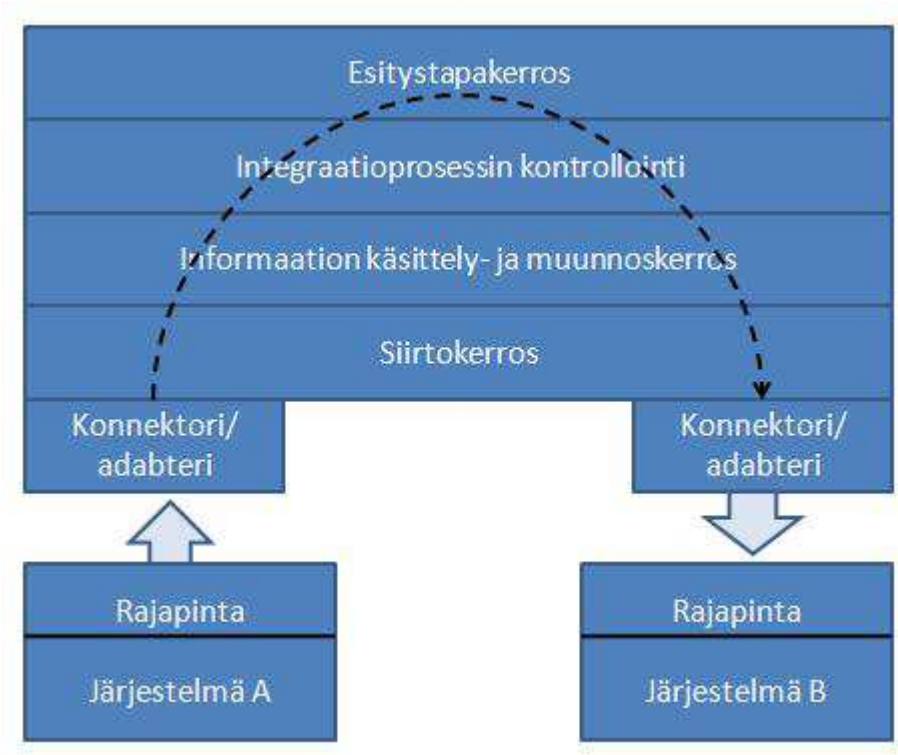
Integraatioratkaisun hankinnan ulkoisia syitä olivat siis asiakkaat ja yhteistyöhön liittyvät syyt sekä yhteiskunnalliset syyt. Yhteistyöhön liittyvien syiden taustalla on usein erilaisten yhteistyökumppaneiden ja toimittajien sekä hankkijoiden kanssa perustetut verkostot. Kilpailuetua halutaan luoda esimerkiksi automatisoimalla tilaus-toimitusketjuja, jolloin yrityksen tietojärjestelmien tulee olla integroituina yhteistyökumppaneiden järjestelmien kanssa. Ulkoisina syinä integraation valintaan vaikuttavat myös asiakassuhteet ja niiden ylläpitämiseksi tarkoitetut järjestelmät. Asiakkaille halutaan tarjota lisäarvoa mahdollistamalle heille palvelukanava yrityksen järjestelmiin. Myös yhteistyöhön liittyvät syyt ovat taustalla asiakas-syissä. Yhteiskunnallisten syiden taustalla voi olla esimerkiksi lainsäädäntö tai direktiivit, jotka vaativat yrityksiä kehittämään järjestelmiään, kun niistä pitää esimerkiksi siirtää jotain tietoa valtion tai kuntien käyttöön. (Siltanen 2004, 30-31.)

3.3 Integraatioratkaisuiden perusarkkitehtuuri

Integraatioratkaisuissa on yksinkertaisimmillaan siis kyse siitä, että vähintään kaksi keskenään yhteen sopimatonta järjestelmää saadaan keskustelemaan keskenään. Tällöin järjestelmien välillä täytyy pystyä siirtämään tietoa ja tieto pitää pystyä muuntamaan siten, että toinen järjestelmä ymmärtää toisen järjestelmän tuottamaa tietoa. Lisäksi tätä siirtoprosessia pitäisi pystyä kontrolloimaan ja valvomaan. (Tähtinen 2005, 48.)

Seuraavaksi käydään läpi tarkemmin, mitä tämä vaatii. Kuviossa 8 on kuvattu integraatioratkaisun koostumus eli eri kerrokset, joita vaaditaan integraatioratkaisun toteuttami-

seksi. Kuviossa 8 on myös kuvattu nuolilla, miten tieto liikkuu näiden eri kerroksien kautta järjestelmästä A järjestelmään B. Kuviossa 8 on kuvattu kaikki integraation mahdolliset kerrokset, jotka käydään seuraavassa läpi, mutta huomionarvoista on se, että yksinkertaisimmissa toteutuksissa ei välttämättä ole tarvetta kaikille näille kerroksille.



KUVIO 8 Integraatioratkaisun kerrokset (mukaellen Tähtinen 2005, 72)

Informaation siirto kahden eri järjestelmän välillä vaatii, että molemmat järjestelmät tarjoavat jonkinlaisen rajapinnan, jonka avulla järjestelmästä voidaan hakea tai sinne voidaan syöttää informaatiota. Yksinkertaisimmillaan tämä voi tarkoittaa esimerkiksi tekstipohjaista siirtotiedostoa, jonka järjestelmä tuottaa tai osaa lukea sisäänsä. Myös integraatioratkaisulla pitää olla rajapinta, joka taas osaa ottaa vastaan järjestelmästä haetun tiedon, jotta se voidaan välittää eteenpäin toiselle järjestelmälle. Näitä integraatioratkaisun rajapintoja kutsutaan konnektoreiksi ja adabtereiksi. Konnektori on yksinkertaisempi rajapinta, joka vaan välittää saamansa tiedon eteenpäin, adabteri puolestaan ymmärtää hiukan enemmän välitettävästä aineistosta ja voi tulkita sitä. (Tähtinen 2005, 49-50 ja 117-120.)

Lisäksi informaation siirto vaatii jonkin fyysisen siirtotien (kuviossa 8 kohta ”siirtokerros”). Siirtotie voi olla yleinen tietoverkko esimerkiksi yrityksen sisäverkko tai joku

sanomajärjestelmä. Ulkoisissa integraatioissa siirtoteissä tulee ottaa huomioon tietoturva, kun tietoa joudutaan siirtämään julkisen verkon yli. (Tähtinen 2005, 50-52.)

Osana siirtokerrosta on monimutkaisemmissa ratkaisuissa välityskerros, jossa toimivat erilaiset etäkutsut ja sanomansiirto. Etäkutsut tarkoittavat synkronista tiedonsiirtoa, jossa sovellus pystyy kutsumaan toisen sovelluksen tarjoamaa palvelua ja odottaa sen antamaa vastausta. Sovellus ei pysty jatkamaan toimintaansa ilman, että se vastauksen, siksi tiedonsiirto on siis synkronista. Tästä voisi olla esimerkkinä vaikka asiakkaan saldon automaattinen tarkistus toiminnanohjausjärjestelmästä myyntitilanteessa. Sanomansiirto tai –välitys tarkoittavat asynkronista tiedonsiirtoa, jossa sovellus julkaisee sanoman yhteiseen sanomajonoon, josta muut sovellukset voivat lukea sanoman, kun se niille parhaiten sopii. Sanoman otsikkotiedoissa kerrotaan muun muassa, kuka viestin on lähettänyt ja kenelle se on tarkoitettu, jotta oikea sovellus osaa ottaa viestin käsittelyyn jonosta. Viesti odottaa siis jonossa käsittelyä määritellyn ajan ja sanoman julkaissut sovellus voi jatkaa toimintaansa julkaisun jälkeen. Sanomansiirto mahdollistaa joustavan toiminnan; viestin julkaissut sovellus voi jatkaa toimintaansa jouhevasti ja toisaalta taas esimerkiksi huolto- tai verkkokatkot eivät aiheuta viestien katoamista, kun vastaanottava osapuoli voi noutaa viestin jonosta katkon jälkeen. (Tähtinen 2005, 52-53 ja 122-125.)

Integraatioratkaisuiden tarkoituksena on saada yhteen sopimattomat järjestelmät keskustelemaan keskenään. Rajapinnan ja siirtokerroksen avulla saadaan tieto siirtymään sovellukselta A sovellukselle B, mutta yleensä sovellukset on toteutettu eri lailla siten, että ne eivät suoraan ymmärrä toistensa tuottamaa informaatiota. Tällöin integraatioratkaisun avulla tieto tulee muuntaa toisen sovelluksen ymmärtämään muotoon (kuviossa 8 kohta ”Informaation käsittely- ja muuntokerros”). Integraatioratkaisun tehtävänä onkin siis tulkita tietyissä tapauksissa lähettävän järjestelmän informaatiota ja muuntaa se vastaanottavan järjestelmän ymmärtämään muotoon. Integraatioratkaisun tulee myös osata ohjata käyttäytymistään lähettävän järjestelmän informaation perusteella. Jos esimerkiksi informaatio on puutteellista, tulee integraatioratkaisun reagoida hälytyksin tai tarvittaessa yrittää hakea tietoa uudelleen. Integraatioratkaisun tulee osata myös täydentää lähetettävää informaatiota erilaisten vakioasetusten avulla esimerkiksi muuntamalla erilaisia mittayksiköitä. (Tähtinen 2005, 54-59.)

Koska integraatioratkaisu on tärkeä osa yrityksen toimintaa ja se palvelee yrityksen liiketoimintaprosesseja, tarvitsee tiedon siirtyä järjestelmästä toiseen kontrolloidusti. Edellä esitettyjen siirto- sekä käsittely- ja muuntokerroksien tehtävät eivät kuitenkaan tapahdu spontaanista tai itsekseen vaan tarvitaan integraatioprosessin kontrollointikerros (katso kuvio 8), jonka avulla kontrolloidaan sitä, milloin tietoa noudetaan, siirretään, käsitellään ja muunnetaan, jotta se palvelee yrityksen prosesseja. Nämä tehtävät voidaan määritellä alkamaan tiettyinä ajankohtina ajastuksien avulla tai sitten niihin voidaan määritellä ulkoisia herätteitä esimerkiksi käyttäjän antama pyyntö. (Tähtinen 2005, 58-65.)

Kuviossa 8 ylimpänä kerroksena on esitystapakerros. Tämä kerros toimii rajapintajana integraatioratkaisun ja ihmisten välillä. Tämän kerroksen tarkoituksena on tuottaa näkymä, jonka avulla yrityksen johto ja työtekijät saavat tietoa integraatioprosessien tilasta ja toiminnasta. Yksinkertaisimmillaan tämä voi tarkoittaa sähköpostilla lähetettävää raporttia, jossa kerrotaan esimerkiksi jonkin tietyn integraatioprosessin läpimenoajoista. (Tähtinen 2005, 71.)

Integraatioratkaisun tehtävänä on siis siirtää tietoa paikasta toiseen määritellyn toimintamallin mukaisesti. Integraatioratkaisu voi myös tarvittaessa muuntaa tietoa ja valvoa sen sisältöä ja siirron onnistumista. Kehittyneemmissä ratkaisuissa on myös käyttöliittymät, joiden avulla tavallinenkin käyttäjä pääsee kurkistamaan, miten siirrot etenevät ja onnistuvat. Kuten jo luvun alussa mainittiin, kaikkia näitä ominaisuuksia ei välttämättä aina tarvita, vaan esimerkiksi joissain yksinkertaisimmissa järjestelmäintegraatioissa pelkkä tiedostosiirto ja sen valvonta voi tulla kysymykseen.

3.4 Integraatioratkaisuiden haasteet

Keskitetyn integraatioratkaisun ongelmakohdaksi voi muodostua niin sanottu single point of failure, kun yrityksen kaikki liiketoimintaprosessien kannalta oleellinen informaatio kulkee yhden pisteen kautta. Jos tämä yksi piste vikaantuu, pahimmassa tapauksessa koko yrityksen toiminta voi pysähtyä. (Tähtinen 2005, 65-69.) Integraatioratkaisu saattaa jonkin teknisen vian vuoksi olla pois kokonaan käytöstä, jolloin liikenne tuon pisteen kautta lakkaa kokonaan. Tämä voi tapahtua jos esimerkiksi palvelin, jolla integ-

raatioratkaisuun liittyvät sovellukset pyörivät, hajoaa. Ongelmaksi voi muodostua myös integraatioratkaisun hidastuminen tai tökkiminen, jos tämän yhden integraatioratkaisun kautta ohjataan niin paljon liikennettä ja tapahtumia, että palvelimen suorittimen tehot tai muisti loppuu. Ongelmia voi aiheuttaa myös tietoliikenne-yhteyksien kuormittuminen.

