Tietotekniikan koulutusohjelma

Ohjelmistotuotanto

2011

Riikka Kaarima

JATKUVA INTEGROINTI QT-YMPÄRISTÖSSÄ

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Ohjelmistotuotanto

Kesäkuu 2011 | 37 sivua

Ohjaajat: Jani Ruotsalainen, Tiina Ferm

Riikka Kaarima

JATKUVA INTEGROINTI QT-YMPÄRISTÖSSÄ

Tässä opinnäytetyössä perehdytään jatkuvaan integrointiin, joka on eräs ketteristä ohjelmistokehitysmenetelmistä. Tavoitteena on kuvata menetelmän tarjoamat edut ja sen käyttöönottamiseen liittyvät haasteet ja niihin vastaaminen Qt-ohjelmointiympäristössä, jollainen on käytössä opinnäytetyön toimeksiantajalla Nokia Oyj:n DSS-osastolla (Device Sw Update and Security Solutions).

Jatkuvan integroinnin prosessiin ja Qt-ohjelmointiympäristöön syvennyttiin sekä kirjallisuuden avulla että toimeksiantajan luona. Työn teoriaosuudessa esitellään jatkuvan integroinnin periaatteet ja käytännöt sekä prosessi ja edut. Myös Qt-kehitysympäristö kuvataan pääpiirteissään. Lopuksi pohditaan jatkuvaa integrointia Qt-ympäristössä tarvittavien työkalujen sekä uuden toimintatavan käyttöönoton tuomien haasteiden hallinnan kannalta. Haasteista suurimmaksi ilmeni muutoksen aikaan saaminen henkilöstön toimintatavoissa ja asenteissa.

Opinnäytetyön käytännön osuus toteutettiin Turku AMK:n atk-laboratoriossa pystyttämällä sinne pienimuotoinen jatkuvan integroinnin ympäristö Qt-ohjelmointiympäristöön. Tämä tehtiin maksuttomia ohjelmia käyttäen. Ympäristön pystytys sujui helposti, mutta tässä työssä on otettava huomioon testiympäristön minimaalisuus.

Työssä kävi ilmi, että kunhan käyttöönotto valmistellaan hyvin koko ohjelmistokehitysprosessi huomioon ottaen ja kaikki ohjelmistokehitysprosessiin liittyvät henkilöt siihen sitouttaen, jatkuvan integrointikäytännön käyttöönotto tarjoaa mahdollisuuden saada ohjelmistokehitysprojektin onnistumisen kannalta kriittinen vaihe, integrointi, stabiilimmaksi. Vakaan integrointikäytännön tärkeys vielä korostuu monialustaympäristössä, jollainen Qt on.

ASIASANAT:

Jatkuva integrointi, Agile, Qt

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Software Engineering

June 2011 | 37 pages

Instructors: Jani Ruotsalainen, Tiina Ferm

Kaarima Riikka

CONTINUOUS INTEGRATION IN QT ENVIRONMENT

This thesis looks into the continuous integration, which is one of the agile software development methods. The goal is to describe the advantages that continuous integration offers for a sw development process. Also the challenges and how to handle them when deploying a new practice generally and specifically in a Qt programming environment are discussed. The commissioner of the thesis was DSS (Device Sw Update and Security Solutions), Nokia PLC. Salo.

The literature, sw developing processes and the environment at DSS were used to engross in the continuous integration process and the Qt programming environment. In the theoretical part of this thesis the principals and practices as well as the process and advantages of continuous integration are explained. Also the basic functions of Qt programming environment are described. Finally, the continuous integration in Qt environment is thought about from the perspective of the needed tools and of controlling the challenges the deployment will cause.

The practical part of the thesis was implemented at Turku University of Applied Scienses IT laboratory, by setting up a basic environment for continuous integration in Qt environment. This was carried out by using only open source applications. Setting up the environment was easier than expected, but the simplicity of the test environment explains some of it.

It appeared, that as long as the deployment of the continuous integration process is prepared the whole software development process in mind, and committing each and everyone in the process it self, deploying the continuous integration practice will make it possible to stabilize the software integration phase. This is a critical part of the software development process. In a cross-platform environment, such as Qt is, the importance of a smooth integration practice is highlighted.

KEYWORDS:

Continuous integration, Agile, Qt

SISÄLTÖ

1 JOHDANTO 5

2 JATKUVA INTEGROINTI 7

2.1 Jatkuvan integroinnin periaatteet ja käytännöt 7

2.2 Jatkuvan integroinnin prosessi ja edut 12

3 QT 17

4 JATKUVA INTEGROINTI QT-YMPÄRISTÖSSÄ 22

4.1 Tarvittavat työkalut 22

4.2 Haasteiden hallinta uuden toimintatavan käyttöönotossa 25

5 JATKUVAN INTEGROINNIN PROSESSI DSS:SSÄ 26

6 JATKUVAN INTEGROINTIYMPÄRISTÖN PYSTYTYS QT-KEHITYSYMPÄRISTÖÖN 31

7 YHTEENVETO 36

LÄHTEET 37

KUVAT

Kuva 1. Jatkuvan integroinnin järjestelmän komponentit 13

Kuva 2. Integrointinappula 15

Kuva 3. Qt Creatorin tarjoamat palvelut 18

Kuva 4. Quillotine-projektin koostamisprosessi 27

Kuva 5. Ote kompleksisuusmittauksen yksityiskohtaisesta raportista 29

Kuva 6. Qt-projektin koostamisprosessi 30

Kuva 7. C++-testiohjelma Qt-ympäristössä 32

Kuva 8. Git-käyttöliittymä 33

Kuva 9. Jenkins-ikkuna 34

Kuva 10. Koostamistiedot 35

1 Johdanto

Tämä opinnäytetyö sai alkunsa tarpeesta kuvata jatkuvan integroinnin prosessi ja sen käyttöönotto jo olemassa olevassa Qt-kehitysympäristössä. Opinnäytetyö on tehty PC-ohjelmistoja kehittävälle osastolle Nokia Oyj:n Salon toimipisteessä. Kyseinen osasto on nimeltään DSS (Device Sw Update and Security Solutions), joka työllistää n. 100 henkilöä, joista noin 30 % toimii alihankkijoina.

Vuonna 1996 norjalainen Trolltech toi markkinoille ohjelmistojen kehitysympäristö Qt:n. Nokia hankki Trolltechin omistukseensa vuonna 2008 monialustastrategiansa vahvistamiseksi. Qt-kehitysympäristö on pohjana kymmenissätuhansissa avoimen sekä kaupallisen puolen sovelluksissa. Qt-ohjelmistoympäristö tarjoaa ympäristön ja työkalut alustariippumattoman ohjelmiston tuottamiseen. (Qt Who we are 2011; Qt – Cross-platform application and UI framework 2011.)

Toisaalla Amerikan Yhdysvalloissa Snowbirdin hiihtokeskuksessa Utahissa helmikuussa 2001 kokoontui joukko henkilöitä, jotka edustivat useita erilaisia, tuolloin kevyt-menetelmiksi kutsuttuja, ketteriä (agile) ohjelmointikäytäntöjä. Tapaamisen lopputuloksena syntyi julistus, Agile Manifesto, jota pidetään ketterän kehityksen perusmääritelmänä. Ketteriä menetelmiä ovat esimerkiksi ASD (Adaptive Systems Development), Crystal-menetelmät, DSDM (Dynamic Systems Development Method), FDD (Feature Driven Development), Scrum ja XP (Extreme Programming). (Agile Alliance 2010; Agile Manifesto 2001; Ketterä ohjelmistokehitys 2010.)

Jatkuva integrointi, joka terminä on lähtöisin XP-prosessista, yhtenä sen 12 käytännöstä, on ohjelmistokehityskäytäntö, jossa ohjelmointitiimin jäsenet integroivat kirjoittamaansa koodia pääohjelmistoon säännöllisesti, vähintään päivittäin, mikä johtaa useisiin integrointeihin päivässä. Jokainen integrointi verifioidaan ohjelmiston automaattisella koostamisella (software build), joka

sisältää myös testauksen, jotta integrointivirheet saadaan mahdollisimman nopeasti kiinni. (Fowler 2006.)

Tämän opinnäytetyön tavoitteena on tutkia jatkuvan integroinnin soveltamista Qt-ohjelmointiympäristössä. Tämän opinnäytetyön rakenne koostuu seitsemästä osasta. Ensimmäisessä pääluvussa esitetään opinnäytetyön johdanto, jossa käsitellään työn tausta ja tavoitteet, esitellään tutkimusmenetelmä ja aineisto sekä opinnäytetyön rakenne. Toisessa pääluvussa esitellään jatkuva integrointi, sen periaatteet ja käytännöt sekä prosessi ja edut. Kolmannessa pääluvussa perehdytään Qt-ohjelmointiympäristöön. Neljännessä pääluvussa kuvaillaan jatkuvan integroinnin prosessia Qt-ohjelmointiympäristössä, tarvittavia työkaluja ja uuden toimintatavan käyttöönoton tuottamiin haasteisiin vastaamista. Viidennessä pääluvussa esitellään DSS:ssä käytössä oleva toimiva jatkuvan integroinnin prosessi sekä lyhyesti kehitysvaiheessa oleva Qt-ympäristön prosessi. Kuudennessa pääluvussa kuvataan jatkuvan integrointiympäristön pystytys koulun atk-laboratoriossa Qt-ohjelmointiympäristöön. Lopuksi esitetään tutkimuksen yhteenveto sekä keskeiset johtopäätökset.

Tutkimusmenetelmänä käytetään kirjallisuustutkimusta sekä empiiristä tutkimusta. Tutkimusaineistona käytetään jatkuvaa integrointia ja muutosvastarintaa käsittelevää kirjallisuutta. Qt:n osalta, kuten myös jatkuvan integroinnin laajemman taustatiedon tueksi, käytetään sähköisessä muodossa olevaa tietoa eli internetistä löytyviä artikkeleita ja sivustoja. Toimeksiantajaan liittyvä erityistieto on kerätty haastattelemalla ja oman aikaisemman työkokemuksen perusteella.

