Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

OHJELMISTOTUOTANNON TOTEUTUSMENETELMIEN YHTENÄISTÄMINEN

Diplomityön aihe on hyväksytty Tietotekniikan osaston osastoneuvostossa 26.1.2005.

Työn 1. tarkastaja: Prof., Heikki Kälviäinen
Työn 2. tarkastaja ja ohjaaja: DI Jyri Syväoja

Jarkko Sikiö
Huhtiniemenkatu 25 C 25
53810 Lappeenranta
Puh. 040 595 0068
Email. jarkko.sikio@pp1.inet.fi

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

Jarkko Sikiö

Ohjelmistotuotannon toteutusmenetelmien yhtenäistäminen

Diplomityö

2005

86 sivua, 6 kuvaa ja 2 liitettä.

Tarkastajat: Prof., Heikki Kälviäinen
DI Jyri Syväoja

Hakusanat: ohjelmistotuotanto, toteutusmenetelmät, prosessimalli
Keywords: software engineering, implementation methods, process model

Organisaatio, prosessimalli ja menetelmät vaikuttavat toisiinsa sekä suorasti että prosessien ja tavoitteiden kautta epäsuorasti. Prosessimallit vaihtelevat eri organisaatioiden välillä, mutta työkalut ja menetelmät, erityisesti toteutusmenetelmät, saattavat vaihdella jopa eri projektien ja sovelluskehittäjien välillä. Toteutusmenetelmien yhtenäistämisellä tavoitellaan ohjelmistokehityksen tehokkuuden parantamista, ohjelmistojen laatutason nostamista ja työmotivaation kohottamista.

Tämän diplomityön käytännön osuudessa selvitettiin ohjelmistokehitysorganisaation asenteita ja edellytyksiä toteutusmenetelmien yhtenäistämistä kohtaan. Diplomityön tuloksena laadittiin suositus siitä, kuinka parhaat käytännöt -dokumentti voidaan toteuttaa. Suosituksen mukaan kyseinen dokumentti tulisi jakaa kahdeksi dokumentiksi siten, että toinen dokumenteista kattaisi käytännöllisimmät toteutusmenetelmät, toinen sisältäisi suunnittelumenetelmät.

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Jarkko Sikiö

Standardizing implementation methods of software engineering

Master's thesis

2005

86 pages, 6 figures and 2 appendices.

Examiners: Prof., Heikki Kälviäinen
M.Sc. Jyri Syväoja

Keywords: software engineering, implementation methods, process model

An organization, a process model and the methods are interacting with each other both directly and indirectly via the processes and the goals. The process models vary between the organizations but the tools and methods, especially the implementation methods, might vary even between the projects and the application developers. The benefits of standardizing the implementation methods are improved performance of the software development and increased quality level of software, as well as the higher work motivation of the employees.

In the practical section of this master's thesis, the attitudes and prerequisites towards standardization of the implementation methods were investigated. As a result, a recommendation about how to implement a best practices document was created. According to that recommendation, the document should be divided into a two separate documents. The first document should include only the most practical implementation methods, and the second document should include the design methods.

ALKUSANAT

Tämä diplomityö on tehty TeliaSonera Finland Oy:n Lappeenrannan yksikössä.

Työn tarkastajana Lappeenrannan teknillisessä yliopistossa on toiminut professori Heikki Kälviäinen, jolle tahdon esittää kiitokseni työn ohjauksesta.

Työn ohjaajana TeliaSonera Finland Oyj:n puolesta on toiminut Jyri Syväoja, jota kiitän, samoin kuin Petri Ahosta, keskusteluista jotka antoivat suunnan diplomityölleni sen alkuvaiheessa. Haluan esittää kiitokseni myös Antti Ollilaiselle, Ari Tikalle, Ville Kanervalle ja Jani Huttuselle heidän näkemyksistään. Lisäksi kiitän Mika Kataikkoa, joka myötävaikutti työni aiheen valintaan, samoin kuin esimiehiäni Harri Tumeliusta ja Ilkka Toivasta siitä, että sain vapaat kädet valitsemani aiheen käsittelyyn.

Lappeenrannassa 5. huhtikuuta 2005

SISÄLLYSLUETTELO

LYHENNELUETTELO

AIM	Application Implementation Method
AOP	Aspect-Oriented Programming
ASD	Adaptive Software Development
AM	Agile Modeling
ASAP	Accelerated SAP
ASCII	American Standard Code for Information Interchange
BSD	Berkeley System Distribution
CASE	Computer Aided Software Engineering
CVS	Concurrent Versions System
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
DSDM	Dynamic Systems Development Method
FDD	Feature-Driven Development
FSF	Free Software Foundation
GPL	General Public License
IDE	Integrated Development Environment
IEEE	Institution of Electrical and Electronics Engineers
ILDE	Integrated Life-Cycle Development Environment
IRC	Internet Relay Chat
ISD	Internet-Speed Development
ISO	International Organization for Standardization
J2EE	Java 2 Platform, Enterprise Edition
JIT	Just-In-Time
JVM	Java Virtual Machine
OMG	Object Management Group
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OOP	Object Oriented Programming
OSS	Open Source Software
PP	Pragmatic Programming
RAD	Rapid Application Development

RPD	Rapid Program Development
RSD	Rapid System Development
RUP	Rational Unified Process
SDI	Solution Development and Integration
SEI	Software Engineering Institute
SPICE	Software Process Improvement Capability Determination
UML	Unified Modeling Language
XP	Extreme Programming

1 JOHDANTO

1.1 Tausta

TeliaSonera Finland Oyj:ssä on kehitetty uusi prosessimalli, joka on otettu käyttöön vuonna 2004. Prosessimalli on korkean abstraktiotason kuvaus yrityksen tuotekehitysorganisaation toiminnasta. Ohjelmistotuotantoon liittyy kuitenkin monia prosessiriippumattomia toteutusmenetelmiä, joita ei niiden yksityiskohtaisuuden vuoksi ole tarkoituksenmukaista kuvata prosessimallissa. Kaikkia organisaatiossa vaikuttavia tekijöitä ei myöskään voi eksplisiittisesti kuvata, sillä ne ovat organisaation jäsenten yksilöllisiä työskentelytapoja ja tottumuksia.

Koska kohdeyrityksessä ei ole toteutusohjeistusta ohjelmistomoduulien suunniteluun, toteutukseen ja testaukseen, toiminta suunnittelu-, toteutus- ja testausvaiheissa vaihtelee eri projektien ja kehittäjien välillä. Tästä seuraa, että kyseisten vaiheiden aikana tuotetut ohjelmistomoduulit eroavat toisistaan. Tällaisia eroja ovat esimerkiksi eri tavoin muotoiltu ohjelmakoodi ja vaihtelevat suunnitteluratkaisut.

Kohdeyrityksessä on päätetty yhtenäistää toteutusmenetelmiä kehittämällä toteutusohjeistus Implementation Policy -nimisen parhaat käytännöt -dokumentin muodossa.

1.2 Tavoite ja rajaukset

Diplomityö oli selvitystyö, jonka kohdeyritys oli mm. Suomessa toimiva teleoperaattori, TeliaSonera Finland Oyj. Diplomityön tavoite oli selvittää, kuinka ohjelmistotuotannon toteutusmenetelmät voidaan yhtenäistää ja sen perusteella laatia suositus parhaat käytännöt -dokumentin sisällöstä.

Kohdeorganisaatio, SDI-yksikkö (Solution Development and Integration), on kohdeyrityksen ohjelmistokehitysyksikkö, jonka sisälle diplomityö rajattiin. Peruste rajaukselle on, että yksikön toiminta eroaa merkittävästi yrityksen muiden yksiköiden, kuten asiakaspalvelun, toiminnasta. Tästä syystä diplomityössä kartoitettuja toteutusmenetelmiä ei myöskään voida soveltaa muissa yksiköissä.

Diplomityössä tarkasteltiin ohjelmistotuotannon suunnittelu- ja toteutusvaiheita. Suunnitteluvaihe jakautuu arkkitehtuurisuunnitteluun ja moduulisuunnitteluun [1]. Diplomityössä keskityttiin yksistään moduulisuunnitteluun, koska ohjelmistomoduulit olivat kehitettävän toteutusohjeistuksen tärkein kohde.

Toteutusvaiheessa käytettävistä teknologioista keskityttiin Sunin kehittämään Java-ohjelmointikieleen ja OMG:n (Object Management Group) standardoimaan UML-mallinnuskieleen (Unified Modeling Language), koska ne ovat keskeisiä kohdeorganisaatiossa käytettyjä teknologioita. Parhaat käytännöt -dokumenttia voidaan myöhemmissä versioissa laajentaa kattamaan muita keskeisiä toteutusteknologioita, joita ovat Oracle-relaatiotietokannat ja muut kohdeorganisaatiossa käytetyt ohjelmointikielet. Lisäksi myöhemmät versiot voivat sisältää arkkitehtuurisuunnitteluun liittyviä asioita.

1.3 Rakenne

Kappaleessa 2 esitellään, kuinka diplomityön käytännön osuus, eli selvitystyö, liittyy teoriaosuuteen. Kappale 2.1 käsittelee perusteet sille, kuinka selvitystyö toteutettiin, kappale 2.2 mikä on selvitystyön tutkittava ilmiö ja kappale 2.3 esittelee teoreettisen viitekehyksen, jonka puitteissa teoriaosuudessa tutkittavaa ilmiötä tarkastellaan.

Diplomityön teoreettinen osuus koostuu yhteensä kolmesta kappaleesta. Kappaleessa 3 esitellään ohjelmistotuotannon peruskäsitteet ja käydään läpi teoreettisen viitekehyksen keskeisimmät tekijät. Kappaleessa 4 määritellään toteutusmenetelmät, käsitellään moduulisuunnittelu ja -toteutus sekä loput teoreettisen viitekehyksen tekijät. Kappaleessa 5 pohditaan toteutusmenetelmien vaihtelemisen syitä ja edellytyksiä niiden yhtenäistämiselle sekä nostetaan esiin syyt, miksi ne pitäisi yhtenäistää ja mitä etuja niiden yhtenäistämisellä tavoitellaan.

Kappaleessa 6 kuvataan, kuinka diplomityön käytännön osuus toteutettiin ja kuinka sen vaatima data kerättiin. Kappaleessa 7 esitellään diplomityön tulokset ja niiden perusteella esitetään suositellut toimenpiteet.

2 SELVITYSTYÖN TEORIA

Tässä kappaleessa käsitellään selvitystyön toteuttamiseen, kuten datan keräämiseen, liittyvä teoreettinen tausta. Lisäksi esitellään teoreettinen viitekehys, joka muodostaa rungon, johon diplomityön teoriaosuus perustuu.

2.1 Tutkimusmetodi ja datan kerääminen

Tämän diplomityön käytännön osuudessa, eli selvitystyössä, käytetty tutkimusmetodi oli tapaustutkimus, jossa käsiteltiin yhtä tapausta [2]. Tutkimuskohde on TeliaSonera Finland Oyj:n SDI-yksikkö, johon viitataan jatkossa kohdeorganisaationa.

Tapaustutkimuksessa datan kerääminen tapahtuu kuudesta eri lähteestä, jotka ovat dokumentit, arkistot, haastattelut, suora havainnointi, osallistuva havainnointi ja fyysiset artefaktit [3].

Diplomityössä datan keräämiseen käytetyt menetelmät olivat kysely, haastattelut ja havainnointi. Näiden menetelmien yhdistelmä valittiin siksi, että haastattelujen avulla selvitettiin, kuinka haastateltavat henkilöt kokevat sen, miten kohdeorganisaatiossa toimitaan. Havainnoinnin avulla voitiin puolestaan todeta, kuinka kohdeorganisaatiossa käytännössä oli toimittu. [4] Kyselyn avulla puolestaan kerättiin tilastollista dataa täydentämään muilla menetelmillä hankittua dataa.

