VAASAN YLIOPISTO

TEKNILLINEN TIEDEKUNTA

OPPIAINE TIETOTEKNIIKKA

Jaakko Huhtala

JATKUVA INTEGRAATIO -PROSESSIN KÄYTTÄMINEN LAITEALUSTA-RIIPPUMATTOMASSA OHJELMISTOKEHITYKSESSÄ

Asiakasyrityksen ohjelmistokehitysalustassa käytetyn käännösympäristön selvitystyö

Tietotekniikan diplomityö, joka on jätetty tarkastettavaksi diplomi-insinöörin tutkintoa varten Vaasassa 27.4.2010.

Työn valvoja Merja Wanne

Työn ohjaajat Jouni Lampinen
Tero Vuorela

SISÄLLYS **sivu**

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Jaakko Huhtala
Tutkielman nimi: Jatkuva integraatio -prosessin käyttäminen laitealustariippumattomassa ohjelmistokehityksessä
Ohjaajan nimi: Jouni Lampinen
Tutkinto: Diplomi-insinööri
Laitos: Tietotekniikan laitos
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2002
Työn valmistumisvuosi: 2010 **Sivumäärä:** 55

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee tietokoneohjelmiston kehitystyöprosessia, jossa kehitetään laitealustariippumatonta sovelluskehystä C/C++-ohjelmointikielellä käyttäen jatkuva integraatio -prosessia. Työ on tapaustutkimus, jossa kehitetään työn toimeksiantajan projektissa käytetyn sovelluskehitysjärjestelmän toimintaa. Sovelluskehitysjärjestelmässä käytetään jatkuva integraatio -prosessin toimintatapoja ja työkaluja, joilla helpotetaan ohjelmistokehittäjän työtä sekä mahdollistetaan sovelluksen toimivuus eri laitealustoilla kehitystyön aikana. Työn tarkoituksena on tehdä selvitys käytetystä sovelluskehitysjärjestelmästä sekä havainnoida järjestelmän ongelmia ja löytää ehdotuksia järjestelmän parantamiseksi.

Työssä verrataan aiempaa tutkimusta aiheesta tutkittavan projektin käännösympäristöön. Työssä esitellään jatkuva integraatio -prosessia varten kehitettyjä työkaluja. Lisäksi työssä havainnoidaan toimeksiantajan projektin ohjelmistokehitystyön aikana esiintyneitä käännösympäristön ongelmia ja etsitään parannusehdotuksia käännösympäristön kehittämiseksi.

Käytetyssä ohjelmistokehitysjärjestelmässä on havaittu ongelmia sovelluksen kääntyvyyden ylläpitämisessä kaikilla laitealustoilla. Toinen ongelma on ollut käännöspalvelimella kääntämisen hitaus. Parannusehdotuksena työssä esitetään Pulse-käännöspalvelinohjelmistossa käytetty private build -prosessi, joka helpottaisi ja nopeuttaisi kääntyvyyden testaamista kaikilla laitealustoilla. Työssä esitetään parannusehdotuksina myös seuraavia asioita: integraatiopalvelimen tuottamien binääripakettien lataamisoptio käännösympäristöön, CMake-abstraktiokerroksen luominen CMake-konfiguraatioiden tekemisen helpottamiseksi, kaaviokuvien automaattinen luominen käännöspalvelimella kuvaamaan sovelluskirjastojen välisiä riippuvuuksia ja kehitysympäristön käyttöönoton nopeuttaminen asennussovelluksen avulla.

AVAINSANAT: jatkuva integraatio -prosessi, ohjelmistokäännösympäristö, sulautetut järjestelmät, ohjelmistokehitysympäristö, laitealustariippumaton ohjelmistokehitys

UNIVERSITY OF VAASA
Faculty of technology
Author: Jaakko Huhtala
Topic of the Thesis: Continuous Integration -process in cross-platform software development
Supervisor: Jouni Lampinen
Degree: Master of Science in Technology
Department: Department of Computer Science and Engineering
Degree Programme: Degree Programme in Computer Science
Major of Subject: Computer science
Year of Entering University: 2002
Year of Completing Thesis: 2010 **Pages:** 55

ABSTRACT

The thesis introduces software development process that aims for creating cross-platform software with C and C++ programming languages using continuous integration process. The thesis is case study and it was ordered by a customer to investigate software development system that is used in their software development process. The investigated software development system uses continuous integration process and tools related to it for making developers work easier and for enabling cross-platform software development. The purpose of this study is to examine the software development system in use, observe possible problems in the system and create proposals for making the software development system better.

The thesis compares previous study and literature about the subject to the customers development system. The thesis introduces continuous integration studies and the tools that those studies recommended. The thesis also observes development process in the customer development project and searches for solutions to make the customers build system better.

Some problems were detected in the used software development system. One problem is in keeping the software intact for all supported platforms. Another problem is the usability and speed of the build servers. The thesis proposes usage of private build process that is a technique that can be deployed with Pulse build server system. Private build process would make the testing of changes easier for all supported platforms. Thesis also introduces following improvement suggestions: binary package download option to development environment for downloading and installing binary packages created by integration server, CMake abstraction layer for making build script writing easier, automatic diagram generation on build server for showing dependencies between software libraries and installation script for installing development environment.

KEY WORDS: continuous integration, build systems, embedded systems, cross-platform development, software development environment.

LYHENTEET JA SYMBOLIT

API	Application Programming Interface (Ohjelmointirajapinta, jolla eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään)
ARM	Advanced RISC Machines (Mikroprosessoriarkkitehtuuri, joka on suosittu sulautettujen järjestelmien suorittimissa)
CISC	Complex Instruction Set Computer (Yleisnimitys suorittimille, joiden konekielen käskyt ovat rakenteeltaan monimutkaisia)
CI	Continuous Integration (Englanninkielinen nimi jatkuva integraatio -prosessille)
GCC	GNU Compiler Collection (GNU-projektin kääntäjien kokoelma)
GNU	GNU is Not Unix (Projekti, johon on kerätty olemassa olevia vapaita ohjelmistoja ja jonka tavoitteena on kehittää täysin vapaa käyttöjärjestelmä)
IDE	Integrated Development Environment (Integroitu kehitysympäristö, joka yleensä sisältää tekstieditorin, kääntäjän ja testaustyökalut)
IRC	Internet Relay Chat (Internetin pikaviestintäpalvelu)
PC	Personal Computer (Henkilökohtainen tietokone)
POCO	Portable Components (C++-kirjastokokoelma, jolla pyritään yksinkertaistamaan ja nopeuttamaan laitealustariippumattomien sovellusten kehittämistä)
POSIX	Portable Operating System Interface for Unix (Unix käyttöjärjestelmille kehitetty IEEE-standardikokonaisuus)
PowerPC	Performance Optimization With Enhanced RISC – Performance Computing (RISC-tyyppinen mikroprosessoriarkkitehtuuri, johon on otettu mukaan myös monimut-

	kaisempia käskyjä. Käytössä muun muassa pelikonsoleissa ja henkilökohtaisissa tietokoneissa.)
RISC	Reduced Instruction Set Computer (Yleisnimitys suorittimille, joiden konekielen käskyt ovat rakenteeltaan yksinkertaisia)
STL	Standard Template Library (C++-kielen standardoitu ohjelmistokirjasto)
SVN	Subversion-niminen versionhallintasovellus.
x86	Yleinen nimi Intel-yrityksen kehittämälle suoritinarkkitehtuurille. x86 on CISC-tyyppinen arkkitehtuuri, ja se on käytössä muun muassa henkilökohtaisissa tietokoneissa.

1. JOHDANTO

1.1 Tutkimuksen taustatietoa

Tämä opinnäytetyö on tehty Wapice Oy:n asiakkaan toimeksiantamaan ohjelmistokehitysprojektiin. Toimeksiantajayrityksen nimeä ei tulla työssä esittämään yrityksen tuotteiden salassapitovaatimuksen takia. Tutkittavan projektin oikea nimi on myös salassapidon takia korvattu nimellä A-projekti.

Projektin toimeksiantajayritys käyttää digitaalitekniikkaa tuotteidensa ohjauslaitteissa ja työprosessin valvomisessa. Digitaalisia laiteprojekteja on yrityksellä ollut useita, ja erillään olevat projektit saattavat sisältää päällekkäisyyksiä laitteiden sovelluksien toteutuksissa. Ohjelmointiprojektien yhtenäistämiseksi yritys aloitti A-ohjelmistokehitysalustan toteutusprojektin, joka yhdistää yrityksen tuotteissa käytettyjen ohjelmistojen tyypilliset toiminnot yhteen ohjelmistokehityspakettiin. Ohjelmistokehitysalustalla luodaan yhtenäinen toimintatapa uusien digitaalisten laiteprojektien kehitystyöhön ja helpotetaan tuotekehittäjien työmäärää tarjoamalla valmiit ratkaisut tuotteiden yleisimpiin kehitystarpeisiin.

1.2 Tutkimuksen tavoitteet ja rajaukset

A-ohjelmistokehitysalustan kehitystyön tukemista varten käytetään projektin vaatimuksien mukaan räätälöityä käännösympäristöä. Käännösympäristö tukee sovelluksen kääntämistä usealle eri laitealustalle ja antaa työkalut käännöksessä tarvittavien työkalujen käyttämiseen, kuten testien ajamiseen, API-dokumenttien automaattiseen luomiseen ja lokalisointitiedostojen luomiseen. A-ohjelmistokehitysalustan kehitystyössä käytetään jatkuva integraatio -prosessia, jolla pyritään nopeuttamaan kehitystyötä, parantamaan

tuotetun sovelluksen laatua ja mahdollistamaan ohjelmistokehitysalustan nopea julkaisusykli. Prosessin tukemiseksi käännösympäristössä käytetään käännöspalvelimia, joilla pyritään varmistamaan versionhallinnassa olevan lähdekielisen sovelluksen toimivuus.

Tämä opinnäytetyö on selvitys A-projektissa käytetystä käännösympäristöstä. Työssä käydään läpi käännösympäristöihin ja jatkuva integraatio -prosessiin liittyvää teoriaa ja esitetään A-käännösympäristössä käytettyjä tekniikoita ja toimintatapoja. Työssä havainnoidaan käännösympäristön toimintaa ja esitetään ratkaisuja sen parantamiseksi.

A-projekti on jaoteltu kahteen alueeseen: mc (machine control level) ja sup (supervisor level). Tämä työ keskittyy sup-tason kehitystyöhön, jossa luodaan ohjelmistokehitysalusta sulautettujen näyttölaiteohjelmistojen kehittämiseksi.

1.3 Tutkimusmenetelmä

Tämä opinnäytetyö on selittävä tapaustutkimus. Selittävässä tapaustutkimuksessa tuotetaan tosiasioiden kirjaamista ja vedetään niistä päätelmiä (Järvinen & Järvinen 2000: 79). Opinnäytetyössä havainnoidaan A-projektin sovelluskehitysjärjestelmän toimintaa. Havaintojen ja aiemman teorian pohjalta tuotetaan ehdotuksia järjestelmän parantamiseksi.

1.4 Tutkimuksen rakenne

Tämän työn rakenne on jaoteltu seuraavasti: kappaleessa kaksi esittelen tutkimuksen lähtökohtia ja kerron aikaisempiin tutkimuksiin tutustumisesta sekä tutustumisesta erilaisiin jatkuva integraatio -järjestelmiin. Kappaleessa kolme käsittelen teoriaa laitealustariippumattoman sovelluksen toteuttamiseksi. Kappaleessa neljä esitän teoriaa jatkuva

integraatio -prosessin käyttämisestä ja sitä, kuinka prosessia on käytetty A-sovelluskehyksen kehitystyössä. Kappaleessa viisi arvioin A-sovelluskehyksen käännösympäristön toimivuutta ja esitän ehdotuksia käännösympäristön parantamiseksi. Kappaleessa kuusi työ saatetaan päätökseen esittelemällä työn tuloksia sekä työn vaikutuksia projektiin.

2. KATSAUS JATKUVA INTEGRAATIO -JÄRJESTELMIIN

2.1 Aiempi kokemus käännösympäristöistä ja jatkuva integraatio -prosessin käytöstä

Tutustuin käännösympäristöihin ensimmäisen kerran työni kautta vuonna 2006. Projekti, johon tulin mukaan, oli ollut jaoteltuna erillisiin komponentteihin ja niitä oli kehitetty erillään toisistaan. Tehtävänäni oli luoda käännösjärjestelmä, joka yhdistää projektin niin, että kaikki komponentit voidaan kääntää käyttäen yhtä käännöskomentoa. Yhdistetyllä käännösjärjestelmällä komponenttien toimivuus yhdessä voitiin varmentaa helposti kehitystyön aikana ja koko projektista saatiin kerralla käännettyä valmis sovelluspaketti.

Kehittäessäni projektiin uusia ominaisuuksia huomasin, että toteutettavaa lähdekieltä on hyvä integroida versionhallintaan mahdollisimman usein. Jos versionhallintaan ehti tulla paljon muutoksia, oli integroinnissa esiin tulleet ongelmat vaikeampi löytää ja korjata. Vaikka projektissa ei käytetty jatkuva integraatio -prosessia, oli jatkuvan integraation tarve kuitenkin olemassa itsestään. Muuten integraatiovaiheessa ilmenneet ongelmat olisivat saattaneet paisua liian isoiksi.