2 Jatkuva integrointi

Jatkuva integrointi on eräs ketteristä käytännöistä, tarkalleen ottaen yksi 12:sta Extreme Programming -menetelmän alkuperäisestä käytännöstä. Extreme Programming (XP) on puolestaan yksi ketteristä menetelmistä.

Jatkuva integrointi on ohjelmistokehityskäytäntö, jossa ohjelmointitiimin jäsenet integroivat kirjoittamaansa koodia pääohjelmistoon säännöllisesti, vähintään päivittäin, mikä johtaa useisiin integrointeihin päivässä. Jokainen integrointi verifioidaan ohjelmiston automaattisella koostamisella (software build), joka sisältää myös testauksen, jotta integrointivirheet saadaan mahdollisimman nopeasti kiinni. (Fowler 2006.)

Jatkuvaa integrointia käyttämällä on mahdollista muuttaa usein tunteja tai jopa päiviä kestävä integrointitapahtuma itsestään selväksi asiaksi, joka vain tapahtuu (Duvall et al. 2007, xix).

2.1 Jatkuvan integroinnin periaatteet ja käytännöt

Jotta jatkuvan integroinnin prosessia on mahdollista käyttää, täytyy tiettyjen edellytysten täyttyä. Seuraavassa on eroteltu lihavoiduilla väliotsikoilla avainkäytäntöjä, joita noudattamalla on mahdollista rakentaa tehokas jatkuvan integroinnin ympäristö. (Fowler 2006.)

Yksi ainoa versio pääohjelmistosta ja sen ylläpito versionhallintajärjestelmässä

Versionhallintajärjestelmä (version control repository) on ehdoton edellytys jatkuvalle integrointikäytännölle. Versionhallintajärjestelmä tarvitaan, jotta voidaan hallita koodimuutoksia ja muita toimivan pääohjelman tarvitsemia tiedostoja. (Duvall et al. 2007, 7.)

Versionhallintajärjestelmään pitäisi tallettaa kaikki tiedostot, joita tarvitaan toimivan ohjelman tuottamiseen, mutta ei niistä syntyviä koostettuja tuotteita.

Pääohjelmistosta pitäisi pyrkiä säilyttämään versionhallintajärjestelmässä yksi päähaara (mainline), ja välttää turhaa haaroittumista (branches). (Fowler 2006.)

Koostamisen automatisointi

Ohjelmiston koostamiseksi toimivaksi ohjelmaksi tarvitaan yleensä monimutkaista prosessia, johon kuuluu esimerkiksi koodin kääntäminen, tiedostojen siirtely, kaavioiden lataaminen tietokantaan jne. Prosessi on kuitenkin yleensä samanlainen joka kerta, jolloin se on syytä automatisoida. (Fowler 2006.)

Koostamisen automatisoinnilla saadaan vähennettyä manuaalisia, toistettavia ja virheille alttiita toimintoja. On kuitenkin tehtävä kompromissi sen suhteen, mitä kaikkea koostamiseen sisällytetään ja kuinka kauan koostaminen kestää. (Duvall et al 2007, 67.)

Koostamisskriptin (build script) pitäisi sallia kulloisenkin tarpeen mukainen koostaminen. Pitäisi olla mahdollista suorittaa koostaminen esimerkiksi testien kanssa, ilman testejä tai erilaisia testejä eri kerroilla sisältäen. (Fowler 2006.)

Kun koostamisprossessi on automatisoitu, se on valmis jatkuvaan integrointikäytäntöön (Duvall et al 2007, 6).

Automatisoitujen testien sisällyttäminen koostamisprosessiin

Automatisoidut koostamiseen sisältyvät testit mahdollistavat virheiden nopean ja tehokkaan kiinnisaamisen. Nämä testit pitää saada päälle yhdellä käskyllä ja niiden pitää olla statuksensa itse tarkastavia. Jos yksikin testi epäonnistuu, koko koostaminen on epäonnistunut. Testikattavuutta voidaan tutkia kattavuustyökaluilla, jollaisen voi liittää koostamisprosessiin. (Duvall et al 2007, 42; Fowler 2006.)

Kehittäjätiimin jokainen jäsen varmistaa päivittäin tekemiensä koodimuutosten ongelmattoman integroitumisen pääohjelmaan

Ainoa tapa tehokkaaseen jatkuvaan integrointikäytäntöön on säännöllinen koodimuutosten tallentaminen versionhallintajärjestelmään. Tämä pitää tehdä vähintään päivittäin, mieluummin vieläkin useammin. Ennen versionhallintajärjestelmään tallentamista kyseessä olevan muutoksen tekijän on varmistettava, että hänen tekemänsä muutos integroituu onnistuneesti pääohjelmaan, mikä tarkoittaa myös koostamistestien onnistunutta ajoa. Varmistaminen tapahtuu niin, että koodimuutoksen tekijä koostaa omalla koneellaan muutoksen pääohjelman viimeisimmän version kopioon. Jos tämä onnistuu ongelmitta, on muutos valmis integroitavaksi varsinaiseen pääohjelmaan. Mitä nopeammin muutos integroidaan pääohjelmaan, sitä nopeammin virheet havaitaan ja saadaan korjattua. Korjaaminen on aina helpompaa, kun korjauksen kohteena on juuri kirjoitettu koodi, jolloin muutos on tuoreessa muistissa. Ongelmat, joita ei havaita viikkoihin, voivat olla todella vaikeita paikallistaa. (Fowler 2006.)

Jokaisen koodimuutoksen pitää johtaa pääohjelmiston koostamiseen integrointikoneella

Päivittäinen koodimuutosten tallentaminen versionhallintajärjestelmään ja integrointi pääohjelmaan tekee pääohjelman koostamisen säännöllisesti testatuksi. Siitä huolimatta virheitä tulee erilaisista syistä, kuten inhimillisestä kurin puutteesta tai eroista ohjelmointiympäristössä kehittäjien välillä. Tästä syystä koostamisen pitää tapahtua säännöllisesti integrointikoostamiskoneella (integration build machine). (Fowler 2006.)

Integrointikoostamiskone isännöi CI-palvelinta (Continuous integration), joka säännöllisesti tarkistaa versionhallintajärjestelmää muutosten varalta (Duvall et al. 2007, 12). Integrointikoostamiskone toimii eräänlaisena turvaverkkona varmistaen ohjelmiston toimimisen niin kuin sen pitäisi toimia. Erillistä integrointikoostamiskonetta käytettäessä koostamisprosessi voidaan ajaa usein

ja koostamisympäristöstä tulee todella toistettava, mikä vähentää ympäristöstä johtuvia olettamuksia. (Duvall et al. 2007, 82.)

Koostaminen on pidettävä nopeana

Jatkuvan integroinnin prosessissa nopea palaute koostamisen onnistumisesta tai epäonnistumisesta on keskeinen elementti. Jotta kehitystyö ei hidastuisi palautteen odottamisen takia, on koostamisprosessi pidettävä nopeana. (Duvall et al. 2007, 87.)

XP:n ohjesäännön mukaan hyvä nyrkkisääntö koostamisen kestolle on 10 minuuttia. Siinä ajassa ei tietenkään ole mahdollista suorittaa kaikkia tarvittavia testejä ja tarkastuksia. Ratkaisuna tähän on vaiheittainen koostaminen (staged builds). Siinä tehdään peräkkäisiä koostamisia tarpeen mukaan. CI-palvelimen havaitsema muutos käynnistää aina ensimmäisen koostamisen, joka pidetään nopeana. Tämä tapahtuu tasapainoillen virheiden löytämisen ja nopeuden välillä niin, että ensimmäisen koostamisen tuottama ohjelmaversio on riittävän vakaa toisten kehittäjien jatkaakseen työtään. Ensimmäisen koostamisen jälkeen ajetaan toinen koostaminen, joka sisältää loput tarvittavat testit ja muut tarkastukset. Toinen koostaminen voi olla kestoltaan muutamia tuntejakin. Ensimmäistä koostamista seuraavia koostamisia voi lisätä tarpeen mukaan ja nopeuttaa ajamalla niitä esimerkiksi rinnakkain. (Duvall et al. 2007, 88; Fowler 2006.)

Testaus suoritetaan pääohjelman lopullista käyttöympäristöä vastaavassa ympäristössä

Testaaminen ohjelman loppukäyttöympäristöstä eroavassa ympäristössä voi johtaa virheisiin, jotka tapahtuvat testausympäristössä, mutta eivät loppukäyttöympäristössä, ja päinvastoin. Tästä johtuen on pyrittävä tekemään testausympäristö mahdollisimman loppukäyttöympäristöä vastaavaksi. Yksi vaihtoehto tämän toteuttamiseksi on käyttää virtualisoituja koneita virtualisoinnin kehittyessä koko ajan suorituskykyisemmäksi. (Fowler 2006.)

Kenen tahansa pitää olla helppo saada viimeisin ohjelmaversio käyttöönsä

Kaikilla ohjelmistoprojektiin kuuluvilla pitäisi olla mahdollisuus saada käyttöönsä viimeisin versio ohjelmasta ja ajaa se esimerkiksi demona, testauksen vuoksi tai vain nähdäkseen, mitä muutoksia siihen on tullut sitten viime version. Sitä varten on oltava paikka, josta kaikki tietävät sen löytävänsä. (Fowler 2006.)

Kaikkien pitää pystyä näkemään, mitä tapahtuu

Kommunikointi on tärkeää jatkuvassa integroinnissa. Kaikkien projektiin kuuluvien pitää helposti nähdä järjestelmän tila ja mitä muutoksia on tehty. Yksi tärkeimmistä kommunikoitavista asioista on pääohjelmakoosteen (mainline build) tila. Tämän seuraamiseksi voidaan käyttää vaikka laavalamppuja, vihreää ja punaista, jotka tilan lisäksi voivat kertoa, kuinka kauan ko. tila on vallinnut. (Fowler 2006.)