Haastattelun lajeista diplomityössä käytettiin teemahaastattelua, koska haastattelujen aihepiiri eli teema-alue – toteutusmenetelmien yhtenäistäminen – oli etukäteen valittu. Haastattelu toteutettiin yksilöhaastatteluina. [4]

Havainnoinnin lajeista valittiin osallistuva havainnointi, jossa tarkkailija ei ole passiivinen, vaan osallistuu tutkimuksen kohteena oleviin tapahtumiin [3]. Valinnan tärkein syy on se, että diplomityön tekijä oli ollut kohdeyrityksen palveluksessa vuodesta 2000 lähtien ja tunsi sen työskentelytavat. Toisin sanoen, tekijä oli havainnoitavan ryhmän jäsen ja siten jakoi samat kokemukset sekä tunsi havainnoitavien kielenkäytön [4].

Osallistuvaan havainnointiin liittyvä haitta, tarkkailijan havainnointien tekeminen tutkimuskohteen ”sisällä”, pyrittiin ottamaan huomioon välttämällä tiedonkeräämistarkoituksissa harjoitettua kohdeyrityksen tapahtumien manipulointia [3]. Käytännössä tämä tarkoitti, että havainnot kohdistuivat ensisijaisesti dokumentteihin, joita olivat kohdeorganisaation tuottamat suunnittelu- ja toteutusdokumentit sekä lähdekoodit. Diplomityössä ei käsitelty dokumenttien ja lähdekoodien sisältöä, sillä ne olivat luokitukseltaan luottamuksellisia tai salaisia. Huomiot kohdistuivat siis siihen, kuinka ohjelmakoodia oli kirjoitettu ja miten dokumentit oli tehty.

2.2 Tutkittava ilmiö

Tässä diplomityössä tutkittu ilmiö oli toteutusmenetelmien vaihteleminen. Tällä tarkoitetaan ohjelmistotuotannon toteutusvaiheen menetelmien vaihtelemista eri projektien ja työntekijöiden välillä. Tutkimusongelma oli, kuinka toteutusmenetelmät voidaan yhtenäistää. Tätä lähestyttiin seuraavilla jatkokysymyksillä: mitkä toteutusmenetelmät voidaan yhtenäistää, kuinka toteutusmenetelmät vaihtelevat ja mistä niiden vaihteleminen johtuu.

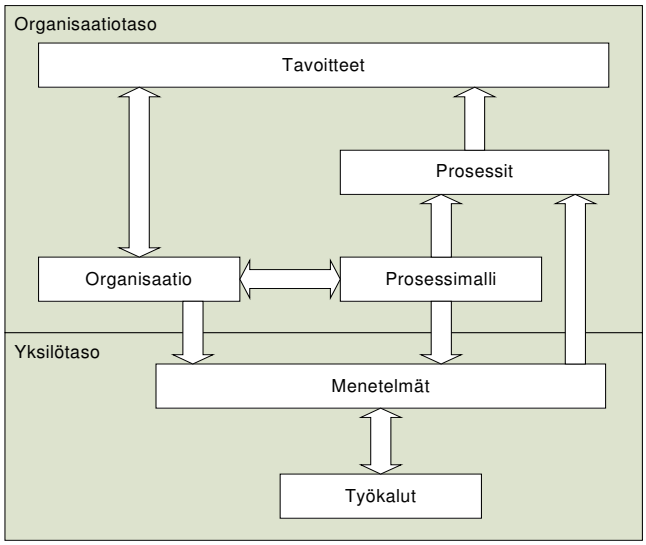
Jotta voitiin määritellä, mitkä toteutusmenetelmät voidaan yhtenäistää, määriteltiin toteutusmenetelmien vaihtelemiseen vaikuttavat tekijät teoreettisen viitekehyksen avulla. Kohdeorganisaation nykytila puolestaan selvitettiin haastattelujen ja havaintojen avulla, jotta voitiin vastata kysymyksiin, kuinka toteutusmenetelmät vaihtelevat ja mistä niiden vaihteleminen johtuu. Kyselyn ja haastattelujen avulla kartoitettiin myös asenteita toteutusmenetelmien yhtenäistämiseen tähtääviä toimenpiteitä kohtaan, koska oletettiin että organisaatiolla ja sen yksilöillä olisi suuri merkitys siihen, kuinka toteutusmenetelmiä voitaisiin yhtenäistää.

2.3 Teoreettinen viitekehys

Koska ei ole sellaista vakiintunutta viitekehystä, jossa esiintyisivät kaikki tekijät, joiden oletettiin vaikuttavan toteutusmenetelmien vaihtelemiseen kohdeorganisaatiossa, tässä kappaleessa esitetään oma teoreettinen viitekehys.

Ohjelmistokehityksen viitekehyksen tekijät ovat tavoitteet, organisaatio, prosessimalli, prosessit, menetelmät ja työkalut (kuva 1). Kaikki nämä tekijät ovat löydettävissä

ohjelmistotuotantoa käsittelevästä kirjallisuudesta. Sen sijaan niiden keskinäisistä yhteyksistä on useita eri näkemyksiä ja painotuksia.



Kuva 1. Diplomityön teoreettinen viitekehys.

Yhteys organisaation ja tavoitteiden välillä on johtamista. Tavoitteet on asetettava organisaation mukaan, koska ohjelmistokehitysorganisaation tavoitteet ovat erilaiset kuin myynti- tai asiakaspalveluorganisaation. Tavoitteiden asettaminen vaikuttaa myös toiseen suuntaan. Organisaation on sitouduttava tavoitteiden saavuttamiseen. Tämä vaikuttaa yrityskulttuuriin. Jos tavoitteena on laatu, myös yrityskulttuurin tulisi olla sellainen, että koko organisaatio tavoittelee laatua. Laatu ei kuitenkaan ole usein ainoa tavoite, jota organisaatiossa tavoitellaan. Tavoitteisiin voi kuulua mm. tuotekehityksen nopeus tai tietoturva sekä liiketoimintatavoitteet.

Organisaation ja prosessimallin välinen yhteys on prosessikehitystä. Prosessimalli vaikuttaa aikanaan organisaatioon muotoutuvan yrityskulttuurin kautta. Vastaavasti organisaatio itsessään vaikuttaa siihen, millainen prosessimallista käytännössä muotoutuu. Molemmat näistä tekijöistä vaikuttavat menetelmiin, joihin kuuluvat myös toteutusmenetelmät. Prosessimalli tarvitsee siihen soveltuvat menetelmät, kun taas organisaatio hyväksyy kyseiset menetelmät, soveltaa niitä tai hylkää ne. Kolmas menetelmiin vaikuttava tekijä on menetelmiä tukevat työkalut. Menetelmien valinta vastavuoroisesti ohjaa työkalujen valintaa.

Prosessimalli määrittelee prosessit. Tässä diplomityössä viitataan prosesseilla vain ohjelmistokehityksen prosesseihin, kuten suunnittelu- ja toteutusprosesseihin, ei yleisiin liiketoimintaprosesseihin, kuten tuotekehitysprosessiin. Nämä ohjelmistokehityksen prosessit itsessään ohjaavat toimintaa kohti asetettuja tavoitteita. Tässä on siis takaisinkytkentä prosessimalliin: kun tavoitteet vaikuttavat organisaatioon, ne vaikuttavat myös prosessikehitykseen. Mikäli prosessit eivät ohjaa organisaatiota tavoitteisiin, organisaation on muutettava prosesseja prosessimallin kautta.

Kaikki nämä viitekehyksen tekijät voidaan jakaa kahdelle tasolle: organisaatiotasolle ja yksilötasolle. Organisaatiotasolla ovat tavoitteet, organisaatio, prosessimalli ja prosessit. Yksilötason muodostavat menetelmät ja työkalut. Jaon perusteena ovat viitemallin tekijöiden keskinäiset riippuvuussuhteet. Organisaatiotason muodostavat ne tekijät, jotka liittyvät kiinteästi toisiinsa. Niistä mitään tekijää ei voi muuttaa ilman, että vaikutukset koskevat kaikkia viitemallin tekijöitä. Yksilötason tekijöitä sen sijaan voidaan muuttaa ilman, että muutoksilla olisi merkittäviä vaikutuksia organisaatiotason tekijöihin, prosesseja lukuun ottamatta.

Menetelmien ja prosessien välinen yhteys on diplomityön kannalta viitekehykseen merkittävin yhteys. Tämä johtuu siitä, että diplomityölle on asetettu reunaehto, jonka mukaisesti toteutusmenetelmiä on tarkasteltava siitä lähtökohdasta, että ne ovat prosessiriippumattomia. Viitemallin mukaan menetelmien ja prosessien välillä on kuitenkin kaksi yhteyttä. Prosessit vaikuttavat menetelmiin välillisesti tavoitteiden ja organisaation kautta, kun taas menetelmät vaikuttavat prosesseihin suoraan.

Kohdeyrityksen ja diplomityön kannalta suora yhteys toteutusmenetelmien ja prosessien välillä on epätoivottava. Prosessiriippumattomuus ei viitemallin mukaan tarkoita, että toteutusmenetelmiä voitaisiin muuttaa ilman vaikutusta prosesseihin. Tätä yhteyttä ei viitemallissa kuitenkaan voida jättää huomioimatta, sillä se esiintyy useissa kirjallisuuslähteissä, joskin ohjelmistokehityksen ja sen prosessien määritelmät vaihtelevat huomattavasti esimerkiksi perinteisten prosessimallien ja ketterien menetelmien välillä.

3 OHJELMISTOTUOTANTO

Tässä kappaleessa esitellään aluksi keskeiset diplomityössä käytetyt termit ja käsitteet, minkä jälkeen käsitellään yksityiskohtaisesti teoreettisen viitemallin organisaatiotason tekijät.

3.1 Määritelmä

IEEE (Institution of Electrical and Electronics Engineers) määrittelee ohjelmistotuotannon vapaamuotoisesti käännettynä olevan mm. systemaattinen ja kurinalainen lähestymistapa ohjelmiston kehitykseen, toimintaan ja ylläpitoon [5]. Tämä määritelmä ei yksin riitä kuvaamaan ohjelmistotuotantoa, vaan kirjallisuudesta löytyy useita malleja sitä varten. Ohjelmistotuotanto voidaan jakaa esimerkiksi neljään kerrokseen. Kerrokset ylhäältä alas lukien ovat työkalut, metodit, prosessit ja laatu. [6] Tässä diplomityössä väitetään, etteivät nämäkään mallit kata kohdeorganisaation tapauksessa kaikkia ohjelmistotuotannon keskeisiä tekijöitä ja siitä syystä diplomityön teoriaosuudessa edetään kappaleessa 2.3 esitellyn teoreettisen viitekehyksen mukaan.

3.2 Peruskäsitteet

Osa ohjelmistokehityksen termeistä ovat vakiintumattomia. Eri yhteyksissä ja organisaatioissa käytetään toisistaan eroavaa terminologiaa. Käytännössä ohjelmistotuotannon käsitteitä ”prosessimalli”, ”elinkaarimalli” ja ”menetelmä” käytetään osin ristikkäin. Näistä jälkimmäinen korvaa usein sanan ”metodologia” ja sen paremmat käännökset ”kehittämismalli” ja ”menetelmämalli”, jotka eivät tässä yhteydessä ole vakiintuneet suomen kieleen [7] [8].

Prosessimallilla ja elinkaarimallilla viitataan käytännössä vaiheistettuun, tai muutoin jaoteltuun kehitystoimintaan, johon kuuluvat mm. ohjelmiston suunnittelu- ja toteutusvaiheet [5]. Menetelmän sanakirjamääritelmä on: ”määrämuotoinen tapa tuottaa tietyn ohjelmistokehitystyön tehtävän lopputulokset” [8]. Tämän diplomityön kannalta kattavampi määritelmä kuuluu vapaasti käännettynä seuraavasti: ”menetelmä on suositeltu joukko vaiheita, menettelytapoja, sääntöjä, tekniikoita, työkaluja, dokumentaatiota, johtamista ja koulutusta, joita käytetään järjestelmäkehityksessä” [9].

Yksinkertaisemman näkemyksen mukaan kyseessä on ”joukko käytäntöjä, jotka sovelluskehitystiimi ottaa käyttöönsä” [10]. Jälkimmäinen määritelmä sisältää diplomityön kannalta merkittävän huomion siitä, etteivät kaikki organisaation valitsemat käytännöt ole tosiasiassa käytössä. Asiaa käsitellään tarkemmin kappaleissa 4 ja 5.