Integraatioratkaisu tuleekin toteuttaa siten, että se ei saa muodostua tällaiseksi yksittäiseksi virhe-pisteeksi vaan integraatioratkaisun tulee olla erittäin vikasietoinen. Vikasietoisuutta tulee pohtia kahdella eri tasolla: teknisen alusta toimivuutena ja integraatioratkaisun loogisena toimivuutena. Alustan tekninen toimivuus voidaan varmistaa samoin kuin minkä muunkin tapansa yrityssovelluksen toimivuus. Integraatioratkaisujen kohdalle täytyy kyllä lisäksi ottaa kantaa esimerkiksi palvelimien kahdentamiseen ja tarvittavan nopeisiin tietoliikenneyhteyksiin. Loogisella vikasietoisuudella tarkoitetaan sitä, että mikäli joku integraatioprosessiin kuuluva järjestelmä ei vastaa, esimerkiksi huoltokatkon vuoksi, ei integraatioprosessi voi toimia oikein. Loogista vikasietoisuutta voidaan parantaa varautumalla ongelmatilanteisiin seuraavilla toimenpiteillä:

- ongelman tiedostaminen
- uudelleenyritykset
- vaihtoehtoisen ratkaisun löytäminen
- hälytykset (Tähtinen 2005, 105-107 ja 144.)

Integraatioratkaisun avulla siirrettävä informaatio on yleensä yrityksen liiketoiminnan kannalta kriittistä. Tämän vuoksi tietoturvaan onkin kiinnitettävä erityistä huomiota. Tietoturvan tarkoituksena on yleisesti järjestelmäintegraatioiden kannalta varmistaa, että tieto pystyy luottamuksellisena, eheänä ja saavutettavissa. Luottamuksellisuus tarkoittaa sitä, että tietoon pääsee käsiksi vain ne henkilöt ja ohjelmistot, joilla on siihen oikeus. Tiedon eheys tarkoittaa sitä, että tiedon siirrossa ja säilytyksessä tietoa pysyy muuttumattomana. Eli esimerkiksi taloushallinnon luvuissa pilkut pysyvät kohdallaan. Saatavuus tarkoittaa sitä, että tieto saadaan käsiteltäväksi viiveettä kaikissa tilanteissa. Yrityksen sisäisissä integraatioissa tiedon luottamuksellisuuteen ei yleensä tarvitse kiinnittää niin paljon huomiota, kuin yrityksen ulkoisissa integraatioissa. Eheys ja saavutettavuus ovat kuitenkin tärkeitä myös sisäisissä integraatioissa. (Tähtinen 2005, 110-112.)

Integraatioratkaisun arkkitehtuuria suunnitellessa päämääränä tulee olla mahdollisimman automatisoitu tiedonsiirto. Manuaalista työtä pitäisi siis välttää. Arkkitehtuurissa pitää ottaa huomioon herätteet, joiden perusteella integraatioratkaisun tulee tietoa siirtää. Nämä herätteet ovat yksinkertaisimmillaan erilaisia ajastuksia. Herätteinä voidaan kuitenkin käyttää esimerkiksi sitä, onko joku tieto olemassa tai onko joku integraatiotehtävä suoritettu. (Tähtinen 2005, 104-105.) Eli esimerkiksi se, että johonkin tiettyyn kansioon ilmestyy tietynniminen tiedosto, voi toimia integraatioratkaisun herätteenä.

Valvottavuus on tärkeää integraatioratkaisun läpinäkyvyyden kannalta. Valvonta voi olla joko tosiaikaista tai viiveellä tapahtuvaa. Integraatioprosessit tulee suunnitella siten, että informaatiota voidaan siirtää prosessin sopivissa vaiheissa. Valvonta voi perustua raportointitehtäviin, mutta yleensä valvonta liittyy vikasietoisuuteen; mikäli integraatioprosessit eivät toimi halutulla tavalla, yrityksessä työskentelevät henkilöt varmasti haluavat tästä tiedon. (Tähtinen 2005, 108.)

Yrityksen integraatioratkaisuun nivotut tietojärjestelmät muuttuvat ja vaihtuvat jatkuvasti, joten integraatioratkaisunkin on oltava laajennettavissa ja muutettavissa. Muutos voi edellyttää vain pieniä parametrointeja integraatioratkaisuun, mutta siinä voi olla kyse myös integraatioprosessien täydellisestä uudelleenmäärittelystä, jos järjestelmän käyttötarkoitus muuttuu. Muutoksiin pitäisi aina varautua hyvän dokumentoinnin avulla. (Tähtinen 2005, 109-110.)

Integraatioratkaisua rakennettaessa tulee siis kiinnittää huomiota moneen seikkaan. Teknisten näkökulmien lisäksi tulee kiinnittää huomiota myös siirrettävään tietoon ja sen tietoturvaan. Myös tulevaisuuden kehitystarpeet ja ratkaisun laajennettavuus on otettava huomioon. Ehkä riskianalyysi on paikallaan vaihtoehtojen punnitsemisen ja päätöksenteon yhteydessä. Niin moni asia voi mennä pahasti pieleen, jos integraatioratkaisu ei toimi niin kuin on suunniteltu. Integraatioratkaisun toiminnalla on kuitenkin yleensä niin suuri liiketoiminnallinenkin merkitys.

4 JÄRJESTELMÄINTEGRAATIOIDEN TESTAAMINEN

Edellisissä luvuissa on käyty läpi testaamisen ja järjestelmäintegraatioiden teoriaa. Testauksen teoriassa on käynyt ilmi testauksen merkitys, tarkoitus ja tarpeellisuus. Ilman testausta voi tuotantoon päästä vakaviakin virheitä, jotka saattavat aiheuttaa suuria ongelmia yrityksen toimintaan ja maineeseen. Järjestelmäintegraatioiden teoriasta on käynyt ilmi järjestelmäintegraatioiden suuri merkitys liiketoiminnassa. Jos järjestelmäintegraatio ei toimi niin kuin pitäisi, saattaa koko yrityksen toiminta vaarantua, kun liiketoimintaan liittyvä informaatio ei kulje yrityksien järjestelmien välillä. Toisaalta toimiessaan järjestelmäintegraatiot mahdollistavat yritykselle kilpailuetua ja taloudellisia säästöjä.

Testaaminen on siis tärkeää ohjelmistotuotannossa ja liiketoiminnan kannalta. Testaamisen merkitys kuitenkin korostuu liiketoimintakriittisten ratkaisuiden, kuten järjestelmäintegraatioiden, yhteydessä. Järjestelmäintegraatio-ratkaisun luotettava toiminta on tärkeää ja miten muuten kuin testaamalla ratkaisun toimivuudesta voidaan päästä varmuuteen.

4.1 Mitä järjestelmäintegraatiotestaaminen on?

Järjestelmäintegraatioiden testaamista voisi verrata integraatio-testaamiseen. Kuten aikaisemmista luvuista kävi ilmi, niin integraatio-testaamista suoritetaan siinä vaiheessa, kun yksittäisiä testattuja komponentteja aletaan yhdistää toimivaksi kokonaisuudeksi. Integraatiotestauksella pyritään siihen, että erikseen toimiviksi todetut komponentit, toimivat myös yhdistettynä kokonaisuutena. Järjestelmäintegraatioiden testaamisessa on sama periaate, mutta testattavat kokonaisuudet ovat isompia; järjestelmäintegraatiotestauksessa liitetään yhteen komponenttien sijaan kokonaisia järjestelmiä ja/tai sovelluksia ja testataan niiden välistä vuorovaikutusta eli järjestelmien välisen rajapinnan toimintaa (ISTQB Foundation level Syllabus 2010, 24).

Järjestelmäintegraatiotestausta tehdään järjestelmätestauksen jälkeen eli järjestelmäintegraatiotestauksessa ei enää kiinnitetä huomiota itse järjestelmien toimintaan, vaan pelkästään rajapintojen toimintaan. Tiettyjen ei-toiminnallisten piirteiden kuten suorituskyvyn testaus voidaan ottaa mukaan järjestelmäintegraatiotestaukseen. (ISTQB Foundation level Syllabus 2010, 24.)

4.2 Järjestelmäintegraatioiden testaamisen haasteet

Kuten aikaisemmin on jo käynyt ilmi, niin järjestelmäintegraatioiden avulla toteutetaan yrityksen liiketoimintaprosesseja, jolloin toimintaketjut voivat olla pitkiäkin ja käsittää useita järjestelmiä. Tämä asettaa haasteita järjestelmäintegraation testaukseen. Testaajat voivat olla esimerkiksi osastokohtaisia, jolloin heillä ei ole välttämättä osaamista, tietoa eikä edes pääsyä kaikkiin järjestelmiin toimintoketjuissa. (ISTQB Foundation level Syllabus 2010, 24.) Pitkät toimintoketjut ja laaja kokonaisuus aiheuttavat myös itse testaukselle haasteita; testitapauksia tulee paljon ja ne ovat monimutkaisia.

Testauksessa tulee ongelmia myös löydettyjen virheiden paikallistamisessa ja rajaamisessa. Mitä suurempi integraatiokokonaisuus on, sitä vaikeammaksi tulee häiriöiden eristäminen tiettyyn komponenttiin tai järjestelmään, mikä voi johtaa lisääntyneisiin riskeihin ja ongelmanselvityksen ajantarpeen kasvamiseen. (ISTQB Foundation level Syllabus 2010, 24.)

Monesti järjestelmäintegraatioissa ei ole kyse vain yrityksen sisäisten järjestelmien integroimisesta, vaan kuvioon astuu mukaan yrityksen ulkopuolisia järjestelmiä. Nämä ulkopuoliset järjestelmät saattavat aiheuttaa erityisiä haasteita testaukselle. Näihin järjestelmiin ei ole yleensä yrityksen sisäisillä testaajilla pääsyä, joten testaus pitää yhdessä sopia ja järjestää tämän ulkopuolisen organisaation kanssa. Osa ulkopuolisen järjestelmätoimittajan tarjoamista toiminnallisuuksista voi myös mahdotonta testata; miten esimerkiksi voi testata sitä, että asiakkaan tililtä oikeasti veloitetaan maksu tai että tavara toimitetaan oikeaan osoitteeseen (Hohpe & Istvanick 2002, 7)?

Ulkopuoliset järjestelmät aiheuttavat haasteita myös testiympäristölle. Ne eivät välttämättä tarjoa kaikkiin toiminnallisuuksiinsa erillistä ympäristöä pelkästään testaamista

varten. (Hohpe & Istvanick 2002, 7.) Järjestelmäintegraatioiden testauksen kannalta testiympäristö on jo ilman ulkopuolisia osapuoliakin haasteellinen. Esimerkiksi kaikkia verkkoympäristöön ja kuormitukseen liittyviä tilanteita voi olla lähes mahdoton simuloida. Miten esimerkiksi testiympäristössä voi ottaa huomioon tilanteen, jossa integroidaan esimerkiksi useita eri puolilla maata olevia konttoreita?

Järjestelmäintegraatiot perustuvat yleensä asynkroniseen tiedonsiirtoon. Tämä tarkoittaa siis sitä, että siirrettäessä tietoa, vastaanottava osapuoli ei välttämättä heti ota tietoa käsittelyyn ja vastaa lähettävälle osapuolelle. Tämä aiheuttaa testaukselle haasteita, kun testitapausta ei saada suoritettua välittömästi loppuun asti. (Hohpe & Istvanick 2002, 7.)

Järjestelmäintegraatioiden perustuminen liiketoimintaprosesseihin aiheuttaa sen, että monet tiedonsiirrot integraatioissa ovat ajastettuja. Integraatioratkaisulle kerrotaan esimeriksi erilaisin ajastuksin, milloin pitää mitäkin tietoa siirtää. Lisäksi erilaiset hälytykset poikkeustilanteissa voivat myös olla ajastettuja. Asiakkaalle lähetetään esimerkiksi sähköposti, kun toimitus on myöhässä tietyn ajan tai integraatioratkaisu hälyttää, kun esimerkiksi kassalta ei ole tullut myyntiaineistoa toiminnanohjausjärjestelmään viimeisen vuorokauden aikana. Kuten asynkroninen tiedonsiirto, myös nämä ajastukset aiheuttavat haasteita testaukselle. Testitapauksen suorittamiseksi täytyy odottaa tietty aika eli toiminnon tai hälytyksen aktivoituminen. Toinen vaihtoehto on muuttaa ajastuksien aikavälejä pienemmäksi tai manipuloida järjestelmän kelloa, jotta nuo vaaditut aikarajat täyttyvät nopeammin. (Hohpe & Istvanick 2002, 7.) Testauksen kannalta tällaiset manipuloinnin ovat mielestäni vähän kyseenalaisia; kuvaako testitulos enää todellista tilannetta, jos se on saatu aikaiseksi manipuloimalla?

Järjestelmäintegraatioiden testaukseen haasteita aiheuttaa myös se, että jokaisessa yrityksessä liiketoimintaprosessit, yrityksen käytössä olevat järjestelmät ja tapa käyttää niitä sekä yrityskulttuuri eroavat hieman toisistaan, jolloin järjestelmäintegraatioratkaisuiden toteutuskin eroaa toisistaan. Erilaiset ympäristöt aiheuttavat sen, että testaustapa tarvitsee aina määritellä yrityskohtaisesti eli samoja testauksen periaatteita ei voi suoraan käyttää yrityksestä toiseen vaan yrityksen toimintatavat ja järjestelmäintegraatioratkaisun toteutustapa on otettava huomioon testausta suunniteltaessa.