Käyttöönoton automatisointi

Jatkuvassa integroinnissa tarvitaan useita ympäristöjä, yhtä ensimmäistä koostamista varten, toista tai useampia seuraavia koostamisia varten. Silloin tarvitaan myös tiedostojen ja ohjelman siirtämistä ympäristöstä toiseen monta kertaa päivässä, mikä johtaa toiminnon automatisoinnin tarpeeseen. Samalla automatisoinnin tuomalla helppoudella pitäisi myös pystyä toimittamaan ohjelma loppukäyttäjälle. Loppukäyttäjän tapauksessa on oltava myös tapa palata vanhempaan versioon helposti. (Fowler 2006.)

Jos ohjelmaa ei ole julkaistu, tarkoittaa se melkein samaa kuin ohjelmaa ei olisi olemassa (Duvall et al. 2007, 191).

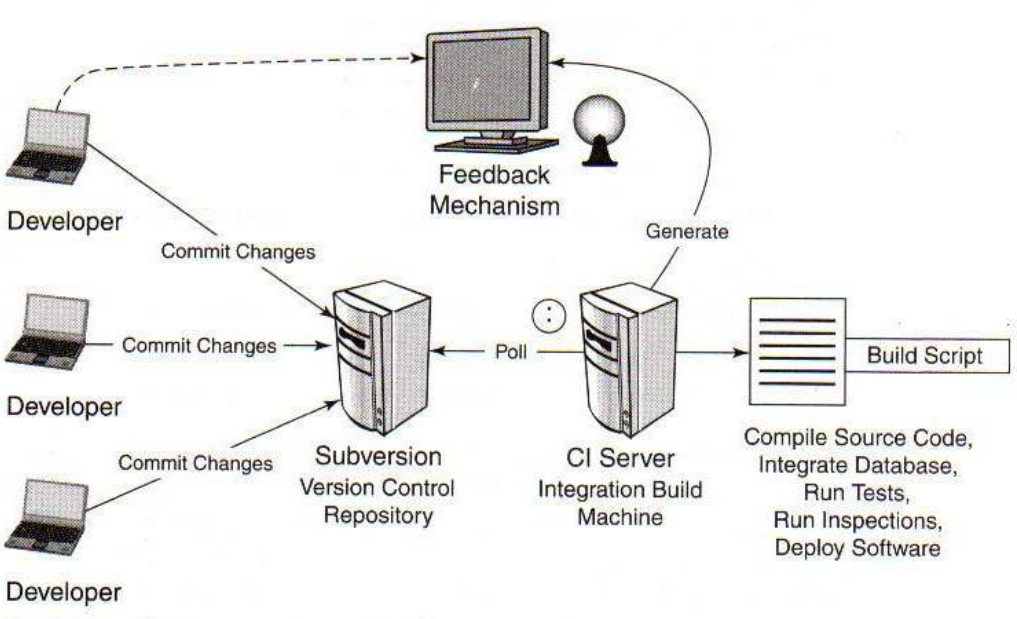
2.2 Jatkuvan integroinnin prosessi ja edut

Jotta jatkuva integrointi toimisi tehokkaasti, on projektin kuuluvien kehittäjien päivittäisiin työtapoihin kuuluttava tehtyjen koodimuutosten tallentaminen versionhallintajärjestelmään, toimimattomien koosteiden korjaaminen, automatisoitujen 100 % ajasta toimivien testien kirjoittaminen ja toimimattoman koodin versionhallintajärjestelmään laittamatta jättäminen tai sieltä ottaminen.

Jatkuvan integroinnin prosessi voidaan jakaa seuraaviin riittävän pienellä taajuudella toistettaviin vaiheisiin:

1. Ohjelmointitiimin jäsen tallentaa tekemänsä koodin tai koodimuutoksen versionhallintajärjestelmään. Samaan aikaan integrointikoostamiskoneella oleva CI-palvelin tarkistaa (poll) versionhallintajärjestelmää muutosten varalta muutaman minuutin välein.
2. Ohjelmointitiimin jäsenen tallennettua koodinsa versionhallintajärjestelmään CI-palvelin huomaa muutoksen, jolloin se noutaa versionhallintajärjestelmästä viimeisimmän version pääohjelmistosta ja ajaa koostamisskriptin (build script), joka integroi muutoksen pääohjelmistoon.
3. Integroinnin jälkeen CI-palvelin lähettää palautteena koostamistuloksen tietyille projektin jäsenille esimerkiksi sähköpostiviestinä.
4. CI-palvelin jatkaa muutosten tarkistamista versionhallintajärjestelmästä.

Edellä mainitut vaiheet esitetään kuvassa 1.



Kuva 1. Jatkuvan integroinnin järjestelmän komponentit(Duvall et al. 2007, 5).

Jatkuvan integroinnin prosessi tuottaa väistämättä seuraavia korkean tason etuja: riskien vähentäminen, toistettavien manuaalisten prosessien vähentäminen, toimivan ohjelmiston tuottaminen milloin tahansa, paremman näkyvyyden mahdollistaminen ja kehitystiimin suuremman luottamuksen vahvistaminen ohjelmistotuotteeseen (Duvall et al. 2007, 29).

Nämä edut esitellään seuraavaksi lihavoiduin väliotsikoin eroteltuina.

Riskien vähentäminen

Projektin riskejä voidaan vähentää integroimalla monta kertaa päivässä, mikä johtaa virheiden nopeaan kiinnisaamiseen, ohjelmiston laadun mittaamiseen ja olettamusten vähenemiseen. Virheet saadaan korjattua nopeammin, kun ne saadaan kiinni heti koodin tultua tallennetuksi versionhallintajärjestelmään. Koska jatkuva testaus ja katselmointi sisältyvät automaattiseen integrointiprosessiin, voidaan ohjelmiston laatua, kuten kompleksisuutta,

seurata koko ajan. Koostamalla ja testaamalla ohjelmistoa puhtaassa ympäristössä jatkuvasti samaa prosessia ja samoja skriptejä käyttäen voidaan vähentää olettamuksia. (Duvall et al. 2007, 29; Duvall et al. 2007, 30.)

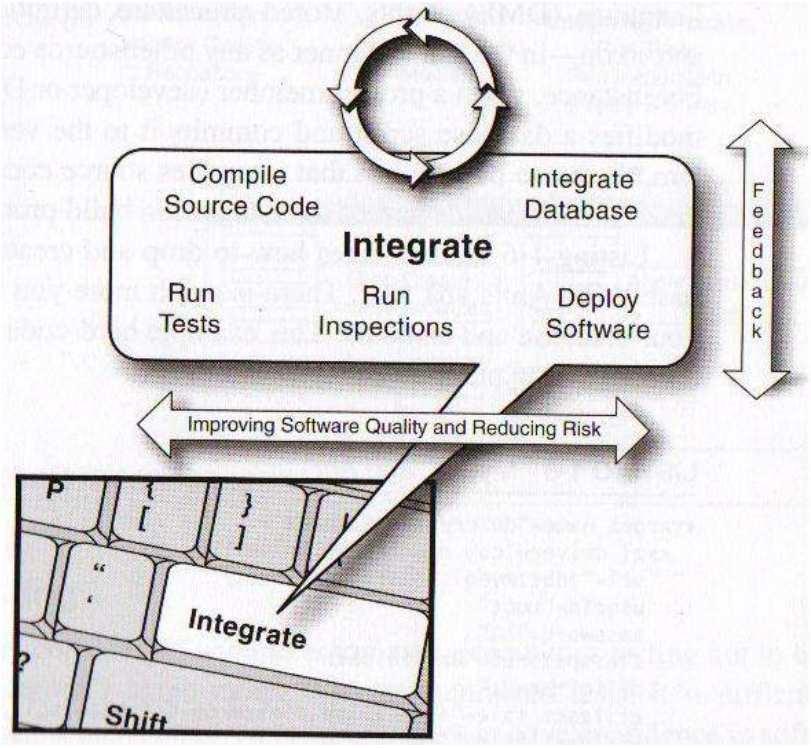
Toistettavien prosessien vähentäminen

Manuaalisesti toistettavien prosessien vähentäminen säästää aikaa, kustannuksia ja työtä. Automatisoidulla jatkuvan integroinnin prosessilla voidaan varmistaa prosessin ajo samalla lailla joka kerralla, määrätyn prosessin seuraaminen ja prosessin ajo aina, kun koodimuutos on tallennettu versionhallintajärjestelmään. (Duvall et al. 2007, 30.)

Toimivan ohjelmiston tuottaminen

Jatkuva integrointi mahdollistaa toimivan ohjelmiston tuottamisen minä tahansa ajan hetkenä. Tätä voisi pitää asiakkaan kannalta jatkuvan integroinnin tärkeimpänä etuna. Tämä on mahdollista, koska pienetkin koodimuutokset integroidaan ja testataan heti ja säännöllisesti, jolloin ongelmien ilmetessä korjaukset voidaan tehdä välittömästi. (Duvall et al. 2007, 31.)

Kuvassa 2 esitetään tiivistetysti toimivan ohjelmiston tuottaminen yhdellä komennolla niin kuin sen täysimittaisessa jatkuvan integroinnin ympäristössä tulisi tapahtua.



Kuva 2. Integrointinappula (Duvall et al. 2007, 13).

Paremman näkyvyyden mahdollistaminen

Koska jatkuvassa integroinnissa saadaan saman tien informaatiota juuri ajetun ohjelmistokoosteen tilasta ja laatumetriikasta, projektiin saadaan parempi näkyvyys. Jatkuvasta integroinnista johtuen on mahdollista nähdä myös koostamisen onnistumis- tai epäonnistumissuuntaukset. Yleisesti ottaen laatu ja muu olennainen projekti-informaatio on saatavissa. (Duvall et al. 2007, 31.)

