Citynomadi Oy:n ohjelmistotestauksen kehittäminen

Antti Juutinen

Kesäkuu 2012
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikan suuntautumisvaihtoehto

JUUTINEN, ANTTI
Citynomadi Oy:n ohjelmistotestauksen kehittäminen

Opinnäytetyö 26 sivua, josta liitteitä 1 sivu
Kesäkuu 2012

Työn tilaaja Citynomadi Oy

TIIVISTELMÄ

Citynomadi Oy:n ohjelmistokehityksessä tehdään ohjelmistoja, joilla asiakkaat voivat selata GPS- tietoon perustuvaa informaatiota erilaisilla mobiililaitteilla. Tämän opinnäytetyön tarkoituksena on kehittää käytännöt, joiden puitteissa yrityksessä kehitettävät ohjelmistot testataan. Näin voidaan varmistaa tuotteiden olevan laadukkaita ja tarkoitukseensa sopivia.

Ohjelmistotestauksen käytäntöjen kehittäminen on yritykselle merkittävässä roolissa yrityksen laaduntarkkailun parantamiseksi. Käytäntöihin voidaan tulevaisuudessa tehdä tarkennuksia ja lisämäärityksiä.

Avainsanat opinnäytetyö, ketterät ohjelmistokehitysmenetelmät , ohjelmistotestaus

TAMK University of Applied Sciences
Degree programme in Computer Science
Option of software Engineering

JUUTINEN, ANTTI
Development of software testing in Citynomadi Ltd.
Bachelor's thesis 26 pages, appendices 1 page
June 2012

Co-operating company: Citynomadi Ltd.

ABSTRACT

In the software development of Citynomadi Ltd. there is a need for management and practices for software testing. Objects for this thesis is to develop an agile and reliable way of software testing process to the company.

Practices for software testing are one of the key aspects of companys strategic evolution. Because of that, there is high probability that those practices introduced in this thesis will be further developed in the utilization of the company.

Keywords thesis, agile software development, software testing

Sisällysluettelo

Termien ja lyhenteiden luettelo

Symbian	Mobiililaitteille suunniteltu käyttöjärjestelmä.
XML	Extensible Markup Language, standardi tekstimuotoisen datan esitykseen
GPS	Global Positioning System, maailmanlaajuinen satelliittipaikannusjärjestelmä
Android	Googlen kehittämä älypuhelinten käyttöjärjestelmä
iOS	iPhone Operating System, Applen kehittämä mobiililaitteiden käyttöjärjestelmä
Qt	Cross-platform application and UI framework, järjestelmäriippumaton sovellus- ja käyttöliittymäkehitysympäristö
Maemo	Nokian kehittämä avoin Linux-pohjainen mobiililaitteiden käyttöjärjestelmä
ISTQB	International Software Testing Qualifications Board, Kansainvälinen ohjelmistotestauksen tutkintolautakunta
Driver	Ajuri, jolla voidaan simuloidaan testattavaa ohjelmistokoodia kutsuvaa toiminnallisuutta. Ajuri siis käyttää olemassa olevaa toiminnallisuutta.
Stub	Tynkätoteutus, jolla kuvataan järjestelmän keskeneräistä osaa. Tyngillä simuloidaan järjestelmän kutsumien funktioiden toteutusta.

1 Johdanto

Citynomadi Oy:n on pieni suomalainen ohjelmistokehitykseen keskittynyt yritys, joka on perustettu vuonna 2009. Yrityksessä kehitetään erityisesti mobiileille päätelaitteille tarkoitettuja ohjelmistoja käyttäen mahdollisimman kehittyneitä tekniikoita ja työkaluja. Tällä hetkellä yrityksessä on kaksi päätoimista ohjelmistokehitykseen keskittynyttä työntekijää. Lisäksi yrityksessä on markkinointiin keskittynyt toimitusjohtaja ja pääomistaja, sekä sivutoiminen graafikko. Citynomadin ohjelmistokehityksessä luotetaan pääasiassa ketteriin menetelmiin.

Tämän opinnäytetyön tavoitteena on kehittää malli ja käytännöt yrityksen kehitystyön kohteina olevien ohjelmistojen testaukseen, jotka tukevat kehitystyötä ja varmistavat julkaistujen tuotteiden korkean laadun.