Projektissa oli mukana vain muutama henkilö, joten integroinnissa esiin tulleet ongelmat oli yleensä helppo korjata yhdessä muiden kehittäjien kanssa. Välillä versionhallintaan jäi kuitenkin siirtämättä osia sovelluksesta tai integraatiota ei ollut tehty kunnolla, jolloin versionhallinnassa oli kääntymätön versio projektista. Tämä tuli ilmi vasta, kun toinen kehittäjä yritti integroida omia muutoksiaan versionhallintaan. Jos ongelman korjaamisesta vastuussa ollut kehittäjä ei ollut paikalla, ongelmaa saattoi olla mahdotonta yrittää ratkaista. Tällaiset ongelmatilanteet olisi voitu korjata ajoissa, jos käytössä olisi ollut integraatiopalvelin. Integraatiopalvelin olisi havainnut heti kun sovellus ei käänny ja ilmoittanut siitä ongelman aiheuttaneelle kehittäjille. Muutaman hengen kokoisissa kehitysprojekteissa ongelmat eivät ehdi kasvaa isoiksi, mutta kun kehityksessä on mu-

kana paljon ihmisiä, ongelmat kasautuvat helposti. Tämän takia sovelluksen toimivuus on hyvä varmistaa automatisoidun käännösjärjestelmän avulla.

Jatkuva integraatio -prosessiin tutustuin ensimmäisen kerran siirtyessäni mukaan A-projektiin. Jatkuva integraatio -prosessi oli projektissa jo käytössä, kun tulin mukaan. Ensimmäinen tehtäväni oli tutustua siihen, kuinka käännösautomaatio toimii ja kuinka integraatiopalvelimia käytetään. Sain projektissa käännösmanagerin vastuun, joten vastaan työssäni sovelluksen kääntyvyydestä ja versioinnista sekä kehitän integraatiojärjestelmään uusia ominaisuuksia.

2.2 Aiempi jatkuva integraatio -järjestelmistä tehty tutkimus

Tutustuin tutkimukseni aluksi jatkuva integraatio -järjestelmistä kirjoitettuun tutkimukseen ja kirjallisuuteen nähdäkseni, kuinka jatkuva integraatio -prosessi on toteutettu muissa projekteissa ja mitä toimintoja suositellaan käytettäväksi. Käytin prosessiin tutustumiseksi seuraavia teoksia:

- Fowlerin (2006) kirjoittama artikkeli *Continous Integration*, jossa hän käy läpi jatkuvan integraatio -prosessin perusteet, selvittää prosessin käytöstä saadut hyödyt ja esittää mitä asioita prosessin käytössä on hyvä ottaa huomioon.
- Duvall, Matyas ja Glover (2007) esittelevät teoksessaan *Continous Integration: Improving Software Quality and Reducing Risk* jatkuva integraatio -prosessin käyttöönottamisen Fowleria (2006) tarkemmin antamalla esimerkkejä integraatiopalvelimen käyttämiseksi ja integraatiotestien ajamiseksi. Kirja keskittyy esimerkeissään pitkälti CruiseControl-nimisen integraatiopalvelimen käyttöön, mutta esitettyjä menetelmiä voidaan seurata myös käytettäessä muita integraatiopalvelinsovelluksia.
- Palviainen (2009) on kirjoittanut jatkuva integraatio -prosessin käyttöönotosta diplomityön *Introducing Continuous Integration for C and C++ Software Development Projects on Linux Platform*. Kirjoituksessa esitetään alustava tutkimus

jatkuva integraatio -prosessin käyttöönottamiselle Sesca Mobile Software Oy:ssä.

Tutustuin myös laitteistoriippumattomasta sovelluskehityksestä kirjoitettuun kirjallisuuteen:

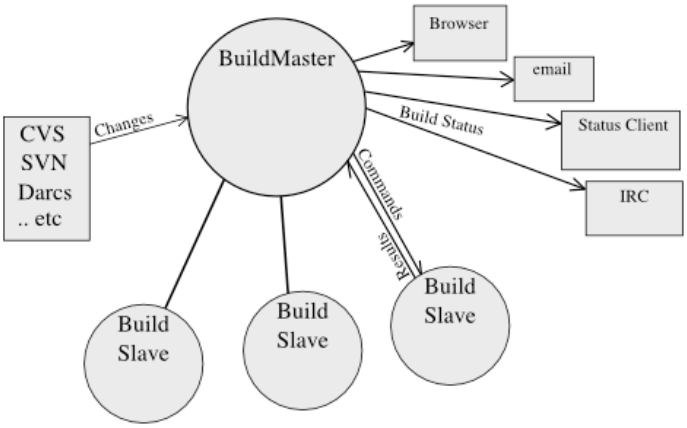
- Logan (2008) käsittelee kirjassaan *Cross-Platform Development in C++* laitteistoriippumattomassa sovelluskehityksessä huomioitavia asioita. Kirjassaan esittelee tekniikoita ongelmiin lähdesovellusesimerkein.
- Yaghmourin, Mastersin, Ben-Yossefin ja Gerumin (2008) teos *Building Embedded Linux Systems* kattaa sulautetun Linux-järjestelmän kehityksen peruskäsitteet ja tekniikat. Kirja sisältää myös paljon neuvoja erilaisten työkalujen käyttämiseksi helpottamaan kehitystyötä ja ongelman ratkaisua.

2.3 Eri tyyppisiä jatkuva integraatio -järjestelmiä

Jatkuva integraatio -palvelinjärjestelmiä on useita. Järjestelmiä on julkaistu sekä Open Source -lisensseillä että maksullisilla lisensseillä. Osa sovelluksista on suunniteltu käytettäväksi tietyn tyyppisissä ohjelmistoprojekteissa, kuten Web-, Windows- tai Java-sovelluksissa. Osa sovelluksista taas on laitealustariippumattomia, joten niillä voidaan kehittää projektia mille tahansa laitealustalle. Laitealustariippumattomuus on yleensä tuettu luomalla mahdollisuus käännössovellusten käynnistämiselle komentorivikäskyllä. Projektityyppikohtaisten ominaisuuksien käyttämisen etuna kuitenkin on, että siinä on projektityypille ominaiset työkalut helpommin käytettävissä. Seuraavassa listassa esittelen lyhyesti muutamia integraatiopalvelin -sovelluksia:

- BuildBot (versio 0.7.12) on Open Source -sovellus integraatiomuutosten validointiin ja testien ajamiseen. Sovellus on kirjoitettu Python-kielellä, ja se tukee laitealustariippumatonta sovelluskehitystä. Sovelluksen arkkitehtuuri erottelee käännöspalvelinympäristöön kuuluvaksi yhden BuildMaster-koneen, johon voi-

daan liittää rajaamaton määrä BuildSlave-käännöskoneita. BuildMaster päättää, milloin ja mitä käännetään ja käskyttää BuildSlave-käännöskoneita hoitamaan kääntämisen. Käännöstulokset voidaan lähettää sähköpostiin tai IRC-palvelimelle, niitä voidaan seurata web-sovelluksesta tai voidaan käyttää erillistä työpöytäsovellusta, jonka kautta voidaan lähettää kehittäjille myös viestejä. (BuildBot 2010.)



Kuva 1. BuildBot-arkkitehtuuri.

- CruiseControl (versio 2.8.3) on yksi ensimmäisistä integraatiopalvelinsovelluksista. Sovellus on kirjoitettu Javalla, ja se on alun perin luotu tukemaan Javassa paljon käytettyä Ant-käännösjärjestelmää. Myös seuraavia käännösjärjestelmiä on nykyään tuettu: NAnt (.NET), Maven (Java), Phing (PHP), Rake (Ruby) ja XCode (OSX). Sovellusta voidaan käyttää myös C- ja C++-projektien kääntämiseen konsolikäynnistyksen avulla, mutta pääosin CruiseControl on tarkoitettu Java-projekteille. Projektin aloitti ThoughtWorks-niminen yritys, joka nykyään tekee kaupallista Cruise-integraatiojärjestelmää. (CruiseControl 2010.)
- Pulse (versio 2.1.23) on Zutubi-nimisen yrityksen kaupallinen integraatiopalvelinsovellus. Pulsessa on mukana paljon BuildBotin ominaisuuksia, kuten useamman käännöspalvelimen käyttö, tuki testien ajamiselle, laitealustariippumattomuus ja käännöstulosten julkaiseminen monella eri tavalla. Pulsessa on lisäksi

muita hyödyllisiä ominaisuuksia, kuten testitulosten näyttäminen, käännöspalvelimen käyttäminen API-rajapinnan avulla, käyttöoikeuksien määrittäminen ja henkilökohtaisten käännösten ajaminen, jonka avulla kehittäjä voi kokeilla kääntää omia muutoksiaan käännöspalvelimella siirtämättä muutoksia ensin versionhallintaan. (Zutubi 2010a.)

- Qt Continuous Integration on sovelluskehys laitealustariippumattomien sovellusten kehittämiseen. Qt-projekti ilmoittaa Internet-sivuillaan ottavansa käyttöön jatkuva integraatio -palvelintuen ennen kesäkuuta julkaistavassa Qt Creator IDE -sovelluksessaan. (Qt 2010a.)

3. LAITEALUSTARIIPPUMATTOMAN SOVELLUKSEN TOTEUTTAMINEN C- JA C++- OHJELMOINTIKIELILLÄ

Laitealustariippumattoman sovelluksen toteutuksessa pyritään kehittämään tietokoneohjelmistoa, jota voidaan käyttää useilla eri laitealustoilla. Ohjelmistokehittäjän näkökulmasta laitealustalla tarkoitetaan sekä tietokoneen fyysisen arkkitehtuurin että tietokoneen ohjelmistoalustan yhdistelmää. (C.Bailey 2000: 263.) Yleisin PC-laitealusta koostuu x86-prosessorin ja Windows-käyttöjärjestelmän yhdistelmästä. Prosessoriarkkitehtuureita ja käyttöjärjestelmiä on kuitenkin olemassa useita, joten laitealustakombinaatioita on myös useita. Tässä luvussa käyn läpi laitealustariippumattoman sovelluksen kehittämiseen liittyvää teoriaa.

3.1 Ohjelmointikielten tavat toteuttaa laitealustariippumattomuus

Eri ohjelmistokielillä on eri tapoja mahdollistaa sovelluksen laitealustariippumattomuus. Java- ja Python-kielet käyttävät sovelluksen ajamiseen virtuaalikoneeksi kutsuttua sovellusta, joka tulkkaa sovellusta ajon aikana laitteen prosessorin ja käyttöjärjestelmän ymmärtämään muotoon. Tämä toteutustapa mahdollistaa sen, että ohjelmistosovelluksesta tarvitsee olla vain yksi jaettava paketti. Tämä yksi jaettava paketti toimii tällöin kaikilla laitealustoilla, joille tulkkaava sovellus on asennettu. Tulkattavan ohjelmistokielen käyttäminen ei kuitenkaan sovellu kaikille laitealustoille, sillä sovelluksen kääntäminen ajon aikana vie paljon prosessoritehoa. Esimerkiksi sulautetuissa laitteissa, joissa käytetään prosessoriteholtaan pienempiä mikroprosessoreita, saattaa tulkattavan kielen käyttäminen olla liian hidasta. Tulkattava ohjelmistokieli ei yleensä sovellu myöskään reaaliaikasovelluksiin, joissa toiminnon suorittamiselle on aikavaatimus. Aikavaatimus tarkoittaa, että funktioiden ajamiselle on määrätty aikaväli, jonka aikana sen täytyy olla suoritettu (Raghavan 2006: 201). Reaaliaikavaatimus voi olla esimerkiksi sovelluksilla, joilla ohjataan isoja mekaanisia laitteita kuten lentokoneita, tehdastyöko-

neita sekä rakennus- ja kaivinkoneita.

C/C++-kielellä toteutettu sovellus käännetään suoraan laitealustan ymmärtämään binäärimuotoon (Prata 2005: 18). Binäärimuotoisen sovelluksen ajaminen on nopeampaa kuin tulkattavan kielen ajaminen, koska erillistä tulkkaavaa sovellusta ei tarvita. Valmis binäärimuotoinen ohjelmistosovellus toimii kuitenkin ainoastaan laitealustalla, jolle se on käännetty, eikä sitä voi käyttää muilla laitealustoilla (Prata 2005: 17).

C/C++-kielellä voidaan kirjoittaa laitealustariippumaton lähdekielinen ohjelmisto, kun käytetään laitealustariippumattomia sovelluskirjastoja. Laitealustariippumattomia sovelluskirjastoja ovat esimerkiksi ANSI/ISO C/C++ -standardissa määritetyt STL-kirjastot ja POSIX-standardissa määritetyt kirjastot. POSIX on alun perin Unix-järjestelmiin kehitetty standardi, mutta sitä tukevat myös monet muut käyttöjärjestelmät, kuten Windows-käyttöjärjestelmät. (Logan 2008: 221–222.)

3.2 C- ja C++-sovelluksen kääntäminen eri mikroprosessoriarkkitehtuureille

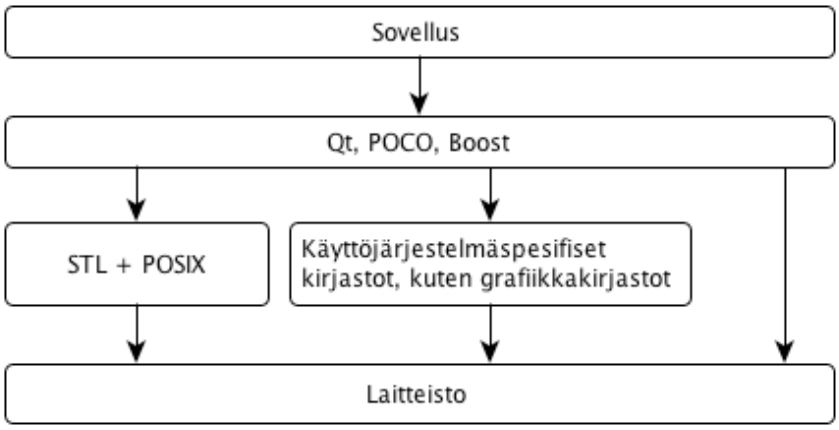
Mikroprosessoriarkkitehtuureja on useita. Näistä yleisimmin käytettyjä mikroprosessoriarkkitehtuureja ovat esimerkiksi x86, ARM ja PowerPC. Jotta sovellus saadaan toimimaan halutulla laitealustalla, on se käännettävä laitealustan ymmärtämään binäärimuotoon. Käyttövalmis binäärimuotoinen ohjelmistosovellus on siis laitealustaspesifinen. Lähdekielinen ohjelmisto, josta sovellus käännetään, voi kuitenkin olla laitealustasta riippumaton, jolloin samaa lähdekieltä voidaan käyttää sovelluksen kääntämiseksi myös toiselle laitealustalle (Prata 2005: 17-18). Tämä kuitenkin vaatii, että käytettävillä laitealustoilla on myös ohjelmistokielen kääntäjä, joka osaa tulkita kieltä samalla tavalla. Lisäksi kaikkien sovelluksessa käytettyjen ulkopuolisten ohjelmistokirjastojen on oltava käännettynä käytettävällä laitealustalla.