Kehitystiimin suurempi luottamus ohjelmistotuotteeseen

Koska jatkuvassa integroinnissa järjestelmä informoi kehittäjiä heti virheen ilmetessä, on kehittäjien helpompaa tehdä muutoksia koodiin. Jatkuvassa integroinnissa kaiken ohjelmistokoostamisen pitäisi perustua yhteen lähteeseen, ja näin sen virheettömyyteen on suurempi luottamus. (Duvall et al. 2007, 32.)

3 Qt

Qt on alustariippumaton ohjelmistojen ja graafisten käyttöliittymien kehitysympäristö, joka sisältää alustariippumattoman luokkakirjaston ja ohjelmointiympäristön. Qt:llä voidaan kirjoittaa kerralla ohjelmistoja graafisine käyttöliittymineen ja käyttää niitä sitten monilla eri käyttöjärjestelmillä kirjoittamatta koodia uudestaan. Tällä hetkellä tuettuja käyttöjärjestelmiä ovat Embedded Linux, Mac OSX, Windows, Linux/X11, Windows Mobile, Windows CE, Symbian, Maemo ja MeeGo. (Qt – Cross-platform application and UI framework 2011.)

Qt-ohjelmointiympäristö on nimeltään Qt Creator, joka on siis alustariippumaton, integroitu kehitysympäristö. Se tarjoaa koodieditorit C++ ja JavaScript, integroidun käyttöliittymäsuunnittelijan, projekti- ja koostamisen hallinnan työkalut, gdb- ja CDB-debuggerit, tuen versionhallinnalle, simulaattorin mobiilikäyttöliittymille ja mahdollisuuden koostaa työpöytä- tai mobiililaitteissa käytettäviä ohjelmia. Nämä palvelut esitetään kuvassa 3.

Kuva 3. Qt Creatorin tarjoamat palvelut (Qt Reference Documentation 2011).

Alustariippumaton ohjelmistokehitys

Yksi Qt Creatorin suurimmista eduista on mahdollisuus yhdellä työkalulla kehittää ja testata (debug) samaa projektia eri käyttöliittymäympäristöissä. Eri käyttöliittymäympäristöille on mahdollista määritellä erilliset koostamisasetukset. Erillisiä projektitiedostoja voidaan käyttää pitämään alustariippuvaiset koodit erillään. Qt-kirjastojen lisäksi on mahdollista yhdistää tekeillä oleva oma ohjelma myös muihin kirjastoihin, kuten järjestelmäkirjastoihin tai omiin kirjastoihin. Jos omat kirjastot riippuvat vielä muista kirjastoista, pitää ne lisätä projektiin. Kirjastojen lisäämisprosessi riippuu käytössä olevasta koostamisjärjestelmästä. (Qt Reference Documentation 2011.)

Projektit

Qt Creator tarvitsee samat tiedot kuin kääntäjä tarvitsisi koostaakseen ja ajaakseen ohjelmaa. Tätä varten pitää luoda projekti, jonka koostamis- ja ajoasetuksissa tarvittavat tiedot on määritelty. Projektin luominen mahdollistaa tiedostojen ryhmittelyn, haluttujen koostamisvaiheiden lisäämisen, muodollisuuksien ja resurssitiedostojen lisäämisen ja asetusten määrittelyn ajettaville ohjelmille. Projekti voidaan luoda tyhjästä tai jo olemassa olevasta projektista. Qt Creator luo kaikki tarvittavat tiedostot luotavan projektin tyypistä riippuen. Qt Creatorin sisältämät qmake ja CMake mahdollistavat automatisoidun alustariippumattoman koostamisen. (Qt Reference Documentation 2011.)

Editorit

Qt Creator sisältää koodieditorin ja kaksi integroitua visuaalista editoria. Koodieditori ymmärtää C++ ja QML -kieliä ei pelkästään tekstinä, vaan koodina, mikä mahdollistaa hyvin muotoillun koodin kirjoittamisen, ennakoinnin ja koodin valmiiksi muotoilemisen, avointen virhe- ja varoitusviestien näyttämisen, semanttisen navigoinnin luokkien, funktioiden ja symbolien välillä ja niihin liittyvän avun saatavuuden, symbolien älykkään uudelleennimeämisen ja funktion määritelmä- ja kutsukohtien näyttämisen koodissa. QML on JavaScript-kielen laajennos.

Visuaaliset editorit ovat Qt Designer ja Qt Quick Designer. Ensin mainittu on tarkoitettu graafisten käyttöliittymien suunnitteluun ja koostamiseen Qt-mokkuloiden pohjalta. Qt Quick Designer puolestaan on tarkoitettu animaatioiden kehittämiseen QML-kieltä käyttäen. Siinä käyttöliittymä määritellään puuna, jossa on objekteja, jotka sisältävät ominaisuuksia. (Qt Reference Documentation 2011.)

Kohteet

Qt Creatorilla voidaan luoda Qt-ohjelmia sekä työpöytäympäristöön (Windows, Linux ja MacOS) että mobiililaitteisiin (Symbian, Maemo ja MeeGo). Koostamisasetuksilla voidaan kohdetta vaihtaa nopeasti. Koostamiskohtaisten tiedostojen pitämiseen erillään lähdekoodista Qt Creator käyttää oletuksena varjokoosteita (shadow builds). Alustariippuvaisten koodien erillään pitämiseksi kehittäjä voi käyttää erillisiä versioita projektitiedostoista. (Qt Creator Whitepaper 2011, 6.)

Työkalut

Qt Creator sisältää tarvittavia työkaluja kuten versionhallintajärjestelmä ja Qt Simulator. Qt Creator käyttää versionhallintajärjestelmän komentoriviasiakasohjelmia päästäkseen käsiksi tietosäilöön. Tuettuja versionhallintajärjestelmiä ovat Git, Subversion, Perforce, CVS, Bazaar ja Mercurial. Käytettävissä olevat Qt-funktiot riippuvat käytetystä versionhallintajärjestelmästä. Perusfunktiot, kuten tiedostojen vertaaminen viimeisimpiin versioihin ja eroavaisuuksien näyttäminen, versiointihistorian ja muutosten yksityiskohtien katsominen, lisähuomautustiedostot ja muutosten tallentaminen ja paluu vanhaan versioon, ovat saatavilla kaikille tuetuille järjestelmille.

Qt Simulatoria käytetään mobiililaitteisiin tarkoitettujen ohjelmien testaamiseen. Qt Simulator tulee Nokia Qt SDK:n mukana. (Qt Reference Documentation 2011.)

Debuggerit

Qt Creator ei sisällä debuggeria, vaan se tarjoaa debuggeriliitännäisen (plug-in), joka toimii rajapintana Qt Creatorin ytimen ja ulkoisten natiividebuggereiden välillä. Ulkoisia natiividebuggereita ovat GNU Symbolic Debugger (gdb), Microsoft Console Debugger (CDB) ja JavaScript debugger.

Qt Creatorin Debug-moodia voidaan käyttää sovelluksen tilan tarkkailuun debugattaessa. Käytössä ovat esimerkiksi toiminnot, kuten eteneminen ohjelman läpi rivi tai ohje kerrallaan, ohjelman ajon pysäyttäminen haluttuun kohtaan (breakpoint), paikallisten ja globaalien muuttujien rekistereiden ja muistin sisällön tutkiminen ja muokkaaminen. (Qt Reference Documentation 2011.)

4 Jatkuva integrointi Qt-ympäristössä

Jatkuvan integroinnin tärkeys korostuu Qt-ympäristössä, joka on monialustaympäristö, koska ohjelman kääntyvyys ja toimivuus on tarkistettava joka kerta monella eri alustalla. Jatkuvan integrointiympäristön pystyttämisestä Qt-ympäristöön ei juuri löydy esimerkkejä. Toimivan prosessin aikaansaamiseksi tarvitaan erilaisia keskenään yhteensopivia työkaluja esimerkiksi vaatimusten hallintaan, versionhallintaan, eri testausvaiheisiin, itse ohjelmointiin ja jatkuvaan integrointiin. Olemassa oleva Qt-kehitysympäristö on lähtökohtana työkalujen valinnalle.

Työkaluvalintojen lisäksi uuden toimintatavan käyttöön ottamisessa ehkä vielä suurempana haasteena on muutoksen aikaan saaminen henkilöstön toimintatavoissa ja asenteissa.

4.1 Tarvittavat työkalut

Kun jatkuvan integroinnin ympäristöä ollaan pystyttämässä jo käynnissä olevaan projektiin, on työkaluvalintojen lähtökohdaksi otettava olemassa oleva työkaluvalikoima.

Työkalut koodin kehitysprosessissa ohjelmoinnista julkaisuun

Tässä opinnäytetyössä kyseessä olevassa tapauksessa Qt-kehitysympäristö on jo käytössä. Ohjelmiston koostamis–testaus–käyttöönotto -prosessissa käytössä on koostamisen automatisoinnin työkalu Electric Commander, joka on kaupallisen puolen työkalu. Yksi avoimen puolen vastaavista työkaluista on Jenkins (aiemmin Hudson), joka osalla projekteista on käytössä. Eri vaiheissa tapahtuvaan testaukseen on käytössä useita erilaisia työkaluja. (Electric Cloud 2011; Jenkins 2011; Hudson 2011.)

Yksikkötestauskäytössä on NUnit-niminen avoimen lähdekoodin työkalu, joka on tarkoitettu .NET-koodille. Qt-kehitysympäristö sisältää oman yksikkötestaustyökalunsa QTestLib:in. Yksikkötestauksen testikattavuuden

tutkimiseen käytössä olevia työkaluja ovat .NET-puolella NCover ja C++-puolella BullseyeCoverage, jotka molemmat ovat kaupallisen puolen työkaluja. Koodin kompleksisuuden mittaamiseen käytetään SourceMonitor-työkalua, joka on ilmaisohjelma. (NUnit 2011; NCover 2011; Bullseye 2011; Campwood Software 2011.)