Diplomityössä käytetään johdonmukaisuuden vuoksi termiä ”prosessimalli”, kun viitataan ohjelmistokehityksen prosessimalleihin, vaikka toisinaan kuvaavampi sana voisi olla ”ohjelmistokehitysmalli”, sillä osa diplomityön kappaleessa 3.6 käsittelemistä prosessimalleista eivät ole prosessikeskeisiä vaan menetelmäkeskeisiä. Käsitettä ”elinkaarimalli” ei diplomityössä käytetä kuin satunnaisesti selkeyden vuoksi, koska valituilla termeillä tahdottiin korostaa prosessimallin ja prosessien välistä yhteyttä. Diplomityössä ei myöskään käsitellä koko ohjelmiston elinkaarta, vaan mm. vaatimusten hallinta ja ylläpito on kappaleessa 1.2 rajattu diplomityön ulkopuolelle.

Termiä "menetelmä” käytetään silloin, kun korostetaan eri mallien taustalta löytyviä filosofisia eroja, joiden esitetään myöhemmin tässä työssä vaikuttavan yrityskulttuuriin. Tällaisia filosofisia suuntauksia on löydettävissä mm. ketterien menetelmien taustalla.

Vapaasti käännettynä IEEE määrittelee ohjelmistokehitysprosessin olevan tiettyä tavoitetta varten suoritettuja, toisiaan seuraavia vaiheita [5]. Sanakirjamääritelmä puolestaan kuuluu: ”Prosessimalli on tehtävien ja päätösten sarja tai verkko, joka tietyssä järjestyksessä systemaattisesti toteutettuna saa aikaan tiettyjä lopputuloksia. Se tuottaa syötteen perusteella tuotoksia. Tuotoksia hyödyntää prosessin asiakas, joka voi olla sisäinen tai ulkoinen. Prosessien yhteydessä puhutaan myös prosessin toimittajasta, omistajasta, resursseista, palautteesta ja mittareista. Prosessista voidaan tehdä prosessikuvauksia graafisessa muodossa, tekstimuodossa tai näiden yhdistelmänä.” [8] Molemmat määritelmät sisältävät oletuksen lineaarisuudesta. Todellisuudessa ohjelmistokehitys on kuitenkin usein iteratiivista, mitä käsitellään tarkemmin prosessimallien yhteydessä kappaleessa 3.6.

Termiä ”organisaatio” ei diplomityössä käytetä sen yleismerkityksessä, vaan sillä viitataan aina ohjelmistokehitystä tekevään organisaatioyksikköön, joka voi olla yrityksen yksikkö, osasto tai projektitiimi [11]. Näin termistä on suljettu pois yritystaso,

mikä on diplomityön puitteissa johdonmukaista, sillä kohdeyritys on ensisijaisesti teleoperaattori, ei ohjelmistoyritys.

Käytännössä ohjelmistot kehitetään useimmiten projekteissa [1]. Sovelluskehittäjä, lyhyemmin kehittäjä, toteuttaja tai ohjelmoija, kirjoittaa editorin tai IDE-kehitysympäristön (Integrated Development Environment) avulla ASCII-muodossa (American Standard Code for Information Interchange) ohjelmakoodia, eli lähdekoodia, joka Java-ohjelmointikielen tapauksessa tallennetaan java-päätteisiin tiedostoihin. Lähdekooditiedostot käännetään yleensä binäärikoodiksi, mutta Java-ohjelmointikielen tapauksessa tulkattavaksi tavukoodiksi. Useimmat JVM-toteutukset (Java Virtual Machine) joko tulkkaavat tämän tavukoodin tai suorittavat JIT-käännöksen (Just-In-Time), jossa tavukoodi käännetään ajonaikaisesti. [12] Kaikkiin näihin työvaiheisiin on olemassa erilaisia työkaluja ja käytännöt niiden hyödyntämiseen vaihtelevat eri organisaatioissa.

Ohjelmistotuote itsessään koostuu komponenteista. Ohjelmistokomponentti on kokonaisuuden osa, jolle on määritelty mm. rajapinnat. Komponentin on usein oltava uudelleenkäytettävä eri versioiden, tuotteiden ja organisaatioiden välillä. [13] Myös kohdeorganisaatiossa tavoitellaan uudelleenkäytettävyyttä komponenttitasolla. Komponentit puolestaan koostuvat ohjelmistomoduuleista. Moduulit, jotka on ensin testattu moduulitestausvaiheessa, testataan integrointitestausvaiheessa, jotta saadaan selville kuinka ne toimivat yhdessä. Lopulta järjestelmätestauksen aikana testataan koko ohjelmisto. Moduulitestauksen tekee tavanomaisesti moduulin toteuttanut kehittäjä, järjestelmätestauksen varsinainen testaaja. [1]

3.3 Ohjelmistotuotannon menetelmät

Kaikkiin ohjelmistotuotannon vaiheisiin liittyy menetelmiä. Ohjelmistotuotannon menetelmät koostuvat vesiputousmallin vaiheisiin suhteutettuina mm. määrittely-, suunnittelu-, toteutus- ja testausmenetelmistä. Näin määriteltynä esimerkiksi dokumentaatio tulisi käsitellä toteutusmenetelmien yhteydessä vain toteutusdokumenttien osalta. Dokumentaation kaltaiset asiat koskettavat kuitenkin ohjelmistotuotannon kaikkia vaiheita, joten tällainen käsittely ei olisi johdonmukainen. Myös raja suunnit-

telumenetelmiin on tässä yhteydessä liukuva, koska toteutusvaiheessa on usein tehtävä sellaisia päätöksiä, jotka tavallisimmin kuuluvat suunnitteluvaiheeseen.

Kappaleessa 2.3 esitetyn viitekehyksen mukaan ohjelmistotuotannon menetelmät ovat osittain riippuvaisia organisaatiosta, jossa niitä sovelletaan. Organisaatiossa vallitsevaan yrityskulttuuriin puolestaan vaikuttavat tavoitteet, prosessit ja valitut menetelmät. Koska menetelmiin vaikuttaa moni asia, ei toteutusmenetelmien yhtenäistämistä voida tarkastella vain osana prosessikehitystä. Prosessimallien lisäksi on huomioitava viitekehyksen mukaisesti myös organisaatio ja työkalut. Viitekehyksessä esitetyllä tavalla nämä tekijät ovat vuorovaikutuksessa myös muiden tekijöiden kanssa. Näiden tekijöiden konkretisoimiseksi kukin niistä käsitellään tässä teoriaosuudessa. Toteutusmenetelmiä käsitellään tarkemmin kappaleessa 4.

3.4 Organisaatio ja tavoitteet

Organisaatio asettaa kehittämälleen tuotteelle tai palvelulle tietyt tavoitteet, joihin ohjelmistokehitysprosesseilla tähdätään ja jota organisaation yrityskulttuuri, sekä käytännön tasolla toteutusmenetelmät, tukevat. Näitä tavoitteita voivat olla liiketoimintatavoitteiden lisäksi esimerkiksi laatu, tietoturva ja uudelleenkäytettävyys. Kaikki nämä tavoitteet lasketaan usein osaksi ohjelmistoteknistä laatua, mikä on yksi kokonaislaadun osa. Toisinaan esimerkiksi tietoturvan ja ohjelmiston laadun välille vedetään yhtäläisyysmerkki [14]. Perinteisen laatuajattelun näkökulmat ovat kuitenkin tarpeettoman yksioikoisia, sillä laatu ei ole aina asiakkaille se merkittävin ominaisuus [15].

Asiakkaan tärkein vaatimus saattaa olla tuotteen tuominen markkinoille mahdollisimman nopeasti, jolloin yrityksen tavoitteeksi voidaan asettaa ohjelmistokehitysprosessin tehokkuus. Kuitenkin työntekijät yleensä haluavat säilyttää tietyn laatutason, mikä pitkällä tähtäimellä johtaa myös asiakastyytyväisyyteen [15]. Tavoitteet siis tässä yhteydessä eivät rajoitu koskemaan vain lopputuotetta, vaan niillä on vaikutus yrityskulttuuriin ja sen kautta prosesseihin ja menetelmiin.

Yrityskulttuurin on oltava yhtenäinen. Jos yrityksen tavoitteena on pyrkiä kohti laatua, kaikkien organisaation tasojen on sitouduttava ohjelmiston laadun tuottamiseen. [16] Samanlainen näkemys esiintyy kirjallisuuslähteissä yleisesti olivat tavoitteet

mitä tahansa. Korostettaessa tietoturvaa, on organisaation sitouduttava siihen kaikilla tasoilla, johdosta sovelluskehittäjiin [14].

Organisaatiot koostuvat yksilöistä ja organisaation kyvykkyys on yksilöiden kyvykkyyttä toimia yhdessä [15] [17]. Tällainen yhteistoiminta syntyy yhteisistä tavoitteista ja yhtenäisestä yrityskulttuurista. Ohjelmistoyrityksen yrityskulttuurin muodostavat kaikkien organisaation jäsenten toiminta, johdon asettamat prioriteetit, projektin tavoitteet ja tekniset käytännöt. [16] Tämä määritelmä on käytännönläheinen, mutta lähtökohdiltaan kiistanalainen, sillä eri projektin tavoitteet vaihtelevat suuresti ja käytännössä työntekijöiden on vaikea sopeutua nopeasti suuriin muutoksiin, jotka kohdistuvat heidän toimintaansa. Tällöin on tärkeää, että ”ihmiset tuntevat tekevänsä muutosta, eivätkä tunne olevansa muutoksen uhreja”. [16] Tästä syystä kappaleessa 2.3 esitetyssä teoreettisessa viitekehyksessä on yhtenä tekijänä prosessimalli, joka asettaa organisaation toiminnan tiettyihin rajoihin. Näiden rajojen puitteissa organisaatio pyrkii tavoitteisiinsa toimimalla prosessimallin mukaisten prosessien kuvaamalla tavalla.

3.5 Prosessit

Ohjelmistokehitysprosesseihin keskittyvät tutkimukset pohjautuvat yleisesti siihen olettamukseen, että prosessin ja kehitetyn ohjelmiston välillä on olemassa suora yhteys [18]. Tätä yhteyttä ei diplomityössä kyseenalaisteta, mutta näkemystä laajennettiin jo edeltävissä kappaleissa kattamaan muutkin tavoitteet kuin laatu. Lyhyesti sanottuna, prosessien avulla organisaatio pyrkii asettamiinsa tavoitteisiin, olivat ne mitä hyvänsä.

Ohjelmistokehitys nähdään tavanomaisesti yhtenä yrityksen liiketoimintaprosessina muiden joukossa, tai se asetetaan osaksi suurempaa kokonaisuutta, eli tuotekehitystä. Nämä lähestymistavat ovat tämän diplomityön aiheeseen nähden liian yleismaailmallisia. Tässä yhteydessä diplomityössä vastaavan kokonaisuuden kattaa käsite ”prosessimalli”, joka määrittelee varsinaiset prosessit, joiden mukaan organisaatio tuottaa tuotteita tai palveluja. Kappaleessa 3.2 esitettiin prosessin teoreettisia määritelmiä, mutta kirjallisuudessa ohjelmistokehityksen prosessi määritellään tavallisimmin joukoksi menettelytapoja, menetelmiä ja työkaluja, joiden avulla ohjelmistotuote luo-

daan [16]. Toisinaan prosessi voidaan myös kuvata joukkona toimintaperiaatteita, organisaatiorakenteita, tekniikoita, menetelmätapoja ja artefakteja [18]. Tässä yhteydessä on huomattava, että erityisesti menetelmien ja prosessien välinen suhde vaihtelee lähteestä riippuen. Kappaleessa 2.3 esitetyssä viitekehyksessä on otettu lähtökohdaksi, että menetelmät eivät ole osa ohjelmistotuotannon prosesseja. Sama lähtökohta esiintyy myös osassa kirjallisuuslähteitä [6].