Lähdekielisen sovelluksen kääntämiseen konekieliseksi binääritiedoksi on kaksi tapaa. Lähdekieli voidaan kääntää joko kohdelaitealustalla, jolla sitä tullaan käyttämään, tai se voidaan kääntää toisella laitealustalla käyttäen kohdelaitealustan kääntäjää. Jälkimmäistä tapaa kutsutaan ristikääntämiseksi. Ristikääntämistä käytetään yleensä käännettäessä sovellusta sulautetuille laitealustoille, joiden prosessoritehot ovat pienemmät kuin PC-koneissa. Kääntäminen sulautetulla kohdelaitealustalla olisi kehitystyön aikana hidasta ja hankalaa. (Yaghmour 2008: 91–92.)

C- ja C++-kielen ristikääntämistä varten tarvitaan kohdekoneen kääntäjä, kohdekoneen ohjelmistokirjastot sekä ohjelmistokirjaston kuvaustiedostot (engl. *header files*, *.h). Tällaista kääntämisessä käytettyä pakettia kutsutaan työkaluketjuksi (engl. *toolchain*). Työkaluketju sisältää yleensä myös lisäkirjastoja (kuten zlib-kirjasto, joka tarjoaa palveluja pakkaukseen) ja lisätyökaluja virheidenjäljitykseen, profilointiin ja muistin tarkistukseen. (Yaghmour 2008: 91.)

3.3 Ohjelmistokirjastoja laitealustariippumattoman sovelluksen kehitykseen

Laitealustariippumattoman sovelluksen luomiseen on muitakin sovelluskirjastoja kuin STL- ja POSIX-kirjastot. Näitä ovat esimerkiksi Open Source -ohjelmistokirjastot, kuten Poco, Qt ja Boost. Nämä kirjastot toteuttavat suurimman osan laitealustariippumattomuudesta käyttäen STL- ja POSIX- kirjastoja. Lisäksi niissä saatetaan käyttää tuetuille laitealustoille ominaisia sovelluskirjastoja. (Poco 2010; Qt 2010; Boost 2010.) Nämä standardit mahdollistavat laitealustariippumattoman lähdekielisen ohjelmiston toteuttamisen tarjoamalla oman kirjastorajapintansa, jota käyttämällä voidaan varmistaa sovelluksen toimiminen sovelluskirjaston tukemilla laitealustoilla.



Kuva 2. Sovelluskirjastojen kutsuhierarkia käytettäessä Qt-, POCO- tai Boost-sovelluskirjastoja.

A-projektissa käytetään Qt-sovelluskehyksen kirjastoja. Qt on C++-kielelle tehty Open Source -sovelluskehitysalusta, jonka avulla voidaan luoda usealla käyttöjärjestelmällä toimivia sovelluksia. Qt tukee Windows-, Mac OS X- ja Linux-käyttöjärjestelmiä, sekä useita Unix-käyttöjärjestelmiä. Suurin osa Qt:sta on omistettu laitealustaneutraalin sovellusrajapinnan luomiseksi, jota voidaan käyttää minkä tahansa toiminnon luomiseksi ilman, että tarvitsee käyttää laitealustakohtaisia toteutustapoja. Vaikka Qt on alun perin suunniteltu C++-ohjelmointiin, on sen käyttämiseksi olemassa rajapintoja myös muille ohjelmointikielille. Qt:n viralliset rajapinnat on tehty C++-, Java- ja JavaScript-kielille. Kolmansien osapuolien tuottamia epävirallisia Qt-rajapintoja on luotu myös näille ohjelmointikielille: Python, Ruby, PHP ja .NET (Thelin 2007: 3).

3.4 Ongelmien välttäminen laitealustariippumattoman sovelluksen kehityksessä

Laiteriippumattomassa sovelluskehityksessä C- ja C++-kielillä tulee usein esiin ongel-

mia, joita ei yhdellä laitealustalla kehitettäessä tarvitse huomioida. Suurin osa näistä ongelmista johtuu kääntäjien välisistä eroista. Tässä luvussa esitellään tapoja, kuinka voidaan välttää laiteriippumattomassa sovelluskehityksessä esiintyviä ongelmia.

3.4.1 Kääntäjäongelmien välttäminen

Laitealustariippumattoman C++-sovelluksen kääntämisessä pitää kaikille käytetyille laitealustoille olla oma kääntäjänsä. C- ja C++-kääntäjien toimittajat kehittävät kääntäjiin jatkuvasti uusia ominaisuuksia ja parantavat vanhoja ominaisuuksia. Standardit myös muuttuvat kehityksen mukana, ja toimittajat toteuttavat uudet standardien mukaiset toiminnot kääntäjiinsä. Prosessissa on monta asiaa, jotka voivat mennä vikaan. Eri laitealustoilla käytetyt kääntäjät saattavat tukea eri versioita standardeista, tai jokin standardi on toteutettu kääntäjään väärin tai vain osittain. (Logan 2008: 52–54.) Eri kääntäjät siis saattavat tulkita sovelluksen lähdekieltä eri tavoilla, jolloin yhdellä kääntäjällä toteutettu sovellus ei ehkä käänny toisen laitealustan kääntäjällä. On myös mahdollista, että jokin kääntäjäominaisuus toimii eri tavoilla eri kääntäjillä. Tällöin ominaisuutta käyttävä sovellus saattaa toimia eri laitealustoilla eri tavalla.

Laiteriippumattomassa ohjelmistokehityksessä sovellusta kannattaa kääntää ja testata kehitystyön aikana kaikilla sovelluksen tukemilla laitealustoilla, jotta mahdolliset ongelmat havaitaan ajoissa (Logan 2008: 52). Logan (2008: 54) esittää, että uusimpia kääntäjien ominaisuuksia ei käytettäisi, ennen kuin ominaisuus on varmistettu toimivaksi kaikilla sovelluksen tukemilla laitealustoilla. Kääntäminen eri kääntäjillä saattaa myös parantaa sovelluksen laatua. Tähän Logan (2008: 53) esittää neljä syytä:

- Koska kääntäjien erot tulevat esiin käännettäessä usealla kääntäjällä, erot kääntäjien ominaisuuksissa on helpompi huomioida.
- C- ja C++-kielien eri standarditulkintojen vaikutus minimoituu ja ei-hyväksyttyjen kääntäjäominaisuuksien käyttöä on helpompi välttää.
- Jokainen kääntäjä tuottaa erilaisia varoitus- ja virheviestejä. Virheilmoitus toisella kääntäjällä saattaa olla selkeämpi, mikä helpottaa virheen etsimistä. Toinen

kääntäjä saattaa huomioida virheen, jonka toinen kääntäjä jättää huomioimatta. Tällaisten virheiden löytäminen parantaa sovelluksen laatua.

- Eri kääntäjät kääntävät sovellusta eri tavoilla, mikä saattaa tuoda esiin ongelmia, joita ei muuten olisi havaittu.

Standardien noudattamiseksi kääntäjille voidaan myös asettaa käännöslippuja. Linuxissa käytetyn GNU-kääntäjän käännöslippu *-std=c++98* määrää kääntäjän käyttämään vuoden 1998 ISO C++ -kielen standardia. Toinen standardiin vaikuttava käännöslippu on *-pedantic-errors*. Sillä voidaan määrätä, että ainoastaan ISO-standardin kanssa yhdenmukainen lähdekieli sallitaan. Se myös estää kaikkien GNU-kääntäjän laajennuksien kääntymisen. Vastaavan tyyppinen Windows-kääntäjän käyttämä lippu on */Za*. Sitä käyttämällä voidaan Microsoftin laajennukset C++-kieleen laittaa pois päältä.

3.4.2 Varoitusilmoitusten huomioiminen

Kääntäjät antavat varoituksia, jotka usein jäävät suunnittelijoilta huomioimatta. Varoitukset eivät estä ohjelman kääntymistä eivätkä usein vaikuta sovelluksen toimintaan, joten kehittäjät eivät aina jaksa käyttää aikaa varoitetun ongelman korjaamiseen. Varoitukset ovat kuitenkin kääntäjän tapa kertoa epävarmasta käännöstilanteesta, joka saattaa johtaa määrittelemättömän tilanteeseen tai sovelluksen virheelliseen käyttäytymiseen. Kun ohjelmaa ajetaan myös muilla laitealustoilla, tämä epävarmuus korostuu, koska eri laitealustat saattavat hoitaa varoitetun ongelman eri tavalla ajettaessa ohjelmaa. Tämä johtaa usein siihen, että sovellusta ei ole siirrettävissä kaikille laitealustoille. (Logan 2008: 62.)

Kääntäjän varoituksien esiin saamiseksi voidaan käyttää käännöslippuja. GNU-kääntäjän käännöslippu *-Wall* mahdollistaa, että suurin osa käännösvaroituksista näytetään. *-Wall* jättää kuitenkin näyttämättä esimerkiksi *-Weffc++*-lipun näyttämät varoitukset. (Logan 2007: 63–64.) *-Weffc++*-käännöslippua voidaan käyttää tarkastamaan, että Meyersin (1998) määrittelemiä tyyliohjeita noudatetaan. Tyyliohjeet määrittävät muun

muassa, että luokkien destruktorit on asetettava virtuaalisiksi (Meyers 1998: 59) ja kopiointikonstruktori ja asetusoperaattori (operator=) on luotava, jos luokassa varataan dynaamisesti muistia (Meyers 1998: 49). Kääntäjä antaa varoituksen, jos lähdekieli on ristiriidassa tyyliohjeiden kanssa. *-Werror*-käännöslipulla saadaan kääntäjä keskeyttämään käännös, jos käännettäessä ilmenee varoitus. Tämä pakottaa suunnittelijan korjaamaan virheen sen sijaan, että virhe sivuutettaisiin. Windows-kääntäjä */WX*-käännöslippu toimii samaan tapaan kuin *-Werror*. (Logan 2007: 63–64.)

4. JATKUVA INTEGRAATIO -PROSESSI A-SOVELLUSKEHITYS-YMPÄRISTÖSSÄ

A-projektin tarkoituksena on luoda ohjelmistokehitysalusta, joka helpottaa asiakkaan tuotteisiin tehtävien ohjaus- ja hallintasovellusten kehittämistä. Projektissa on mukana useita ohjelmistokehittäjiä, jotka tekevät töitä eri toimistoissa. Tämä luo haasteen integraatioprosessin toimivuudelle ja muutosten hallinnalle, jotta tieto muutoksista saadaan välitettyä kehittäjien välillä joustavasti ja nopeasti.

A-projektissa käytetään jatkuva integraatio -prosessin mukaisia toimintatapoja. Tämä luku kuvaa A-projektissa käytetyn ohjelmistokehitysprosessin toimintatavat ja sen, kuinka A-projektin kehitystyö käyttää jatkuva integraatio -prosessia.

4.1 A-sovelluskehitysprosessi

A-sovelluskehitysalusta on jaettu komponentteihin. Näistä jokaiselle on nimetty ohjelmistokehittäjä, joka on vastuussa komponentin kehityksestä. Komponentti on yleensä ohjelmistokirjasto, joka toteuttaa tietyn itsenäisen toiminnon. Se voi myös olla käynnistettävä itsenäinen sovellus. A-ohjelmistokehitysalustaa käyttävät asiakassovellusprojektit voivat itse valita, mitä komponentteja haluavat käyttää ja mitä eivät. Jos jotain komponenttia ei projektissa tarvita, se voidaan jättää pois eikä se silloin vie tilaa laitteiden ohjelmamuistissa.

A-projektin lähdekielinen sovellus sijaitsee versionhallintapalvelimella, jossa SVN-palvelinohjelmisto (*Subversion*) säilyttää lähdekielisen sovelluksen ja tiedot sovellukseen tehdyistä muutoksista. Versionhallintaa käytetään SVN-asiakasohjelmiston avulla, mikä mahdollistaa versionhallintapalvelimen käyttämisen ohjelmistokehittäjien omilta

tietokoneilta käsin.

Projektin kehittäjät hakevat versionhallintapalvelimelta viimeisimmän version sovelluksesta ja toteuttavat siihen halutut toiminnot. Kehityksessä käytetään jatkuva integraatio -prosessia, joten kehittäjiltä toivotaan, että he integroivat muutoksensa versionhallintaan usein. Sovelluksen lähdekieltä integroidessaan ohjelmistokehittäjä testaa ensin, että sovellus kääntyy ja toimii hänen omalla tietokoneellaan. Sen jälkeen hän päivittää versionhallinnasta viimeisimmän version sovelluksesta omalle koneelleen ja testaa, että sovellus toimii yhä muiden tekemien muutosten kanssa. Jos ongelmia ei ole, kehittäjä päivittää omat muutoksensa versionhallintaan. Muussa tapauksessa hän korjaa ongelmat ja tekee sen jälkeen päivityksen versionhallintaan. Integraatiotyö ei kuitenkaan ole valmis ennen kuin sovellus on käännetty myös käännöspalvelimella. Käännöspalvelinkäännöksellä halutaan varmistaa, että kaikki käännöksessä tarvittavat muutokset ovat siirtyneet kehittäjän koneelta versionhallintaan.