4.3 Näkökulmia järjestelmäintegraatiotestaukseen

Kuten juuri edellisestä kappaleesta kävi ilmi, yrityksen järjestelmäkokonaisuudet ja niiden käyttötavat eroavat yleensä toisistaan niin paljon, että mitään yleistä testausmenetelmää järjestelmäintegraatioiden testaamiseksi, joka soveltuisi jokaisen yrityksen tarpeisiin, ei voi olla olemassa. Tämä fakta ei mitenkään eroa testauksesta yleensä sillä jo testauksen yleisiä periaatteita läpikäydessä kävi ilmi, että yleensäkään testausta tehdessä ei ole käytettävissä mitään universaaleja testitapoja tai -suunnitelmia.

Joitain periaatteita tai näkökulmia järjestelmäintegraatioiden testaamiseksi voi kuitenkin johtaa, kun yhdistää testauksen yleisiä periaatteita ja järjestelmäintegraatioiden yleisiä ominaisuuksia, joita on käyty läpi aikaisemmissa luvuissa. Seuraavissa luvuissa on käyty läpi muutamia näkökulmia ja lähtökohtia järjestelmäintegraatioiden testaamiseksi. Testaamisen suunnittelun lähtökohdaksi voidaan ottaa järjestelmäintegraatio-ratkaisun arkkitehtuuri, järjestelmäintegraatioiden yleiset haasteet tai muista testaustyypeistä poimittuja näkökulmia. Testauksen suunnittelun lähtökohdaksi voi ottaa myös liiketoiminnallisen näkökulman.

Mikään näistäkään näkökulmista ei sellaisenaan ratkaise kaikkien yrityksien järjestelmäintegraatioiden testaamistarvetta, mutta ehkä näitä seuraavissa luvuissa mainittuja näkökulmia voi käyttää hyödyksi ja yhdistellä niistä kuhunkin tapaukseen sopiva kokonaisratkaisu. Testaamismenetelmää – ja tapaa valittaessa tulee aina ottaa huomioon kyseisen yrityksen tai järjestelmäintegraatioratkaisun erityispiirteet

4.3.1 Järjestelmäintegraatioiden arkkitehtuuri

Järjestelmäintegraatio-testauksen lähtökohdaksi voi ottaa järjestelmäintegraatioiden arkkitehtuurin. Tämä tarkoittaa sitä, että otetaan testauksessa huomioon arkkitehtuurin eri kerrokset ja edetään testauksessa sen mukaan. Testausta ja testitapauksia suunnitellaan ja toteutetaan arkkitehtuurin kerros kerrallaan. Koska arkkitehtuurin alimmat kerrokset vastaavat yksinkertaisesta tiedonsiirrosta, kannattaa testaus aloittaa alimmista kerroksista ja edetä kerros kerrallaan kohti arkkitehtuurin ylimpiä kerroksia ja moni-

mutkaisempia testitapauksia, joissa otetaan huomioon muun muassa tiedonmuunnoksia ja tiedonsiirron kontrollointia.

Ensimmäiseksi testataan siis järjestelmien rajapintojen toimivuus eli se, että järjestelmäintegraatioratkaisu saa noudettua järjestelmästä A siirrettävän aineiston ja toisaalta siirrettyä aineiston sitä odottavalle järjestelmälle B. Rajapintoja testattaessa kannattaa testaus aloittaa yksinkertaisella tapauksella, jossa integraatio-ratkaisun siirrettäväksi annetaan ensin vain yksi aineisto, joka on kooltaan pieni ja muodoltaan oikeanlainen. Testauksen edetessä testitapauksien monimutkaisuutta kannattaa lisätä esimerkiksi kasvattamalla siirrettävien aineistojen kokoa ja lukumäärää sekä antamalla siirrettäväksi väärän nimisiä ja muotoisia aineistoja sekä keskeneräisiä / rikkinäisiä aineistoja. Useiden aineistojen siirron yhteydessä kannattaa ottaa huomioon myös aineistojen nimeäminen.

Järjestelmäintegraatioratkaisun arkkitehtuurin siirtokerroksen testaaminen voidaan linkittää osaksi rajapintojen testausta. Järjestelmältä A, järjestelmälle B siirrettävä aineisto voidaan rajapintojen testauksessa antaa siirtokerroksen kuljetettavaksi. Siirtokerroksen testauksessa tulee ottaa huomioon siirtokerroksen toimivuus ja siirtokerrosta tulee kuormittaa suurillakin tietomäärillä. Ulkoisissa integraatioissa tulee kiinnittää erityistä huomiota siirtokerroksen tietoturvan testaamiseen.

Integraatioratkaisun käsittely- ja muuntokerroksen tarkoituksena on tulkita siirrettävää aineistoa ja ohjata käyttäytymistään sen perusteella. Näiden kerroksien testaus kannattaa myös aloittaa yksinkertaisista testitapauksista ja edetä monimutkaisempiin testitapauksiin. Käsittely- ja muuntokerroksen testauksen voi siis aloittaa esimerkiksi pienellä oikean muotoisella aineistolla, joka ei vaadi muunnoksia. Testauksen edetessä kannattaa lisätä monimutkaisuutta testaukseen käyttämällä muun muassa erikokoisia aineistoja, jotka vaativat muunnoksia tai käyttää aineistoja, jotka vaativat järjestelmäintegraatioratkaisulta muutakin aineiston tulkintaa.

Kontrollointikerroksen avulla kontrolloidaan sitä, milloin tietoa noudetaan, siirretään ja käsitellään. Tiedon siirron voi aktivoida esimerkiksi ajastus tai käyttäjän antaman pyyntö. Testauksessa tulee ottaa huomioon käytössä olevat eri ajastukset ja testata niiden toimivuus eli siis testata, että aineisto noudetaan tai siirretään kohdehakemistoon silloin

kun ajastus käynnistyy. Testauksessa voi ottaa huomioon erilaisia tilanteita; mitä tapahtuu jos ajastusta muutetaan tai mitä jos ajastuksen käynnistymisen hetkellä hakemistossa ei ole yhtään noudettavaa tiedostoa?

Esitystapakerroksen testaaminen kannattaa nivoa yhteen näiden muiden kerroksien testaamisen kanssa ja seurata miten eri tilanteet näkyvät esitystapakerroksessa. Miten onnistunut siirto ja sen kesto näkyy entä virheeseen mennyt siirto tai ajastuksella käynnistynyt siirto?

Vaikka tässä luvussa onkin esitetty, että testausta kannattaisi tehdä arkkitehtuurin kerros kerrallaan, täytyy kuitenkin jossain vaiheessa tehdä myös niin sanottu end-to-end –testaus, jossa testataan järjestelmäintegraatio-ratkaisun toimivuus kokonaisen liiketoimintaprosessin kannalta. Se, että järjestelmän A ja integraatioratkaisun välinen yhteys toimii saumattomasti ei riitä, jos tieto ei integraatioratkaisusta välity eteenpäin järjestelmälle B, jossa tietoa tarvitaan.

4.3.2 Järjestelmäintegraatioiden haasteet

Yksi testauksen yleisistä periaatteista on se, että testausta tulee priorisoida eli testauksessa pitäisi keskittyä niihin kohtiin, jotka todennäköisemmin aiheuttavat riskejä ja ongelmia tuotteen käytössä. Järjestelmäintegraatioiden haasteita käytiin läpi aikaisemmassa luvussa ja yksi lähtökohta järjestelmäintegraatioiden testaamiseksi voikin olla se, että keskitytään näihin mahdollisiin ongelmakohtiin.

Järjestelmäintegraation tulee liiketoiminnallisen luonteensa vuoksi olla erittäin vikasietoinen. Jotta järjestelmäintegraation vikasietoisuutta saadaan testaamalla tutkittua, tulee suorittaa paljon niin sanottu dirty testing –testausta, jossa yritetään rikkoa järjestelmä. Testitapauksissa ei käytetä juurikaan oikeanmuotoisia aineistoja vaan nimenomaan virheellisiä, vääränmuotoisia ja keskeneräisiä aineistoja. Lisäksi tiedonsiirtoja katkaistaan kesken suoritusten esimerkiksi katkaisemalla virransyöttö tai sammuttamalla tarvittavia palveluita. Tiedonsiirtoa ja järjestelmäintegraation toimintaa yritetään siis kaikin mahdollisin keinoin hankaloittaa ja seurataan, huomaako järjestelmä ongelmat ja

osaako se toipua tilanteesta suunnitellusti eli esimerkiksi yrittämällä uudelleenlähetystä tai antamalla hälytyksen epäonnistuneesta tai liian kauan kestävästä tiedonsiirrosta.

Toinen järjestelmäintegraation ongelmakohdista oli tietoturva. Tietoturvan rooli korostuu erityisesti ulkoisissa integraatioissa, mutta myös sisäisissä integraatioissa, jolloin tietoturvaan kuuluu erityisesti siirrettävän tiedon eheys ja saatavuus. Näihin näkökulmiin tulee myös testauksessa kiinnittää huomiota. Tiedon eheyttä kannattaa erityisesti testata tietomuunnoksia silmälläpitäen. Testaamisessa kannattaa kiinnittää huomiota myös siihen, mitä tapahtuu jos aineisto esimerkiksi suuren kokonsa vuoksi siirtyy vain osittain. Tiedon saatavuuden osalta on ratkaisevaa saadaanko tieto integraatioratkaisun käsiteltäväksi silloin kun pitää. Mitä tapahtuu jos esimerkiksi verkkohakemisto, josta tietoa yritetään siirtää, ei ole saatavilla tai siirrettävä tiedosto on lukittunut tai jonkin muun sovelluksen käytössä?

Järjestelmäintegraatioiden ehkä suurimmaksi ongelmakohdaksi saattaa muodostua niin sanottu single point of failure. Tämän ongelman ehkäisemiseksi järjestelmäintegraatiolle kannattaa tehdä erilaisia suorituskykyyn ja kuormitukseen liittyviä testejä. Järjestelmäintegraatioratkaisun kautta liikkuva tietomäärä, siirrettävien tiedostojen määrä, koko ja siirtotiheys, kannattaa arvioida ja testata järjestelmäintegraatioratkaisun suorituskyky vähintään tämän tietomäärän kanssa. Tässä täytyy ottaa huomioon myös mahdolliset tulevat muutokset järjestelmäintegraation käytössä yrityksessä. Testauksessa tulee ottaa huomioon sekä käytettävän verkkoyhteyden että fyysisten laitteiden suorituskyky. Suorituskykytestauksen yhteydessä testiympäristöllä on suuri merkitys. Testaaminen tulisi suorittaa mahdollisimman hyvin tuotantoympäristöä simuloivassa ympäristössä. Jos testiympäristö ei vastaa tuotantoa, ei suorituskykytestaukselle saada oikeanlaista kuvaa järjestelmän toimivuudesta.

4.3.3 Muut testaustyypit

Järjestelmäintegraatioiden testaamiseen voi ottaa keinoja ja menetelmiä myös muista testaustyypeistä, joita on käyty läpi luvussa 2.5. Lähimpänä järjestelmäintegraatioiden testaamista ovat integraatiotestaaminen ja järjestelmätestaaminen. Integraatiotestaamisessa testataan yhdistettyjen ohjelmistokomponenttien testaamista kokonaisuutena ja

järjestelmätestauksessa testataan kokonaisen toimivan järjestelmän toimintaa monista eri näkökulmista.

Integraatiotestauksessa käytetään eri tekniikoita testaamiseen muun muassa bottom-up, top-down ja incremental. Järjestelmäintegraatioita voi lähteä myös testaamaan näistä näkökulmista. Bottom-up ja top-down tarkoittavat sitä, missä järjestyksessä ohjelman komponentteja testataan ohjelman rakenteeseen nähden. Järjestelmäintegraatioiden osalta ohjelman rakennetta voi ajatella järjestelmäintegraatioratkaisun arkkitehtuurilla ja tätä näkökulmaa testaukseen käsiteltiinkin jo luvussa 4.3.1. Incremental-tekniikka tarkoittaa sitä, että suoritettavia testikierroksia on useita ja testattavaa kokonaisuutta kasvatetaan koko ajan joka kierroksella. Järjestelmäintegraatioiden osalta tätä voisi soveltaa jälleen arkkitehtuurin ja integraatioratkaisun monimutkaisuuden näkökulmasta eli laitetaan ensin esimerkiksi toimimaan vain tiedonsiirrot ja rajapinnat ja testataan ne ja kun ne toimivat, otetaan käyttöön esimerkiksi tiedonmuunnokset.