Ohjelmistotuotannon kehittyessä myös näkemykset sen prosesseista ovat muuttuneet. Alunperin ohjelmistotuotannon prosessien, kuten kaikkien muiden alojen prosessien, on tarkoitettu olevan hallittavia ja toistettavia. Tämä käsitys on kiistelty, koska ohjelmistotuotannon asema insinööritieteenä on ajoittain kyseenalaistettu – osin siitä syystä, että sen harjoittajat ovat eri mieltä mitä ohjelmistotuotanto on [19]. Uusien näkökulmien myötä joissain yhteyksissä ohjelmistotuotannon prosessia pidetään ennemmin kommunikaatiovälineenä kuin tuotteen tuottamiseen tähtäävänä toimintona. [20]

Määritelmästä riippumatta ohjelmistotuotannon prosessien merkitys ohjelmistojen tuottamisessa on huomattava, sillä organisaatioissa on aina prosesseja, vaikka niitä ei olisi eksplisiittisesti kuvattu tai edes tiedostettu [16].

Tapaustutkimusten kautta on havaittu, että suurin osa ohjelmistoyrityksistä voi parantaa prosessejaan, mikä johtaa tuottavuuden ja tuotteen laadun paranemiseen sekä, toisin kuin yleensä uskotaan, parantaa myös työpaikan viihtyisyyttä ja tuottavuutta [21]. Organisaatiot usein pyrkivät tuottavuuden paranemiseen lisäämällä työmäärää ajallisesti, mekanisoimalla tuotekehitysprosessia, tekemällä kompromisseja tuotteen laadusta ja standardoimalla menettelytapoja. Prosesseihin ja työn tekemiseen liittyvä yhtenäistäminen voi kuitenkin saada aikaan vastareaktion, jolloin työstä tulee vähemmän tyydyttävää. [15]

Koska diplomityön tavoitteen, toteutusmenetelmien yhtenäistämisen, rajoitus on että yhtenäistettävien toteutusmenetelmien tulee olla prosessiriippumattomia, ei tässä yhteydessä käsitellä prosessikehitykseen liittyviä asioita, kuten prosessin mittaamista ja arviointia. Keskeisimmät prosessimallit kuitenkin esitellään yhtenä teoreettisen viitekehyksen tekijänä.

3.6 Prosessimallit

Prosessimalli kuvaa organisaatiossa käytettävät prosessit. Ohjelmistotuotannon prosessimalli poikkeaa muiden alojen prosessimalleista siten, että se on ennemminkin strategia, joka sisältää tavanomaisesti prosessit, menetelmät ja työkalut. Prosessimalli voidaan valita projektin tai kehitettävän ohjelmiston mukaan. [6] Diplomityö poikkeaa viitekehyksensä osalta tästä määritelmästä siten, että prosessimallin ei tulisi vaihdella projektista toiseen. Tätä perustellaan sillä, että eri prosessimalleja käyttävien organisaation eri kehitystiimien yhteistoiminta ja keskinäinen kommunikointi vaikeutuu [22]. Prosessimallin on tarjottava organisaation jäsenille runko, jota eri projekteissa soveltuvin osin seurataan. Prosessimallin tavoitteena on siten vakiinnuttaa ohjelmistokehitysorganisaation toimintaa. Tämä näkökulma otetaan perustaksi, kun seuraavissa kappaleissa käydään lyhyesti lävitse viimeisten kahden vuosikymmenen aikana eniten ohjelmistotuotantoon vaikuttaneet prosessimallit. Esiteltyjen prosessimallien avulla voidaan arvioida diplomityön kohdeorganisaatiossa vaikuttavia suuntauksia, jotka heijastuvat prosessimallin soveltamiseen käytännössä.

3.6.1 Prosessimallit ohjelmistokehityksessä

Organisaation sovelluskehittäjille ja muille asiantuntijoille prosessimalli tarjoaa yhteisen ajatusmallin, johon tukeutua. Johdon näkökulmasta prosessimalli on puolestaan yksi projektijohtamisen väline. Prosessimalli soveltuu myös alihankkijoiden toiminnan ohjaamiseen ja kommunikointiin muiden sidosryhmien kanssa, joihin kuuluvat myös asiakkaat. Käytännössä on kuitenkin havaittu, että toisistaan suuresti poikkeavia prosessimalleja käyttävien kehittäjätiimien on vaikea kommunikoida keskenään jopa saman organisaation sisällä [22].

Prosessimalli on prosessien ohella tärkein organisaatiossa dokumentoitu ohjelmistotuotannon tekijä. Siinä missä prosessit kuvaavat sitä, kuinka yrityksessä on tarkoitus toimia, prosessimallien valinta vaikuttaa mm. organisaation pelisääntöihin ja siihen kuinka prosesseja sovelletaan – toisin sanoen yrityskulttuuriin. Lisäksi prosessimallien ja menetelmien taustalla vaikuttaa filosofinen taso: joukko niihin liittyviä uskomuksia ja olettamuksia [9].

Kaikissa organisaatioissa ei ole lainkaan prosessimalleja tai valittuja menetelmiä, tai niitä ei ole ilmaistu eksplisiittisesti. Silti menetelmäsuuntaukset vaikuttavat näihinkin organisaatioihin tiettyjen työkalujen ja tekniikoiden kautta. [9] Käytännössä useissa yrityksissä on niiden itse kehittämiä prosessimalleja tai vähintäänkin yrityskulttuuri tukee tiettyjä toimintatapoja ja käytäntöjä. Tällöin organisaatio ottaa vapaasti, tai tiettyjen sääntöjen rajoissa, käyttöönsä ideoita useasta prosessimallista, koska mikään prosessimalli ei sellaisenaan sovellu kaikille organisaatioille [10].

Prosessimalleihin usein kohdistettu kritiikki kohdistuu siihen, että ne ovat tyypillisesti suunniteltu vain suuria ja monimutkaisia projekteja varten. Tällöin niiden soveltaminen on vaikeaa ja organisaatiot kohtaavat sekä asiakkaiden että sovelluskehittäjien vastustusta. Tällaisia sosiaalisia, poliittisia ja organisaatioon liittyviä ulottuvuuksia eivät prosessimallit tyypillisesti edes huomioi. [9] Suurimmat prosessimalleihin liittyvät ongelmat kuitenkin ovat juuri sosiaalisia: kuinka prosessimallin elementit sopivat työntekijöiden käyttäytymiseen ja kuinka nämä elementit sulautetaan osaksi organisaatiota [23].

Vahva yhteys prosessimallin ja organisaation välillä aiheuttaa myös käytännön ongelmia. Yritykset toimivat budjetin, aikataulun ja henkilöresurssien rajoissa. Tällaisessa ympäristössä uudet prosessimallit, etenkään kokeelliset, eivät saa jalansijaa, koska ne vaativat muutoksia työrutiineihin. [23] Prosessimallin tulee siis alusta lähtien soveltua organisaatiolle, koska prosessimalliin on vaikea tehdä nopeasti suuria muutoksia. Osin tästä syystä diplomityössä ei huomioida mm. formaaleja menetelmiä, sillä niiden sovellettavuus kohdeyrityksessä sen nykyisen yrityskulttuurin ja toimialan huomioon ottaen olisi erittäin pieni.

Ohjelmistokehityksen prosessimallit voidaan jakaa usealla tavalla, mutta tässä diplomityössä ne ryhmitellään karkeasti perinteisiin prosessimalleihin ja ketteriin menetelmiin. Perustelu ryhmittelylle on käytännössä se, että ketterät menetelmät nähdään usein kokonaan uudenlaisina vaihtoehtoina vanhemmille prosessimalleille. Yksikön prosessimallilla on yhteneväisyyksiä vesiputousmallin ja RUP-prosessimallin (Rational Unified Process) kanssa, mutta projekteissa toimivien yksilöiden toiminnassa voidaan nähdä yhtäläisyyksiä mm. tiettyjen ketterien menetelmien ja avoimen lähdekoodin kehityksen kanssa, jota käsitellään muiden prosessimallien yhteydessä.

3.6.2 Perinteiset prosessimallit

Perinteisillä prosessimalleilla tarkoitetaan tässä yhteydessä sellaisia perustavaa laatua olevia prosessimalleja, jotka ovat vaikuttaneet merkittävästi ohjelmistotuotantoon ja joiden vaikutus näkyy niitä seuranneissa prosessimalleissa. Tällaisia prosessimalleja ovat vaihejakomallit, joita kappaleessa 3.3 mainitun vesiputousmallin lisäksi ovat esimerkiksi spiraali-, evoluutio- ja protoilumalli [6]. Spiraalimalli perustuu ohjelmistokehityksen vaiheiden toistamiseen, jolloin ohjelman jokainen uusi versio pohjautuu aina edeltävään versioon. Evoluutiomallissa puolestaan nopeutetaan uusien versioiden tuottamista ja parhaimmillaan uusi versio on valmiina päivittäin. [24] Protoilumallissa kehitetty ohjelmistoversio hylätään kokonaan ennen uuden version kehittämistä [6].

Perinteisiin prosessimalleihin lasketaan tässä yhteydessä myös perinteiset menetelmät, joilla viitataan mm. laajaan joukkoon erilaisia oliomenetelmiä, kuten oliomäärittely (OOA, Object Oriented Analysis), oliosuunnittelu (OOD, Object Oriented Design) ja olio-ohjelmointi (OOP, Object Oriented Programming) [1]. Yhteensä erilaisia oliokeskeisiä menetelmiä lasketaan olevan useita kymmeniä, mutta UML:n standardoinnin myötä niiden kehitysvauhti tasaantui ja vähitellen tiede- ja yritysmaailman kiinnostus on siirtynyt ketteriin menetelmiin [25].

Perinteiset prosessimallit ovat saaneet osakseen paljon kritiikkiä, koska niihin liittyy paljon dokumentointia ja niiden soveltaminen on vaikeaa. Kritiikkiä on esitetty myös siitä syystä, etteivät ne myöskään tyydyttävästi kuvaa kesken ohjelmistokehityksen muuttuvia vaatimuksia tai lähtevät johdonmukaisesti liikkeelle vääristä oletuksista; käytännössä ympäristö ei ole vakaa, liiketoimintastrategia hyvin dokumentoitu tai vaatimuksien osalta ei päästä yhteisymmärrykseen asiakkaan kanssa. [9] Osittain tähän kritiikkiin on vastattu siten, että iteratiivisiakin prosessimalleja sovelletaan ohjelmistokehitysorganisaatioissa, kuin ne olisivat lineaarisia [26]. Toisaalta on muistettava, etteivät kaikki prosessimallit sisällä iteratiivisuutta, vaan ovat todella lineaarisia, kuten vesiputousmalli [24]. Perinteisten prosessimallien, erityisesti vesiputousmallin, hyödyllisyys onkin asetettu kokonaan kyseenalaiseksi sillä perusteella, että ne luovat harhakuvan hallittavasta ja mitattavasta prosessista, vaikka ohjelmistokehitys yrityksessä ei todellisuudessa olisikaan sellaista. [27] Käytännössä projektien tuotok-

set ovat nähtävissä vasta myöhäisessä vaiheessa, mikä asettaa haasteita toteuttajille motivaation säilyttämisessä ja johdolle projektin edistymisen arvioinnissa [24]. Osa tästä voidaan sälyttää myös sen varaan, että prosessimallit ovat raskaita, eli niihin liittyy paljon dokumenttien tuottamista.

Kritiikin osalta on kuitenkin muistettava, että prosessimallit eivät vaadi kaikkia organisaatiossa tuotettuja dokumentteja, vaan käytännöt, kuten kokousmenettelyt, raportit ja katselmoinnit johtavat niiden kirjoittamiseen. Lisäksi organisaatioita ja prosesseja mitataan ja arvioidaan, mikä vaatii lisätyötä. Arviointimalleista tunnetuimmat ovat ISO:n (International Organization for Standardization) kehittämä SPICE (Software Process Improvement Capability Determination), eli ISO 15504 -standardi, sekä SEI:n (Software Engineering Institute) kehittämä CMM-malli (Capability Maturity Model), joka nykyisin tunnetaan CMMI:nä (Capability Maturity Model Integration) [28]. Tässä yhteydessä näihin ei kuitenkaan syvennytä tarkemmin, vaan jatketaan diplomityössä esitellyn viitemallin mukaisesti prosessimallien parissa.