A-projektissa sovelluskehittäjä testaa tekemänsä muutokset omalla tietokoneellaan. Tätä varten hän kääntää sovelluksen omalle PC-laitealustaympäristölleen, joka on yleensä x86 ja Linux. Testaaminen tehdään sovelluskehittäjän omalla tietokoneella, jotta sovelluskehitys olisi nopeaa. Toinen tapa olisi testata sovellusta sulautetulla kohdelaitteella, mutta tämä on yleensä hitaampaa eikä aina tarpeellista. Sovelluksen kääntyminen muilla tuetuilla laiteympäristöillä varmistetaan kääntämällä sovellus käännöspalvelimilla kaikille tuetuille laiteympäristöille.

A-kehityksessä käytetään Qt-sovelluskehyskirjastoja, jotta vältyttäisiin laitealustakohtaisten toteutusten tekemiseltä (ks. lukua 3.3). Aina tämä ei kuitenkaan riitä, vaan eri laitealustat saattavat vaatia sovellukselta omia laitealustalle ominaisia toteutustapoja. Tällöin sovelluksen lähdekielessä käytetään kääntäjä-makroja, joilla valitaan käännöksen aikana sovellukseen mukaan otettava toteutus laitealustamääritysten mukaan (Logan 2008: 84–85). Kääntäjä-makrojen käyttöä pyritään kuitenkin välttämään, koska aina kun sovelluksen halutaan tukevan uutta laitealustaa, makroilla tehdyt järjestelmäspesifiset toteutukset täytyy tarkistaa ja lisätä myös uuden laitealustan käännökseen.

4.2 Jatkuva integraatio ja käännöspalvelimet

Projektin koon kasvaessa tulee järjestelmälle suurempi tarve varmistua eri kehittäjien toteuttamien komponenttien integraation toimivuudesta. Jos komponenttien integrointi tehdään vasta projektin lopussa, mahdolliset integraatio-ongelmat selviävät vasta silloin ja yhteensopivuuden korjaaminen saattaa vaatia isojakin muutoksia. Korjaus saatetaan myös joutua tekemään tavalla, joka heikentää sovelluksen laatua. Integraatio-ongelmat johtavat projektin viivästymiseen ja luovat projektille lisäkustannuksia. Jatkuva integraatio -prosessilla pyritään varmistamaan, että kehittäjät integroivat tuotoksiaan versionhallintaan jatkuvasti, jolloin ongelmat löytyvät ajoissa. (Duvall 2007.)

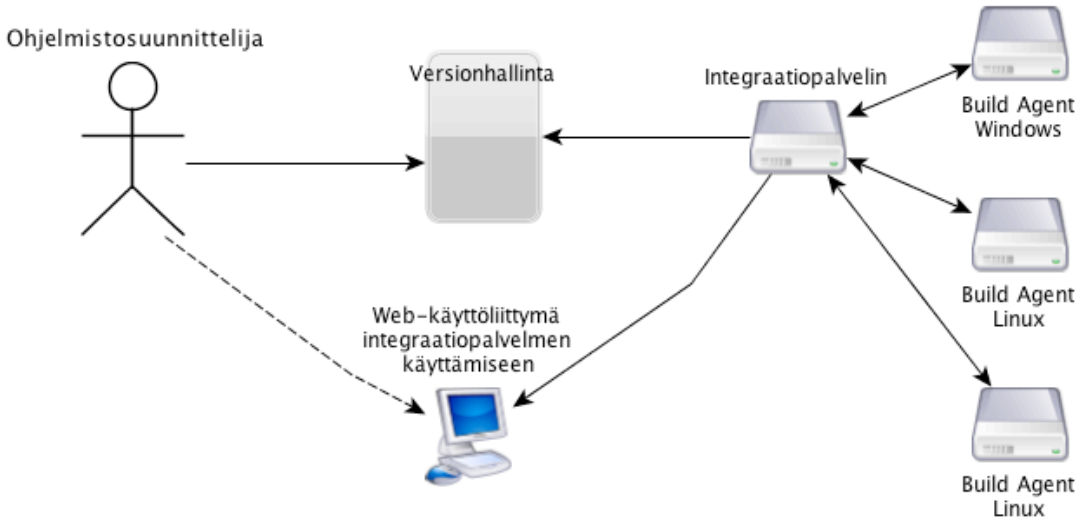
Jatkuva integraatio -prosessin käyttämiseksi Duvall (2007: xxi) listaa seitsemän kohtaa, jotka prosessissa täytyy toteuttaa.

- Sovelluskehittäjät ajavat käännöksen työkoneellaan ennen muutosten siirtämistä versionhallintaan varmistaakseen, ettei heidän muutoksensa riko integraatiokäännöstä.
- Kehittäjät siirtävät muutoksensa versionhallintaan vähintään kerran päivässä.
- Integraatiokäännökset tehdään useasti päivässä erillisellä käännöspalvelimella.
- Sovellukselle on ajettava kaikki testit integraatiokäännöksissä.
- Käännös luo sovelluksen, jolle voidaan tehdä funktionaalisia testejä.
- Käännöksen rikkoutuessa sen korjaaminen on tärkeysjärjestyksessä korkeimpana.
- Osa kehittäjistä käy läpi käännöksen luomat raportit, kuten sovelluksen laatua ja riippuvuuksia analysoivat raportit.

A-projektissa käytetään Zutubi-nimisen yrityksen kehittämää Pulse -käännöspalvelinohjelmistoa. Pulse on Javalla kirjoitettu palvelinohjelmisto, joka on suunniteltu jatkuva integraatio -prosessin avuksi automatisoimaan sovelluksen kääntämistä ja testien

mistä ja testien ajamista. Se sisältää web-käyttöliittymän, jolla sovelluksen kääntymistä voi seurata, sekä työkalut käännöstöiden ajamiseen ja testitulosten lukemiseen. Pulse -käännöspalvelimen yhteyteen voidaan lisätä useita palvelimia, joille käännöstöitä voidaan jakaa. Jokaiseen käännöspalvelimeen pitää asentaa Pulsen agenttiohjelmisto. Sen kautta Pulse käynnistää käännökset ja hakee käännöstulokset, jotka näytetään web-käyttöliittymän kautta.

Käännösprosessissa palvelin hakee versionhallinnasta lähdekielisen sovelluksen ja käynnistää käännöksen. Kaikki käännöksestä saadut tulokset on mahdollista nähdä web-käyttöliittymän lokista, joten käännöksen epäonnistuessa voidaan virheen syy löytää helposti. Valmiit binääripaketit kopioituvat palvelinkoneelle, josta ne ovat web-selaimen kautta kaikkien saatavissa.



Kuva 3. Jatkuva integraatio -prosessi Pulse-käännöspalvelinympäristössä.

Jatkuva integraatio -prosessissa sovellukselle pitää ajaa myös testejä, jotta voidaan varmistua versionhallinnassa olevan sovelluksen toimivuudesta. Testien luomiseen A-projektissa käytetään Unittest++-kirjastoja. Unittest++ on yksikkötestien tekemiseen

C++-kielellä tarkoitettu sovelluskehys (Unittest++ 2010). Testit ajetaan käännöspalvelimilla onnistuneen sovelluskäännöksen jälkeen ja tuloksena saadaan xml-raportti testien onnistumisesta. Pulse-palvelinohjelmisto osaa lukea testien tulokset xml-raportista ja esittää ne web-käyttöliittymässä, josta tulokset ovat kehittäjien luettavissa.

A-sovelluskehystä käännetään sekä Windows- että Linux-käyttöjärjestelmille ja prosessoriarkkitehtuureille kuten ARM:lle, PowerPC:lle ja x86:lle. Lisäksi ARM-prosessoriversioita on käytössä kahta eri tyyppistä, ja PPC-arkkitehtuurin kanssa on käytössä kaksi eri Linux-versiota: Linux 2.4 ja 2.6. Laitealustaympäristöjä, joille sovelluskehys täytyy pystyä kääntämään, on siis useita ja niitä tulee lisää sovelluskehystä käyttävien asiakasprojektien tarpeiden mukaan. Käännöspalvelimia on käytössä sekä Windows- että Linux-käyttöjärjestelmäympäristöille. Linux x86-palvelimilla voidaan sovellusta kääntää x86-laitealustan lisäksi ristikääntäjillä myös ARM- ja PowerPC-laitealustoille.

4.3 Kehitysympäristön käyttöönotto ja käyttäminen

A-projekti tarjoaa kehitystyökalut projektin kääntämistä varten. Kehitystyökalujen käyttöönoton helpottamiseksi on kehitystyökalut paketoitu VMWare-levykuvaan. Levykuvaa käytetään VMWare Player -ohjelmistolla. VMWare-levykuvan etuna on, että tietokoneella voidaan ajaa useampaa käyttöjärjestelmää samanaikaisesti (VMWare 2010). Tämä mahdollistaa sovelluksen testaamisen eri käyttöjärjestelmillä nopeasti. VMWare-levykuva sisältää Linux-käyttöjärjestelmän ja A-projektin käännöstyökalut kaikille tuetuille laitealustoille. Käyttäjän tarvitsee vain asentaa VMWare Player ja käynnistää sillä levykuva, jonka jälkeen Linux-ohjelmistokehitysympäristö on valmis käytettäväksi. Jos VMWare-levykuvaa ei haluta käyttää, ohjelmistosuunnittelijan on asennettava käännöstyökalut itse. Ohjeet tähän löytyvät projektin wiki-sivulta. Sivuilta löytyvät myös ohjeet Windows-kehitysympäristön asentamiseen.

Helpoin tapa aloittaa kehitystyö A-projektissa on ottaa käyttöön VMWare-levykuva ja hakea versionhallinnasta uusin versio sovelluksen lähdekielestä. Käännös käynnistetään yhdellä komennolla, joka kääntää kaikki A-kehitysympäristön komponentit. Käännöksen tuottamat binääriartefaktit ja kääntämisessä tarvittavat tiedostot, kuten C++:n käyttämät .h-tiedostot, kopioituvat käännöksessä käännösympäristön määrittämään kansioon. Komponenteilla on riippuvuuksia toisiinsa, joten kääntäminen tapahtuu ennalta määrätyssä järjestyksessä. Komponentteja voidaan kääntää myös yksittäin käännöskomennolle annettavalla käännöslipulla.

Koko A-projektin kääntäminen saattaa kestää joillakin tietokoneilla kymmeniä minuutteja. Myös integroitaessa muutoksia versionhallintaan saatetaan joutua kääntämään useita komponentteja uudestaan, mikä saattaa viedä paljon aikaa. Koska käännöspalvelin kääntää sovelluksen aina, kun versionhallintaan tehdään muutoksia, voidaan käännöspalvelimen tuottama A-kehityspaketti kopioida kehittäjän omaan käännösympäristöön ja käyttää sitä kehittäjän muutosten testaamiseen muiden komponenttien kanssa.

4.4 Laitealustariippumaton käännösjärjestelmä

A-käännösympäristössä käytetään CMake-käännöstyökalua, joka on suunniteltu tukemaan lähdekielisen sovelluksen kääntämistä monella eri laitealustalla. CMake on geneerinen käännösjärjestelmä, jota voidaan käyttää minkä tahansa tyyppisen sovelluksen kääntämiseen. CMake toimii niin, että se luo projektista laitealustaspesifisen käännösjärjestelmän, jota se käyttää varsinaisen käännöksen tekemiseen. Esimerkiksi Linux:lle ja Mac OS X:lle oletuskäännösjärjestelmä on GCC-käännösjärjestelmä. Muita tuettuja käännösjärjestelmiä ovat Microsoft-kääntäjät, Borjand, Watcom, MinGW ja geneerinen Unix Makefile –järjestelmä. (Thelin 2007: 457–459.)

A-käännösympäristö tukee PC-koneilla Linux- ja Windows-käyttöjärjestelmiä. Ohjelmistokehittäjät käyttävät kuitenkin yleensä Linux-käyttöjärjestelmää, sillä sitä käytetään

tällä hetkellä myös tuetuissa sulautetuissa näyttölaitteissa. A-sovelluskehykseltä vaaditaan tuki myös Windows-käyttöjärjestelmälle, koska osa asiakasprojekteista käyttää tai tulee jatkossa käyttämään näyttölaitteissaan Windows-käyttöjärjestelmää. Windows-tuen etuna on myös, että sillä luotua referenssisovellusta on helppo käyttää projektin esittelyyn ja opetustarkoituksessa.

5. A-KEHITYSYMPÄRISTÖN ARVIOINTI JA PARANNUSEHDOTUKSET

Tässä luvussa käsittelen A-käännösjärjestelmässä havaitsemiani ongelmia. Esittelen ongelmiin kehittelemiäni ehdotuksia, joilla käännösjärjestelmän toimintaa voidaan parantaa. Esittelemäni ehdotukset eivät sisällä valmiita ratkaisuja, vaan pyrkimykseni on esittää tapoja, kuinka ehdotuksia kannattaa lähteä toteuttamaan. Työssä pystyn näin löytämään ratkaisuja useampaan ongelmaan.