Integraatiotestauksessa korostetaan myös end-to-end –testausta sekä stressi- ja kestävyys –testausta. Näitäkin näkökulmia käsiteltiin jo luvuissa 4.3.1 ja 4.3.2. Myös siis integraatiotestauksen näkökulmasta ajateltuna end-to-end –testaus eli kokonaisten toiminnallisuuksien testaaminen on tärkeää. Stressi- ja kestävyystestauksien osalta integraatiotestauksessa on mainittu pari asiaa, joita ei vielä tullut esille luvussa 4.3.2, jotka voisi ottaa järjestelmäintegraatioiden testauksessa huomioon. Stressi- ja kestävyystestauksessa tulee kiinnittää virhetilanteissa huomiota järjestelmän antamien virheilmoituksien ja hälytyksien lisäksi lokien toimintaan, jotta mahdollisia virhetilanteita voidaan tutkia myöhemminkin. Lisäksi stressi- ja kestävyystestauksessa tulee ottaa huomioon perinteisen prosessoritehon ja muistin riittävyyden sekä verkon toimivuuden lisäksi myös se, miten paljon yhtäaikaisia toimintoja ohjelmisto pystyy suorittamaan eli miten se toimii kiireessä; jääkö kenties joitain toimintoja tai viestejä kokonaan suorittamatta? Testauksessa tulisi tutkia myös sitä, kuinka pitkään ohjelmistoa voi käyttää, ilman että se pitää uudelleenkäynnistää.

Järjestelmätestauksessa ensinnäkin testataan, että järjestelmä vastaa tehtyjä määrityksiä. Asiakas ei välttämättä osaa, eikä hänen tarvitsekaan osata määritellä järjestelmäintegraation teknistä toteutusta, mutta ehkä järjestelmäintegraation muunnoksien ja tiedonsiirron kontrolloinnin osalta tätäkin näkökulmaa voisi soveltaa järjestelmäintegraatioi-

den testaamisessa. Eli testauksessa voisi varmistua siitä, että muunnokset toimivat niin kuin asiakas on halunnut ja että tietoa siirtyy silloin kun asiakas on tarkoittanut.

Järjestelmätestauksessa testataan myös tuotteen ympäristö ja sen toimivuus. Tässä otetaan huomioon muun muassa laitteet, käyttöjärjestelmä ja verkkoyhteydet sekä järjestelmän asentaminen ja päivittäminen. Järjestelmän tulee myös toimia päivityksien jälkeen normaalisti. Järjestelmäintegraatioratkaisun toimivuutta olisi hyvä myös testata näistä näkökulmista. Testauksessa voisi ottaa huomioon esimerkiksi sen, miten järjestelmäintegraatioratkaisu toimii esimerkiksi käyttöjärjestelmän tai itse järjestelmäintegraatioratkaisun päivittämisen jälkeen.

Järjestelmätestauksessa testataan myös tuotteen käytettävyys ja tarkoituksenmukaisuus. Järjestelmäintegraatioratkaisut ovat yleensä teknisesti niin monimutkaisia, että asiakkaat eivät itse tee muutoksia tietojen siirtoihin, mutta käytettävyyden testausta voi toki suorittaa ajatelleen järjestelmätoimittajan asiantuntijoita, jotka näitä muutoksia joutuvat tekemään. Asiakkaan näkökulmasta on kuitenkin tärkeää testata järjestelmäintegraatioratkaisun esitystapakerroksen tuottamaa näkymää, jonka kautta asiakas voi itse seurata järjestelmäintegraatiossa liikkuvia tiedonsiirtoja ja muun muassa niiden onnistumista ja mahdollisia virheitä.

Järjestelmätestauksessa otetaan kantaa myös tuotteen luotettavuuteen ja toimintaan kuormituksen alla. Kuormitukseen liittyvää testausta on jo käyty edellisessä luvussa läpi, mutta järjestelmätestauksessa on tähän vielä pari näkökulma lisää, joita ei ole edellisessä luvussa otettu huomioon, ja jota voisi soveltaa järjestelmäintegraatioiden testauksessa. Kuormitustestauksella ei vaan pyritä varmistamaan tuotteen toimivuutta, vaan tällä pyritään myös etsimään tuotteen mahdollisia virhetilanteita ja ongelmia tuotteen toiminnassa. Tärkeää on myös testata se, että tuote palautuu ongelmien jälkeen. Eli esimerkiksi laitevian tai tuotteen kaatumisen jälkeen tuote toimii taas normaalisti.

Tietoturvallisuuteen on myös otettu kantaa edellisessä luvussa, mutta vain siirrettävän tiedon osalta. Testauksen avulla pitäisi tietoturvallisuuden osalta varmistua muun muassa myös siitä, että vain sallitut henkilöt pääsevät tuotetta käyttämään ja tekemään muutoksia järjestelmäintegraatioratkaisun toimintaan ja tiedonsiirtoihin. Tässä vaiheessa voidaan testata esimerkiksi käyttäjän sisäänkirjautumiseen ja tunnistautumiseen liittyviä asioita.

4.3.4 Liiketoiminnan näkökulma

Järjestelmäintegraatioilla toteutetaan yrityksen liiketoimintaprosesseja. Teknisten lähtökohtien, kuten järjestelmäintegraation arkkitehtuurin sijaan, testauksen lähtökohdaksi voi ottaa liiketoiminnallisen näkökulman. Tällöin testausta lähdetään suorittamaan esimerkiksi yrityksen liiketoimintaprosessien näkökulmasta.

Testauksen lähtökohdaksi voi siis ottaa esimerkiksi yrityksen liiketoimintaprosessit, joita toteutetaan järjestelmäintegraatioratkaisulla. Ennen testausta yrityksen liiketoimintaprosessit ja niissä käytettävä tieto tulee tunnistaa ja määritellä. Kun nämä vaiheet on suoritettu, voidaan aloittaa testaaminen.

Testausta suoritettaessa ajetaan järjestelmäintegraatiossa läpi jokainen tunnistettu prosessi ja varmistetaan niiden läpimeno. Koska yksi liiketoimintaprosessi sisältää yleensä useita erilaisia tietoja, joutuu samaa prosessia toistamaan useaan kertaan jotta mahdollisimman monta eri skenaariota saadaan suoritettua. Esimerkiksi tilaus sisältää asiakkaan ja tilattujen tuotteiden tietoja. Asiakkaita voi olla erilaisia; tilauskiellossa oleva asiakas, laskutusasiakas ja käteisasiakas. Tilattavissa tuotteissakin voi olla esimerkiksi sellaisia tuotteita, joita ei tilaushetkellä ole varastossa. Tilausprosessi tulisi siis ajaa läpi huomioiden kaikki nämä eri tiedot.

5 CASE: JÄRJESTELMÄINTEGRAATIOIDEN TESTAAMINEN SOLTEQ OYJ:SSÄ

5.1 Taustaa ja tavoite

Case-yritykseni on Solteq Oyj, joka on vuonna 1982 perustettu ohjelmistoalan yritys, joka toimittaa it-ratkaisuja kaupan toimialalle. Solteq Oyj toimittaa asiakkailleen muun muassa ketjuuntuneen erikoistavarakaupan kokonaisratkaisua, johon kuuluu kassajärjestelmä ketjunohjauksineen, ERP-toiminnanohjausjärjestelmä ja näiden välisestä tiedonsiirrosta huolehtiva Solteq Hub. Solteq Hub on Solteq Oyj:n oma ratkaisu vaativiin järjestelmäintegraatioihin.

Tällaisen kokonaisratkaisun toimittaminen asiakkaalle on aina haasteellista ja jokaisen asiakkaan ympäristö pitää ottaa huomioon varsinkin järjestelmäintegraation kannalta. Unohtaa ei voi myöskään sitä, mikä rooli järjestelmäintegraatioissa on asiakkaan liiketoiminnan kannalta. Muun muassa näiden syiden vuoksi järjestelmäintegraatio-ratkaisun tulee olla laadukas ja yksi keino järjestelmäintegraation laadukkuuden varmistamiseksi on testaaminen.

Opinnäytetyön tavoitteena on selvittää liittymien testauksen nykytila Solteq Oyj:ssä ja sitä kautta identifioida ja löytää järjestelmäintegraatiotestauksen nykyiset ongelmat ja puutteet sekä myös hyväksi havaitut toimintamallit. Opinnäytetyön tavoitteena on myös löytää, kehittää ja kerätä ylös eri asiantuntijoiden ideoita järjestelmäintegraatioiden testauksen parantamiseksi ja yhdenmukaistamiseksi. Tarkoituksena olisi näitä tietoja hyödyntämällä saada aikaan yhtenäinen käytäntö ja toimintamalli liittymien testaukseksi ja dokumentoida se. Opinnäytetyön tavoitetta on vielä rajattu siten, että tässä vaiheessa järjestelmäintegraatioiden testauksen yhdenmukaistaminen koskee vain myymäläjärjestelmäympäristöjä, vaikka Solteq HUB -tuotetta käytetään myös muissa toimintaympäristöissä järjestelmäintegraatioiden ratkaisuna.

Yhtenäisellä käytännöllä ja toimintamallilla pyritään siihen, että järjestelmäintegraatioiden testaus olisi mahdollisimman laadukasta. Laadukkuuden ja sen varmistamisen taus-

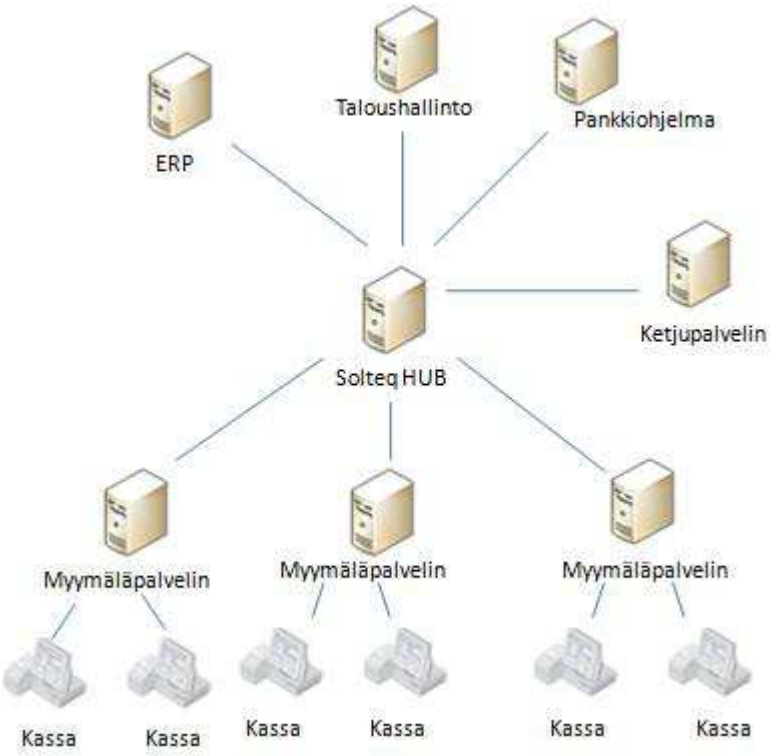
talla ovat asiakkaamme ja heidän tarpeensa. Asiakkaiden liiketoiminta ei salli käyttökatkoksia ja kustannustehokas toiminta vaatii muun muassa ajantasaisia myyntitietoja ketjusta; molempien näiden tavoitteiden saavuttamisessa on suuri rooli järjestelmäintegraation toimivuudella. Laadukkuus lisää myös asiakkaidemme ja potentiaalisten asiakkaiden luottamusta yrityksemme toimintaan ja sitä kautta tarjoavat Solteq:lle suurempia asiakkuuksia ja kasvattavat meidän markkinaosuuttamme erikoistavarakaupan järjestelmätoimittajana.

Lisäksi johdonmukainen ohjeistus ja laadunvarmistus takaavat sen, että järjestelmäkokonaisuuden toimitus onnistuu niin sanotusti ”kerralla oikein”. Tämä taas pitää huolen toiminnan kustannustehokkuudesta ja kannattavuudesta myös Solteqin näkökulmasta. Luonnollisesti ”kerralla oikein” -toimintaperiaate vaikuttaa myös asiakkaidemme tyytyväisyyteen.

Työn tavoitteena on myös hyödyttää järjestelmäintegraatioita toteuttavien asiantuntijoiden työskentelyä. Heidän päivittäinen työtehtävänsä helpottuu ohjeistuksien avulla.