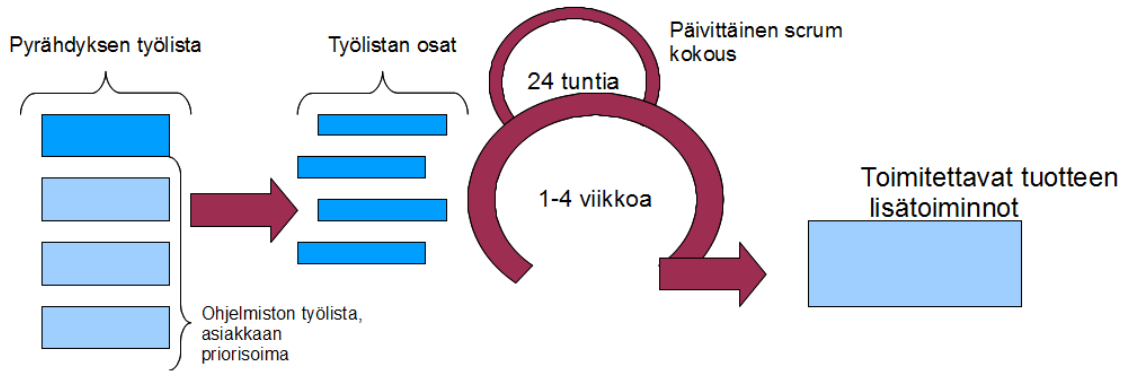
Aluksi tutustutaan lyhyesti ketteriin ohjelmistokehitysmenetelmiin. Seuraavaksi perehdytään ohjelmistotestaukseen liittyvään suunnitteluun ja perusteisiin niissä tehtäville valinnoille. Tämän jälkeen keskitytään dokumentteihin, joita käytetään testauksen aikana. Itse testaustyön vaiheita kuvataan luvuissa 5-12, ja lopuksi tarkastellaan Citynomadi Oy:n kehittämän ohjelmiston testauksen suunnittelua.

2 Ketterät ohjelmistokehitysmenetelmät

Agile manifesto koostettiin vuonna 2001, ja se on perusta erilaisille ohjelmistokehityksessä käytettäville ketterille menetelmille, joita ovat mm. Scrum, eXtreme Programming ja Rational Unified Process (RUP). Julistuksessa nostetaan tärkeimmäksi tavoitteeksi ohjelmiston toimivan version aikaisen toimituksen, asiakkaan tarpeet täyttäen. Lisäksi korostetaan lyhyttä päivitysaikataulua, motivoitunutta henkilöstöä ja kehitystiimin itseohjautuvuutta. (Agile manifesto 2001)

2.1 Scrum

Scrum on eräs laajimmin käytetyistä ketteristä menetelmistä. Se on lähtökohtaisesti iteratiivinen ja inkrementaalinen. Kehitystyö tapahtuu pyrähdyksissä, jotka kestävät 1-4 viikkoa. Pyrähdyksen alussa valitaan kehitettävät ominaisuudet ohjelmiston työlistalta. Listalla on kuvattu ohjelmiston tavoitteet arvioituna työmäärältään ja tärkeydeltään. Päivittäisissä palavereissa kehitystiimi kokoontuu enintään 15 minuutin tapaamiseen, jossa keskustellaan, mitä on tehty, mitä tulee päivän aikana tekemään ja onko etenemiselle esteitä. Kehitysryhmä vastaa ominaisuuksien toteutuksen valmistumisesta, testauksesta ja integroinnista pyrähdyksen loppuun mennessä. Pyrähdyksen päätteeksi pidetään demonstraatiotilaisuus, jossa tulokset esitellään asiakkaalle ja sidosryhmille. Lisäksi pyrähdyksen lopussa pidetään vielä kehitysryhmän sisäinen palaveri, jossa analysoidaan ryhmän toimintaa ja keskustellaan potentiaalisista kehityskohteista. (Wikipedia, scrum-menetelmä) Kuvassa 1 havainnollistetaan scrum-prosessin etenemistä. Kuva 1 on myös suurennettuna liitteessä 1.



Kuva 1: Scrum-menetelmän prosessikaavio (Certiconglobal.com, Scrum-process)

2.2 Ohjelmistotestaus ketterillä menetelmillä

Perinteisesti ohjelmiston testaus on alkanut vasta, kun kehitystyö on valmis. Tämä ei kuitenkaan sovi ketterästi kehitettyihin sovelluksiin, sillä ohjelmisto kehitetään pyrähdyksissä. Erilliselle testausjaksolle ei ole pyrähdyksessä aikaa. Tästä syystä testaus aloitetaan mahdollisimman varhain, jo ohjelmistoa kehitettäessä.

Kun testataan ketterillä menetelmillä kehitettyä sovellusta, on testien muodostamisessa haasteita, koska jotkin määrittelydokumenteista puuttuvat. Dokumentit kuuluvat oleellisena osana perinteisiin, niin sanottuihin rakenteellisiin ohjelmistonkehitysmalleihin. Testitapaukset voidaan kuitenkin muodostaa käyttötapausten ja skenaarioiden pohjalta. Ketterillä menetelmillä voidaan myös käyttää testauslähtöistä ohjelmointitapaa, jossa ohjelmoija kirjoittaa perusmoduulitestit ennen itse moduulin kehittämistä.

3 Testauksen suunnittelu ja testausstrategia

Testaus on suunniteltava kehitystyön kanssa tiiviisti. Varsinkin ketterillä menetelmillä kehitettävät sovellukset ovat alttiina laajoille muutoksille ohjelmiston rakenteessa. Näin ollen testausryhmän on oltava koko ajan yhteydessä kehitysryhmän kanssa, sillä ohjelmiston määritykset voivat muuttuvat kehitystyön aikana varsin paljon. Vaatimukset taas toimivat testauksen perustana, testauksen varmistaessa sen, että ohjelmisto toimii siten, kuin se on määritetty ja niillä vaatimuksilla mitä sille on annettu.