5.1 Tapaustutkimuksen järjestelyt

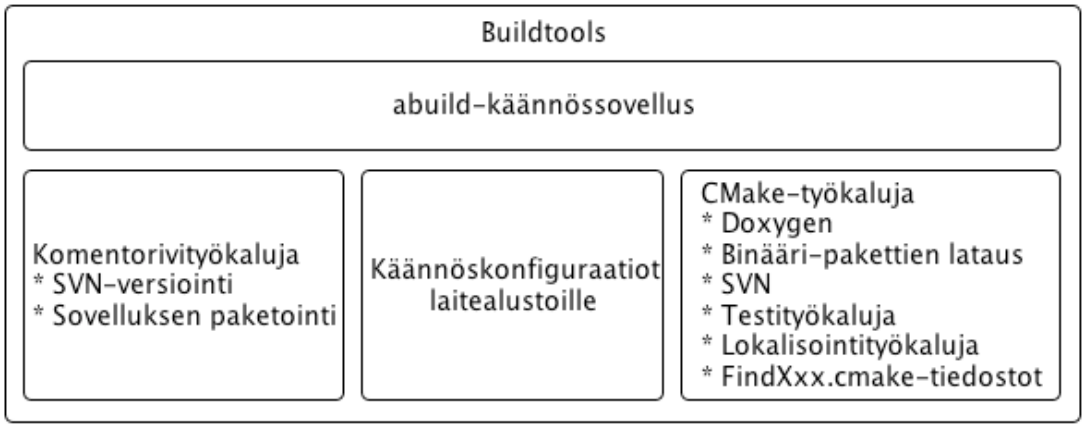
Tulin A-projektiin mukaan auttamaan käännösympäristön kehityksessä ja käännöspalvelimien ylläpitotehtävissä. Yksi työtehtävistäni on käännösmanagerina toimiminen, mikä tarkoittaa, että vastaan työssäni projektin käännöspalvelimien toimivuudesta sekä viikoittaisien ja kuukausittaisien käännöspakettien tekemisestä. Käännösympäristön kehitystyöhön kuuluu muun muassa erilaisten työskentelyä tukevien työkalujen tekeminen, käännöstyökalujen päivittäminen ja testien automatisointi. Toimenkuvaani kuuluu myös ongelmien havainnointi, parannusehdotusten tekeminen ja parannusten toteuttaminen virhetilanteiden ennalta ehkäisemiseksi.

Työskenteleminen käännösmanagerina on tarjonnut hyvät puitteet tapaustutkimuksen tekemiselle ja parannusehdotusten suunnittelemiselle. Olen saanut tiedon käännösympäristössä havaituista ongelmista monesti ensimmäisenä ja olen päässyt heti miettimään ongelmiin ratkaisuja. Järjestelmän testaamisen olen tehnyt A-projektin tarjoamalla Ubuntu Linux VMWare -levykuvalla.

5.2 Käännösympäristön automaation toiminta

Lähdekielisen ohjelmiston kääntäminen toimivaksi järjestelmäksi on usein monimutkainen prosessi, johon kuuluu lähdekielen kääntäminen, tiedostojen siirteleminen ja jaettavan sovelluspaketin luominen (Fowler 2006). A-käännösautomaatiossa käytetään prosessin apuna käännöstyökalupakettia, joka on nimetty Buildtools-paketiksi. Buildtools-paketti sisältää abuild-käännössovelluksen, joka hoitaa käännösympäristön asetukset, ja valmiita CMake-konfiguraatioita, joihin on asetettu projektin käyttämiä parametrimäärityksiä sekä makroja[1]. Lisäksi abuild-käännössovellus tarjoaa erilaisia toimintoja kehitystyön tueksi, kuten yksittäisten komponenttien kääntäminen, edellisten käännöstulosten poistaminen ja API-dokumenttien luominen. Buildtools-paketin tarjoamia parametrimäärityksiä ja makroja voidaan käyttää komponenttikohtaisissa CMake-konfiguraatioissa muun muassa tiedostojen asentamisessa sekä testien ajamisessa. Näin yksittäisten komponenttien konfiguraatioissa voidaan määrittää, mitä asennetaan, mutta käännösympäristön parametrit määrittävät, mihin asennus tehdään. Asiakkaiden projekteille suositellaan myös Buildtools-paketin käyttöä, jotta heidän ei tarvitse luoda omaa käännösympäristöään. Tällöin kaikkien A-projektin aliprojektien käännösympäristöt pysyvät yhtenäisenä. Yhtenäisen käännösympäristön käyttäminen tukee myös kehittäjien siirtämistä projektista toiseen, koska yhdessä A-projektissa työskennelleellä ohjelmistosuunnittelijalla ei ole oppimiskynnystä käyttää toisen A-projektin käännösympäristöä. A-projektin käännösympäristön käyttäminen ei kuitenkaan ole vaatimus projektin ohjelmistokirjastojen käyttämiselle.

[1] Makro on CMake-konfiguraatiokielessä käytetty lähdekielen komentoja sisältävä lohko, jota voidaan käyttää yhdellä kutsulla suorittamaan makron sisällä määritetyn joukon kutsuja.



Kuva 4. Buildtools-paketti.

Projektin kääntämisessä on kaksi vaihetta: lähdekielen kääntäminen konekieliseksi ja tiedostojen asentaminen. CMake-konfiguraatiossa kerrotaan käännöstyökalulle käännöksen tekemiseen ja tiedostojen asentamiseen tarvittavat tiedot. Käännöksen tekeminen vaatii konfiguraatioon asetettavaksi käännettävien tiedostojen nimet, käytettävien ohjelmistokirjastojen nimet, tieto siitä, käännetäänkö sovellus vai ohjelmistokirjasto ja käännöslippujen asettaminen. Alla on esimerkki CMake-konfiguraatiosta, jolla voidaan kääntää yksittäinen komponentti käyttäen Qt-ohjelmistokirjastoja ja projektin GUI-nimistä komponenttia.

```
find_package(Qt4 REQUIRED)
find_package(GuiPackage REQUIRED)

include_directories(
    ${GUI_INCLUDE_DIR}
    ${QT_INCLUDE_DIR}
)

set (BASE_LIBRARIES
    ${QT_LIBRARIES}
    ${GUI_LIBRARIES}
)
```

```
set_compile_flags_if_win32(Runtime   "-D_CRT_NONSTDC_NO_DEPRECATE
-DQ_CC_MINGW ${QT_DEFINITIONS}")

# -- Compile main executable
add_executable(runtime main.cpp application.rc)

target_link_libraries(runtime
    RuntimeLib
    ${BASE_LIBRARIES}
    ${QT_MAIN_LIBRARY}
)
```

Esimerkki 1. CMake-konfiguraatio sovelluksen kääntämistä varten

Buildtools -paketti määrittää tällä hetkellä yhden CMake-makron yksinkertaistamaan ohjelmistokirjastojen ja ohjelmistojen asentamista. Tämä on riittävä asennus mekanismi useimmille komponenteille, mutta usein tarvitaan myös muiden tiedostojen asentamista. Tällaisia ovat esimerkiksi CMake:ssa käytettyt FindXXX-konfiguraatiot[2], kuvat ja konfiguraatio-tiedostot. Näiden asentamista varten Buildtools-paketissa määritetään asennuspolun sisältäviä muuttujia, joita käytetään komponenttien kopiointikutsussa.

```
install   (DIRECTORY   ${CMAKE_CURRENT_SOURCE_DIR}/../bin/runexe/
DESTINATION ${INSTALL_DIR}
         PATTERN "*"
         PATTERN ".svn" EXCLUDE
         PATTERN "RUNEXE_LICENSE.txt" EXCLUDE
         PERMISSIONS    OWNER_READ    OWNER_WRITE    OWNER_EXECUTE
GROUP_READ WORLD_READ)
```

Esimerkki 2. Kansion asennus esimerkkisovelluksen CMake-konfiguraatiossa.

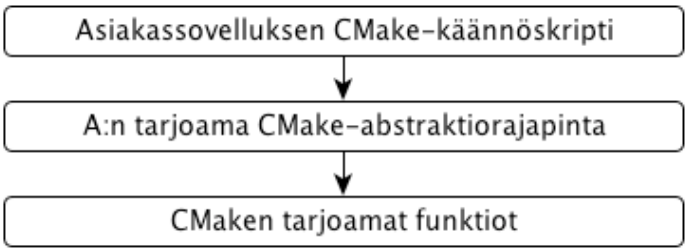
[2] FindXXX-konfiguraatio on CMake-sovelluksen tapa määritellä toiminnot tiedostojen etsimiseen. Projektissa luodaan jokaiselle komponentille FindXXX-konfiguraatio, jossa määritetään, miten komponentin tiedostot löytyvät.

Buildtools-paketin ongelma on tällä hetkellä CMake-konfiguraatioiden osaamisvaatimus. CMake on edelleen melko uusi työkalu, joten kaikilla ohjelmistosuunnittelijoilla ei ehkä ole tarvittavaa osaamista sen käyttämiseen. CMake saattaa siis muodostaa asiakasprojekteille kynnyksen A-käännösympäristön käyttöönotolle. Oppimiskynnyksen vähentämiseksi ehdotan yhtenäisen makro-kirjastokokonaisuuden kehittämistä A-käännösympäristöön. Makro-kirjastolla luotaisiin abstraktiokerros CMake-funktiokutsujen ja asiakasprojektien CMake-konfiguraatioiden välille. Abstraktiokerroksen avulla ohjelmoijien ei tarvitse tietää toteutuksen yksityiskohtia (Logan 2008: 14–15). Abstraktiokerroksella voitaisiin yksinkertaistaa asiakasprojektien CMake-konfiguraatioiden luomista ja tarjota valmiita ratkaisuja, joissa mahdolliset ongelmat on otettu huomioon. Esimerkiksi kopioitaessa versionhallinnassa säilytettävää kansiota täytyy kopiointikutsuun muistaa laittaa SVN-versionhallinnassa käytetyn piilokansion (.svn) ohituskäsky, jotta se ei kopioituisi pakettiin mukaan. Valmiit ratkaisut tällaisten ongelmatilanteiden ratkaisemiseen säästäisivät ohjelmistokehittäjiltä aikaa.

```
a_install (DIRECTORY ../bin/runexe/ DESTINATION A_INSTALL_DIR
      PATTERN "RUNEXE_LICENSE.txt" EXCLUDE
)
```

Esimerkki 3. Näin asentaminen voitaisiin tehdä käyttämällä abstraktia rajapintaa.

Abstraktiokerroksella voitaisiin ratkaista myös CMake-työkalun päivittämiseen liittyvät ongelmat. CMake-työkalun päivittäminen uuteen versioon saattaa muuttaa CMake-funktioiden kutsuja, jolloin asiakasprojektien CMake-konfiguraatiot vaatisivat funktiokutsujen muuttamisen toimiakseen päivitetyllä käännösympäristöllä. Abstraktiokerroksella vältettäisiin asiakasprojekteissa CMake-funktioiden kutsujen käyttäminen suoraan, jolloin CMake-työkalun päivitys vaatisi mahdollisia muutoksia ainoastaan abstraktiokerrokseen.



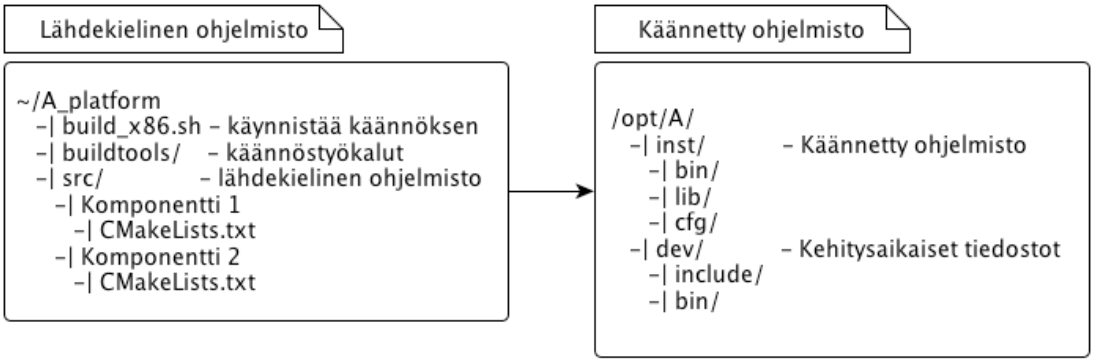
Kuva 5. Kuvaus abstraktiosta, jossa CMake-funktioiden kutsuminen korvataan abstraktiorajapinnalla.

5.3 Käännösympäristön käyttäminen kehittäjän näkökulmasta

Käännösprosessin on oltava automatisoitu niin, että se tekee kaiken tarvittavan toimivan ohjelmiston luomiseksi. Käännösprosessin on oltava myös yksinkertainen, koska jos prosessi vaatii pitkien käskyjen kirjoittamista, saattaa työhön tulla helposti virheitä. (Fowler 2006.) A-käännösympäristössä eri laiteympäristöille kääntäminen on tehty helpoksi luomalla jokaisen laitealustan kääntämiseksi oma shell- tai bat-sovelluksen, joka voidaan käynnistää konsolisovelluksella tai asettaa käynnistymään käytetyn IDE:n kautta. Käännössovellukset välittävät käännöksessä tarvittavat konfiguraatiotiedostot varsinaisen käännöksen käynnistävälle abuild-käännösojelmalle. Abuild on Python-kielellä ohjelmoitu käännösohjelma, joka on suunniteltu tukemaan käännöstyötä monelle laitealustalle. Se on myös suunniteltu komponenttikohtaiseen kääntämiseen, jossa komponenttilista välitetään ohjelmalle parametrina. Komponenttilista sisältää komponenttien nimet, sijainnit kansiohierarkiassa ja komponenteille tehtävät toimenpiteet. Lisäksi abuild-käännösojelmalle välitetään parametrina konfiguraatiotiedosto, jossa määritetään muun muassa käännöksessä käytettävän kääntäjän sijainti, käännösparametrit ja ympäristömuuttujat.