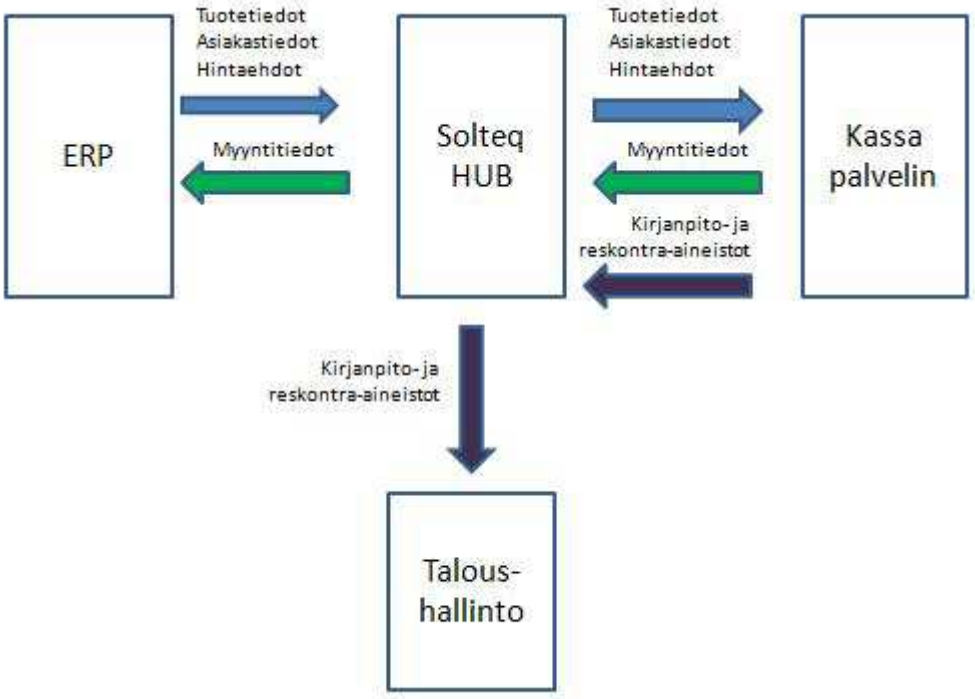
Solteqin Solteq HUB –tuotetta voidaan käyttää järjestelmäintegraatioissa monissa eri ympäristöissä, mutta kuten edellä on mainittu, niin tässä opinnäytetyössä keskitytään Solteq HUB –pohjaisten integraatioiden testaamiseen myymäläjärjestelmäympäristössä.

Myymäläjärjestelmäympäristö koostuu vähintään kassajärjestelmästä, toiminnanohjausjärjestelmästä ja näiden välisestä tiedonsiirrosta vastaavasta Solteq HUB:sta. Lisäksi myymäläjärjestelmäympäristöön voi kuulua muun muassa ketjunohjauspalvelin, erillinen taloushallinto-ohjelmisto tai pankkiohjelmisto. Tätä ympäristöä on kuvattu kuviossa 9.



KUVIO 9 Myymäläjärjestelmäympäristö

Myymäläjärjestelmäympäristössä liikkuu paljon tietoa, jonka liikuttamisesta Solteq HUB vastaa. Kuviossa 10 on kuvattu näitä tietovirtoja. Myymäläjärjestelmäympäristössä siirretään muun muassa kaikki perustiedot ERP toiminnanohjausjärjestelmästä kassajärjestelmään. Perustiedot käsittävät muun muassa tuote-, asiakas- ja hintatiedot. Kassajärjestelmästä taas siirretään myyntitietoja ERP toiminnanohjausjärjestelmään sekä kirjanpitoon taloushallinnon vaatimia tietoja myyntitapahtumista.



KUVIO 10 Myymäläjärjestelmäympäristön tietovirrat

Tietovirtojen siirtotiheys on aina asiakaskohtaisesti konfiguroitavissa. Yleensä osa tietovirroista on lähes tai kokonaan reaaliaikaisia; myyntiaineistoa ERP toiminnanohjausjärjestelmään voidaan siirtää kassajärjestelmästä esimerkiksi muutaman minuutin välein. Yleensä myös perustietojen siirrot on määritelty reaaliaikaisiksi eli esimerkiksi tuotteeseen tehty hintamuutos saadaan kassajärjestelmään nopeasti. Osa siirroista taas tapahtuu vain esimerkiksi kerran vuorokaudessa; tällainen tietovirta on muun muassa taloushallintoon vietävät kirjanpidon aineistot.

Solteqin myymäläjärjestelmäympäristö on erityisesti tarkoitettu ketjuuntuneen kaupan ratkaisuksi. Tämä tarkoittaa sitä, että asiakkaalla on useita myymälöitä, mitä on havainnollistettu kuviossa 9. Myymälöitä saattaa olla alle viisi tai jopa vain yksi mutta toisaalta niitä voi olla yli satakin. Muun muassa tämän vuoksi Solteq HUB:n on tuettava erilaisia siirtotapoja. Solteq HUB voi joutua monistamaan tietoa kun sama tuoteaineisto halutaan toimittaa asiakkaan jokaiseen myymälään. Toisaalta siirrettävälle tiedolle voi olla tällaisessakin ympäristössä tarkka osoite; esimerkiksi myymäläkohtaisien hintojen tapauksessa Solteq HUB:n on tunnistettava ERP–toiminnanohjausjärjestelmän muodostamasta aineistosta myymälä johon ja vain johon aineisto halutaan siirtää.

Isojen asiakkuuksien kohdalla Solteq HUB:n on siis pystyttävä käsittelemään suuriakin tietovirtoja. Toisaalta palvelun on oltava joustava ja ketterä pienempiä asiakkuuksia ajatellen.

5.2 Tutkimusstrategia eli keinot tavoitteen saavuttamiseksi

Opinnäytetyöni pohjaa laadulliseen tutkimukseen. Tutkimusmenetelminä käytin tapaus- ja toiminnallista tutkimusta sekä bechmarking-tutkimusta. Tutkimusmetodeinani oli havainnointi, haastattelut ja tekstianalyysit.

Ensimmäisenä minun tarvitsi selvittää liittymien testauksen nykyinen toimintamalli ja sen ongelmat, kehitettävät asiat ja toimivat ratkaisut. Näiden asioiden selvittämiseksi haastattelin yrityksemme järjestelmäintegraatio-asiantuntijoita (katso liite 1). Minulla itselläni oli myös jonkinlainen kuva järjestelmäintegraatioiden toimivuuden nykytilasta ja niissä havaituista ongelmista sen vuoksi, että olen itse työskennellyt Solteqin asiakastuessa kassajärjestelmien parissa jo viisi vuotta. Olin mukana myös testaamassa järjestelmäintegraatioita yhdessä käytännön projektissa, jonka avulla sain lisätietoja testauksen nykytilasta.

Kun nykytila oli saatu kartoitettua, aloin seuraavaksi selvittää, miten testausta voisi parantaa ja kehittää. Tekemieni haastatteluiden perusteella olin saanut kerättyä järjestelemäintegraatio-asiantuntijoiden mielipiteitä testauksen parantamiseksi. Haastattelin myös yrityksen muiden osastojen edustajia ja tiedustelin heiltä, miten järjestelmäintegraatioiden toteutus on heillä hoidettu. Tutustuin myös alan kirjallisuuteen järjestelmäintegraatioiden testaamisesta sekä testaamisen että järjestelmäintegraatioiden yleiseen teoriaan.

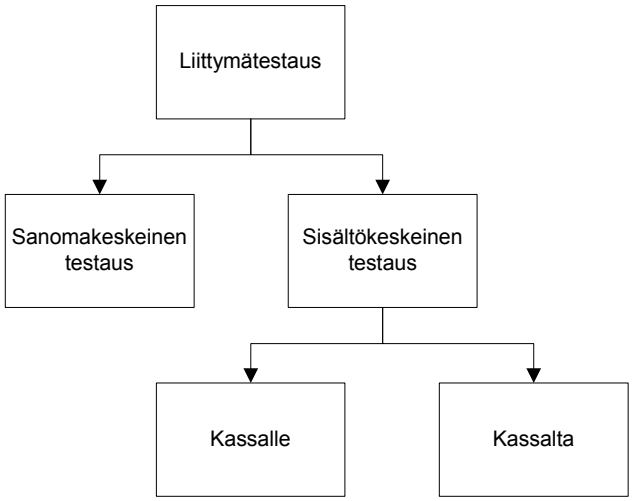
Olen itse työssäni testannut kassajärjestelmiä ja itselläkin oli pari ajatusta järjestelmäintegraatioiden testaamisen parantamiseksi ja yhdenmukaistamiseksi. Kuten aiemmin jo mainitsin, pääsin myös seuraamaan yhtä käytännön järjestelmäintegraatioprojektia, jossa pääsin kokeilemaan näitä ideoita testaamisen parantamiseksi käytännössä.

Haastatteluiden, alan teorian ja käytännön kokeilujen perusteella kirjoitin ohjeistuksen järjestelmäintegraatioiden testaamiseksi Solteqin myymäläjärjestelmäympäristössä.

5.3 Tutkimustulos eli ohjeistus järjestelmäintegraatioiden testaamiseksi Solteqin myymäläjärjestelmäympäristössä

Ohjeistuksessa, joka on liitetty tähän opinnäytetyöhön liitteeksi 2, on aluksi kerrottu yleisesti mikä on testauksen tavoite ja mitä testaaminen yleensä pitää sisällään; testaaminen ei siis ole pelkkää manuaalisten testitapauksien suorittamista. Ohjeistuksessa on myös kerrottu muutamia tärkeitä testauksen yleisiä periaatteita, kuten se, että testitapauksen ja testien tulokset tulee dokumentoida.

Järjestelmäintegraatioiden testaaminen jaettiin kahteen eri näkökulmaan. Jo järjestelmäintegraatio-asiantuntijoiden haastatteluissa nämä kaksi eri näkökulmaa kävivät ilmi ja järjestelmäintegraatioiden testaamisen teoriastakin löytyy perusteluita näille valinnoille. Solteqin myymäläjärjestelmäympäristöjen järjestelmäintegraatioiden testaus jaettiin siis sanoma- ja sisältökeskeiseen testaukseen. Sisältökeskeinen testaus on vielä edelleen jaettu kahteen eri vaiheeseen: kassalta ja kassalle. Tätä jakoa on havainnollistettu kuviossa 11.



KUVIO 11 Järjestelmäintegraatioiden testaaminen Solteqin myymäläjärjestelmäympäristössä

Sanomakeskeisen testauksen avulla varmistetaan, että tieto siirtyy paikasta toiseen eli esimerkiksi ERP -toiminnanohjausjärjestelmästä kassajärjestelmään. Sisältökeskeisen testauksen avulla taas varmistetaan siirrettävän tiedon oikeellisuus. Sisältökeskeinen testaus on vielä jaettu kahteen eri osa-alueeseen sen mukaan mihin suuntaan tietovirrat

kulkevat. Tietovirrat kulkevat joko kassajärjestelmään tai kassajärjestelmästä pois. Tietovirtojen suunta määrittelee aineiston luonteen ja tyypin ja sen mukaan myös sen, miten aineistoa oli paras testata.

Kuten aikaisemmin jo mainitsin, niin tämä jako tehtiin haastatteluissa ilmi käyneiden näkökulmien ja järjestelmäintegraatioiden testaamisen teorian perusteella. Järjestelmäintegraatioiden testauksen teoriassa tätä jakoa tukee muun muassa järjestelmäintegraatioiden testauksen haasteet ja erityisesti tietoturvallisuus. Tietoturvallisuushan oli jaettu tiedon eheyteen ja saatavuuteen. Sanomakeskeisen testauksen taustalla on tiedon saavutettavuus ja sisältökeskeisen testauksen taustalla taas tiedon eheys. Myös testauksen haasteista vikasietoisuutta ja single point of failure –näkökulmaa on otettu huomioon sanomakeskeisessä testauksessa. Sanomakeskeinen testaus perustuu myös paljolti järjestelmäintegraatioratkaisuiden arkkitehtuuriin. Sisältökeskeinen testaus taas on ottanut näkökulmia liiketoiminnallisesta testauksesta.

5.3.1 Sanomakeskeisen testaamisen menetelmät

Sanomakeskeisen testauksen tarkoituksena on siis varmistaa tiedonsiirron onnistuminen paikasta toiseen. Tiedonsiirroissa ja niitä testattaessa on otettava huomioon muun muassa siirrettävän sanoman muoto, koko ja siirtotiheys eli kuinka usein aineistoa siirretään. Lisäksi on otettava huomioon myös muun muassa se, kuinka siirtoa pitää monistaa, kun esimerkiksi siirretään samaa aineistoa useaan myymälään tai miten siirtojen onnistumisesta tai epäonnistumisesta annetaan hälytyksiä. Huomioon on otettava myös asiakkaan ympäristö ja erilaisen ongelmatilanteet siirroissa.

Laadin sanomakeskeistä testausta varten listauksen huomioon otettavista asioista. Listauksen on tarkoitus toimia tukena testauksen yhteydessä ja auttaa testaajaa muistamaan tärkeät asiat, jotka on otettava huomioon testauksessa. Listauksen ei ole tarkoitus toimia tarkkana, kaikenkattavana testitapausluettelona, vaan pohjana testaukselle ja luoda siten yhteneväistä käytäntöä testauksen tueksi.

Listausta alettiin rakentaa siten, että se ei olisi käyttökelpoinen pelkästään yhdentyyppisille asiakkaille ja että sitä voitaisiin hyödyntää jatkossa myös muissa ympäristöissä

kuin myymäläjärjestelmäympäristöissä. Tämän vuoksi sen lähtökohdaksi otettiin järjestelmäintegraatioiden ominaisuudet, joita ovat muun muassa siirtojen monistaminen, siirtojen muunnokset, hälytykset ja siirrettävien tiedostojen koko ja volyymi. Järjestelmäintegraatioiden testaamisen teoriassa yhtenä näkökulmana oli järjestelmäintegraatioiden testaaminen arkkitehtuurin kautta ja tämä valitsemamme näkökulmahan liittyy läheisesti arkkitehtuuriin. Järjestelmäintegraatioiden arkkitehtuurissahan on nimenomaan kyse siitä, että arkkitehtuurin eri kerrokset edustavat järjestelmäintegraation tiettyjä ominaisuuksia.