3.1 Testiluettelon laadinta

Alustavaa testiluetteloa laadittaessa voidaan tarkastella ohjelmiston suunnitteludokumentteja. Jos soveltuvaa dokumentaatiota ei ole, luodaan testiluettelo siten, että kirjataan ylös ohjelmiston jokainen merkittävä toiminto ja tiedetty tai epäilty ohjelmiston realiteetti. Nämä toiminnot voivat tulla ohjelmiston markkinointi- ja suunnittelukuvauksista, käyttöliittymän valikoista, tai ohjelmiston funktioista. Tämä alustava testiluettelo toimii lähtökohtana testityön määrän arvioinnissa.

3.2 Testausstrategia ja priorisointi

Koska kokonaisvaltainen testaus on monissa tapauksissa mahdotonta, on määriteltävä prioriteetteja. Erilaiset testaustekniikat ja testauksen päättymiskriteerit on muodostettava riippuen ohjelmistoon liittyvistä riskeistä. Kriittiset järjestelmän komponentit on testattava intensiivisesti. Sen sijaan vähemmän kriittisillä komponenteilla testaukseen riittää vähemmän kokonaisvaltainen testaus tai joissain tapauksissa niistä voidaan jopa luopua. Päätös komponenttien testauksesta tulee olla tarkkaan harkittu, jotta voidaan saavuttaa mahdollisimman hyvä testauksen optimointi ohjelmiston kriittisten komponenttien ja testien kattavuuden suhteen. (Spillner, Andreas 2007, 12)

Ohjelmistoprojekteissa aika on usein rajallinen. Tämä on otettava huomioon myös testauksen suunnittelussa. Testien priorisoinnilla voidaan varmistaa, että järjestelmän kriittisimmät komponentit testataan, jos kaikkia suunniteltuja testejä ei voida suorittaa ajan tai resurssien puutteen vuoksi.

4 Dokumentaatio

Testauksessa käytettävä dokumentaatio koostuu seuraavista osa-alueista: testaussuunnitelma, bug-tracker -vianhallintaohjelmisto, testiluettelo sekä mitoituslaskentataulukko. Lisäksi projektin edetessä hallinnolle voidaan tehdä myös muita dokumentteja, kuten väliraportteja ja loppuyhteenveto. Dokumentaation on syytä olla avoin ja jokaisen avainhenkilön saatavissa, esimerkiksi inter- tai intranetissä tarvittavilla toimenpiteillä suojattuna. Näin jokaisella asianosaisella on aina uusin informaatio saatavilla. Myös muokkausoikeus on hyvä antaa useammalle henkilölle. Dokumentaation ylläpitoon on kuitenkin nimitettävä vastuuhenkilö, jolloin varmistutaan materiaalin yhdenmukaisuudesta. Dokumentaatio voidaan rakentaa esimerkiksi työryhmäohjelmien, kuten TWikin tai Microsoftin SharePoint Services -ohjelmiston avulla. Testausdokumentaatio ei ole testauksen varsinainen päämäärä, joten siihen sisällytetään vain oleelliset asiat.

4.1 Testaussuunnitelma

Testaussuunnitelmaan kirjataan ominaisuudet, jotka testataan ja joita ei testata, testien läpäisykriteerit, erilaisten testausympäristöjen tarve, henkilöstö- ja koulutustarpeet, testauksen riskit, testien suoritusaikataulu sekä muut käytettävät resurssit.

4.2 Bug tracker -vianhallintaohjelmisto

Virheiden seurantaan tarvitaan järjestelmä, johon kirjata löytyneet epäkohdat, niiden vakavuusaste ja korjaustoimet sekä aikataulu ja henkilö, joka varmistaa virheen korjauksen. Lisäksi merkitään, kuinka virhe löydettiin merkitsemällä testin tunnus tai tutkiva testaus. Markkinoilla on olemassa monia kehittyneitä seurantatyökaluja, kuten esim. MantisBT, Bugzilla ja Flyspray.

4.3 Testiluettelo

Testiluettelo on dokumentti, joka kokoaa kaikki testitapaukset. Luetteloon merkitään testin yksilöity tunnus, testin syöttö- ja odotetut paluuarvot, testin suorittamiseksi tarvittavat riippuvuudet, suunniteltu testien suoritusjärjestys ja prioriteetti. Luettelo tehdään samalla työryhmäohjelmistolla, kuin muutkin testausryhmän tekemät yleiset dokumentit, esimerkiksi Twikillä.

4.4 Mitoituslaskentataulukko

Tämä dokumentti sisältää suhteelliset osuudet, kuten testien kattavuus, ajankäytön, testauskustannukset ja mahdollisesti myös testauksen/testaamatta jättämisen kustannukset. Se tulee sisältämään olettamuksia ja arvioita, jotka tulee selkeästi erottaa todellisista arvoista. Testiprosessissa nämä arviot tullaan korvaamaan oikeilla arvoilla. Tämän dokumentin avulla voidaan seurata projektin toimituskelpoisuutta. Lisäksi mitoituslaskentataulukon avulla voidaan parantaa arvioita seuraavissa projekteissa.