Käännösohjelmisto on helppokäyttöinen ja se sisältää silti riittävästi toimintoja erilaisten käännösten tekemiseen. Koko projektin kääntäminen voidaan tehdä yhdellä komen-

nolla, joka kääntää kaikki komponentit komponenttilistan järjestyksessä, tai komponentteja voidaan kääntää yksittäin lisäämällä komennon loppuun komponenttikäännösparametri ja komponentin nimi. Komponenttien kääntäminen erikseen kuitenkin vaatii, että komponentit, joihin käännettävällä komponentilla on riippuvuuksia, on käännetty ensin. Komponenttien käännöstulokset kopioituvat käännöspakettiin, josta muut komponentit voivat käyttää niitä riippuvuuksiensa ratkaisemiseen.



Kuva 6. Lähdekielisen ohjelmiston kääntäminen ja kopioiminen käännöspakettiin.

Versionhallintaan tehdään jatkuvasti muutoksia usean suunnittelijan toimesta, joten kun yksittäinen suunnittelija haluaa integroida omat muutoksensa versionhallintaan, on hänen päivitettävä lähdeohjelmisto versionhallinnasta omalle koneelleen ja käännettävä projekti kokonaan. Näin voidaan tarkistaa, vaikuttavatko muutokset versionhallinnassa suunnittelijan omiin muutoksiin. Koko projektin kääntäminen on kuitenkin hidasta, mikä hidastaa integraatiotyön tekemistä. Integraatiotyön nopeuttamiseksi ehdotan, että yksittäisen komponentin käännöksessä voitaisiin käyttää käännöspalvelimen tuottamia käännöspaketteja. Koska käännöspalvelin tekee ohjelmistosta uuden käännöspaketin aina, kun versionhallintaan tehdään muutos, on palvelimen tuottamat käännöspaketit käännetty aina uusimmasta lähdeohjelmiston versiosta.

Käännöspaketin asentamista varten ehdotan, että abuild-käännösohjelmistoon tehdään uusi optio käännöspakettien kopioimiseksi. Tein esimerkkisovelluksen käännöspaketin asentamiseksi konsolisovelluksella, joka voidaan ottaa kehittäjien käyttöön, ennen kuin abuild-käännösoptio on saatu tehtyä. Esimerkkisovellus löytyy liitteestä numero 1. Esimerkkisovellus hakee uusimman A-käännöspaketin palvelimelta ja purkaa sen käännösympäristön käyttämään hakemistoon. Esimerkkisovellus voidaan käynnistää yhdellä komennolla:

```
./install_x.sh xxx_arm
```

Tämä hakee viimeisimmän Linux ARM -laitealustalle käännetyn A-paketin ja asentaa sen käännösympäristön määräämään paikkaan. Asennuksen jälkeen voidaan mikä tahansa komponentti kääntää ilman, että muita komponentteja tarvitsee kääntää ensin. Kehittäjän täytyy kuitenkin muistaa tarkistaa, että lähdeohjelmisto on kääntynyt integrointipalvelimella ja testit ovat menneet läpi, jotta asennettu A-paketti on tarpeeksi uusi integrointityön tekemiseksi.

5.4 Projektin käännösympäristön käyttöönotto

Käännösympäristön käyttöönotto onnistuu helpoiten käyttämällä A-projektin ylläpitämää VMWare-levykuvaa, johon käännöksessä tarvittavat työkalut on asennettu. VMWare-levykuvan käyttöä esiteltiin kappaleessa 5.3. Monet A-projektin kehittäjät kuitenkin käyttävät mieluummin tietokoneen varsinaiseksi käyttöjärjestelmäksi asennettua Linuxia, koska se toimii nopeammin. Tällöin käännösympäristössä tarvittavat työkalut täytyy asentaa erikseen. Myös Windows-käännösympäristön asentaminen on tehtävä erikseen, koska Windows-käyttöjärjestelmälle ei ylläpidetä VMWare-levykuvaa. Fowlerin (2006) mukaan kenen tahansa pitäisi pystyä hakemaan tietokoneella versionhallinnasta lähdekielinen sovellus ja saada tietokoneelleen toimiva järjestelmä ajamalla yksi komento. Projektin wiki-sivulla on ohjeet A-käännösympäristön käyttämiseksi ja työkalupakettien asentamiseksi. Kääntämisessä tarvittavia työkalupaketteja säilytetään projektin verkkolevyllä, jolta ne voi ladata URL-osoitteella. A-projektin työkalupaketteihin

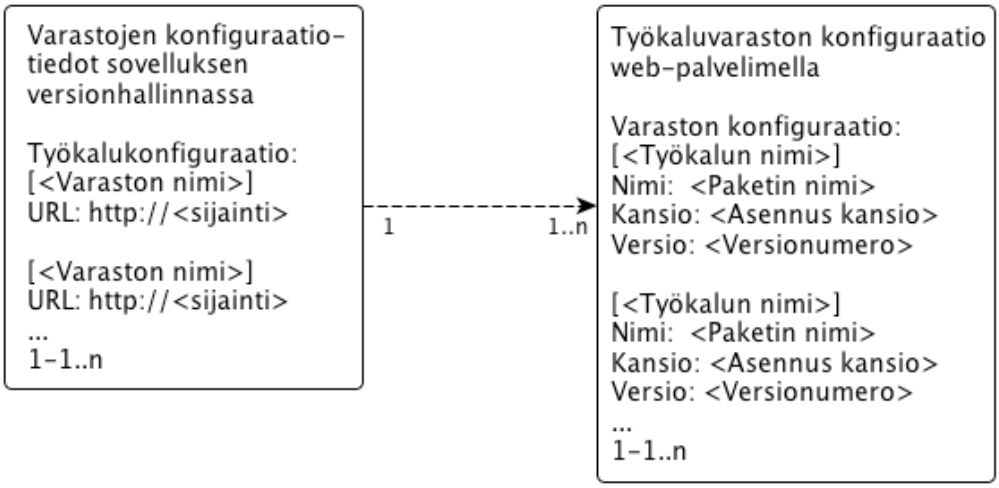
kuuluu yleensä Qt-sovelluskehyskirjastot ja työkaluketju, joka sisältää kääntäjän ja laitealusta sisältämät ohjelmistokirjastot. Muita A:n aliprojektien käyttämiä työkalupaketteja, joita ylläpidetään, ovat Python ja Poco. Ohjeet pakettien asentamiseksi sisältävät monta kohtaa, joissa neuvotaan, mitä komentoja ajamalla asennus onnistuu. Vaikka komentojen kopioiminen wiki-sivulta ja ajaminen komentorivillä on helppoa, on asennustyö silti aikaa vievää ja virhealtista. Wiki-sivujen ohjeiden ylläpito vaatii myös päivitystyötä aina, kun työkaluja päivitetään ja työkalujen nimet sekä sijainnit muuttuvat.

Etsiessäni toimivaa ratkaisua työkalujen asentamiseen tarkastelin seuraavia Linux-käyttöjärjestelmissä käytettyjä asennustyökaluja: RPM, Debian-package ja dpkg. Näiden käyttö vaati asennettavan ohjelmiston paketoimisen asennustyökalun käyttämään muotoon ja asennuskonfiguraatioiden luomisen. Windows-työkalujen asennus ei kuitenkaan olisi onnistunut näillä asennusohjelmilla helposti ja pakettien tekeminen vaati monimutkaisen paketointiprosessin ylläpitämistä sekä erillisen palvelinsovelluksen käyttöönottoa. A-projektin jakamien käännöstyökalujen asentaminen vaatii työkalupakettien purkamisen ja muutamien käyttöjärjestelmän ympäristömuuttujien asettamista. Tähän tarkoitukseen sopivan asennusohjelman tekeminen tuntui helpoimmalta ratkaisulta.

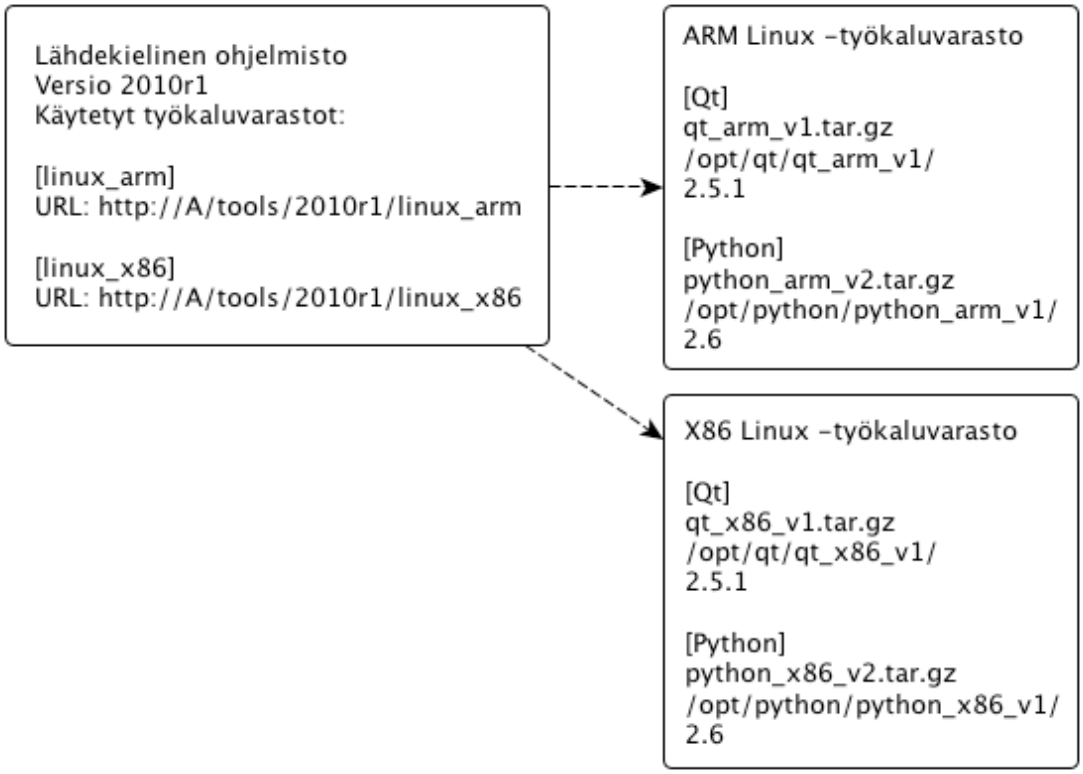
Asennusohjelman vaatimuksena on toimiminen Linux- ja Windows-käyttöjärjestelmissä, joten ohjelman lähdekielen täytyy olla laitealustariippumatonta. Käännöstyökaluja päivitetään ajoittain uudempiin versioihin, joten asennustyökaluun pitää pystyä helposti konfiguroimaan uuden paketin nimi, sijainti ja tieto siitä, mihin se asennetaan. Koska A-projektin lähdekielinen ohjelmisto versioidaan aika-ajoin, pitää myös työkalupaketit samalla versioida. Näin toimimalla tiedetään, mikä työkalupaketti milläkin versiolla on käytössä, ja taataan versioidun ohjelmiston ja työkalupaketin yhteensopivuus.

Suunnitelmani on luoda työkalujen asentamiseen sovellus, joka voidaan konfiguroida asentamaan eri ohjelmistoversioille niille tarkoitetut työkalupaketit. Työkalu käyttäisi konfiguraatiota, joka asennetaan versionhallintaan. Versioidessa ohjelmistoa sen hetki-

nen konfiguraatiotieto jäisi näin talteen versionhallintaan. Konfiguraatio sisältäisi tiedon siitä, mitä työkaluvarastoja käytetään ja missä työkalut sijaitsevat. Työkaluvarastot on jaoteltu laitearkkitehtuurien mukaan, ja jokainen varasto sisältää kutakin laitealustaa varten tarvittavat käännöstyökalut. Työkaluvarasto sisältäisi myös konfiguraatiotiedoston, joka kertoo, mitä tiedostoja työkaluvarastossa on ja mihin kansioon ne asennetaan kehittäjän tietokoneella.



Kuva 7. Kuvaus konfiguraatiotiedostojen sisällöstä ja riippuvuuksista



Kuva 8. Esimerkkikuva mahdollisista konfiguraatiotiedoista.

Asennussovelluksen täytyy pystyä toimimaan eri laitealustoilla, joten se on tehtävä laitealustariippumattomalla kielellä. Python-kielellä voidaan luoda laitealustariippumattomia sovelluksia, se sisältää toiminnot asennussovelluksen luomiseksi. Lisäksi sitä on jo käytetty A-käännösympäristössä, joten sen käyttäminen asennussovelluksen toteuttamiseksi vaikuttaisi hyvältä vaihtoehdolta.

5.5 Tuki laitealustariippumattomalle ohjelmistokehitykselle

A-käännösympäristöllä voidaan kääntää A-ohjelmisto monelle eri laitealustalle. A-ohjelmiston kehittäjät käyttävät kuitenkin pääsääntöisesti Linux x86 -ympäristöä, koska testaaminen kaikilla tuetuilla laitealustoilla on aikaa vievää. Ohjelmiston kääntyminen

ja testien ajaminen muilla laitealustoilla varmennetaan integraatiopalvelimella ja testaajien toimesta. Integraatiopalvelimella testit ajetaan tällä hetkellä ainoastaan Windows x86- ja Linux x86 -laitealustoilla, koska testauksen automatisointi olisi hankala toteuttaa ja ylläpitää käytettäessä näyttölaitteita, joissa on vähemmän laskentatehoja. Testien ajaminen sulautetuilla näyttölaitteilla hidastaisi myös käännöspalvelimien toimintaa. Näyttölaitteet sisältävät Linux-käyttöjärjestelmän, joten käyttämällä niitä testaamisessa ei todennäköisesti edes löydettäisi ohjelmistovirheitä juuri enempää kuin Linux x86 -laitealustalla testattaessa. Vastaan on kuitenkin tullut muutamia laitealustakohtaisia ongelmia, jotka ovat ilmenneet vasta, kun testaajat ovat tehneet testejä näyttölaitteilla. Nämä ongelmat olisi pystytty havaitsemaan aiemmin automaattisella testauksella, jos sellainen olisi ollut. Ongelman ratkaiseminen olisi saattanut tällöin olla helpompaa.