Tässä tehdyssä listauksessa on otettu huomioon myös järjestelmäintegraatioiden testauksen teoriasta näkemyksiä luvuista järjestelmäintegraatioiden haasteet ja muut testaustyypit. Näissä kahdessa luvussa korostettiin järjestelmäintegraatioiden vikasietoisuutta ja volyymi- ja rasitustestauksen tärkeyttä, joita olen yrittänyt siis ottaa huomioon myös laaditussa listauksessa.

Ohjeistukseen on laadittu myös prosessikaavio siitä, miten sanomakeskeinen testaus pitäisi suorittaa. Prosessissa on neljä vaihetta:

- testauksen suunnittelu
- testauksen valmistelut
- testaaminen
- testauksen lopettaminen

Testauksen suunnittelu-vaiheessa tutustutaan määrittelyihin, resursoidaan testaus, määritellään testaustehtävät ja tavoitteet. Testauksen valmistelut vaiheessa laaditaan testitapaukset, testiaineistot ja pystytetään testiympäristö. Testaaminen-vaiheessa suoritetaan varsinainen testaus ja testauksen lopettaminen –vaiheessa kirjoitetaan loppuraportti ja tallennetaan testitapaukset ja testiaineisto tulevaa käyttöä varten sekä päätetään pystytetyn testiympäristön jatkosta.

5.3.2 Sisältökeskeisen testaaminen menetelmät

Sisältökeskeisen testaamisen tarkoituksena on siis varmistaa tiedonsiirtojen eheys. Myymäläjärjestelmäympäristössä siirretään tietoa sekä kassajärjestelmään että kassajär-

jestelmästä. Yleensä tieto siirretään xml-muodossa tai joitain kirjanpidon aineistoja voidaan siirtää myös tekstimuodossa.

Haasteena sisältökeskeisessä testauksessa on se, että aineiston muodostuksessa ja aineiston sisään luvussa toiseen järjestelmään osataan ottaa kaikki variaatiot huomioon siten, että aineiston sisältö ja tarkoitus eivät vääristy toisessa järjestelmässä. Tässä testauksessa on tärkeää ottaa huomioon kaikki mahdolliset testitapaukset ja testauksen tulisi olla järjestelmällistä ja hallittua. Hallittavuutta ja järjestelmällisyyttä lisätäkseni laadin sisältökeskeiseen testaukseen prosessikuvaukset siitä, miten testauksen tulisi edetä.

Sisältökeskeisen testauksen perusteena toimii järjestelmäintegraatioiden testaamisen teoriassa esitetty liiketoiminnallinen näkökulma. Sekä kassalta että kassalle tapahtuvassa testauksessa tunnistetaan siis erilaisia liiketoiminnallisia prosesseja, joiden kautta testaus tapahtuu käytännössä.

Kassalle päin siirretään perustietoja toiminnanohjausjärjestelmästä. Näitä perustietoja ovat muun muassa tuotteet, asiakkaat ja hintaehdot. Kun testataan kassalle päin siirrettäviä tietoja, aloitetaan testaaminen siitä, että tunnistetaan kaikki kassalle siirtyvät perustiedot. Kun nämä on tunnistettu, jatketaan vielä niiden käsittelyä yksityiskohtaisemmalle tasolle eli tunnistetaan jokaisesta perustietojen siirrosta siirtojen eri tyypit ja parametrit. Tyypillä tarkoitan esimerkiksi erityyppisiä hintaehtoja kuten kampanja tai asiakaskohtainen alennus. Parametreilla taas tarkoitan alennustyypin siirrettäviä tietoja esimerkiksi alennuksen voimassaoloa, alennusmäärää ja asiakasrajausta. Kun nämä kaikki tyypit ja parametrit on tunnistettu, saadaan aikaiseksi kattava testitapausluettelo. Testitapausluetteloon tulee vielä lisätä jokaisen tapauksen eri variaatiot. Näillä tarkoitan perustiedon siirtoa kassalle uutena, sen muuttamista ja sen poistamista. Kun kaikki nämä testitapauksen on käyty läpi, on testaaminen suoritettu.

Kassalta poispäin siirretään myynti- ja kirjanpitoaineistoja. Näiden aineistojen oikeellisuuden testaaminen vaatii kassamyynnin tekemistä. Ilman kassamyyntiä ei näitä aineistojakaan voi muodostaa eikä testata niiden oikeellisuutta. Testaaminen alkaakin siitä, että kassalle pitää tehdä testimyyntiä sekä yhteenveto testimyynneistä. Yleensä myymäläjärjestelmäympäristön toimituksen yhteydessä toimitetaan asiakkaalle myös kassajärjestelmä, jolloin sen testauksen yhteydessä tehtyjä testimyyntejä voidaan käyttää järjes-

telmäintegraation testaamisessa hyödyksi. Jos tilanne ei kuitenkaan näin ole, olen ohjeistukseen liittänyt listauksen tehtävien kuittien minimivaatimuksista. Kun testimyynnit ja yhteenveto myynneistä on kassalla tehty, muodostetaan kassalla myynti- ja kirjanpitoaineistot ja luetaan ne ERP:iin tai kirjanpitoon sisään. Tämän jälkeen tarkistetaan sekä ERP:stä ja kirjanpidosta, että niiden luvut täsmäävät kassalla tehtyyn yhteenvetoon.

6 JOHTOPÄÄTÖKSET

Tämän opinnäytetyön tavoitteena oli tutustua testauksen ja järjestelmäintegraatioiden teoriaan sekä erityisesti järjestelmäintegraatioiden testauksen teoriaan. Teoriaosuuden lisäksi opinnäytetyö sisälsi case-tutkimuksen, jonka tavoitteena oli selvittää järjestelmäintegraatioiden testauksen nykytila toimeksiantajayrityksessäni Solteq Oyj:ssä myymäläjärjestelmien osalta. Nykytilan selvityksen kautta oli tarkoitus identifioida järjestelmäintegraatiotestauksen nykyiset ongelmat ja puutteet sekä myös hyväksi havaitut toimintamallit. Nykytilan selvityksen jälkeen tavoitteena oli myös löytää, kehittää ja kerätä ylös eri asiantuntijoiden ideoita järjestelmäintegraatioiden testauksen parantamiseksi ja yhdenmukaistamiseksi. Näiden palautteiden ja omien kokemuksieni perusteella oli tarkoitus saada aikaan yhtenäinen käytäntö ja toimintamalli liittymien testaukseksi ja dokumentoida se.

Teoriaosuudessa onnistuin mielestäni hyvin ja kattavasti esittämään testauksen ja järjestelmäintegraatioiden osalta tärkeimmät osa-alueet tämän työn kannalta. Näihin osa-alueisiin löytyi kattavasti uudehkoja lähdeteoksia.

Järjestelmäintegraatioiden testaamisen osalta teoriaa ei kuitenkaan ollut juurikaan saatavilla, joten jouduin tässä opinnäytetyössä pohjaamaan siitä kertovan luvun testauksen ja järjestelmäintegraatioiden yleiseen teoriaan. Tein tämän nivomalla yhteen testauksen yleisiä periaatteita järjestelmäintegraatioiden yleiseen teoriaan ja kirjoitin näiden perusteella lähes kokonaan opinnäytetyön neljännen luvun. Tämän opinnäytetyön neljäs luku eli järjestelmäintegraatioiden testaus –luku on siis lähes kokonaan opinnäytetyön kirjoittajan omaa tuotosta, joten luvun sisältämä informaatio ei ole yleisesti hyväksyttyä ja perusteltua alan piireissä. Tästä huolimatta luvun sisältämää informaatiota ei tule jättää täysin omaan arvoonsa, perustin sen kuitenkin testauksen ja järjestelmäintegraatioiden yleisiin teorioihin ja onnistuin tässä mielestäni melko hyvin.

Tämän opinnäytetyön tavoitteena oli siis myös selvittää järjestelmäintegraatioiden testauksen nykytila Solteq Oyj:ssä myymäläjärjestelmien osalta ja luoda Solteq Oyj:lle

yhtenäinen käytäntö ja toimintamalli niiden järjestelmäintegraatioiden testaamiseksi ja dokumentoida se.

Onnistuin tässäkin tavoitteessa mielestäni ihan hyvin. Sain selvitettyä järjestelmäintegraatioiden nykytilan haastattelemalla järjestelmäintegraatio-asiantuntijoita Solteq Oyj:ssä ja seuraamalla yhtä käytännön projektiakin. Nykytilan kartoituksen kautta lähdin yhdessä järjestelmäintegraatio-asiantuntijoiden kanssa miettimään parannusehdotuksia testaamiseen ja saatujen kommenttien perusteella sekä omien havaintojeni valossa laadin ohjeistuksen järjestelmäintegraatioiden testaamiseksi.

Ohjeistus tuli laatia melko yleisellä tasolla sillä kuten tämän työn teoriaosuudesta käy ilmi järjestelmäintegraatioiden toteutukset vaihtelevat niin paljon asiakkaittain, että yleistä universaalia toimintamallia testaukseen, joka sopisi jokaiselle asiakkaalle, on mahdoton laatia. Huomasinkin, että aluksi ohjeistus aiheutti hämmennystä Solteq Oyj:n järjestelmäasiantuntijoiden parissa; he olivat olettaneet, että ohjeistus tarjoaisi heille valmiimpia työkaluja ja ratkaisuja testauksen suorittamiseksi.

Jatkoa ajatellen laatimani ohjeistus pitäisi seuraavaksi ottaa käyttöön ja testata sen toimivuus ja hyödynnettävyys käytännössä. Käytännön kokemuksien jälkeen ohjeistusta tulee korjata ja päivittää nyt ja myös jatkossa.

Koska nyt tämän opinnäytetyön tekemisen ja kirjoittamisen hetkellä järjestelmäintegraatioiden testaamisen osalta ei ollut teoriaa juurikaan saatavilla, pitäisi jatkossakin seurata alan kirjallisuutta. Jossain vaiheessa varmasti tästäkin aiheesta aletaan kirjoittaa enemmän ja silloin yleisen teorian valossa laadittuun ohjeistukseen voidaan saada uusia näkökulmia.

LÄHTEET

Hohpe, Gregor & Istvanick, Wendy 2002. Test-Driven Development in Enterprise Integrations Projects. [online] [viitattu 5.9.2011]
http://eaipatterns.com/docs/TestDrivenEAI.pdf

ISTQB Foundation level Syllabus 2010 [online] [viitattu 18.11.2010]
http://istqb.dedicated.adaptavist.com/download/attachments/2326555/Foundation+Level+Syllabus+(2010).pdf

ISTQB Standard glossary of terms used in software testing 2010.[Online] [viitattu 17.11.2010]
http://istqb.dedicated.adaptavist.com/download/attachments/2326555/ISTQB+Glossary+of+Testing+Terms+2+1.pdf

Jokinen, Juha-Veli 2011. Järjestelmässä on ollut ongelmia ennenkin, sanoo ex-ylilääkäri. Aamulehti 21.9.2011, A04.

Kaner, Cem 2006. Publications: Exploratory Testing [Online] [viitattu 17.11.2010].
http://kaner.com/?page_id=7

Maes, Dominic & Mertens, Steven 2008. 10 tips for succesfull testing. Testing Experience The Magazine for professional testers, 52-53.
http://www.testingexperience.com/testingexperience03_08.pdf

Naik, Kshirasagar & Tripathy, Priyadarshi 2008. Software testing and quality assurance Theory and practice. Hoboken, New Jersey: John Wiley & Sons, Inc.

Tamres, Louise 2002. Introduction Software Testing. Great Britain: Pearson Education limited.

Schaeref, Hans 2008(1). What a tester should know, at any time, event after the midnight. Testing Experience The Magazine for professional testers, 41-46. [online] [viitattu 16.11.2010]. http://www.testingexperience.com/testingexperience01_08.pdf

Siltanen, Juha 2004. Tietoarkkitehtuuriin perustuva sovellusintegraatiometodi – tapaus Liikelaitos. Tampereen Yliopisto: Pro Gradu –tutkielma. [online] [viitattu 7.9.2011].
http://www.cs.uta.fi/research/theses/masters/Siltanen_Juha.pdf

Tähtinen, Sami 2005. Järjestelmäintegraatio. Jyväskylä: Gummerus kirjapaino Oy.

VR maksaa hvyitystä puolitoistakertaisena. 2011. Aamulehti 28.9.2011, A07.

Zimmerer, Peter 2010(11). The Value of Testing in 5 dimensions. Testing Experience The Magazine for professional testers, 7-9. [online] [viitattu 17.11.2010].
http://www.testingexperience.com/testingexperience11_09_10.pdf

LIITTEET

Liite 1 Haastattelun teemarunko

Haastattelun tavoite: selvittää järjestelmäintegraatioiden testauksen nykyinen toimintamalli Solteq Oyj:ssä ja sen ongelmat, kehitettävät asiat ja toimivat ratkaisut. Tarkoituksena on myös kerätä ehdotuksia järjestelmäintegraatiotestauksen parantamiseksi.