Käyttöliittymätestauksessa on käytössä kaksi eri työkalua. Alustariippumattoman käyttöliittymätestauksen työkalu on Squish, joka on tarkoitettu juuri Qt-ympäristön käyttöön, ja jota käytetään graafisen käyttöliittymän toiminnallisen regressiotestauksen automatisointiin. Windows- ja Web-sovellusten käyttöliittymätestauksen automatisoinnin työkaluna käytössä on Ranorex. Sekä Squish että Ranorex ovat kaupallisen puolen työkaluja. (Froglogic 2011; Ranorex 2011.)

Versionhallinta

Kehitysympäristön ja koostamis–testaus–käyttöönottoprosessin lisäksi ehdoton välttämättömyys jatkuvan integroinnin ja yleensäkin laadukkaan ohjelmistotuotannon kannalta on versionhallintajärjestelmä. Versionhallintaa varten on tarjolla useita keskenään samankaltaisia työkaluja sekä kaupallisella että avoimen lähdekoodin puolella. Sekä Qt, Electric Commander että Hudson asettavat rajoitteita valittavissa oleville versionhallintatyökaluille. Tässä tapauksessa on päätetty keskittyä kolmeen mahdolliseen vaihtoehtoon, Mercurialiin ja Gitiin, jotka molemmat ovat avoimen lähdekoodin järjestelmiä ja joiden molempien kehitystyö alkoi toisistaan muutaman päivän erotuksella vuonna 2005 sekä Synergyyn, joka on kaupallisen puolen järjestelmä, ja tällä hetkellä vielä käytössä suurelta osin toimeksiantajan projekteilla.

Mercurial on vapaan lähdekoodin hajautettu versionhallintajärjestelmä. Se on kirjoitettu pääosin Pythonilla, ja pieni osa c-kielellä, mistä johtuen binäärijulkaisut ovat saatavilla kaikille käytetyimmille alustoille. Qt Creatorin IDE (integrated development environment) sisältää tuen Mercurialin käytölle suoraan Qt Creatorista. Mercurial asentuu yhtenä binäärinä. Perusasennus on helppokäyttöinen ja vaikeasti rikottava. Lisätoimintoja, esimerkiksi

edistyneemmille käyttäjille luokiteltavia toimintoja, saa mukaan erillisillä Mercurialin mukana tulleilla virallisilla laajennoksilla, jotka voi aktivoida käyttöön. Laajennoksia voi myös ladata wikistä tai kirjoittaa ihan omia. (Mercurial source control management 2011; Thomson 2008.)

Kuten Mercurial myös Git on vapaan lähdekoodin hajautettu versionhallintajärjestelmä. Alun perin Linus Torvalds suunnitteli ja kehitti Gitin Linux kernelin kehitystyötä varten. Nykyään Gitiä ylläpitää Junio Hamano. Toisin kuin Mercurial, Git ei asennu yhtenä binäärinä, vaan se on ennemminkin suuri joukko erillisiä työkaluja. Se on kirjoitettu c-kielellä, Bourne Shellillä ja Perlillä ja se on saatavilla POSIX- ja Windows-alustoille. Toimiakseen Windowsilla Git vaatii joko msysgitin tai Cygwinin asentamista. Msysgit on natiivi Microsoft Windows -portti, jolla Git toimii Windowsissa vähän hitaammin kuin Linuxilla. Cygwin puolestaan on POSIX-emulaattorikerros. Samoin kuin Mercurialin kohdalla, Qt Creatorin IDE sisältää tuen Gitin käytölle suoraan Qt Creatorista. (Git the fast version control system 2011.)

Telelogic Synergy on kaupallisen puolen versionhallintatyökalu, jonka juuret juontavat vuoteen 1988 Computers West yhtiöön Kaliforniaan. Synergy-nimi tuli kuvioihin vasta 2000-luvulla, ja vuodesta 2008 asti Synergy on ollut IBM Rationalin omistuksessa. Synergy on globaalin ohjelmistokehityksen tehtäväperusteinen (task based) integroitu versionhallintajärjestelmä. Se on saatavilla Linuxille, Unixille ja Windowsille. (Rational Synergy 2011.)

4.2 Haasteiden hallinta uuden toimintatavan käyttöönotossa

Kun muutetaan jo olemassa olevaa prosessia, ihmisellä tyypillisesti ilmenee muutosvastarintaa. Muutoksen toteuttamisen tietä voidaan tasoittaa riittävällä perehdytyksellä uuteen toimintatapaan. Pekka Mattilan (2008) mukaan onnistuneen muutoksen johtaminen ja eteenpäin viemisen avaintehtävät voidaan jakaa neljään vaiheeseen:

1. perustan luominen
2. käynnistystoimet
3. hallittu eteneminen
4. vakiinnuttaminen

Ensimmäisen vaiheen perustan luomisessa tarkoituksena on luoda kokonaiskuva muutoksesta ja sen vaikutuksista, pohtia kyseessä olevan työyhteisön lähtötilannetta ja siihen liittyviä riskejä, tiivistää muutoksen visio ja perusteet ja konkretisoida muutokselle asetetut tavoitteet.

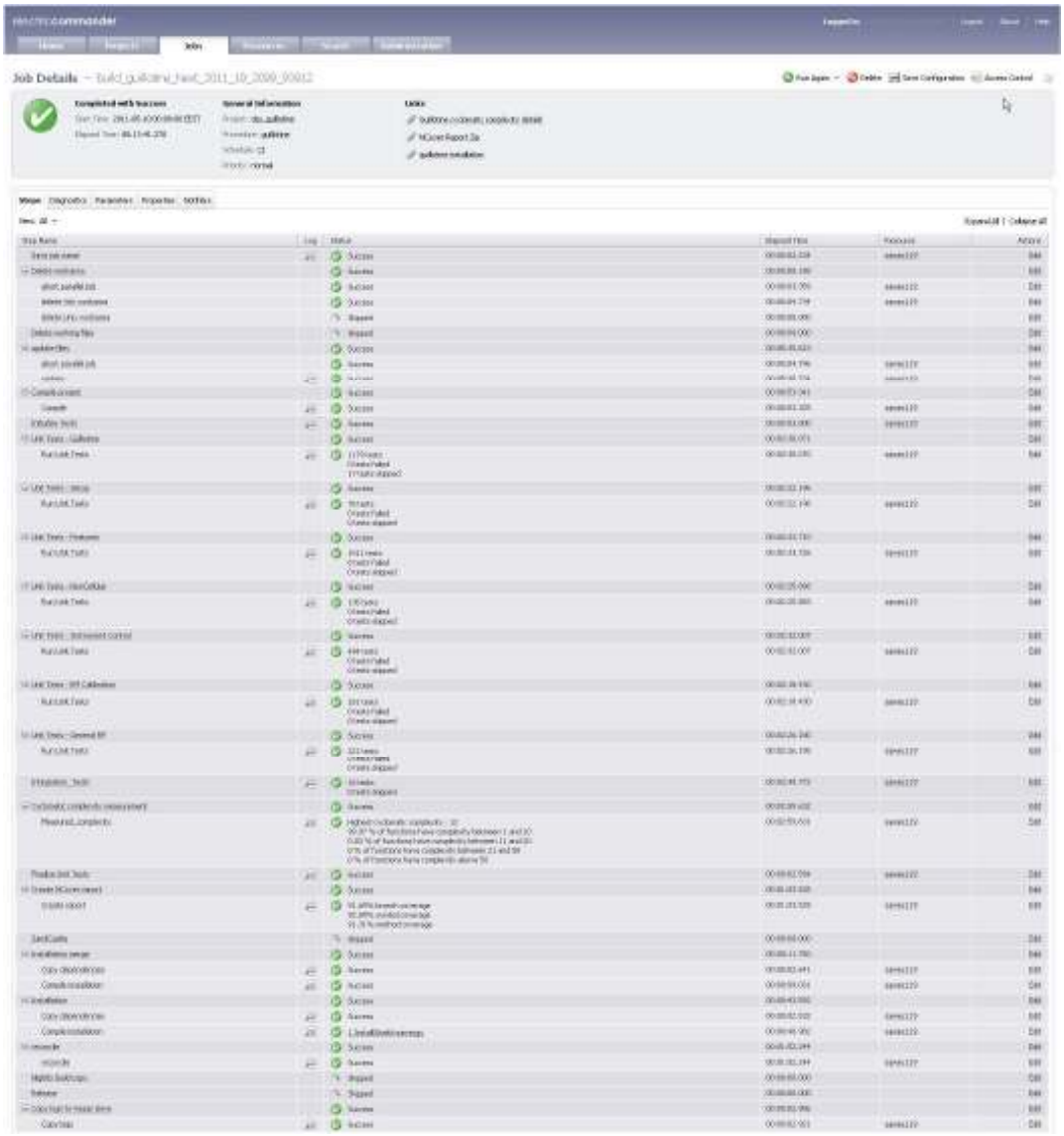
Toisen vaiheen käynnistystoimien tarkoituksena on varmistaa sujuva liikkeellelähtö kaikilla halutuilla rintamilla, luoda innostusta ja sitoutumista sekä sulattaa muutosvastarintaa näin heti alkuvaiheessa. Alkuvaihe on syytä hoitaa ripeästi ja määrätietoisesti, jotta voitetaan aikaa mahdollisten myöhemmin ilmenevien ongelmien varalta ja jotta saadaan luotua uskottavuutta uudelle suunnalle.

Hallitun etenemisen vaiheessa hoidetaan suurin osa muutosjohtamisen tehtävistä ja arkipäivästä. Tyypillisiä tämän vaiheen ilmentymiä ovat uusien alkavien tehtävien ja päättyvien vaiheiden limittyminen, ensimmäisten onnistumisien saavuttaminen sekä tarvittavien korjaavien toimenpiteiden tekeminen.