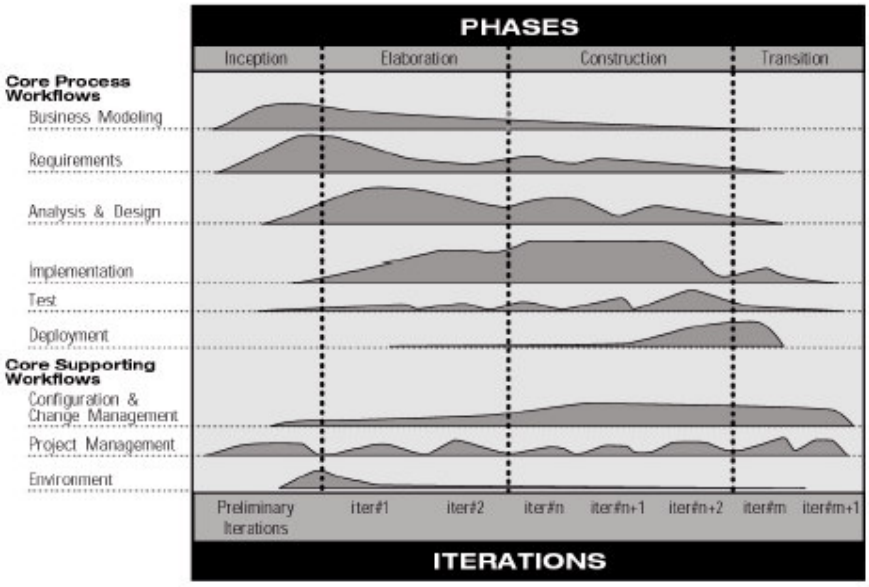
Yksi raskaimpana pidetyistä prosessimalleista on RUP, joka lasketaan tässä diplomityössä perinteisiin prosessimalleihin sillä perusteella, että sitä voidaan pitää ylätasonsa osalta vesiputousmallin johdannaisena, johon liittyy iteratiivisuutta vasta alemmilla tasoilla [28]. RUP soveltuu parhaiten suurille organisaatioille ja vaikka sitä suositellaan sovellettavaksi vain niiltä osin, kuin se organisaatiolle sopii, RUP sisältää useita kaavioita ja dokumentteja. Tämä riitelee ketteriin menetelmiin olennaisesti liittyvän dokumentoinnin vähentämistavoitteen kanssa, vaikka RUP:n kehittäjät itse näkevät mallien luomisen ja ylläpitämisen paperidokumentaatiota merkittävämmäksi [26].

1990-luvun puolivälissä kehitetty RUP on tarkoitettu käytettäväksi UML:n kanssa, joka itsessään on suurilta osin prosessiriippumaton. RUP on tarkoitettu iteratiiviseksi prosessiksi, vaikka houkutus käyttää sitä vaihemaisesti on suuri. Se on tässä mielessä siis evoluutio- ja spiraalimallin sovellus [1]. Keskeistä RUP-prosessissa on mallien korostamisen lisäksi arkkitehtuurikeskeisyys, koska sen tavoitteena on mm. rinnakkainen kehitys, uudelleenkäytettävyys ja järjestelmän ylläpidettävyys. [26]

RUP voidaan räätälöidä sitä käyttävän organisaation tarpeiden mukaisesti, sillä sen kehittäjät toteavat, ettei mikään yksittäinen prosessi sovellu kaikille ohjelmistokehi-

tysorganisaatioille. RUP on myös skaalautuva, eli se sopii erikokoisille organisaatioille. [26] Tältä osin RUP on kuitenkin saanut osakseen paljon kritiikkiä, koska se ei tarjoa ohjeita, kuinka sitä tulisi soveltaa [29].

RUP koostuu neljästä iteratiivisesta vaiheesta, jotka ovat voimaantulo, valmistelu, toteutus ja siirtymä. Kuva 2 havainnollistaa vaiheiden sijoittumista ohjelmistokehityksen elinkaarelle. Voimaantulovaihe keskittyy liiketoiminnan mallintamiseen. Valmisteluvaiheen aikana voidaan toteuttaa järjestelmän prototyyppi, jonka jälkeen analysoidaan ongelma-alue ja laaditaan ohjelmistoarkkitehtuuri, jonka perusteella järjestelmä voidaan toteuttaa. Toteutusvaiheessa kehitetään valmis ohjelmistotuote, joka siirtymävaiheessa toimitetaan asiakkaan käyttöön. [26]



Kuva 2. Ohjelmistokehityksen elinkaari RUP-mallin mukaan. [30]

UML:n ja RUP:n saama huomio kirjallisuudessa on ollut suuri. Silti tutkimusyhtiö Meta Group arvioi, että vain 10 % sovelluskehittäjämarkkinoista on UML:n hallussa. Sovelluskehittäjät eivät siis käytä sitä samassa määrin. [31] Tähän arvioon perustuen voidaan tehdä oletus, että RUP-prosessia käytetään vieläkin vähemmän.

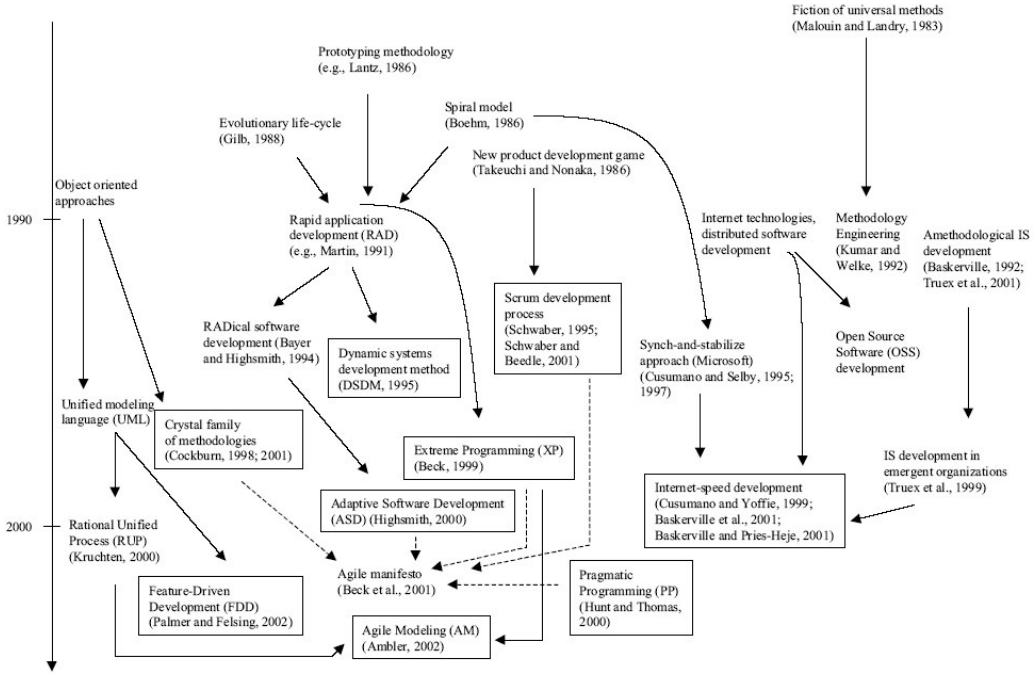
3.6.3 Ketterät menetelmät

Vastapainoksi perinteisille, raskaina pidetyille prosessimalleille, James Martin kehitti 1990-luvun alussa RAD-prosessimallin (Rapid Application Development). Sen tavoite on nopeuttaa ohjelmistokehitystä välttämällä perinteisten prosessimallien sisältämiä laadunhallinta ja -tarkistuskäytäntöjä. [32]

RAD perustuu erittäin lyhyisiin kehityskierroksiin ja on tästä syystä yksi iteratiivisuutta korostavista prosessimalleista, kuten spiraalimalli. RAD korostaa ohjelmistoprosessin nopeutta ja tavoittelee tehokkuutta, mikä saavutetaan yhtäaikaisen kehityksen ja uudelleenkäytettävien komponenttien avulla. [6]

Tarkalleen ottaen RAD:lle ei ole mitään täysin yleispätevää määritelmää [33]. Käytännössä eri ohjelmistotarjoajilla ja konsulttiyrityksillä on omia versioitaan RAD:sta [34]. Yleistasolla RAD voidaan kuitenkin jakaa kahteen päämuotoon. Näistä ensimmäinen, RPD (Rapid Program Development), lähtee tilanteesta, jossa projektitiimillä on käsissään valmis määrittely, jonka mukainen ohjelmisto on toteutettava tiukassa aikataulussa. RSD (Rapid System Development) alkaa puolestaan ydinideasta, joka koskee uutta tai parannettua tietojärjestelmää. RSD:n tapauksessa kommunikaatio asiakkaiden kanssa on erittäin tärkeää. [32] Kirjallisuuslähteistä voidaan myös löytää RAD-prosessimalli vaiheisiin jaettuna. Nämä vaiheet ovat liiketoiminnan mallinnus, datan mallinnus, liiketoimintaprosessien mallinnus, sovelluksen toteuttaminen, testaus ja käyttöönotto. [6]

Kritiikkiä RAD on saanut siitä syystä, että seurauksena on toisinaan ollut huonolaatuisia ohjelmistoja, koska kehitystiimi ei ole toiminut kurinalaisesti [34]. Monissa organisaatioissa on myös havaittu, että jo yksistään RAD:n vaatima yrityskulttuurin muutos on ollut mahdotonta [32]. Diplomityön kohdeorganisaatiossa, jossa on totuttu selkeästi määriteltyihin prosessimalleihin, RAD:lla tuskin tulee olemaan käytännön merkitystä. On kuitenkin perusteltua käsitellä RAD-prosessimallia tässä, koska se voidaan nähdä kehitysfilosofisiansa osalta ketterien menetelmien yhtenä tärkeimmistä edeltäjistä (kuva 3). Ketterät menetelmät edustavat tässä yhteydessä todellista vaihtoehtoa perinteisille, raskaana pidetyille prosessimalleille. Kohdeorganisaation yksilöiden toiminnassa voidaan myös nähdä yhteneviä piirteitä ketterien menetelmien kanssa.



Kuva 3. Ketterien menetelmien kehittyminen. [35]

Ketterät menetelmät ovat saaneet 2000-luvulla paljon huomiota, osittain ns. ketterän manifestin ansiosta. Kyseessä on periaatteellinen julistus, jonka allekirjoittivat spontaanisti joukko ohjelmistokehitysmallien kehittäjiä vuonna 2001. Julistuksen painottamat asiat antavat diplomityölle mielenkiintoisen perspektiivin tarkastellessa kohdeorganisaatiota ja sen toimintaa. Julistuksen mukaan yksilöt ja heidän keskinäinen toimintansa ovat tärkeämpiä kuin prosessit ja työkalut, toimiva ohjelmisto on tärkeämpi kuin kattava dokumentaatio, samoin kuin yhteistyö asiakkaiden kanssa on tärkeämpää kuin sopimusten neuvotteleminen ja muutokseen vastaaminen on merkittävämpää kuin suunnitelman noudattaminen. [10]

Ketterät menetelmät ovat saaneet osakseen samaa kritiikkiä kuin RAD vuosia aiemmin, eli ne nähdään vain perusteluna suunnittelemattomalle ohjelmistokehitykselle. Ketterien menetelmien korostamat käytännön menetelmät, kuten suunnittelumallit, vaativat kuitenkin kurinalaisuutta. Samoin ketterien menetelmien tavoitteet ovat asiakkaalle tuotettu arvo ja korkea laatu, niihin vain pyritään eri lähtökohdista ottamalla samalla huomioon mm. aikataulupaineet. [10] Samankaltaisista painotuksistaan huolimatta, ketterät menetelmät poikkeavat toisistaan merkittävästi, kuten sen

suhteen millaista tukea ne tarjoavat projektinhallintaa varten ja mitä ohjelmistokehityksen elinkaaren vaiheita ne kattavat [35].

Ketteriä menetelmiä ovat ASD (Adaptive Software Development), AM (Agile Modeling), Crystal-menetelmäperhe, DSDM (Dynamic Systems Development Method), XP (Extreme Programming), FDD (Feature-Driven Development), ISD (Internet-Speed Development), PP (Pragmatic Programming) ja Scrum. Tällä hetkellä on vain vähän luotettavaa tai empiiristä tietoa, joka kertoisi missä määrin näitä menetelmiä käytetään. [35] Poikkeuksen muodostaa kuitenkin XP, jota voidaan mm. tästä syystä pitää merkittävimpänä ketteränä menetelmänä. XP-menetelmän osalta on saatavilla sekä tutkimuksista saatua että muutoin luotettavaa empiiristä tietoa. Käytännön kokemuksia XP:stä ovat raportoineet myös suuryritykset, kuten Motorola, Nokia ja ABB. [22] Näiden syiden perusteella tässä kappaleessa käsitellään vain XP-menetelmää.