Ennen haastattelun aloittamista:
- esittele itsesi (haastattelija)
- kerro tehtävästä opinnäytetyöstä

Haastattelurunko:

1. Järjestelmäintegraatiotestauksen nykyinen toimintamalli
- miten itse olet testannut järjestelmäintegraatioita?
- mitä toimintatapoja/työkaluja olet käyttänyt?
- miten testaus on edennyt?
- testataanko mielestäsi järjestelmäintegraatioita tarpeeksi?

2. Mitä hyvää nykyisessä toimintamallissa mielestäsi on?
- mikä toimii, mitä tehdään järkevästi?
- mistä toimintatavoista ei kannata luopua?

3. Mitä kehitettävää nykyisessä toimintamallissa mielestäni on?
- mikä on vaikeaa testauksessa?
- mitä puutteita testauksessa mielestäsi on?
- mitä asioita tai kokonaisuuksia ei oteta lainkaan testauksessa huomioon?

4. Ehdotuksesi liittymien testauksen parantamiseksi
- mitä pitäisi tehdä toisin?
- miten nykyistä toimintamallia voisi parantaa?

Liite 2 Ohjeistus järjestelmäintegraatioiden testaamiseksi Solteqin myymäläjärjestelmäympäristössä

LIITTYMIEN TESTAAMINEN

Tässä dokumentissa esitellään ehdotus liittymien testaamiseksi Solteqin myymäläjärjestelmäympäristössä. Myymäläjärjestelmäympäristöllä tässä tarkoitetaan kokonaisuutta, johon kuuluu ERP toiminnanohjausjärjestelmä, kassajärjestelmä ja tiedonsiirroista vastaava Solteq Hub.

Tarkemmin tässä dokumentissa esitetään yleiset periaatteet testauksen suorittamiseksi sekä testaamismalli varsinaisten liittymien testaamiseksi. Liittymien testaamismalli on tarkoitettu erityisesti uuden asiakasympäristön käyttöönottotestaamiseen. Tässä ei ole otettu huomioon niin sanottu on-line kyselyitä.

Tässä dokumentissa esitetty testaamismalli ei sisällä tarkkoja testitapausluetteloita, vaan testimallin on tarkoitus toimia ohjenuorana ja muistilistana testauksen yhteydessä. Kuten seuraavasta Testaaminen yleisesti –luvusta käy ilmi, testauksessa käytettävä testaamismalli on aina valittava tuote- ja asiakaskohtaisesti, joten tämä dokumentti ei voi tarjota yleispätevää testimallia kaikkeen testaukseen.

Tässä dokumentissa esitettyä testimallia saa ja pitää päivittää käytännön kokemuksien myötä.

TESTAAMINEN YLEISESTI

Testaamisella pyritään tuotteen laadun parantamiseen ja mittaamiseen muun muassa tuotteen virheiden löytämisen kautta ja tuotteen käytön riskien vähentämisellä. Testauksessa otetaan kantaa myös siihen, soveltuuko tuote käyttötarkoitukseensa ja vastaako tuote tehtyjä määrityksiä.

Testaaminen ei ole pelkästään testitapauksien manuaalista suorittamista vaan testauksen suorittamiseen kuuluu muitakin vaiheita. Testausta tulee suunnitella etukäteen ja ennen varsinaisen testauksen aloittamista pitää muun muassa laatia testitapaukset, testiaineisto ja pystyttää testiympäristö. Testauksen jälkeen, käydään läpi testitulokset ja ryhdytään sen mukaan toimenpiteisiin.

Testaamisessa tulee ottaa huomioon yleisiä testauksien periaatteita, joita ovat muun muassa:

- testaus tulee aloittaa ajoissa. Tämä ei tarkoita vain testitapauksien manuaalisen suorittamisen aloittamista ajoissa vaan testaus voidaan aloittaa jo määrittelyvaiheessa katselmoimalla tehtyjä dokumentteja
- universaalia testausmenetelmää ei ole vaan, testimenetelmä tulee valita asiakasympäristön ja testattavan sovelluksen mukaan

jatkuu

- kaikkien toimintojen testaaminen ei ole mahdollista, vaan testausta tulee priorisoida ja testaus tulee myös osata lopettaa ajoissa
- testaamisessa testiympäristöllä ja käytettävällä testiaineistolla on suuri merkitys. Testiympäristön ja testiaineiston pitää mahdollisuuksien rajoissa simuloida todellista tuotantoympäristöä
- testitapaukset tulee laatia etukäteen ennen testausta ja niistä pitää käydä ilmi syöte, joka tuotteelle annetaan sekä odotettu tulos. Testitapauksen suorittamisen jälkeen testitapauksen yhteyteen merkitään saatu tulos. Testitapaukset voidaan esittää esimerkiksi taulukko-muodossa;

	Initial state					
Test case id	**username**	**password**	**cursor location**	**Input**	**Expected results**	**Actual result**
tc-401	(empty)	(empty)	Place cursor in username field	a	'a' appears in username field	
tc-402	m	(empty)	place cursor after 'm' in username field	b	'mb' appears in username filed	
tc-403	jasmine	(empty)	place cursof after 'j' in username field	a	'jasmine' appears in username field	
tc-404	joelouis	(empty)	place cursor at end of username field	2	ignore keystoke (maximum number of characters exceeded): no change	

- testitapauksia laadittaessa on tärkeää tuntea ja ottaa huomioon testauksen eri kategoriat: dataa ei anneta, suorita kahteen kertaan, oikeanmuotoinen data, vääränmuotoinen data, keskeytys, virtakatko, järjestelmän rasittaminen ja suorituskyky.
- testitapaukset ja niiden tulokset tulee kirjata ylös
- laadittavien testitapauksien pohjana toimii yleensä määrittelydokumentaatio
- jo kerran laadittuja testitapausluetteloita kannattaa käyttää uudelleen myöhemmissä testauksissa
- testauksen kannalta on tehokkaampaa, jos testitapaukset laativat eri henkilö kuin se, joka suorittaa varsinaiset testauksen

LIITTYMIEN TESTAAMINEN

Liittymätestaus on jaettu kahteen osa-alueeseen

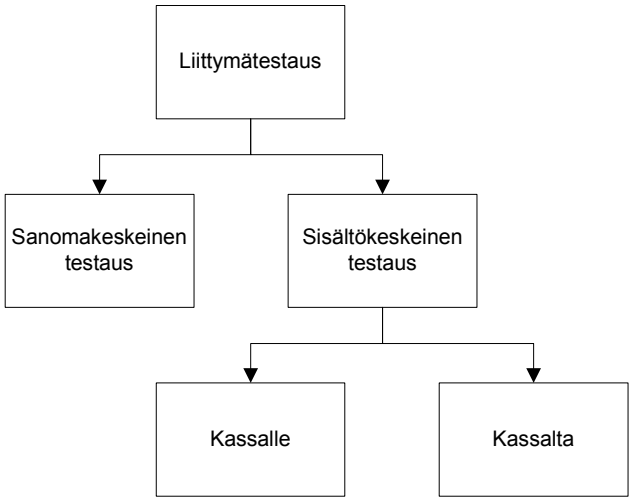
- sanomakeskeinen testaus
- sisältökeskeinen testaus

Sanomakeskeisen testauksen tarkoituksena on varmistaa tiedonsiirto kohteesta a kohteeseen b.

Sisältökeskeisen testauksen tarkoituksena on varmistaa siirrettävän tiedon oikeellisuus. Sisältökeskeinen testaus on vielä jaettu kahteen osa-alueeseen;

- tiedonsiirto ERP:stä kassalle
- tiedonsiirto kassalta ERP:iin

jatkuu



SANOMAKESKEINEN TESTAUS

Sanomakeskeisen testauksen tarkoituksena on varmistaa, että data saadaan siirrettyä sille määriteltyyn kohteeseen.
Tässä on koottu yhteen testauksessa huomioon otettavia asioita, joiden tarkoitus on tukea testausta ja auttaa testaajaa muistamaan tärkeät asiat, jotka on otettava testauksessa huomioon.

Testaus tulisi suorittaa asiakkaan testiympäristössä, joka mahdollisuuksien mukaan simuloi asiakkaan tuotantoympäristöä.

Suoritettavissa testitapauksissa ota huomioon

- sanomamuoto
  - testaa eri tiedostotyypeillä mm. txt ja xml
- merkistöt
  - aineisto sisältää erikoismerkkejä, ääkkösiä
- reititys / monistus
  - aineiston lähetys yhteen kohteeseen
  - aineiston lähetys useaan kohteeseen
- kuittaukset vastaanottajalta
  - vertaa kuittauksien määrä lähetyksien määrään
  - tarkista kuittauksen sisältö (viittaa oikeaan lähetykseen)
- volyymi
  - aineistojen lukumäärä
  - aineistojen koko
  - lähetyksen tiheys

jatkuu

Testaa perustapaukset, ota huomioon edellä mainitut tekijät yhdistelemällä niistä erilaisia testitapauksia.

- Esimerkiksi testaa, miten onnistuu siirto kun kyseessä on kooltaan iso tekstitiedosto, joka sisältää erikoismerkkejä ja jota ollaan siirtämässä yhteen kohteeseen; tarkista myös mahdollisen kuittauksen siirtyminen Hub:lle.

Testattavia erikois- ja ongelmatapauksia

Ota huomioon, miten seuraavat tapaukset vaikuttavat siirtoihin; onnistuuko siirto ongelmasta huolimatta, miten siirto jatkuu ongelman jälkeen ja miten Hub informoi käyttäjää siirron ongelmista.

- Yhteysongelmat
  - verkkoyhteys katkeaa siirtokohteen ja Hub:n välillä siirron eri vaiheissa
  - tietokantayhteys katkeaa siirron eri vaiheissa Hub:lla tai kohteessa
  - kohde ei vastaa (siirron ainoa kohde / yksi siirron kohteista)
  - virtakatko siirron eri vaiheissa

  Huomioi myös yhteysongelmien kesto, miten se vaikuttaa tilanteeseen.

- Siirrettävän datan eheys
  - siirrettävä xml-tiedosto ei ole eheä
  - siirrettävä txt-tiedosto katkeaa kesken
  - siirrettävä tiedosto on tyhjä
  - saman tiedoston siirtäminen useaan kertaan

- Käyttöoikeusongelmat
  - kansioon/hakemistoon, josta siirrettävä data pitäisi noutaa, ei ole käyttöoikeuksia
  - kansioon/hakemistoon, johon data pitäisi siirtää, ei ole käyttöoikeuksia
  - käyttöoikeuksiin määritellyt käyttäjätunnukset ovat väärin
  - siirrettävä tiedosto on lukittu

- Volyymi / rasitustestaus

  Arvioidaan asiakasympäristön todellinen siirtojen määrä ja siirrettävien tiedostojen koko. Otetaan huomioon siirtojen ajoittuminen päivän aikana ja mahdolliset raja-arvot eli pienin ja suurin mahdollinen siirtomäärä (pyhäpäivä vs. tankkaus). Pyritään simuloimaan mahdollisuuksien mukaan ko. tilannetta ja varmistetaan volyymin sietokyky siirroissa.

jatkuu

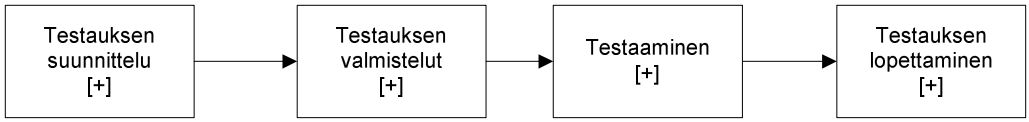
Hälytyksien toiminta

- Testaa kaikki käytettävät hälytystyypit ja niiden toiminta. Esimerkiksi jos hälytyksiin on määritelty jollekin siirrolle tietty aikaväli tapahtumille, testaa sen toiminta ylittämällä tuo aikaväli ja varmista, että hälytys toimii.
- Tarkista hälytyksien parametrit; kenelle, mitä.
- Huomioi hälytyksien toiminnassa poikkeukset;
    - aukioloajat, viikonloput, pyhäpäivät

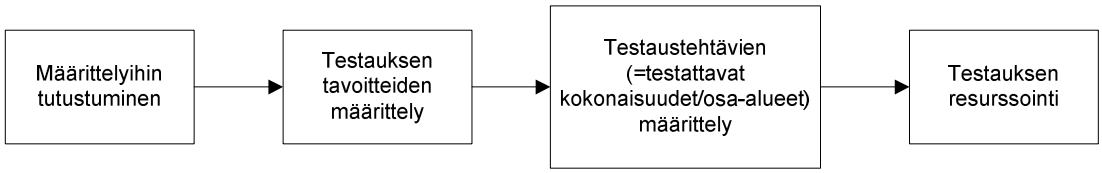
Muuta huomioitavaa

- varmistukset ja niiden ajankohdat
- muut mahdolliset ajastukset ja niiden ajankohdat
- palvelimien uudelleenkäynnistykset
- levytilan riittävyys

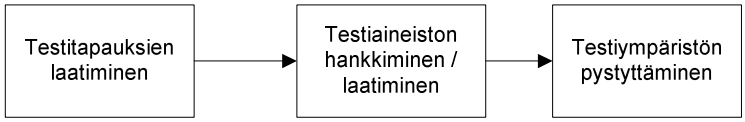
Ohessa prosessikaavio, jossa on kuvattu sanomakeskeisen testauksen suorittaminen käytännössä.