4.5 Ohjelmakoodin hallinta

Jaottelemalla ohjelmakoodi eri hakemistoihin voidaan projektin edistymistä seurata paremmin, ja näin hakemistoissa ei ole ylimääräistä materiaalia. Hakemistohierarkian hallintaan ja muokkaukseen on kehittyneitä työkaluja, kuten Git ja Subversion. Ohjelmakoodi jaetaan hakemistoihin seuraavasti:

Development: Tässä hakemistossa on kehitystyön alainen ohjelmakoodi. Ohjelmistoa ei ole tarkastettu eikä testattu, joten se on erittäin epävakaata.

Module testing: Ohjelmakoodi on tarkastettu ja on moduuli- ja integraatiotestauksessa.

Release candidate: Tässä hakemistossa on integraatiotestattua ohjelmakoodia. Kun hakemistossa on tärkeimmät ohjelmiston komponentit, voidaan järjestelmätestaus aloittaa.

5 Lähdekoodin tarkastus

Lähdekoodin tarkastus on tärkeä osa sekä kehitys- että testaustyötä. On osoitettu, että hyvin järjestetyissä tarkastustilaisuuksissa paljastuu jopa 30-70% logiikkasuunnittelu- ja ohjelmointivirheistä (Myers, Glenford J. 2004, 23). Tarkastusryhmä koostuu yleensä neljästä jäsenestä. Yksi henkilöistä toimii puheenjohtajana. Puheenjohtajan odotetaan olevan pätevä ohjelmoija, muttei tarkastelun kohteena olevan ohjelmakoodin kirjoittaja, eikä hänellä tarvitse olla yksityiskohtaisia tietoja ohjelmasta. Puheenjohtajan tehtäviin kuuluvat tarkastustilaisuuden aikataulutus, tilaisuuden johtaminen, sekä löytyneiden virheiden korjausten varmistus. Muut tilaisuuden jäsenet ovat ohjelman kehittäjä, suunnittelija (jos eri henkilö kuin kehittäjä) sekä testausspesialisti.

Tarkastettavan moduulin kehittäjä toimittaa ohjelman suunnitteluspesifikaatiot sekä ohjelmakoodin ryhmälle muutamaa päivää ennen ohjelmakoodin tarkastusta, ja ryhmän jäsenet tutustuvat materiaaliin ennen tilaisuutta. Itse tarkastustapahtumassa ohjelmoija kertoo sovelluksen logiikan käsky kerrallaan. Muut osallistujat voivat kysyä tarkentavia kysymyksiä, ja kertoa mahdollisista huomaamistaan virheistä, jotka moduulin kehittäjä kirjaa ylös. Sovellus analysoidaan aiemmin löytyneiden virheiden perusteella tehdyn tarkastuslistan mukaan.

Puheenjohtaja varmistaa, että osallistujat keskittyvät virheiden löytämiseen, eikä niiden korjaamiseen. Tarkastuksen jälkeen ohjelmoijalla on lista virheistä ja päätetään, tarvitaanko kyseiselle sovellukselle jatkotarkastusta. Lisäksi virheet analysoidaan, ja tehdään tarvittaessa muutokset tarkastuslistalle. Moduulin hyväksytyn tarkastuksen jälkeen se voidaan luovuttaa testausryhmälle.

6 Testausmetodit

Testauksessa voidaan käyttää erilaisia strategioita virheiden esilletuomiseen ja niiden selvittämiseen. Tällaisia metodeja ovat mustalaatikkotestaus, lasilaatikkotestaus ja harmaalaatikkotestaus.

6.1 Mustalaatikkotestaus

Mustalaatikkotestaus perustuu toiminnallisiin ja laadullisiin vaatimuksiin. Tutkittavana ovat kohteen tulosteet erilaisilla syötteillä. Testattavan kohteen ominaisuuksia tarkastellaan vertaamalla saatuja tulosteita odotettuihin tulosteisiin. Testitapaukset muodostetaan moduulin määrittelyn perusteella.

6.2 Lasilaatikkotestaus

Lasilaatikkotestauksessa tarkastellaan lähemmin ohjelmakoodia. Tässä testaustavassa testitapaukset perustuvat ohjelman rakenteeseen. Tavoitteena on valita testitapaukset niin, että kaikki testattavan kohteen ohjelmapolut tulisi käytyä läpi. Toisin kuin mustalaatikkotestauksessa, ei lasilaatikkotestauksessa riitä, että toiminnan tuloksen arvo on oikea, vaan myös ohjelman sisäinen rakenne on tutkittava. Näin voidaan varmistua siitä, että tuloksen arvo on toteutettu oikein.

6.3 Harmaalaatikkotestaus

Yksittäisinä musta- ja lasilaatikkotestaukset eivät riitä täysipainoiseen testaukseen. Mustalaatikkotestauksessa riskinä on, ettei sillä voida paljastaa tietyntyyppisiä virheitä, esimerkiksi raja-arvovirheitä tai tietovirrassa tapahtuvia häiriöitä. Harmaalaatikkotestaus yhdistää osia musta- ja lasilaatikkotestauksesta. Siinä tarkastellaan käyttäjäpuolen lopputulosta, järjestelmäkohtaista teknistä tietoa sekä ympäröivää käyttöjärjestelmää. Tämä lähestymistapa sopii hyvin niihin ohjelmistoihin, jotka vaativat useita ohjelmisto- ja laitteistokomponentteja. Harmaalaatikkotestauksessa voidaan paljastaa virheitä tiedonkulussa sekä vialliset ohjelmiston määritykset ja yhteensoveltuvuudet. (OAMK, testausstrategiat.)