Kent Beckin kehittämän XP-menetelmän perusidea on sama kuin RAD:n, eli kehityskierroksia lyhennetään mahdollisimman lyhytkestoisiksi. XP vie tämän pidemmälle kuin perinteiset iteratiivisuutta korostavat prosessimallit ja tavoittelee ohjelmistoon kohdistuvien muutosten aiheuttamien kustannuksien vähentämistä pienentämällä kehitysaktiviteetit minimiin. [36]

Yksinkertaistettuna XP:n voidaan sanoa olevan koottu joukko parhaita käytäntöjä, joita noudatetaan kurinalaisesti. XP-menetelmän käytännöt vaihtelevat yleisluontoisista periaatteista, kuten yksinkertaisesta suunnittelusta ja ohjelmakoodin jaetusta omistajuudesta, käytännön menetelmiin, kuten pariohjelmointiin ja jatkuvaan integrointiin, sekä sosiaalisiin seikkoihin, kuten avoimeen työtilaan ja 40-tuntiseen viikkoon. [36] Nämä parhaat käytännöt eivät ole ohjelmistotuotannossa uusia, vaan osa niistä on jo vuosikymmeniä sitten keksittyjä [37].

Kun käytäntöjä tutkii tarkemmin, käytäntöjen yhdistelmä ja painotukset ovat sosiaalisten ja konkreettisten tekijöiden johdonmukainen yhdistelmä. XP:n mukaisen kehitysprosessin perusta on kiinteä yhteistyö sekä tiimin sisällä että asiakkaan kanssa. Asiakas valitsee tärkeimmät ominaisuudet, jotka toteutetaan ensimmäiseen julkaistavaan versioon, jotta asiakas saa mahdollisimman pian käyttökokemuksia järjestelmästä ja kehittäjätiimi puolestaan asiakaspalautetta. Järjestelmä toteutetaan siten, että

koodia kirjoitetaan ns. pariohjelmointina, eli toinen kehittäjä istuu koneen ääressä kirjoittaa ohjelmakoodia toisen tarkkaillessa, kommentoidessa ja miettiessä integrointiin ja arkkitehtuuriin liittyviä seikkoja. [36]

Sosiaalisten seikkojen lisäksi XP korostaa myös yhtenäisiä koodauskäytäntöjä. Koodauskäytännöt ovat kehittäjätiimin itse vapaasti valittavissa, mutta kun yhteen koodauskäytäntöön on päädytty, kaikkien on noudatettava sitä. Tällä tavoitellaan sitä, että kaikki tuotettu ohjelmakoodi näyttää siltä kuin se olisi yhden kehittäjän kirjoittamaa, eikä kehittäjäkohtaisia eroja ole. [37]

Ohjelmakoodi integroidaan aina mahdollisimman nopeasti osaksi järjestelmää ja testit ajetaan päivittäin automaattisesti. Kehittäjä on suunnitellut moduulitestit, joita tässä yhteydessä nimitetään yksikkötesteiksi, ennen ohjelmakoodin kirjoittamista. [36]

Organisaation kannalta XP-menetelmään uskotaan liittyvän myös suuria riskejä. XP on saanut erityisen paljon kritiikkiä siitä syystä, ettei dokumentteja tuoteta kuin vähin mahdollinen määrä ja koko kehitysprosessissa asetetaan yksilöiden kyvykkyyden varaan [38]. Kent Beck torjuu kritiikin vastaamalla, ettei pariohjelmoinnin vuoksi yhden kehittäjän lähteminen tiimistä ole riski, koska koodin tuottamiseen on osallistunut koko ajan vähintään kaksi ohjelmoijaa, jaetun omistajuuden -periaatteen ansiosta useampikin [36].

Tätä merkittävämpi argumentti XP:tä vastaan on kuitenkin, että sen uskotaan aiheuttavan vastustusta suurissa ja vakiintuneissa organisaatioissa [20]. Käytännön kokemukset ovat myös osoittaneet, että XP:n integroiminen olemassa oleviin prosesseihin ja kehitystiimeihin aiheuttaa kehitysvaiheiden päällekkäisyyksistä ja kommunikaatiovaikeuksista aiheutuvia ongelmia. Niissäkin on kuitenkin todettu, että haluttu laatutaso on säilytetty. [22] Koska näissä käytännön tilanteissa on jouduttu joustamaan mm. yrityksissä käytettyjen laatujärjestelmien hyväksi, eivät XP:n tavoitteet nostaa tehokkuutta ja samalla säilyttää tai parantaa laatua, ole toteutunut siinä määrin, että voitaisiin väittää XP:n käytön tulosten olevan ristiriidassa perinteisen laatuajattelun kanssa.

3.6.4 Muut prosessimallit

Vakiintuneiden prosessimallien lisäksi on olemassa prosessimalleja, joiden merkityksestä ei ole yksimielisyyttä. Tällaisia ovat ennemminkin kehitysfilosofiana pidetty avoimen lähdekoodin kehitys (OSS, Open Source Software) ja sen vastakohtana pidetyt kaupalliset prosessimallit.

Organisaatio ei välttämättä kehitä ohjelmistojaan itse. Tällöin ohjelmistotuotanto ulkoistetaan osittain tai täysin kolmannelle osapuolelle, jolloin yritykselle ei enää ole merkitystä sillä, kuinka ohjelmistoja kehitetään, vaan ohjelmistokehityksen tehokkuus ja sopimusten neuvottelu korostuvat [9]. Organisaatio saattaa kuitenkin hyötyä kolmannen osapuolen ohjelmistokehityksestä tai tuotteista muulla tavoin. Tästä on kyse myös avoimen lähdekoodin kehityksessä, jonka lähestymistapa ohjelmistotuotantoon on täysin erilainen, pääosin sen taustalla vaikuttavien BSD:n (Berkeley System Distribution) ja FSF:n (Free Software Foundation) GNU-lisenssin eli GPL:n (General Public License) johdosta [39].

Avoimen lähdekoodin projektit ovat yksittäisen kehittäjän tai pienen ydinkehittäjien joukon vetämiä vapaaehtoisprojekteja, joilla ei ole määriteltyä projekti- tai prosessimallia. Tästä huolimatta avoimen lähdekoodin projektit, kuten Linux-käyttöjärjestelmä ja Apache-palvelinohjelmisto, ovat kuitenkin menestyneet mm. saavuttamalla merkittäviä markkinaosuuksia. [39] Näistä syistä avoimen lähdekoodin kehitys antaa mielenkiintoisen näkökulman tälle diplomityölle. Ilman mitään eksplisiittisesti ilmaistavissa olevaa prosessimallia voidaan saavuttaa merkittäviä tuloksia. Mitkä ovat olleet siis avoimen lähdekoodin kehityksen keskeiset menestystekijät? Kiistämättä niitä ovat olleet motivoituneet kehittäjät, yhteiset tavoitteet ja tietyt käytännöt, joihin kuuluvat vapaasti valittavat työtehtävät ja rinnakkaisten kehityshaarojen välttäminen. Näiden lisäksi kehittäjäyhteisöllä on omat sääntönsä, joita noudatetaan, vaikkei niitä ole dokumentoitu [39]. Toisin sanoen kehittäjäyhteisöllä on oma yrityskulttuuria vastaava yhteishenki ja yleisesti hyväksytyt toteutusmenetelmät.

Vaikka avoimen lähdekoodin kehittäjäyhteisö poikkeaa monin tavoin mistä tahansa tavanomaisesta organisaatiosta, sen esimerkkiä voidaan pitää kannustavana, kun harkitaan joko oman prosessimallin kehittämistä tai olemassa olevan prosessimallin so-

veltamista. Myös RUP:n ja XP:n kehittäjät rohkaisevat soveltamaan prosessimallejaan [26] [36].

Täysin päinvastaisena vaihtoehtona yrityksen omalle prosessimallille voidaan nähdä kaupalliset, suurten ohjelmistotoimittajien kehittämät prosessimallit. Nämä kuitenkin ovat marginaalisessa asemassa ja tunnetuimpienkin prosessimallien käyttö perustuu täysin ohjelmistotoimittajien suureen markkinaosuuteen. Tunnetuimpia näistä ovat Oraclen AIM (Application Implementation Method) ja SAP:n ASAP (Accelerated SAP), joista etenkin jälkimmäisen laskeminen ohjelmistotuotannon prosessimalliksi ei ole täysin perusteltua, vaan kyseessä on ennemminkin pitkälle tuotteistettua konsultointia.

Vaikka ohjelmistotoimittajien prosessimallien kaupallisuusaste vaihtelee, yhteistä niille kaikille on, että ne perustuvat ohjelmistotoimittajien omien tuotteiden käyttämiselle. Tällä perusteella kaupallisiin prosessimalleihin voitaisiin luokitella myös RUP, sillä vaikkei se itsessään ole kaupallinen, se on pitkälle tuotteistettu mm. kurssimateriaalia myöten ja sen tavoitteena voidaan nähdä mm. Rational Rose ja Rational ClearCase -ohjelmistojen käytön lisääminen.

4 TOTEUTUSMENETELMÄT

Tässä kappaleessa käsitellään teoreettisen viitemallin yksilötason tekijät. Diplomityön rajaukseen perustuen, menetelmien kohdalla keskitytään yksinomaan toteutusmenetelmiin.

4.1 Toteutusmenetelmät ohjelmistokehityksessä

Kappaleessa 3.3 ohjelmistotuotannon menetelmät määriteltiin yleisesti ottaen. Tässä diplomityössä ohjelmistotuotannon toteutusmenetelmän käsitettä kuitenkin laajennetaan siitä, miten se yleisesti määritellään. Raja suunnittelu- ja toteutusvaiheen välillä saattaa olla häilyvä jo esimerkiksi työkalujen käyttötapojen vuoksi. Esimerkiksi oliomalli voidaan päivittää sen mukaan, millaisia muutoksia toteutusvaiheessa on tarvittu. Vastaavasti ohjelmistomoduulien toteutus ja testaus vuorottelevat keskenään, kun kehitys tapahtuu iteratiivisesti. Rajat eri vaiheiden välillä voivat olla liukuvia myös muista käytännön syistä. Toteutusvaiheessa on palattava päivittämään suunnitteludokumentteja tai kaavioita ja testauksesta on suora takaisinkytkentä toteutukseen, kun havaitut viat korjataan. Pahimmillaan myös testausvaiheessa voidaan todeta, ettei järjestelmä täytä sille asetettuja vaatimuksia.

Tässä diplomityössä viitataan toteutusmenetelmillä ohjelmistotuotannossa sovellettuihin toimintatapoihin, koodauskäytäntöihin ja työkaluihin. Menetelmien valintaan vaikuttaa myös organisaatio arvojen, pelisääntöjen ja yrityskulttuurin kautta.

Toteutusmenetelmiä käytetään ohjelmistomoduulien toteuttamiseen. Toteuttaminen ei ole yksinomaan ohjelmistokoodin kirjoitusta, vaan arkkitehtuurista ja suunnitteludokumenteista riippuen, suunnittelumallien soveltamista, rajapintojen toiminnallisuuden tarkistamista ja ohjelman toiminnan ymmärtämistä teknisellä tasolla. Tämän lisäksi, organisaatiosta ja sen prosesseista riippuen, toteutusvaiheeseen liittyy dokumentointia ja moduulitestausta.