Sovelluksen kääntäminen Windows-käyttöjärjestelmälle ja testaaminen sillä on kehittäjien näkökulmasta ollut ongelmallista, sillä kaikilla kehittäjillä ei ole ollut mahdollisuutta käyttää kahta käyttöjärjestelmää yhtä aikaa. Windows-kääntäminen ja automatisoitu testaaminen on tehty ainoastaan integraatiopalvelimella, ja kehittäjät ovat voineet tarkistaa käännöksen toimivuuden integraatiopalvelinsovelluksesta web-käyttöliittymän kautta. Virheiden korjaaminen on kuitenkin ollut ongelmallista, jos kehittäjällä ei ole ollut käytettävissä Windows-käyttöjärjestelmää. Tämän takia kehittäjä on joskus joutunut tekemään korjauksen versionhallintaan testaamatta korjauksen toimivuutta. Korjauksen toimivuus on varmistettu käynnistämällä käännös integraatiopalvelimella. Kääntäminen integraatiopalvelimella on kuitenkin aikaa vievää, sillä integraatiopalvelin kääntää koko projektin kerrallaan. Tämä johtaa siihen, että korjauksen tekemiseen kuluu aikaa ja ongelma jää versionhallintaan korjaamatta pitkäksikin aikaa. Seuraavissa alaluvuissa esitän kaksi tapaa helpottaa korjaustyötä. Toisessa tavassa käytetään Pulse-sovelluksen personal build -toimintoa ja toisessa Pulsen tarjoamaa API-rajapintaa

5.5.1 Integraatiotyön helpottaminen Personal build -toiminnon avulla

Pulse-sovelluksessa on personal build -toiminto, jolla kehittäjän koneella olevat ohjel-

miston lähdekieleen tehdyt muutokset voidaan kääntää integraatiopalvelimella siirtämättä muutoksia versionhallintaan. Personal build toimii niin, että integraatiopalvelin hakee ensin versionhallinnasta sovelluksesta puhtaan version, johon liitetään mukaan kehittäjän työstämän version muutokset. Personal buildin käynnistävä sovellus käynnistetään kehittäjän tietokoneella, joka luo .diff-tiedostot lähdeohjelmiston muutoksista kehittäjän työasemalla ja lähettää ne integraatiopalvelimelle. Integraatiopalvelin hakee versionhallinnasta lähdekielisen ohjelmiston, asentaa muutokset .diff-tiedostoista ja käynnistää käännöksen. Näin ohjelmistokehittäjän paikallisella koneella olevat lähdekielisen ohjelmiston muutokset voidaan kääntää integraatiopalvelimella ilman, että muutokset siirretään versionhallintaan. (Zutubi 2010b.)

Testasin personal build -toimintoa A-käännösympäristössä. Personal buildin käynnistävä konsolisovellus piti ensin ladata Pulse-palvelinohjelmiston web-sivulta ja asentaa omalle koneelle. Sovelluksen ajamiseksi asetettiin ensin sovelluksen käyttämiseen vaaditut ympäristömuuttujat.

```
export PULSE_HOME=/opt/<käyttäjä>/pulse-dev
export PATH=$PULSE_HOME/bin:$PATH
```

Esimerkki 4. Ympäristömuuttujien asettaminen

Personal build -sovellus käyttää konfiguraatiotiedostoa nimeltä .pulse.properties. Sovellus lukee tiedostosta asetukset, kuten Pulse-palvelimen osoiteen, salasanat ja käynnistettävän Pulse-projektin nimen. Personal build käynnistetään konsolista näin:

```
pulse personal <Tiedosto 1> <Tiedosto 2> …
```

Esimerkki 5. Personal buildin käynnistäminen konsolista

Personal build -sovellus siirtää parametrina annettujen tiedostojen muutokset Pulse-palvelimelle ja käynnistää käännöstyön. Toinen vaihtoehto on jättää tiedostojen määrittäminen pois, jolloin kaikki projektin tiedostot käydään läpi ja kaikki muutokset siirretään integraatiopalvelimelle. (Zutubi 2010b.) Testasin myös tätä vaihtoehtoa, mutta sovellus epäonnistui muutosten siirtämisessä ensimmäisellä kerralla versionhallinnassa käytettyjen svn:external[3]-linkkien takia. Pulsen dokumenteista löytyi ohjeet siitä, kuinka svn:external-linkkien käytön rajoitukset voidaan poistaa muuttamalla konfigurointitiedostoa. Poiston jälkeen personal build -käännöksen tekeminen onnistui ongelmitta. Dokumenteissa kuitenkin varoitettiin, että svn:external-linkkien käyttäminen saattaa aiheuttaa ongelmia personal build -käännöksen tekemisessä.

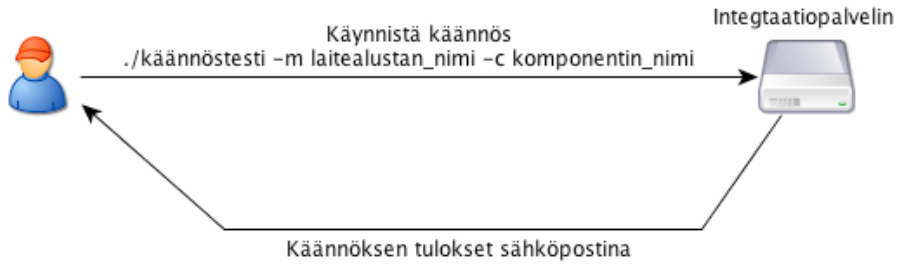
Personal build -käännöksen käyttäminen helpottaa kääntymisen testaamista eri laitealustoilla, koska kehittäjällä ei tällöin tarvitse olla kaikkia laitealustoja käytettävissään. Kääntäminen voidaan tehdä millä tahansa integraatiopalvelimen käytössä olevalla laitealustalla, joten myös Windows-käännöksen voi testata, vaikkei itsellä olisi käytettävissä Windows-käyttöjärjestelmää. Kääntäminen käännöspalvelimella on kuitenkin edelleen hidasta, jos koko projekti joudutaan kääntämään kokonaan. Käännöksen nopeuttamiseksi kannattaa ottaa käyttöön kappaleessa 5.3 esitetty tapa yksittäisen komponentin kääntämiseksi. Tällöin saadaan nopeasti testattua, ovatko tehdyt korjaukset toimivia kaikilla laitealustoilla.

5.5.2 Integraatiotyön helpottaminen konsolisovelluksen avulla

Integraatiotyön helpottamiseksi voitaisiin luoda komentoriviltä käytettävä sovellus, jolla käännöstyön voisi käynnistää integraatiopalvelimella komentorivikäskyllä. Kun versionhallintaan tulee muutoksia, A-integraatiopalvelin käynnistää automaattisesti Linux x86 -laitealustakäännöksen ja virhetilanteessa ohjelmistosuunnittelija saa virheen sähköpostilla. Jos kehittäjä haluaa varmistua, että omat muutokset toimivat myös jollain

[3] svn:externals on SVN-versionhallinnassa käytetty määritys, jolla eri versionhallinnoissa olevien kansioiden sisältö voidaan liittää toisiinsa.

muulla laitealustalla, on hänen käytettävä web-sivua käännöksen käynnistämiseen. Web-sivun käyttäminen taas vaatii sisäänkirjautumista web-sovellukseen, mikä voi tuntua hankalalta ja aikaa vievältä. Tarkistuskäännöksen käynnistäminen komentoriviltä olisi nopeaa, koska se ei vaatisi kuin yhden komennon kirjoittamisen. Samalla konsolisovelluksella olisi hyvä pystyä myös tarkastamaan käännöksen tulokset. Komentorivisovellus käyttäisi Pulse-sovelluksen tarjoamaa API-rajapintaa käskyjen välittämiseksi ja tulosten hakemiseksi integraatiopalvelimelta.



Kuva 9. Konsolisovellus integraatiotestin käynnistämiseksi.

5.6 Työkalu riippuvuuksien analysointiin

A-kehitysalustan asiakasprojektit voivat valita, mitä komponentteja he käyttävät ohjelmistokehitysalustasta. Uusia komponentteja luodaan projektiin jatkuvasti, ja riippuvuussuhteet saattavat muuttua. Riippuvuuksien tarkastelua varten olisi projektiin hyvä saada työkalu, joka loisi käännösten yhteydessä integraatiopalvelimella kaaviokuvan komponenttien välisistä riippuvuussuhteista. Kaaviota voitaisiin käyttää riippuvuussuhteiden analysointiin ja kehitysalustan oppimistarkoitukseen. Tässä luvussa esittelen kaksi tapaa kaaviokuvien luomiseksi automaattisesti. Ensimmäisessä alaluvussa näytän, kuinka ohjelmiston kirjastojen välisistä riippuvuuksista saadaan luotua kaavio. Toisessa alaluvussa esitän, kuinka sovelluksen C++-luokkahierarkiasta ja luokkien välisistä kutsuista luodaan kaaviot.

5.6.1 Depot- ja Graphviz-työkalut kirjastoriippuvuuksien kuvaamiseen

Kaaviokuvan luomiseen voidaan käyttää kahta Open Source -ohjelmistoa: Depdot:ia ja Graphviz:ia. Depdot on C++:lla ohjelmoitu komentorivisovellus, joka tuottaa Graphviz:n käyttämiä dot-tekstitiedostoja kirjastojen riippuvuuksien kuvaamiseen (Depdot 2010). Dot-tiedosto on tekstitiedosto, jossa on listattuna kaaviokuvan solmut ja solmujen riippuvuudet. Tässä tapauksessa solmut edustavat ohjelmistokirjastoja. Alla on esitettynä bash-sovellus, joka luo dot-tiedoston.

```
#! /bin/sh

for i in lib*.so
do
nm $i 2>/dev/null | deplib.pl | awk "{printf(\"$i|%s\n\",
\$0);}"
done | depdot > lib_dependencies.dot
```

Esimerkki 6. Shell-sovellus dot-tiedoston luomiseksi

Depdot-sovellus käyttää nm-sovelluksen tulostetta, joka muutetaan sovelluksen mukana tulleella Perl-kielisellä sovelluksella ja awk-sovelluksen avulla Depdotin ymmärtämään muotoon. Tämä toimenpide on tehty, jotta Depdot-sovellus ei olisi riippuvainen nm-sovelluksen tulosteesta. Esimerkiksi Solaris-käyttöjärjestelmässä käytetty nm-sovellus tulostaa riippuvuustiedot eri formaatissa kuin GNU-paketin nm-sovellus. Perl-sovellusta muokkaamalla voidaan tuloste muuttaa haluttuun muotoon. (Depdot 2010.) Awk-sovellus lisää tulosteeseen kirjaston nimen.

Graphviz on kaaviotiedon visualisointiin tarkoitettu sovellus, joka luo kuvattavien komponenttien riippuvuustiedosta kaaviokuvan. Graphvizia voidaan käyttää käyttöliittymäsovelluksena, jolla avataan dot-muotoisia tekstitiedostoja ja valitaan käyttöliittymästä

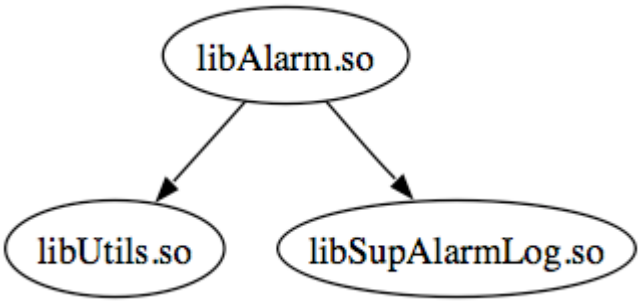
kaaviokuvan tyyppi. Erilaisia kaaviotyyppejä ovat muun muassa puukaavio ja relaatiomalli. Kaaviokuvien luontiin on Graphviz-projektissa tarjolla myös konsolisovellukset kaikille tuetuille kaaviotyypeille. (Graphviz 2010.) Depdot-sovelluksen tuloste voidaan siis tekstitiedoston sijasta ohjata myös suoraan Graphviz-konsolisovellukselle. Alla on esimerkki bash-sovelluksesta, jossa depdot-ohjelman tuloste ohjataan putkella suoraan Graphviz-konsolisovellukselle.

```
#! /bin/sh

for i in lib*.so
do
nm $i 2>/dev/null | deplib.pl | awk "{printf(\"$i|%s\n\",
\$0);}"
done | depdot | dot –Tjpg:cairo –Gmodel=subset >
lib_dependencies.jpg
```

Esimerkki 7. Kaaviokuvan luominen

A-projektin sovelluskirjastoja on kymmeniä, joten yhden kaavion luominen kaikista kirjastoista saa aikaan sotkuisen kuvan. Kuva kannattaa siis luoda ainoastaan tärkeimpien komponenttien riippuvuuksista. Olisi myös selkeää luoda jokaisen sovelluskirjaston riippuvuuksista kaavio, jotta yksittäisten kirjastojen riippuvuuksia olisi helpompi lukea. Tätä varten voidaan käyttää gvpr-sovellusta, jolla voidaan muokata kaaviotietoja konfiguraatioiden avulla. Gvpr-sovelluksen käytöstä voi lukea enemmän web-osoitteesta http://linux.die.net/man/1/gvpr.