Vakiinnuttamisen vaiheessa vasta käyvät todeksi muutoshankkeen hyödyt ja lopputulos kirkastuu. On tärkeä olla hellittämättä liian aikaisin. (Pekka Mattila 2008, 50).

5 Jatkuvan integroinnin prosessi DSS:ssä

Opinnäytetyön toimeksiantajan DSS:n (Nokia Oyj.) sisällä tuotetaan satoja pieniä projekteja, joita koostetaan tällä hetkellä kolmella eri työkalulla projektista riippuen. Uudempia useammin koostettavia projekteja koostetaan Electric Commanderilla. Joillakin projekteilla on käytössä avoimen puolen ohjelma Hudson. Vanhempia, lähinnä ylläpitovaiheessa olevia projekteja koostetaan AutoBuilderilla, joka on DSS:n omaa tuotantoa kymmenen vuoden takaa. Osalla DSS:n projektiryhmistä on jo käytössä jatkuva integrointi. Esimerkki todellisen toimivan jatkuvan integroinnin prosessin vaiheista esitetään kuvassa 4.



Kuva 4. Quillotine-projektin koostamisprosessi.

Koostamisprosessi alkaa, kun on havaittu versionhallintajärjestelmään (tässä tapauksessa Synergyyn) tallennettu muutos. Koostaminen tapahtuu Electric Commanderilla. Aluksi puhdistetaan palvelinkoneen työalue (Delete workarea) mahdollisista vanhoista versioista, jonka jälkeen päivitetään palvelimelle versionhallintajärjestelmästä viimeisin versio koostettavasta projektista (Update files). Samassa yhteydessä estetään kyseisen projektin rinnakkaisajo (abort

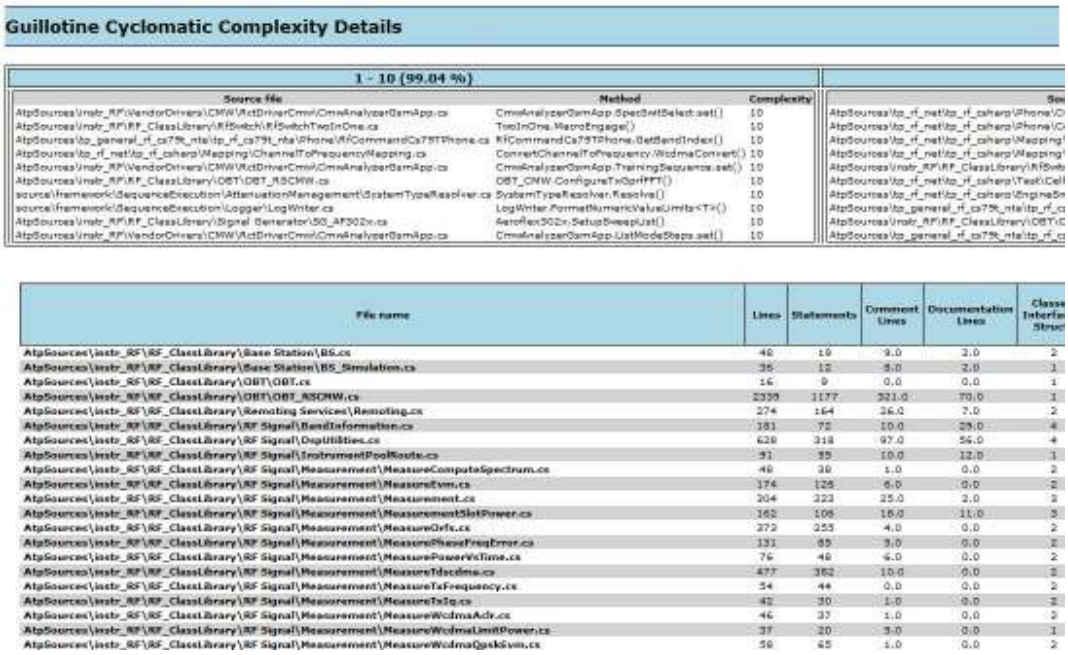
parallel job). Tämän jälkeen suoritetaan varsinainen käännös (Compile) ja alustetaan testit ja testikattavuuden mittaus (Initialize tests).

Yksikkötestejä ajetaan (Unit Tests) kaikille komponenteille erikseen, tässä esimerkkitapauksessa 3266 kappaletta, minkä jälkeen suoritetaan vielä integrointitestejä 16 kappaletta (Integration Tests). Testien aikana suoritetaan myös koodin kompleksisuusmittaus (Cyclomatic complexity measurement). Mitä pienempi kompleksisuusluku, sitä parempi. Yksikkötestien viimeistelyn (Finalize Unit Tests) jälkeen luodaan raportti testikattavuudesta (Create NCover report).

Projektista generoidaan (SandCastle) ohjetiedosto (helpfile), luodaan asennusmoduuli (Installation merge) ja erillinen (standalone) asennuspaketti (Installation). Asennuspaketin luomisen jälkeen palautetaan käännetyt binäärit takaisin versionhallintajärjestelmään (Reconcile) muiden projektien käytettäväksi. Asennuspaketti ja ohjetiedosto tallennetaan tunnettuun paikkaan, josta kaikki tarvitsijat tietävät ne löytävänsä (Nightly build copy). Lopulta päivitetään projektin tila versionhallintajärjestelmässä julkaistuksi (Release) ja jaetaan koostamisen logit asiaankuuluville tahoille nähtäväksi (Copy logs to Vegas drive).

Projektilla on käytössään ns. laavalamppu, joka kertoo liikennevaloväreillä näkyvästi koostamisen tilan. Electric Commanderin koostetun projektin sivulle luodaan myös suorat linkit kompleksisuusmittauksen yksityiskohtaiseen raporttiin (Guillotine cyclomatic complexity details) (kuva 5), testikattavuusraporttiin (NCover Report Zip) ja asennuspakettiin (quillotine installation).

Edellä kuvattu todellinen koostamisprosessi on .NET-projekti, jota kehitetään jatkuvan intgroinnin menetelmällä.



Guillotine Cyclomatic Complexity Details

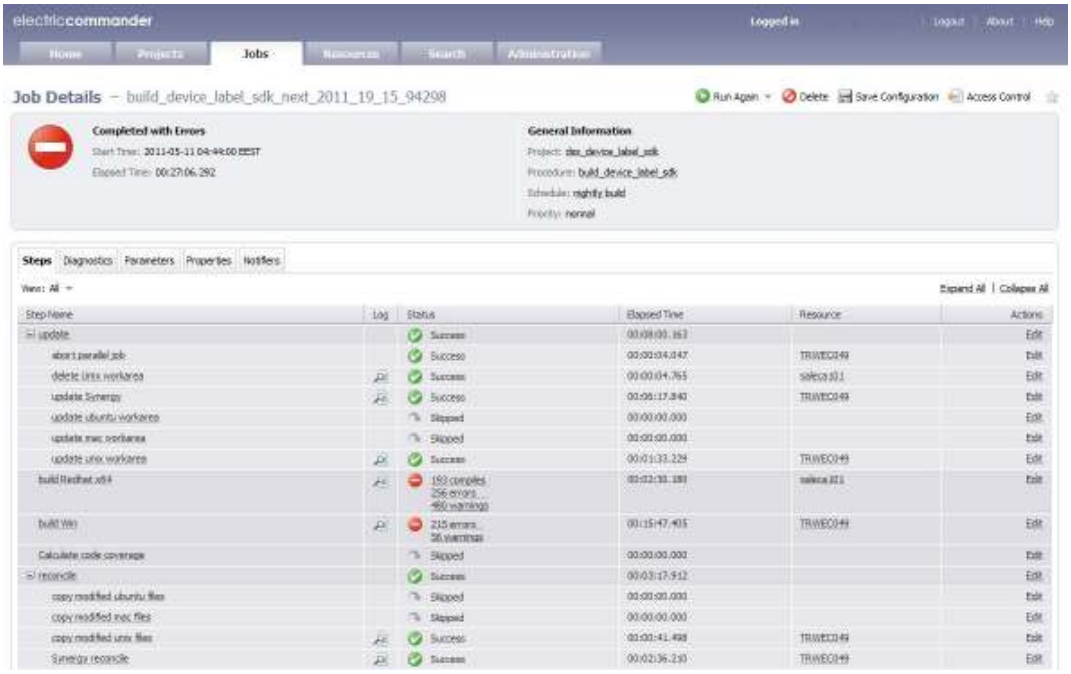
1 - 10 (99.04 %)

Source file	Method	Complexity
AtpSources\instr_RF\VendorDrivers\CMW\RctDriverCmw\CmwAnalyzerSamApp.cs	CmwAnalyzerSamApp.SpecSetSelect.set()	10
AtpSources\instr_RF\RF_ClassLibrary\RfSwitch\RfSwitchTwoInOne.cs	TwoInOne.MacroEngage()	10
AtpSources\tp_general_rf_cs79_nta\tp_rf_cs79_nta\Phone\RfCommandCs79TPhone.cs	RfCommandCs79TPhone.GetBandIndex()	10
AtpSources\tp_rf_net\tp_rf_csharp\Mapping\ChannelToFrequencyMapping.cs	ConvertChannelToFrequency.WcdmaConvert()	10
AtpSources\instr_RF\VendorDrivers\CMW\RctDriverCmw\CmwAnalyzerSamApp.cs	CmwAnalyzerSamApp.TriningSequence.set()	10
AtpSources\instr_RF\RF_ClassLibrary\OBT\OBT_RSCMW.cs	OBT_CMW.ConfigureTxQorFFT()	10
source\framework\SequenceExecution\AttenuationManagement\SystemTypeResolver.cs	SystemTypeResolver.Resolve()	10
source\framework\SequenceExecution\Logger\LogWriter.cs	LogWriter.FormatNumericValueLimits<T>()	10
AtpSources\instr_RF\RF_ClassLibrary\Signal Generator\SG_AF902x.cs	AeroFlex902x.SetupSweepList()	10
AtpSources\instr_RF\VendorDrivers\CMW\RctDriverCmw\CmwAnalyzerSamApp.cs	CmwAnalyzerSamApp.ListModeSteps.set()	10