4.2 Toteutusohjeistukset

Työntekijöillä kuuluisi olla käytettävissään aikaa oman työnsä kehittämiseen, kuten uusien menetelmien tutkimiseen. Usein työntekijät kuitenkin tuntevat, ettei tällaiseen

ole aikaa. Koska näistä asioista ei keskustella sovelluskehittäjien kesken, myöskään tietoa menetelmistä ei vaihdeta, eikä organisaatiossa oteta uusia menetelmiä käyttöön. [15] Innovatiivisuutta käsittelevässä kirjallisuudessa dynaaminen organisaatio nähdään ideaalisena tuotekehitysorganisaationa, kun pyritään innovatiiviseen toimintaan. Innovatiivisuuteen kuuluu keskeisesti tiedon kulkeminen – niin yrityksen sisällä kuin sen sidosryhmien välillä. [17]

Tiedon kulkemista käsittelevä kirjallisuus siis korostaa ihmisten välistä kommunikointia. Toteutusohjeistus ei voi korvata ihmisten keskinäistä tiedonvaihtamista. Parhaat käytännöt -dokumentti voi kuitenkin antaa pohjan sille keskustelulle, jota organisaation jäsenet käyvät toimintatavoistaan ja käyttämistään menetelmistä. Parhaimmillaan tällainen dokumentaatio on jatkuvasti kehittyvä ja heijastaa organisaatiossa kannatusta saavia toimintatapoja ja menetelmiä. Prosessimalli ja suositukset voidaan esitellä alihankkijoille ja vaatia määritellyn mukaista toimintaa. Sen sijaan kommunikointiin muiden sidosryhmien välillä parhaat käytännöt dokumentti ei kelpaa, koska se on liian toteutusläheinen.

Organisaatiossa tehdyt ohjeistukset ja suositukset voivat kattaa organisaation eri tasoja. Konsernisuositukset voivat olla suuria linjauksia, jotka koskevat teknologiavalintoja. Konsernitasoisia sopimuksia solmitaan suurten laitteisto- ja ohjelmistotoimittajien kesken, jolloin suosituksetkin ovat niihin sidoksissa. Suuret alihankintasopimukset voidaan myös tehdä konsernitasolla, jolloin alihankkijasuhde käytännössä rajoittaa alempien organisaatiotasojen päätöksiä.

Myös yksikkö- tai osastotasolla voi olla tiettyjä linjauksia teknologioista ja alustoista. Tällaiseen ohjeistukseen voi kuulua toteutusmenetelmät. Tällöin teknologiavalinnat ovat käytännönläheisempiä ja niiden tarkoituksena on ohjata tiimin työskentelyä. Alimmalla organisaatiotasolla voi jossain määrin olla myös projektikohtaisia sääntöjä. Lisäksi on huomattava, että ohjelmisto- ja laitelisenssit ohjaavat teknologiavalintoja kaikilla tasoilla, riippuen lisenssin kattavuudesta. Tässä diplomityössä parhaat käytännöt -dokumentti on kohdeyrityksen yksikkötason toteutusohjeistus, eli joukko toteutusvaiheen suosituksia.

4.3 Dokumenttipohjat ja dokumentit

Ohjelmistokehityksen prosessit määrittelevät, mitä dokumentteja missäkin vaiheessa on kirjoitettava ja mikä niiden sisältö on. Koska dokumentit ovat prosessiriippuvaisia, ne ovat tämän diplomityön rajauksen ulkopuolella. Dokumenttipohjat sen sijaan ovat prosessiriippumattomia, paitsi sisältönsä osalta.

Dokumenttipohjat nopeuttavat dokumenttien kirjoittamista tarjoamalla valmiin muotoilun ja rungon. Uutta dokumenttia kirjoitettaessa ei siis tarvitse miettiä, mitä asioita sen tulisi kattaa, koska kyseiset asiat on määritelty jo prosessissa. Tällöin kaikissa projekteissa tuotetaan yhtenäisiä dokumentteja.

Koska projektit vaihtelevat kooltaan, dokumenttipohjia varten tulisi olla valmiit ohjeistukset siitä, miten niitä voidaan kussakin tilanteessa soveltaa. Ohjeistuksen tarkoituksena on varmistaa, ettei dokumentista jätetä mitään kriittistä osa-aluetta pois, etenkään jos kyseinen osa-alue on prosessin kannalta tärkeä. Vaihtoehtoisesti organisaatiolla voi olla myös erilaisia dokumenttipohjia eri kokoisten tai eri tyyppisten projektien tarpeisiin. Dokumenttien kattavuutta voidaan valvoa esimerkiksi katselmointien avulla. Katselmointitilaisuuksien ajankohdat ja niihin osallistuvat tahot riippuvat organisaatiosta.

Dokumenttien kohdalla on huomattava, että toteutettu järjestelmä ei aina vastaa dokumentaatiota, eli se voi sisältää dokumentoimattomia ominaisuuksia tai se on kuvaus järjestelmästä sellaisena, kuin se ideaalitilanteessa olisi toteutettu. Syitä järjestelmän dokumentoimattomille ominaisuuksille ovat mm. se, ettei ominaisuutta tueta virallisesti, se on toteutettu tulevaisuutta varten tai se tahdotaan pitää salassa, koska se on huonosti toteutettu tai jopa suoranainen turvallisuusuhka. Usein erot dokumentaation ja toteutuksen välillä johtuvat kuitenkin vain siitä, että dokumentteja ei päivitetä, kun toteutuksen aikana joudutaan tekemään suunnitteluvaiheen ratkaisuja. [40]

Dokumenttien lisäksi dokumentaatiota on kaavioissa. Dokumentaatiota on myös standardeissa, julkaisuissa, testitapauksissa, postituslistoilla, uutisryhmissä, versionhallintatiedoissa, testiohjelmistojen tietokannoissa, markkinointimateriaalissa ja itse lähdekoodeissa sekä niiden kommenteissa [40]. Kaiken tämän yhdistäminen ei ole mahdollista ellei toimi osana ohjelmistokehitysorganisaatiota. Organisaation yksilöi-

den kokemuksen kautta kertynyt hiljainen tieto on siis keskeisessä roolissa ohjelmiston tuntemista silmällä pitäen.

4.4 Moduulisuunnittelu

Moduulisuunnitteluvaiheessa arkkitehtuuri on jo päälinjauksiltaan valmis. Arkkitehtuurisuunnitteluvaiheessa on tehty perusratkaisut, jotka vaikuttavat ohjelmiston laatuun. Käytännössä moduulisuunnittelussa ohjelmiston laatu, tietoturva ja muut vastaavat asiat lyödään pitkälti lukkoon. Tästä syystä moduulisuunnitteluvaiheessa tulisi noudattaa hyviä käytäntöjä, joita ovat mm. tietoturvaan ja henkilötietojen käsittelyyn liittyvät asiat [14]. Myös toteutusvaiheessa suunnittelumenetelmien tunteminen on tärkeää, koska arkkitehtuurisuunnittelun aikana, syystä tai toisesta, jokin komponentti saattaa jäädä suunnittelematta tai muuttuneet vaatimukset vaativat suunnittelemattoman komponentin lisäämistä järjestelmään.

Kaavioiden ja kuvaustekniikoiden myötä moduulisuunnitteluun liittyy paljon menetelmiä. Näitä menetelmiä voidaan nimittää yhtä hyvin suunnittelumenetelmiksi, mutta tässä diplomityössä niihin viitataan toteutusmenetelminä, koska kappaleessa 4.1 todettiin suunnittelu- ja toteutusvaiheen rajan olevan liukuva. Vaikka tiettyihin yksityiskohtiin, kuten kaavioiden piirtämiseen ja koodin generoimiseen kaavioiden pohjalta, on mahdollista laatia ohjeistuksia, ei kaavioiden laatimiseen ole yleisesti hyväksyttyjä menetelmiä. Samaa voitaisiin sanoa suunnittelumalleista, mutta koska niiden käyttäminen on luokiteltavissa konkreettiseksi suunnitteluvaiheen menetelmäksi, niitä käsitellään tässä diplomityössä.

Kokeneet kehittäjät huomaavat ongelmissa ja niiden ratkaisuissa toistuvaa samankaltaisuutta. Kyseisiä ratkaisuja nimitetään suunnittelumalleiksi. Suunnittelumallit ovat paitsi ratkaisuja ongelmiin, ne myös mahdollistavat keskinäisen ymmärryksen saavuttamisen suunnitteluratkaisun osalta. [41] Suunnittelumallit nousivat kasvavan kiinnostuksen kohteeksi oliosuunnittelun myötä. Koska ne ovat yleisluonteisia, niiden avulla suunniteltuja moduuleja voi käyttää uudelleen toisen ohjelmiston osana. Suunnittelumalleja voidaan myös yhdistellä toistensa kanssa. Suunnittelumallien tunnistamisen vuoksi on tärkeää, että moduulit jotka toteuttavat suunnittelumallin, nimetään lisäämällä käytetyn suunnittelumallin nimi moduulin nimeen. [40]

Suunnittelumallit luokitellaan kolmeen pääryhmään seuraavasti [42]:

- Luontimallit: abstrakti tehdas, rakentaja, tehdasmetodi, prototyyppi, ainokainen.
- Rakennemallit: sovitin, silta, rekursiokooste, kuorruttaja, julkisivu, hiutale, edustaja.
- Käyttäytymismallit: vastuuketju, komento, tulkki, iteraattori, välittäjä, muisto, tarkkailija, tila, strategia, operaatiorunko, vierailija.

Tässä esitetty lista ei ole kattava, jos sellaista on edes mahdollista laatia, mutta siinä ovat tunnetuimmat suunnittelumallit. Listasta puuttuvat mm. hajautukseen liittyvät suunnittelumallit ja eri teknologoihin läheisesti liittyvät suunnittelumallit [42]. Javan yhteydessä luokittelumallien jaottelu saattaa vaihdella ja yllä mainittujen kolmen pääluokan lisäksi mainitaan toisinaan muitakin ryhmiä ja suunnittelumalleja. [41]

4.5 Moduulien toteuttaminen

Moduulien toteuttamisella tarkoitetaan tässä yhteydessä ohjelmakoodin kirjoittamista, minkä yhteydessä tärkein yksittäinen asia on koodauskäytännöt. Koodauskäytäntöjen avulla täytetään ohjelmakoodille asetetut vaatimukset, joihin voivat kuulua:

- yhdenmukaisuus: kaikki ohjelmakoodi kirjoitetaan saman ohjeistuksen mukaisesti eikä vaihtelua projektien tai kehittäjien välillä ole.
- johdonmukaisuus: kaikki kehittäjät noudattavat valittua ohjeistusta, eivätkä käytännöt muutu kehittäjäkohtaisesti saman lähdekooditiedoston sisällä.

Koodauskäytännöt ovat sääntöjä, jotka määrittelevät miltä sovelluskehittäjän ohjelmakoodi näyttää. Koodauskäytäntöihin voivat kuulua yksityiskohtaiset ohjeet nimeämiskäytännöistä, koodirivin maksimipituudesta, ohjelmalohkojen vaikutusaluetta ilmaisevien sulkumerkkien käytöstä ja sisennykseen liittyvistä seikoista. Jälkimmäiseen kuuluvat myös valinnat sen suhteen, käytetäänkö sisennyksissä tabulaattorimerkkejä vai välilyöntejä, ja mikä on yhden sisennyksen pituus välilyönneissä laskettuna. Yksinkertaisenkin ohjelmakoodin voi siis muotoilla lukuisilla eri tavoilla. Koodauskäytännöistä tekee merkittävän se, että esimerkiksi sisennyskäytäntöjä sekoittamalla ohjelmakoodista tulee vähintään hämmentävää, pahimmillaan lukukelvotonta [40]. Kun tämä asetetaan siihen kontekstiin, että yhdessä ainoassa ohjelmistos-

sa on usein sadoista tuhansista aina miljooniin riveihin ohjelmakoodia, koodaussääntöjen vaikutus sovelluskehittäjien jokapäiväiseen työhön on huomattava.