7 Moduuli- ja regressiotestaus

Moduulitestaus tai yksikkötestaus on ohjelman itsenäisten aliohjelmien, alirutiinien tai toimintatapojen testausprosessi. Moduulitestauksen tarkoituksena on verrata moduulin toimintoa johonkin toiminnalliseen- tai käyttöliittymämäärittelyyn, joka kuvaa sitä. Tällä testaustavalla on merkittäviä etuja. Moduulitestauksessa voidaan yhdistää erilaisia testaustekniikoita, kuten musta- ja lasilaatikkotestaus. Se myös helpottaa virheiden etsimistä, koska kun virhe löytyy, sen tiedetään esiintyvän juuri tietyssä moduulissa. Lisäksi useita moduuleja voidaan testata samaan aikaan.

Moduulitestaus suoritetaan uudelleen niille kokonaisuuksille, jotka muuttuvat kehitystyön edetessä. Tätä kutsutaan regressiotestaukseksi. Tällä tavoin pyritään varmistamaan, että ohjelmistokoodin vanha osa toimii oikein, eikä uusi toiminnallisuus aiheuta konflikteja.

Moduulitestaukseen on kehitetty helpottavia työkaluja lähes jokaiselle alustalle. Qt:lla kehitettäessä voidaan käyttää QTestLib:ä, Androidille kehitettäessä The Android testing frameworkia ja objective C:lle OCUnit:a.

8 Integrointitestaus

Integrointitestauksen tarkoituksena on varmistaa ohjelmiston suurempien kokonaisuuksien välinen tehokkuus, toiminnallisuus ja luotettavuus. Nämä kokonaisuudet testataan niiden keskinäisten rajapintojen kautta käyttäen mustalaatikkotestausta. Onnistuneet- ja virhetapaukset simuloidaan tarvittavien parametrien ja datasyötteiden avulla. Jaettujen data-alueiden ja prosessien kommunikaatio testataan ja itsenäisten alijärjestelmien toiminnat otetaan käyttöön niiden rajapintojen kautta. Testitapaukset suunnitellaan ja suoritetaan siten, että yhteenliitetyt komponentit kommunikoivat niiden määritysten mukaisesti. Kokonaisvaltainen idea integrointitestauksessa on, että ohjelmiston elementeistä kootaan ja verifioidaan yhä suurempi toimiva järjestelmä. (Wikipedia, integraatiotestaus.)

8.1 Integrointitestauksen strategiat

Moduulien integroimiseen voidaan käyttää useita strategioita. Yleisimpiä näistä ovat top-down- ja bottom-up -strategiat, nk. kertarysäys, ja ydinohjelmaintegrointi. Strategioita voidaan yhdistää, riippuen testattavan ohjelmiston rakenteesta.

8.1.1 Top-down -testausstrategia

Tässä lähestymistavassa testaus aloitetaan päämoduulista tai käyttöliittymästä, joka kutsuu alemman kerroksen toteutuksia. Kutsuville funktioille muodostetaan stubeja, joilla simuloidaan alemman kerroksen toimintaa. Ylemmän tason testauksen jälkeen nämä stubit korvataan seuraavan tason moduuleilla. Tätä jatketaan niin, että lopulta kaikki alemmat kerrokset ovat toiminnassa. Etuna tässä toimintatavassa on se, että kun virheellistä toimintaa löydetään, sen tiedetään sijaitsevan juuri lisätyssä alijärjestelmässä. Lisäksi ohjelman kontrollirakenteiden toiminta voidaan varmistaa varhaisessa vaiheessa. Haasteina tämän lähestymistavan käytössä on se, ettei monissa tapauksissa ole yksittäistä ylhäältä alas johtavaa ohjelmistopolkua, jonka pohjalta integrointi voidaan suorittaa. Lisäksi stubien tekemiseen kuluu arvokasta työaikaa.

8.1.2 Bottom-up -testausstrategia

Toisin kuin top-down mallissa, aloitetaan testaus nyt järjestelmän alemmista kerroksista ja edetään sieltä ylöspäin. Testaus suoritetaan kirjoittamalla kutsuttavalle alemman tason moduulille ajuri eli driver. Ajuriin luodaan tarvittavat syötteet, joilla kutsutaan testattavaa ohjelmistokoodia. Tämän testausmallin etuna voidaan mainita se, että testitapauksien luominen ja niiden tulosten seuraaminen on helpompaa, kuin top-down mallissa. Haittapuolena mallissa on se, että kontrollirakenteiden ja arkkitehtuurin ongelmat paljastuvat vasta myöhäisessä vaiheessa (Spillner, Andreas. 2007, 110-121).