Testauksen suunnittelu –vaihe:

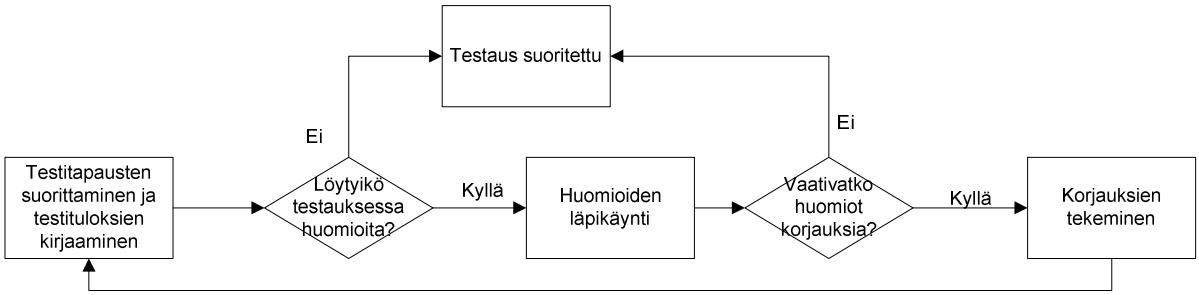


Testauksen valmistelu –vaihe:

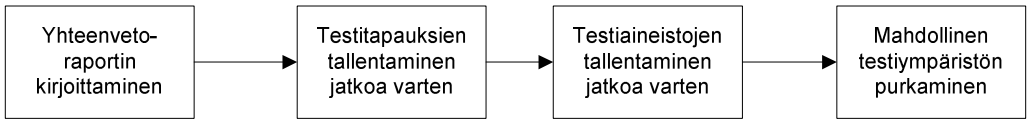


jatkuu

Testaaminen –vaihe:



Testauksen lopettaminen –vaihe:



SISÄLTÖKESKEINEN TESTAUS

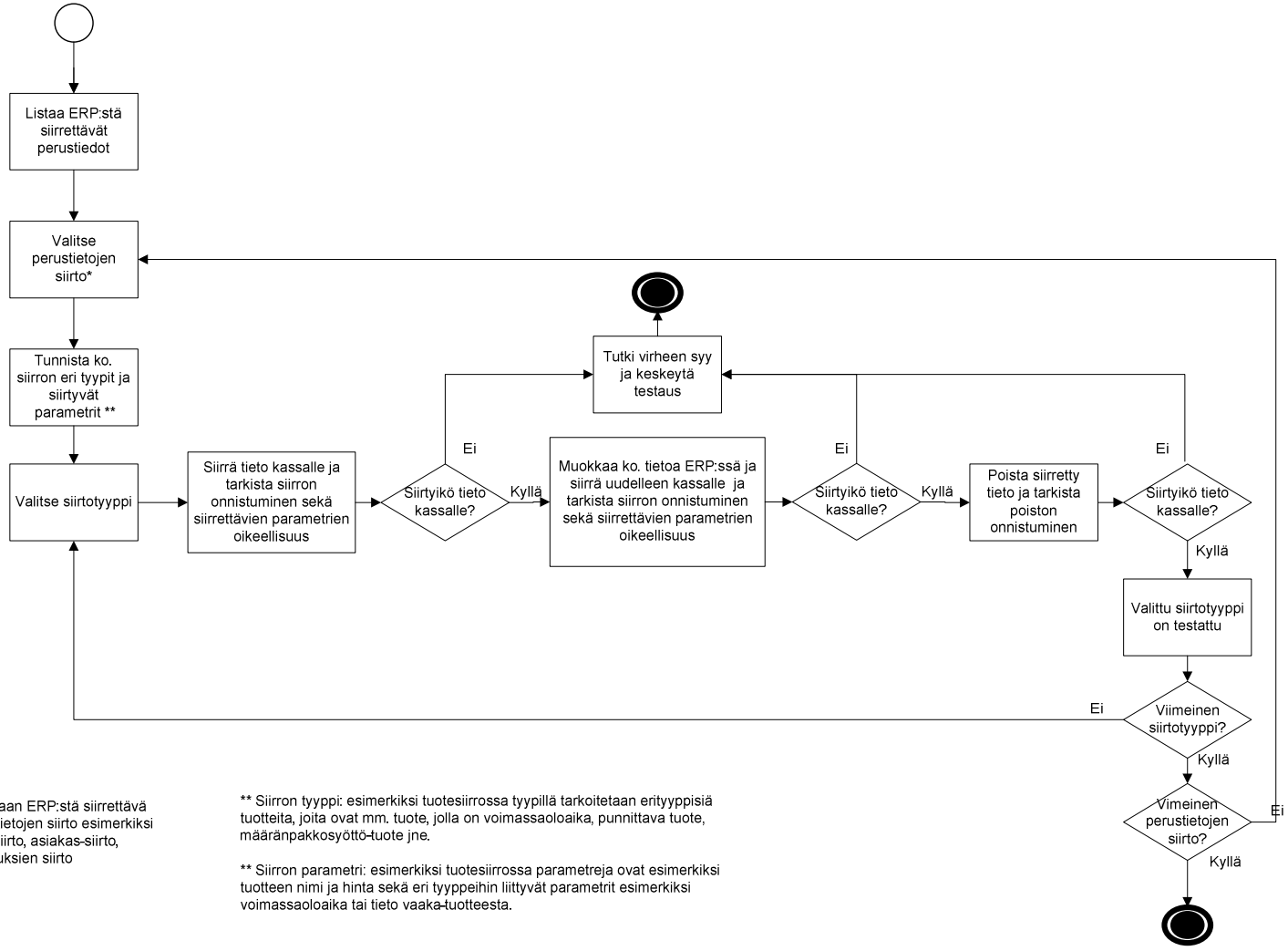
Sisältökeskeisen testauksen tarkoituksena oli siis varmistaa tiedonsiirron oikeellisuus liittymässä. Sisältökeskeinen testaus on jaettu kahteen eri osa-alueeseen eli tiedonsiirto ERP:stä kassalle ja kassalta ERP:iin.

Tiedonsiirto ERP:stä kassalle

Tiedonsiirto ERP:stä kassalle käsittää perustietojen siirron ja niiden oikeellisuuden testaamisen.

Testaaminen alkaa siitä, että tunnistetaan eri perustiedot, jotka siirtyvät ERP:stä kassalle. Tämän jälkeen tunnistetaan näiden siirtojen eri tyypit ja parametrit ja testataan niiden siirto, muutos ja poisto yksitellen kassalle. Ohessa prosessikaavio, jossa on kuvattu tiedonsiirron oikeellisuuden testaaminen tarkemmin.

jatkuu



* Valitaan ERP:stä siirrettävä perustietojen siirto esimerkiksi tuotesiirto, asiakas-siirto, alennuksien siirto

** Siirron tyyppi: esimerkiksi tuotesiirrossa tyypillä tarkoitetaan erityyppisiä tuotteita, joita ovat mm. tuote, jolla on voimassaoloaika, punnittava tuote, määränpakkosyöttö-tuote jne.

** Siirron parametri: esimerkiksi tuotesiirrossa parametreja ovat esimerkiksi tuotteen nimi ja hinta sekä eri tyyppeihin liittyvät parametrit esimerkiksi voimassaoloaika tai tieto vaaka-tuotteesta.

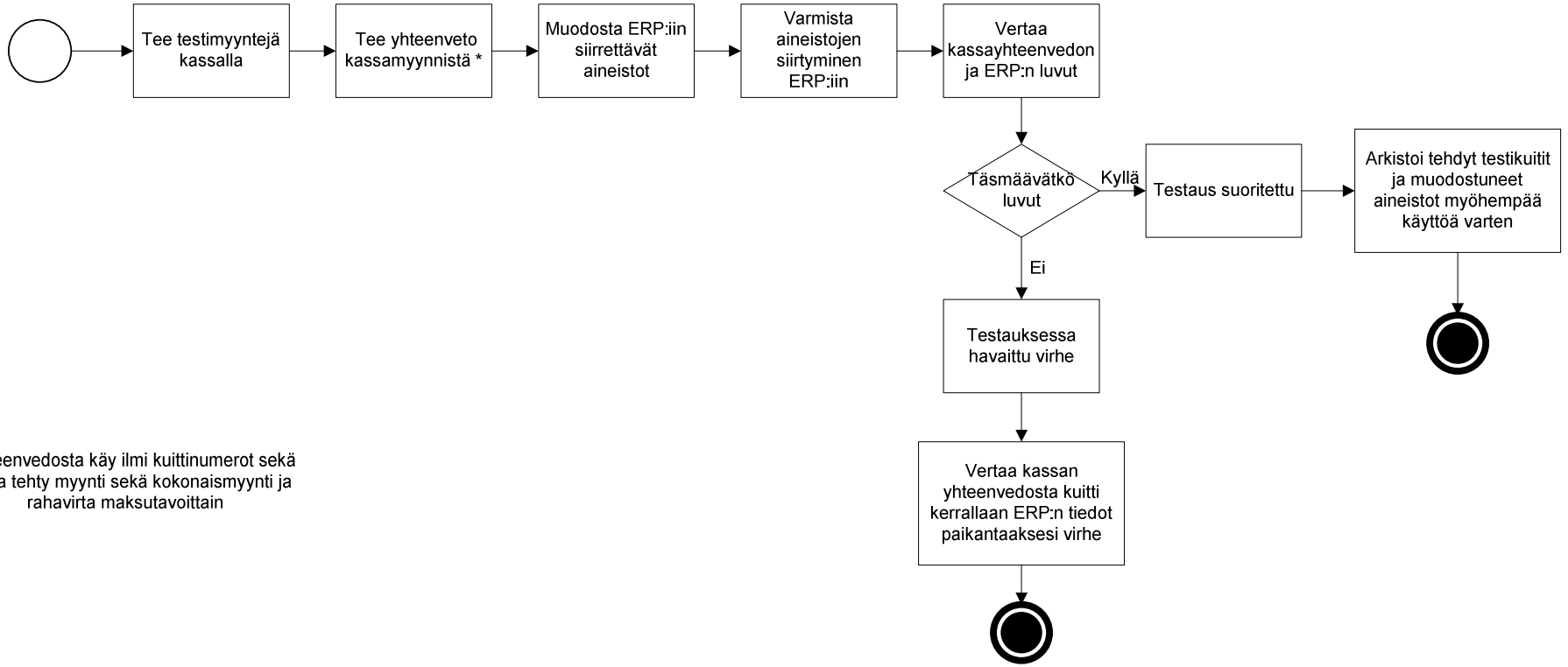
jatkuu

Tiedonsiirto kassalta ERP:iin

Tiedonsiirto kassalta ERP:iin käsittää myynti- kirjanpito- ja reskontra-aineistojen muodostuksen ja näiden aineistojen oikeellisuuden testaamisen.

Testaaminen alkaa siitä, että kassalla tehdään testimyyntiä, josta muodostetaan yhteenveto ja ERP:iin vietävät aineistot. Kun aineistot on sisäänluettu ERP.iin, verrataan kassan yhteenvetoa ERP:ssä näkyviin lukuihin, jolloin voidaan todeta muodostuneiden aineistojen oikeellisuus. Tehdyt testikuitit ja muodostuneet aineistot tulisi arkistoida myöhempää käyttöä varten. Ohessa prosessikaavio, jossa on kuvattu tiedonsiirron oikeellisuuden testaaminen tarkemmin.

jatkuu



* Yhteenvedosta käy ilmi kuittinumerot sekä kuitilla tehty myynti sekä kokonaismyynti ja rahavirta maksutavoittain

jatkuu

Yleensä uuden asiakasympäristön käyttöönotossa tehdään myös laaja kassatestaus, jolloin tuossa testauksessa tehtyjä myyntikuitteja voidaan hyödyntää tässä testauksessa. Mikäli tällaista testausta ei ole suoritettu, pitäisi kassalla tehdyissä testikuiteissa ottaa ainakin huomioon seuraavat testitapaukset;

Perusmyynti

- yksi tuote kuitilla vs. useita tuotteita kuitilla
- eri maksutavat ja niiden yhdistelmät (osa-maksu)
- kappalemäärä
- tekstiä kuitilla

Alennukset

- käy läpi asiakkaalla käytössä olevat eri alennustyypit

Palautus

- palautus eri maksutavoille
- palautus ja myynti samalla kuitilla (kuitin summa positiivinen, kuitin summa negatiivinen, kuitin summa nolla)

Kuitin mitätöinti

- eri maksutavat
- mitätöinti eri päiviltä

Muuta huomioitavaa

- erp-lähetteen rahastus
- kassaan maksu ja kassasta nosto
- asiakas kuitilla
- sovitus
- kuitin tallennus
- tilitys