File name	Lines	Statements	Comment Lines	Documentation Lines	Classe Interfac Struc
AtpSources\instr_RF\RF_ClassLibrary\Base Station\BS.cs	48	19	9.0	2.0	2
AtpSources\instr_RF\RF_ClassLibrary\Base Station\BS_Simulation.cs	36	12	8.0	2.0	1
AtpSources\instr_RF\RF_ClassLibrary\OBT\OBT.cs	16	9	0.0	0.0	1
AtpSources\instr_RF\RF_ClassLibrary\OBT\OBT_RSCMW.cs	2339	1177	321.0	70.0	1
AtpSources\instr_RF\RF_ClassLibrary\Remoting Services\Remoting.cs	274	164	26.0	7.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\BandInformation.cs	181	72	10.0	29.0	4
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\DspUtilities.cs	628	318	97.0	56.0	4
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\InstrumentPoolRoute.cs	91	35	10.0	12.0	1
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureComputeSpectrum.cs	48	30	1.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureEvm.cs	174	126	6.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\Measurement.cs	304	223	25.0	3.0	3
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasurementSlotPower.cs	162	106	16.0	11.0	3
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureOnTs.cs	272	255	4.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasurePhaseFreqError.cs	131	65	3.0	0.0	1
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasurePowerVsTime.cs	76	48	6.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureTdscdma.cs	477	362	10.0	0.0	1
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureTxFrequency.cs	54	44	0.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureTxIq.cs	42	30	1.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureWcdmaAclr.cs	46	37	1.0	0.0	2
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureWcdmaLimitPower.cs	37	20	3.0	0.0	1
AtpSources\instr_RF\RF_ClassLibrary\RF Signal\Measurement\MeasureWcdmaQpskEvm.cs	58	45	1.0	0.0	2

Kuva 5. Ote kompleksisuusmittauksen yksityiskohtaisesta raportista.

Qt-projektin jatkuvan integroinnin prosessi on vielä kehitysvaiheessa. Esimerkki tämän hetkisestä tilanteesta esitetään kuvassa 6. Qt-projektissa tarkoituksena on tässä vaiheessa Linux- ja Windows-versioiden koostaminen samanaikaisesti.



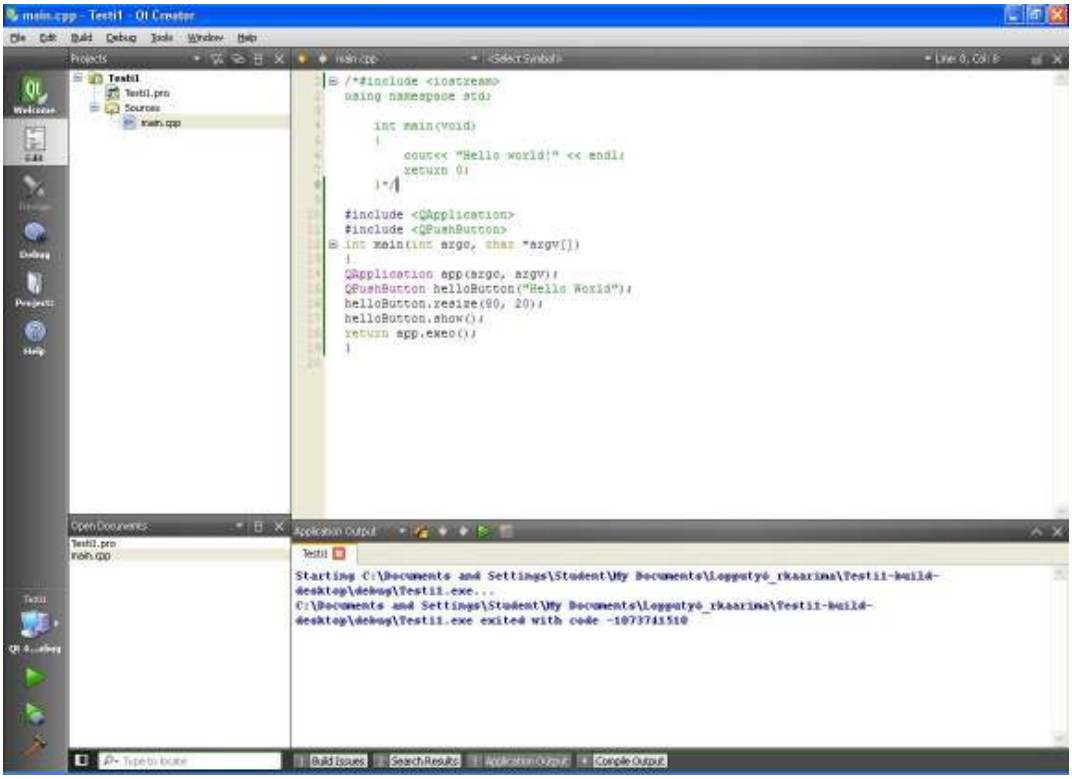
Kuva 6. Qt-projektin koostamisprosessi.

6 Jatkuvan integrointiympäristön pystytys Qt-kehitysympäristöön

Koska opinnäytetyön toimeksiantajan taholla ei kyetty verkko-oikeuksien vaikean saatavuuden vuoksi tässä opinnäytetyön tekoajassa jatkuvaa integrointiympäristöä Qt-kehitysympäristöön pystytttämään, oli teorian toimivuus tarpeen osoittaa toimivaksi tai toimimattomaksi koulun atk-laboratoriossa. Koulun atk-laboratorion koneissa on irrotettavat kovalevyt, jollaisen yhden sain käyttööni. Kovalevyllä oli valmiiksi asennettuna Windows XP-käyttöjärjestelmä. Koeympäristön Qt-kehitysympäristön jatkuvan integroinnin koostamisen työkaluksi valittiin Jenkins ja versionhallintatyökaluksi Git. Kaikki edellä mainitut ovat saatavilla maksutta.

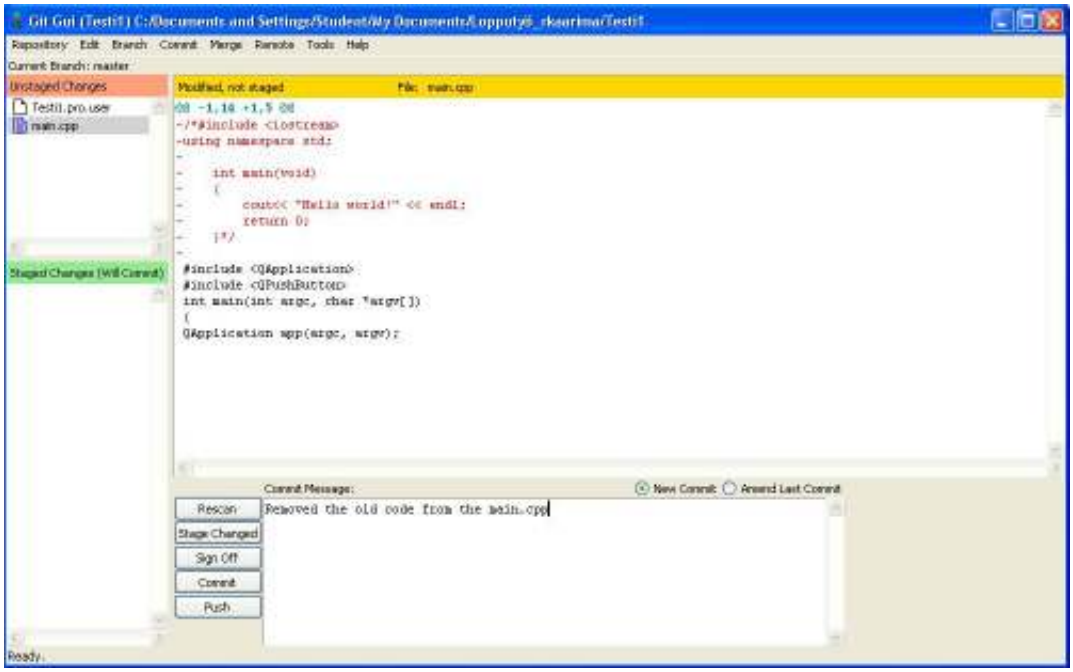
Ympäristön pystytys aloitettiin lataamalla internetistä Jenkins (https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins), Git (http://git-scm.com/) ja Qt (http://qt.nokia.com/downloads). Jenkins asennettiin käytämällä Jenkinsin Test Driveä, joka teki kaiken tarvittavan, jotta Jenkins Kojelaudan (Dashboard) voi käynnistää osoitteessa http://localhost:8080. Tämän jälkeen Hallitse Jenkinsiä (Manage Jenkins) -linkin kautta valittiin Hallitse Liitännäisiä (Manage Plugins) -linkki, jonka Saatavilla (Available) -välilehdeltä valittiin ja asennettiin Git Plugin, joka sisällyttää Git:in Jenkinsiin. Liitännäisen asennuksen jälkeen Jenkins piti käynnistää uudelleen.

Qt SDK versio 1.1.1:n lataus kesti noin kaksi tuntia. Asennuksen tultua valmiiksi Qt Creatorilla kirjoitettiin pieni c++-konsoliohjelma (kuva 7) ja käännettiin se.