8.1.3 Kertarysäys

Kertarysäys ei varsinaisesti ole määritelty integrointistrategia, vaikka se onkin varsin yleisessä käytössä moduulien integroinnissa. Tässä mallissa ohjelman kaikki moduulit kootaan yhteen ja integrointitestaus aloitetaan, mikäli sovellus saadaan toimimaan. Kertarysäyksen etuna on, ettei drivereita tai stubeja tarvita, ja mahdollisesti integrointitestaus voidaan suorittaa nopeasti. Menetelmä soveltuu käyttöön silloin, kun testattava järjestelmä on pieni, rakenteltaan selkeä, ja jonka moduulit on testattu huolellisesti. Mikäli olemassa olevaan järjestelmään tehdään pieniä muutoksia tai uusi järjestelmä kehitetään käytössä olleista komponenteista, voidaan käyttää kertarysäysstrategiaa. Useissa tapauksissa tätä ei kuitenkaan voi suositella, sillä virheiden löytäminen vie aikaa ja on työlästä. Virheet, jotka löytyvät voivat käytännössä olla millä tahansa moduulien välisellä rajapinnalla. Lisäksi tämä malli on siitä haasteellinen, että kaikkien moduulien pitäisi olla valmiina samaan aikaan.

8.1.4 Ydinohjelmaintegrointi

Ydinohjelmaintegrointi aloitetaan yhdistämällä sovelluksen keskeiset toiminnalliset moduulit kertarysäyksellä. Tämä ydinohjelma testataan huolellisesti, jonka jälkeen sitä voidaan käyttää testialustana. Ohjelman toiminnallisuutta lisätään integroimalla uusia moduuleja ja testaamalla kasvavaa kokonaisuutta käyttäen top-down tai bottom-up strategioita. Ydinohjelmaintegrointia voidaankin kuvata edellä mainittujen lähestymistapojen hybridinä. Tämän strategian käyttö edellyttää huolellista sovelluksen rakenteen analysointia ydintoimintojen selvittämiseksi. (Satukangas, Anita 2003.)

9 Järjestelmätestaus

ISTQB:n määrittelemänä järjestelmätestaus on prosessi varmistua siitä, että järjestelmä täyttää sille asetetut vaatimukset. Järjestelmätestauksessa tarkastellaan koko projektissa mukana olevia ohjelmistoja ja laitteistoja. Tuloksia verrataan määrittelyvaiheessa syntyneeseen dokumentaatioon. Ideaalinen tilanne järjestelmätestauksen kannalta olisi, että testaajina on mahdollisimman itsenäinen taho, jolla ei ole tietämystä ohjelman rakenteesta ja yksityiskohdista.

Testitapaukset järjestelmätestaukseen muodostetaan järjestelmän arkkitehtuurista ja suunnittelusta, käyttäjän syötteiden ja käyttötapausten avulla. Tässä vaiheessa ei ole tarpeellista tehdä laajamittaista testausta, sillä suurin osa toiminnallisista virheitä tulisi olla löydetty ja korjattu jo aiemmissa testivaiheissa. (TestingGeek, System Testing)

Järjestelmätestausvaiheessa suoritetaan lisäksi muun muassa ohjelman, käytettävyys-stressi-, suorituskyky- sekä tietoturvatestaus. Virheitä, joita sovelluksessa tässä vaiheessa esiintyy, ovatkin tyypillisesti heikko suorituskyky ja tietoturvan ongelmat.

10 Tutkiva testaus

Ohjelmistotestauksessa merkittäväksi tavaksi etsiä virheitä etenkin ketterillä menetelmillä ohjelmistoja kehitettäessä on noussut tutkiva testaus. Tällä tapaa ohjelmistoa testattaessa ei virheiden etsimistä ole etukäteen valmisteltu, eikä sille ole asetettu tavoitetta. Käytännössä tutkivassa testauksessa käytetään suurelta osin valmista ohjelmistoa ja tutkitaan, miten ohjelma käyttäytyy käyttäjän syötteisiin. Tämä testaustapa vaatii testaajalta luovuutta ja ajattelukykyä testauksen kontrolloimiseksi. Esimerkiksi virheen löytyessä, sen esiintyminen on kyettävä toistamaan, sillä muuten vian löytäminen ja korjaaminen ohjelmakoodista on mahdotonta.

Tutkivassa testauksessa päähyödyksi nousee seuraavat asiat: etukäteisvalmisteluja ei tarvita niin paljon verrattuna perinteisiin testausmetodeihin, ja tärkeät virheet voidaan löytää nopeasti. Lisäksi itse testaustapahtuma vaatii älykästä ajattelua, eikä vain skriptattujen testien suorittamista. (Wikipedia, tutkiva testaus) Virheen esiinnyttyä tutkivan testin pohjalta voidaan tehdä automatisoitu yksikkötesti regressiotestaukseen.

Haittoina tälle testaustavalle ovat muun muassa se, ettei lennossa kehitetyille ja suoritetuille testeille voida suorittaa virhearviointia testitapausten suhteen. Lisäksi on haasteellista määrittää, mitkä testit ovat suoritettu.