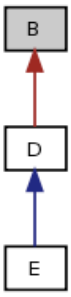


Kuva 10. Esimerkki Graphviz:n luomasta kaaviokuvasta.

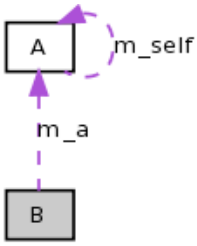
5.6.2 Kaaviokuvan automaattinen luominen luokkien periytymishierarkioista

A-kehitysympäristössä käytetään Doxygen-nimistä Open Source -sovellusta API-dokumentaation luomiseen lähdekielisestä ohjelmistosta. Doxygeniä voitaisiin käyttää myös luokkien periytymishierarkian kuvaamiseen. Doxygenissä on rakennettu tuki, joka luo kaaviot C++-luokkariippuvuuksista ja lisää ne dokumentaatioon. (Doxygen 2010.) Kaavioiden luomiseen Doxygen käyttää Graphviz-sovellusta.

Kaavioiden luominen on helppo asettaa päälle. Integraatiopalvelimelle, jolla Doxygen-dokumentit luodaan, tarvitsee ensin asentaa Graphviz-sovellus ja asettaa sen sijainti PATH-ympäristömuuttujaan. Tämän jälkeen Doxygenin konfiguraatioon asetetaan arvo `HAVE_DOT = YES`. Kaavioiden luomisesta voi lukea lisää Doxygenin Internet-osoitteesta http://www.doxygen.nl/diagrams.html



Kuva 11. Esimerkki Doxygenin luomasta luokkien periytymishierarkiaa kuvaavasta kaaviosta.



Kuva 12. Esimerkki Doxygenin luomasta luokkien välisiä kutsuja kuvaavasta kaaviosta.

6. JOHTOPÄÄTÖKSET

Tutkimuksessa tehtiin selvitystyö toimeksiantajan projektissa käytetyn ohjelmistokehitysympäristön toiminnasta ja projektin kehitystyöhön liittyvistä toimintatavoista. Lisäksi työssä selvitettiin kehitysympäristön käyttöön liittyviä ongelmia ja etsittiin ratkaisuja kehitysympäristön parantamiseksi. Tutkielma lähti liikkeelle aihealueen teoriaan ja aikaisempaan tutkimukseen tutustumisella. Tämän jälkeen käytiin läpi tutkittavan sovelluskehitysympäristön toimintatapoja ja verrattiin sitä kirjoissa esitettyyn teoriaan. Tutkimukseni pääpaino on tutkittavan kehitysympäristön parantaminen. Työn lopputuloksena esitettiin viisi ehdotusta, joilla voidaan helpottaa sovelluskehittäjien työtä, nopeuttaa kehitystyöprosessia ja saada arkkitehtuuri sekä siihen tulevat muutokset helposti kuvattua.

Projektin kehitysympäristön toimintaan liittyy olennaisesti jatkuva integraatio -prosessi ja sen apuna käytettävä integraatiopalvelinjärjestelmä, joten tutkimuksessa esitellään lyhyesti erilaisia integraatiopalvelinratkaisuja. Tutkimuksen teoriaosuus alkaa luvusta kolme ja se keskittyy laitealustariippumattoman sovelluksen kehittämiseen C- ja C++-kielillä ja eri tekniikoihin laitealustariippumattomuuden toteuttamiseksi. Luvussa kolme esitellään myös toimintatapoja mahdollisten ongelmien välttämiseksi käytettäessä eri laitealustoja. Seuraavassa luvussa on jatkuva integraatio -prosessin teoriaosuus, jossa käydään läpi prosessin toimintatapoja ja verrataan esitettyä teoriaa tutkittavan kehitysympäristön toimintatapoihin. Luvussa neljä käydään läpi sovelluskehitysprosessia ja siihen liittyviä toimintatapoja, integraatiopalvelinten teoriaa, kehitysympäristön käyttöönottoa ja käyttöä sekä laitealustariippumattomuuden tukemista käytettäessä jatkuva integraatio -prosessia.

Tutkimukseni työosuudessa arvioidaan tutkittavaa kehitysympäristöä ja esitetään viisi kehitysympäristön parannusehdotusta. Ensimmäisenä parannusehdotuksena esitetään CMake-abstraktiokerroksen luomista käännöskonfiguraatioiden tekemisen helpottami-

seksi. Projektin kääntämisessä tarvittavien konfiguraatiotiedostojen luominen vaatii CMake:n hallintaa. Abstraktiokerroksen avulla voitaisiin konfiguraatioiden tekemistä yksinkertaistaa. Abstraktiokerros tarjoaisi yksinkertaistetun rajapinnan käännöskonfiguraatioiden luomiseen ja piilottaisi kääntämisessä tarvittavat CMake-funktioiden kutsut abstraktiokerrokseen.

Toinen parannusehdotus on binääripakettien lataamisoptio käännösympäristöön. Sovelluskomponenttien kehittäminen vaatii, että versionhallinnasta otettu lähdekielinen sovellus on ensin käännettävä kokonaan. Projekti on käännettävä, jotta binäärikirjastojen väliset linkitykset saadaan luotua käännöksen aikana. Koko projektin kääntäminen on kuitenkin hidasta. Binääripakettien automaattisella lataamisoptiolla saataisiin integraatiopalvelinten tuottamat binääripaketit asennettua kehittäjän työkoneelle. Tämän jälkeen kehittäjä voisi alkaa kääntää ja kehittää yksittäisiä komponentteja ilman, että koko projektia tarvitsee ensin kääntää. Tällä nopeutettaisiin kehitystyön aloittamista.

Kolmantena parannuksena ehdotetaan käännösympäristön käyttöönoton nopeuttamista asennussovelluksen avulla. Käännösympäristön asentaminen vaatii vähintään kahden binääripaketin asentamista jokaista tuettavaa laitealustaa kohden. Lisäksi kehitysympäristössä tarvitaan CMake-sovellus. Binääripakettien asennusta helpottaisi, jos käytettävissä olisi asennussovellus, joka hoitaisi asennuksen yhdellä komennolla.

Neljäntenä parannusehdotuksena esitetään Pulse-käännöspalvelinohjelmistossa olevan private build -toiminnon käyttöönottoa. Private build -toiminto mahdollistaa kehittäjän koneella olevan lähdekielisen ohjelmiston kääntämisen integraatiopalvelimilla. Tätä käyttämällä voidaan sovellusmuutoksien kääntymistä testata eri laitealustoilla nopeasti ilman, että kehittäjän tarvitsee asentaa laitealustan käännösympäristöä tietokoneelleen.

Viides ehdotus on kaaviokuvien luominen sovelluskirjastojen välisistä riippuvuuksista ja luokkien periytymishierarkiasta. Kaaviokuvien avulla projektin arkkitehtuuri olisi helpommin, nähtävissä ja sen avulla voitaisiin tarkistaa, että komponenttien riippuvuudet pysyvät arkkitehtuurin mukaisina kehitystyön aikana.

Työssä käsitellyssä projektissa on useita kehittäjiä, joten kehitysympäristön toimivuus ja helppokäyttöisyys on tärkeää. Toimivalla kehitysympäristöllä voidaan säästää kehittäjien aikaa ja ratkaista ennalta kehitystyössä esiin tulevia ongelmatilanteita. Kehitysympäristön kehitystarpeet on kuitenkin mitattava projektin tarpeiden mukaan. Kaikkia kehitysideoita ei kannata lähteä toteuttamaan, jos niiden toteuttaminen vie liikaa aikaa projektin muulta kehitystyöltä ja jos saatava hyöty ei ole tarpeeksi suuri.

Toiveet projektin kehitystarpeista tulevat pääasiassa kehitysalustaa käyttäviltä aliprojekteilta. Tässä tutkimuksessa esitettyjen parannusehdotuksien toteuttamisesta tullaan päättämään aliprojektien tarpeiden mukaan. Parannusehdotuksista toinen on kuitenkin jo toteutuksessa, ensimmäinen ja viides on jo laitettu toteutuslistalle ja muiden toteuttamista ehdotetaan aliprojekteille.

LÄHDELUETTELO

BuildBot (2010) *BuildBot Manual 0.7.12*. Saatavana World Wide Webistä. <URL:http://djmitche.github.com/buildbot/docs/0.7.12/> (siteerattu 5.4.2010)

C.Bailey Edward (1997). *Maximum RPM*. NC: Red Hat, Inc. 450 s. ISBN 978-0-672-31105-5

CruiseControl (2010) *CruiseControl*. Saatavana World Wide Webistä. <URL:http://cruisecontrol.sourceforge.net/index.html> (siteerattu 30.3.2010)

Depdot (2010). *Library Dependency Graphs*. Saatavana World Wide Webistä. <URL:http://sourceforge.net/projects/depdot/> (siteerattu 30.3.2010)

Doxygen (2010). *Graphs and diagrams.* Saatavana World Wide Webistä. <URL:http://www.stack.nl/~dimitri/doxygen/diagrams.html> (siteerattu 3.4.2010)

Duvall Paul M., Matyas Steve ja Glover Andrew (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison Wesley Professional. 336 s. ISBN 978-0-321-33638-5.

Fowler Martin (2006). *Continuous Integration*. Saatavana World Wide Webistä <URL:http://martinfowler.com/articles/continuousIntegration.html> (siteerattu 23.2.2010)

Graphviz (2010). *Graph Visualization Software.* Saatavana World Wide Webistä. <URL:http://www.graphviz.org/> (siteerattu 30.3.2010)

Järvinen Pertti, Järvinen Annika (2000). *Tutkimustyön metodeista.* Tampereen yliopisto paino Oy. Juveness-Print. ISBN 951-97113-8-4.

Logan Syd (2008). *Cross-Plattform Development in C++*. MA: Adisson Wesley. 547 s. ISBN 987-0-321-24642-4.

Meyers Scott (1998). *Effective C++: 55 Specific Ways to Improve Your Programs and Designs.* 1. painos. MA: Adisson Wesley. s. 256. ISBN 0-201-92488-9.

Palviainen Jarkko (2009). *Introducing Continuous Integration for C and C++ Software Development Projects on Linux Platform.* Saatavana World Wide Webistä. <URL:http://urn.fi/URN:NBN:fi-fe200911122336> (siteerattu 4.4.2010)

Poco (2010). *Poco C++ Libraries: About.* Saatavana World Wide Webistä <URL:http://pocoproject.org/> (siteerattu 3.4.2010)

Prata Stephen (2005). *C++ Primer Plus.* 5. painos. IN: Sams Publishing. s. 1202. ISBN 978-0-672-32697-4.

Qt (2010). *Qt - Cross-platform application and UI framework.* Saatavana World Wide Webistä <URL:http://qt.nokia.com> (siteerattu 3.4.2010)

Raghavan P., Lad Amol, Neelakandan Sriram (2006). *Embedded Linux System Design And Development.* FL: Auerbach Publications. 432 s. ISBN 978-0-849-34058-1.

Thelin Johan (2007). *Foundations of Qt Development.* NY: Apress. 528 s. ISBN 978-1-590-59831-3.

Unittest++ (2010) *Unitest++*. Saatavana World Wide Webistä. <URL:http://unittest-cpp.sourceforge.net/> (siteerattu 2.3.2010)

VMWare (2010). *VMWare Player.* Saatavana World Wide Webistä <URL:http://www.vmware.com/products/player> (siteerattu 4.4.2010)

Zawadzki Maciej (2007). *Drawing the Line: Continous Integration and Build Magement* <URL:http://www.cmcrossroads.com/pdf/DrawingtheLine.pdf> (siteerattu 4.4.2010)

Zutubi (2010a). *New In Pulse 2.1.* Saatavana World Wide Webistä. <URL:http://zutubi.com/products/pulse/new/> (siteerattu 5.4.2010)

Zutubi (2010b). *Configuring the Personal Build Client.* Saatavana World Wide Webistä <URL:http://confluence.zutubi.com/display/pulse0102/Configuring+the+Personal+Build+Client> (siteerattu 28.3.2010)

LIITTEET

LIITE 1. SHELL-KIELINEN SKRIPTI INTEGRAATIOPALVELIMEN TUOTTAMIEN A-PAKETTIEN KOPIOIMISEEN JA ASENTAMISEEN

```
#!/bin/sh
#
# script to copy A-package from binary repository
# ©Jaakko Huhtala 2010
#
# help:
case "$1" in
--help|-h)
       echo Usage: ./install_x.sh \<variant name\>
       echo Example: ./install_x.sh linux_arm
       echo
       echo The script will try to install constant build package
       echo from projects repository
       echo
       echo Script will exit with a return code of 0 in case of
       echo success,return code of 1 indicates check failed,
       echo return code of 2 indicates something went wrong
       echo during the package copying procedure.
       ;;
*)

URL="http://Repository_A"
PACKAGE="Devpack_A_$1.tar.gz"
TARGET_DIR="/opt/"

CHECK=0
if [ -n "$1" ]; then

  CHECK=1
  echo Check passed...

fi

if [ $CHECK -eq 1 ]; then
```

```
      wget -N $URL/$PACKAGE && \
      (echo "Extracting package $PACKAGE... ")  && \
      tar xfz $PACKAGE -C $TARGET_DIR  && \
      (  echo  -n  $PACKAGE;echo  -n  "  sucessfully  extracted  to
$TARGET_DIR ";echo "" ) && \
      exit 0 || echo installing package failed;exit 3
else
      echo CHECK FAILED!
      echo
      echo Something\'s not right here, chummer.
      echo Make sure you have added the variant name.
      echo
      echo Usage: ./install_dev-env.sh \<variant name\>
      echo Example: ./install_dev-env.sh linux_arm
      echo
      echo The script will try to install A constant build
      echo package from:
      echo $URL/$PACKAGE
      exit 2
fi
;;

esac
```