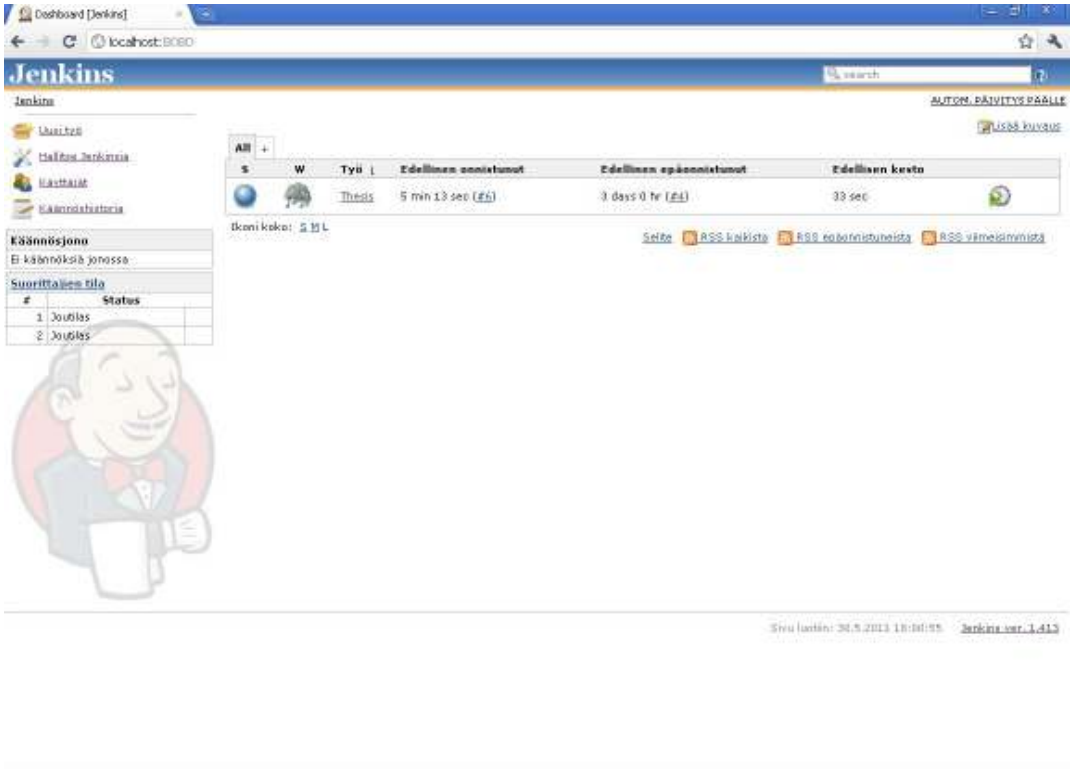
Kuva 7. C++-testiohjelma Qt-ympäristössä.

Git:in asennuksessa piti valita vain käyttöjärjestelmän mukainen asennuspaketti, jota käyttäen asennus sujui ongelmitta. Git-käyttöliittymässä (kuva 8) luotiin uusi tallennuspaikka (repository) edellä mainitun c++-konsoliohjelman tallentamiseksi versionhallintajärjestelmään, minkä jälkeen tallennettiin tehty pieni koodimuutos.



Kuva 8. Git-käyttöliittymä.

Seuraavaksi Jenkinsissä (kuva 9) luotiin Uusi työ (New Job), jonka asetukset säädettiin osoittamaan edellä mainittuun c++-konsoliohjelmaprojektiin. Asetuksissa piti valita Git-versionhallintajärjestelmä ja antaa kyseessä olevan projektin URL-osoite. Lisäksi valittiin koostamisen laukaisin. Valittavissa olivat vaihtoehdot koostaminen toisten projektien jälkeen, koostaminen aikataulutetusti tai koostaminen versionhallintajärjestelmää tarkastelemalla (poll). Myös koostamisen jälkeiset toimenpiteet saa säädettyä, esimerkiksi koostamistuloksen lähettämisen sähköpostilla haluttuun osoitteeseen. Jotta koostaminen Jenkinsistä käsin onnistuisi, oli vielä Jenkinsin asetuksissa säädettävä polku osoittamaan oikein Git-ajotiedostoon.



Kuva 9. Jenkins-ikkuna.

Tämän jälkeen kaikki olikin valmista ensimmäistä koostamista varten. Käännä nyt (Build Now) -komennolla käynnistettiin koostaminen, jonka lopputuloksena saatiin koostamisen status, joka kertoo tehdyt muutokset, koostamisen käynnistäjän ja Git:in koostamistiedot (kuva 10). Näin melko yksinkertaisesti on pystytettävissä jatkuvan integroinnin ympäristö maksuttomilla ohjelmilla, ainakin pienimuotoisesti. Se oli jopa vähän yllättävää. Aihetta saisi saman tien laajennettua testejä lisäämällä tai isomman ja useamman ohjelmistokehitysprojektin lisäämisellä, mikä varmasti kasvattaisi haasteita toimivan jatkuvan integroinnin ympäristön aikaansaamisessa.

Kuva 10. Koostamistiedot.

7 Yhteenveto

Integrointi on yksi ohjelmistokehitysprosessin kriittisimmistä vaiheista. Ohjelmistokehitysprojektien koko ajan kasvaessa on pitänyt kehittää ns. perinteisempien kankeaksi käyneiden ohjelmistokehitysmallien ja erityisesti integrointistrategioiden vaihtoehdoksi ketterämpiä menetelmiä. Erilaiset ohjelmointiympäristöt asettavat myös omat vaatimuksensa integroinnille. Tässä opinnäytetyössä perehdyttiin yhteen kasvavaan trendiin, jatkuvaan integrointiin, ja erityisesti Qt-ohjelmointiympäristössä. Tavoitteena oli tuoda esille jatkuvan integrointikäytännön etuja ja sen kyseessä olleelle ohjelmointiympäristölle asettamia vaatimuksia, unohtamatta myöskään jatkuvan integrointikäytännön käyttöönoton tuomia haasteita.

Tässä melko suppeassakin katsauksessa kävivät jo ilmi jatkuvan integrointikäytännön tarjoamat edut. Havaittiin myös, että tarjolla oleva työkaluviidakko on varsin kirjava. Jokaiseen ohjelmistokehityksen vaiheeseen löytyy työkaluja sekä kaupalliselta että avoimen lähdekoodin puolelta. On vain osattava valita omaan prosessiin sopivin keskenään yhteen toimiva työkaluvalikoima. Lähdettäessä kehittämään jo olemassa olevaa prosessia työkaluvalintojakin suuremmaksi haasteeksi muodostuu helposti henkilöstön asenteiden muokkaaminen muutokselle suotuisaksi.

Kunhan käyttöönotto valmistellaan hyvin koko ohjelmistokehitysprosessi huomioon ottaen ja kaikki ohjelmistokehitysprosessiin liittyvät henkilöt siihen sitouttaen, jatkuvan integrointikäytännön käyttöönotto tarjoaa mahdollisuuden saada ohjelmistokehitysprojektin onnistumisen kannalta kriittinen vaihe, integrointi, stabiilimmaksi, ja sitä myöten lisää resursseja integrointia edeltävään ohjelmistokehitykseen. Jatkuvan integroinnin tärkeys vielä korostuu monialustaympäristössä, jossa ohjelman kääntyvyys ja toimivuus on joka kerta tarkastettava kaikilla tuetuilla alustoilla, ja jollainen tässä opinnäytetyössä kyseessä ollut Qt-ohjelmointiympäristö on.

LÄHTEET

Agile Alliance 2010. Viitattu 15.09.2010. <http://www.agilealliance.org>.

Agile Manifesto 2001. Viitattu 05.10.2010. <http://agilemanifesto.org/>.

Bullseye 2011. Viitattu 3.5.2011. <http://www.bullseye.com/>.

Campwood Software 2011. Viitattu 3.5.2011. <http://www.campwoodsw.com/sourcemonitor.html>.

Duvall, Paul M.; Matyas, Steve & Glover Andrew, 2007. Continuous Integration Improving Software Quality and Reducing Risk. Boston: Addison-Wesley.

Electric Cloud 2011. Viitattu 3.5.2011. <http://www.electric-cloud.com/products/electriccommander.php>.

Fowler, Martin 2006. Continuous Integration. ThoughtWorks. Viitattu 25.10.2010. <http://www.martinfowler.com/articles/continuousIntegration.html>.

Froglogic 2011. Viitattu 3.5.2011. <http://www.froglogic.com/index.php>.

Git the fast version control system 2011. Viitattu 10.4.2011. <http://git-scm.com/>.

Hudson 2011. Viitattu 3.5.2011. <http://hudson-ci.org/ >.

Jenkins 2011. Viitattu 3.5.2011. <http://jenkins-ci.org/>.

Ketterä ohjelmistokehitys 2010. Viitattu 20.10.2010. <http://fi.wikipedia.org/wiki/Ketter%C3%A4_ohjelmistokehitys>.

Mattila, Pekka 2008. Otollinen tilaisuus. Helsinki: Talentum.

Mercurial source control management 2011. Viitattu 5.4.2011. <http://mercurial.selenic.com/about/>.

NCover 2011. Viitattu 3.5.2011. <http://www.ncover.com/>.

NUnit 2011 . Viitattu 3.5.2011. <http://www.nunit.org/>.

Qt Creator Whitepaper 2011. Viitattu 10.1.2011. <http://developer.qt.nokia.com/wiki/pdf/QtCreatorWhitepaper>.

Qt – Cross-platform application and UI framework 2011. Viitattu 10.1.2011. <http://qt.nokia.com/>.

Qt Reference Documentation 2011. Viitattu 10.1.2011. <http://doc.qt.nokia.com/qtcreator-snapshot/creator-overview.html>.

Qt Who we are 2011. Viitattu 10.5.2011. <http://qt.nokia.com/about/who-we-are>.

Ranorex 2011. Viitattu 3.5.2011. <http://www.ranorex.com/>.

Rational Synergy 2011. Viitattu 11.5.2011. <http://www-01.ibm.com/software/awdtools/synergy/>.

Thomson, Patrick 2008. Git vs. Mercurial: Please Relax. Important Shock. Viitattu 5.4.2011. <http://importantshock.wordpress.com/2008/08/07/git-vs-mercurial/>.