Tutkivaa testausta voidaan käyttää etenkin ohjelmiston integrointitestausvaiheessa, jolloin moduulien välinen toiminta on vielä epävarmaa, mutta se ei ole poissuljettu metodi myöskään moduuli- tai järjestelmätestauksessa. Lisäksi tutkivalla testauksella voidaan verifioida, että aikaisemmissa testeissä on löydetty tärkeimmät virheet. (Wikipedia, tutkiva testaus)

11 Käytettävyystestaus

Käytettävyystestauksen tavoitteena on tarkkailla ihmisiä, jotka käyttävät tuotetta, jotta voidaan löytää virheitä ja mahdollisia parannuskohteita. Testaukseen liittyy myös se, kuinka testihenkilöt suhtautuvat neljään sovellukseen liittyvään kategoriaan: tehokkuus, tarkkuus, muistettavuus ja tunteisiin vetoavuus. Ensimmäisen testin tulokset voidaan pitää testin perustasona tai kontrollimittauksena. Sen jälkeiset testit voidaan verrata tähän kontrolliin, jotta voidaan osoittaa järjestelmän kehittymistä. (Wikipedia, käytettävyystestaus.) Seuraavassa kappaleessa esitellään kysymykset, joita käytettävyystestauksessa tarkastellaan.

Kuinka paljon aikaa ja montako vaihetta käyttäjä tekee suorittaakseen tietyn tehtävän tai operaation? Kuinka monta virhettä testihenkilö tekee ja onko niistä mahdollista toipua? Kuinka paljon henkilö muistaa ohjelmistosta testin jälkeen ja mahdollisesti tietyn ajan jälkeen? Mitä tunteita testitapahtuma herätti? Voiko henkilö luottaa ohjelman toimivuuteen vai aiheuttaako se stressiä? Voisiko testaajahenkilö suositella tätä ohjelmistoa ystävälleen?

11.1 Menetelmiä käytettävyystestin suorittamiseksi

Käytettävyystestauksen suorittamiseksi luodaan skenaario tai realistinen tilanne, missä käyttäjä suorittaa sarjan toimenpiteitä samalla, kun tarkkailija seuraa ja kirjaa merkintöjä. Testissä voidaan myös käyttää ennakko- ja jälkikysymyksiä palautteen saamiseksi. Mahdollisimman suuren hyödyn käytettävyystestauksesta saa silloin, kun se aloitetaan jo ohjelmiston prototyyppiasteella.

11.1.1 Käytävätestaus

Käytävätestaus on käytettävyystestauksen perusmenettely. Käytävätestauksessa testausjoukko muodostetaan viidestä tai kuudesta satunnaisesta henkilöstä, jotka kuvastavat ohjelmiston käyttäjien läpileikkausta. Näin voidaan varmistaa, että testaajilla ei ole aikaisempaa kokemusta tarkastelun kohteena olevasta järjestelmästä.

11.1.2 Ammattilaisen arviointi

Toinen yleinen käytettävyystestauksen metodi on käyttää käyttöliittymäammattilaisia. Nämä ammattilaiset käyttävät järjestelmää, kuten kohdeyleisö. Käytön aikana testaajat etsivät poikkeamia hyviksi havaituista käytettävyysperiaatteista. Yleensä normaali käytävätestaus on parempi tapa arvioida tuotteen käytettävyyttä, koska parhaatkaan ammattilaiset eivät voi korvata tuotteen oikeita käyttäjiä. On kuitenkin tilanteita, joissa on hyvä käyttää tätä metodia. Esimerkiksi, jos ohjelmistossa voidaan käyttää monia erilaisia käyttöliittymäkomponentteja, voi ammattilainen suositella niistä hänen mielestään tilanteeseen sopivinta. (Expert Review: alternative to usability testing?) Arviointia ja konsultointia tulee käyttää mahdollisimman varhain, jolloin muutokset käyttöliittymäkomponenteissa voidaan tehdä vielä suhteellisen kivuttomasti.

12 Turvallisuustestaus

Viime aikoina on julkisuuteen noussut yhä enemmän uutisia tietojärjestelmien haavoittuvuuksia hyödyntävistä tietomurroista. Tämän vuoksi ohjelmistolle on syytä tehdä arviointi sen mahdollisista haavoittuvuuksista. Lisäksi tulee varmistaa, ettei käyttäjien henkilökohtaista tietoa saada järjestelmästä ulos. Suojauksessa auttaa ohjelmiston turvalliseksi kehittämisen lisäksi käyttäjätietojen kryptaaminenn ja suolaaminen sopivilla työkaluilla. Näin ne ovat murtajalle hyödyttömiä, vaikka tietokanta altistuisi hyökkäykselle. Myös käyttöjärjestelmien ja erilaisten alustojen tietoturvapuutteet ja haavoittuvuudet tulee ottaa huomioon . (United States department of homeland security, Risk-based and functional security testing)

Yrityksen kehitystyössä tulee henkilöstölle kouluttaa perusteet turvallisesta ohjelmistokehityksestä. Lisäksi lähdekoodin tarkastuksessa on tutkittava, onko moduulin toiminnassa riskejä turvallisuuden kannalta ja mietittävä keinoja välttää tai kiertää niitä. Pienessä yrityksessä on erittäin haasteellista pitää ohjelmistonkehittäjät tietoisina kaikista turvallisuuteen liittyvistä uhista, joten joissakin tapauksissa on hyvä käyttää ulkopuolista konsultointia turvallisuuteen liittyvissä asioissa, etenkin jos henkilö- tai maksuvälinetietoja on ohjelmiston käytössä.

13 Nomadin testaussuunnitelma

Citynomadin keskeisin ohjelmistokehityskohde on Nomadi, jolla esitetään reittejä ja niiden solmupisteitä mobiililaitteissa. Nomadiin kuuluu asiakasohjelmisto, palvelinsovellus sekä tietokanta, joissa reitit säilytetään. Asiakasohjelmisto tarjoaa käyttäjälle mahdollisuuden reittien hakuun ja näyttämiseen. Palvelinsovellus tulkkaa tietokannasta haetut reitit XML-muotoisina asiakasohjelmistolle.

Testausnäkökulmasta Nomadi on varsin haasteellinen ohjelmistokokonaisuus. Nomadin asiakassovelluksesta on saatavana versiot Qt- ohjelmistokehitysympäristöllä tehtynä Symbian- ja Maemo- järjestelmille, sekä omat sovellukset sekä Android-, että iOS-laitteille. Nämä järjestelmät eroavat toisistaan hyvinkin paljon. Palvelinsovellus ja tietokanta tarjoavat saman rajapinnan kaikille asiakasohjelmistoille.

Testauksen ensisijaisena prioriteettina on palvelinsovelluksen ja tietokannan tarjoama rajapinta, koska se on kaikille asiakasohjelmille yhteinen ja näin ollen kriittisin osa järjestelmäkokonaisuutta.

Asiakasohjelmien testauksessa ohjelmistokehittäjät tekevät moduulitestauksen. Testiryhmä keskittyy integraatiotestaukseen, käytettävyystesteihin ja siihen, että asiakasohjelmat ovat toiminnallisuudeltaan yhteneväisiä toistensa kanssa niin hyvin, kuin se ohjelmistoalustojen kesken on mahdollista. Turvallisuustestauksen penetraatiotestit suoritetaan ulkopuolisen yrityksen toimesta, ennen kuin sovelluksesta julkaistaan maksuliikennettä käsittelevää versiota.

14 Yhteenveto

Ohjelmistotestauksen käytännöt, jotka tässä työssä esiteltiin, ovat varsin yleisesti hyväksyttyjä ja käytettyjä. Nykyaikaisen ohjelmistokehityksen haasteina olevat aika ja resurssien niukkuus asettavat usein haasteelliset puitteet testauksen suorittamiselle. Tästä syystä on tärkeää miettiä huolellisesti, kuinka paljon ohjelmistoissa esiintyvät häiriöt ja viat vaikuttavat yrityksen maineeseen ja tuotteiden haluttavuuteen markkinoilla.

Yrityksessä jatketaan tässä työssä esitettyjen mallien ja käytäntöjen kehittämistä yrityksen kehitystyön laadun varmistamiseksi. Tämä onkin tarpeellista, sillä ohjelmistoteollisuus kehittyy nopeammalla tahdilla kuin mikään muu teollisuuden ala.

LÄHTEET

Spillner, Andreas. 2007. Software Testing Practice: Test Management. 1st edition. Rocky Nook Inc. ISBN 978-1-933952-13-0. s.12

Myers, Glenford J. 2004. The art of software testing. 2nd edition. John Wiley & Sons, Inc. ISBN 0-471-46912-2. s. 23

OAMK, testausstrategiat. Luettu 20.05.2011.
http://www.oamk.fi/sbc/testaus/testausstrategiat.htm

Wikipedian artikkeli integraatiotestauksesta. Luettu 23.05.2011.
http://en.wikipedia.org/wiki/Integration_testing

Spillner, Andreas. 2007. Software Testing Practice: Test Management. 1st edition. Rocky Nook Inc. ISBN 978-1-933952-13-0. s.110-121

Satukangas, Anita. 2003. Pro gradu: Yksikkö- ja integrointitestauksen menetelmät ja mittarit. Luettu 25.05.2011.
http://ethesis.helsinki.fi/julkaisut/mat/tieto/pg/satukangas/yksikkoj.pdf

TestingGeek: System Testing. Luettu 26.05.2011. http://www.testinggeek.com/system-testing

Wikipedian artikkeli käytettävyystestauksesta. Luettu 24.05.2011.
http://en.wikipedia.org/wiki/Usability_testing

Expert Review: alternative to usability testing? Luettu 24.05.2011.
http://www.userintelligence.com/ideas/news/expert-review-alternative-usability-testing

Agile manifesto, 2001. Luettu 5.3.2012. http://agilemanifesto.org/iso/fi/

Wikipedian artikkeli scrum-menetelmästä. Luettu 5.3.2012.
http://en.wikipedia.org/wiki/Scrum_(development)

Wikipedian artikkeli tutkivasta testauksesta. Luettu 8.3.2012.
http://en.wikipedia.org/wiki/Exploratory_testing

Kuva 1: Scrum-prosessi. Luettu 8.3.2012. http://www.certiconglobal.com/content/scrum

United States department of homeland security: Risk-based and functional security testing . Luettu 22.5.2012. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing.html

LIITTEET

Liite 1. Scrum-prosessi