Tunnetuimmat koodauskäytännöt ovat Javan tapauksessa Sunin koodauskäytännöt, avoimen lähdekoodin piirissä käytetyt GNU- ja BSD-koodauskäytännöt sekä erityisesti Windows-ohjelmoinnissa käytetty unkarilainen nimeämiskäytäntö. [40]

Koodauskäytäntöjen vaihteleminen johtuu mm. siitä, että eri kehittäjillä on käsitys siitä, mitä tarkoitetaan hyvillä ohjelmointitavoilla. Yhtä lailla kehittäjät ovat eri mieltä siitä, millaista ohjelmakoodia on helppo lukea [43]. Näihin käsityseroihin vaikuttavat todennäköisesti erilaiset näkemykset siitä, mikä on ohjelmakoodin tärkein ominaisuus: painotetaanko ohjelmistoteknistä laatua, ylläpidettävyyttä vai ratkaisun toimivuutta. Ohjelmistotekninen laatu tarkoittaa sitä, että moduuli on toteutettu siten, että se toimii ennakoitavasti kaikilla syötteillä, sen virheenkäsittely on kattava ja se on myös testattu riittävästi. Ylläpidettävä koodi puolestaan on dokumentoitu ja kommentoitu sekä helposti ymmärrettävää. Lähdekoodeihin kirjoitetut kommentit siis auttavat ymmärtämään toteutusvaiheessa tehtyjä ratkaisuja ja oletuksia [14]. Tätä käsitellään lisää kappaleessa 5.3.1, jossa esitellään erilaiset sovelluskehittäjätyypit.

Olivatpa syyt mitä hyvänsä vaihtelevia koodauskäytäntöjä, eli kehittäjäkohtaisista eroja, ei tulisi organisaatiossa hyväksyä, sillä niistä on käytännön haittaa. Kehittäjä tottuu tietyllä tavalla muotoilemaansa ohjelmakoodiin ja eri tavoin muotoiltua ohjelmakoodia on vaikeampi ymmärtää silmäilemällä. Erityisen hankala on tilanne, jossa saman lähdekooditiedoston sisällä koodauskäytännöt vaihtelevat kehittäjäkohtaisesti. Ongelmia syntyy myös, kun kehittäjä käyttää editoriaan muokkaamaan ohjelmakoodin automaattisesti tiettyjen muotoilusääntöjen mukaiseksi. Tällöin riittää että lähdekooditiedosto kerran avataan ja tallennetaan, jolloin koko tiedoston muotoilut ovat muuttuneet. Yllättävä haitta tästä seuraa käytettäessä versionhallintaohjelmistoa, joka tulkitsee kaikki muuttuneet rivit uuden version muutoksiksi. Kun tällaista kokonaan uusiksi muokattua versiota verrataan edelliseen versioon, ei versionhallintaohjelmiston avulla ole mahdollista havaita muutoksia, koska jokainen rivi on versionhallintaohjelman mielestä muuttunut. Ohjelmakoodia on tällöin verrattava jollain muulla ohjelmalla.

4.6 Työkalut

Tässä kappaleessa käsitellään tärkeimmät ohjelmistotuotannossa käytettävät työkaluohjelmat. Kappaleen ei ole tarkoitus olla kattava lista eri työkaluista, vaan käsitellä tärkeimmät Java-ohjelmoinnissa käytetyt työkaluohjelmat.

Tällaisia ohjelmistoja nimitetään CASE-välineiksi (Computer Aided Software Engineering). Koska kohdeorganisaatiossa käytetään lisensoituja, suhteellisen kalliita suunnitteluvaiheen ohjelmistoja, kuten Rational Rose -ohjelmaa, tässä keskitytään tarkastelemaan pelkästään toteutusvaiheen työkaluja, joiden kohdalla sovelluskehittäjillä on enemmän mahdollisuuksia käyttää omaa harkintaansa; riittää että ohjelmistoon on voimassa olevia lisenssejä tai että lisenssi sallii ohjelmiston käyttämisen yrityskäytössä. Tällaisia ovat mm. vapaan lähdekoodin ohjelmat.

4.6.1 Työkalut ohjelmistokehityksessä

Toteuttajilla on usein omat työkalunsa, joita he käyttävät. Yhteisten työkalujen puute vaikeuttaa työntekijöiden keskinäistä kommunikointia. [44] Työkalujen käyttäminen on sidoksissa myös työtapoihin, koska kehittäjillä on usein omat työhakemistonsa. Työkaluja tarkasteltaessa onkin siten laajennettava näkökulmaa käsittämään myös ympäristöt. Kehitysympäristön sanakirjamääritelmä kuuluu ”yleisesti kaikkien laitteiden ja ohjelmistojen joukko, jolla kehitystyötä tehdään” [8]. Yhteinen kehitysympäristö, jossa ohjelmakoodi käännetään, korjataan ja testataan, vaikuttaa yhtenäiseen toimintaan ja ehkäisee yhteensopivuusongelmia eri toteuttajien tuotosten välillä, koska järjestelmät eivät toimi kuten on määritelty [44]. Yhteisten ympäristöjen ja kehitysohjelmistojen käyttäminen ehkäisee laitteisto-, ohjelmisto- ja versioyhteensopivuusongelmat.

Yhteensopivuusnäkökulman lisäksi ohjelmointityökaluilla on perinteisesti tavoiteltu ohjelmoinnin yksinkertaistamista ja sen kautta toteutuksen tehokkuuden parantamista. Käytännössä tehokkuuden nostamisen on nähty liittyvän perustoimintojen automatisointiin ja yksinkertaistamiseen. [45] Tiettyyn rajaan asti tämä pitää paikkansa. Esimerkiksi kääntämistyökalut, kuten Make ja Ant, sekä versionhallintatyökalut ja erilaiset testausta avustavat välineet nopeuttavat työrutiineja.

Työkaluja ja menetelmiä yhdistävä oletus kirjallisuudessa on, että ohjelmistokehitystä, etenkin sen toteutusvaihetta, voidaan oikeilla työkaluilla ja menetelmillä nopeuttaa. Toinen yleinen oletus on, että kun jokin ohjelmistotuotannon työvaihe voidaan tehdä usealla tavalla, jokin niistä on yksiselitteisesti, tai tietyissä olosuhteissa, muita parempi [11]. Ohjelmistokehitykseen, kuten ohjelmakoodin kirjoittamiseen ja dokumentointiin, vaikuttavat kuitenkin myös sosiaaliset näkökohdat, joita korostetaan vain osassa ohjelmistokehitykseen kohdistuneissa tutkimuksissa [46].

Tässä diplomityössä esitetään, ettei yksittäisen ohjelmistokehityksen prosessin, kuten suunnittelu- tai toteutusvaiheen, automatisoiminen tuo tehokkuutta tai edes nopeutta. Sen sijaan prosessimalli ja organisaation kulttuuri ovat keskeisessä roolissa. Väite perustuu siihen empiiriseen havaintoon, että käytännössä yhdessä vaiheessa säästetty aika usein kostautuu siirryttäessä seuraavaan vaiheeseen. Esimerkiksi suunnitteluvaiheessa säästetty aika tarkoittaa usein tavanomaista suurempien suunnitteluratkaisujen tekemistä toteutusvaiheen aikana. Tällöin toteutusvaiheessa esiin nouseva ongelma todellisuudessa saattaa olla oire siitä, ettei prosessi sovellu organisaatiolle.

Havaintoon perustuen voidaan tehdä johtopäätös, että ohjelmakoodin tuottamiseen liittyviin toimintoihin kuluva aika on koko ohjelmistokehityksen kannalta vähemmän merkityksellinen asia kuin kirjallisuudessa on usein väitetty. Toteutuksen tehokkuus ei siis ole sitä, kuinka nopeasti ohjelmakoodia tuotetaan, koska ohjelmointiin kuuluu myös suunnitteluratkaisujen tekeminen ja kirjoitetun ohjelmakoodin testaaminen. Tätä johtopäätöstä tukee se, että ohjelmistoalan tuottavuuden on tiedetty olevan heikko jo vuosikymmeniä ja tutkimuksissa on toistuvasti todettu, etteivät tuotokset vastaa ohjelmistokehitykseen käytettyjä panoksia [47]. Ohjelmistoalan tuottavuuden heikko kehitys on kuitenkin edelleen kiistanalainen ja eri työkaluohjelmistojen toimittajat luonnollisesti kiistävät asian jyrkästi [48].

Lyhyemmin sanottuna on perusteltua väittää, että ohjelmistokehitystä tulee tarkastella kokonaisuutena ja prosessien kautta. Näin ollen tehokkuutta ei voida tavoitella vain työkalujen avulla, vaan tehokkuus toteutuu suoraan prosesseissa ja välillisesti toimintatavoissa. Toimintatavat ovat puolestaan muotoutuneet organisaation ja sen prosessimallin kautta, ne siis ovat osa menetelmiä.

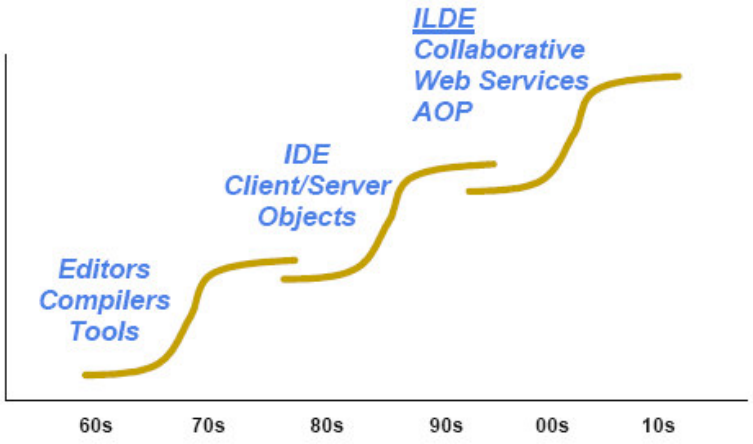
Työkalujen valintaa ei myöskään voida tarkastella ainoastaan tehokkuuden kannalta, vaan tavoitteiden ollessa esimerkiksi laatu tai uudelleenkäytettävyys, on organisaation valittava oikeat toteutusmenetelmät, niitä tukevat työkalut ja kehitettävä prosessimalli, jonka mukaiset prosessit toteuttavat asetetut tavoitteet.

Edellä esitetty ei tarkoita sitä, etteikö työkaluihin kannata kiinnittää huomiota. Päinvastoin, väitetään että ohjelmistokehityksessä käytetyt työkalut ovat monipuolistuneet ja siten nopeuttaneet ohjelmistojen tekemistä. Samojen väitteiden mukaan saavutettu etu kumoutuu ainoastaan siksi, että ohjelmistojen kehittäminen muuttuu vaikeammaksi, koska ohjelmistot itsessään monimutkaistuvat. [48] Seuraavaksi työkaluohjelmistoja ei käsitellä ensisijaisesti tehokkuuden kannalta, vaan tarkastellaan, miten ne vaikuttavat toteutusmenetelmien vaihtelemiseen.

4.6.2 Editorit ja IDE-kehitysympäristöt

Ohjelmakoodia voidaan kirjoittaa millä tahansa editorilla. Useimmissa ohjelmointiin tarkoitetuissa koodieditoreissa on saatavilla valmiita ohjelmointikielen syntaksin mukaan vaihtuvia värikoodeja ja näitä värikoodisääntöjä voidaan määritellä itse. Tällöin editori reagoi mm. ohjelmointikielen varattuihin sanoihin merkitsemällä ne käyttäjän määrittelemällä värikoodilla. Tällaisissa koodieditoreissa voi olla mukana myös ohjelmakoodin muokkaamiseen tarkoitettuja sääntöjä.

Tämän diplomityön kannalta työkalujen kehitysvaiheet eivät ole kiinnostavia, mutta oman näkökulman antaa se, missä määrin niitä käytetään. Kuva 4 esittää työkaluohjelmistojen markkinasyklit sovelluskehittäjän näkökulmasta. Yhdysrakenteiset kehitysympäristöt, lyhyemmin IDE-kehitysympäristöt, alkoivat yleistymään 1980-luvulla Borlandin Turbo Pascal -ohjelman myötä [49]. Kuvan mukaan tällä hetkellä näiden IDE-kehitysympäristöjen huippukausi olisi jo saavutettu ja tuotteiden, kuten Borlandin JBuilder -ohjelman, yhteydessä mainitaan siirtyminen kohti ILDE-kehitysympäristöjä (Integrated Life-Cycle Development Environment), jotka yhdessä Web Services -tekniikan ja AOP:n (Aspect-Oriented Programming) kanssa muodostavat kolmannen suuren syklin ohjelmointityökalujen markkinoilla [31].



Kuva 4. Työkaluohjelmistojen markkinasyklit [31].

Toteutusmenetelmien yhtenäistämisen kannalta IDE-kehitysympäristöt tarjoavat monia etuja ohjelmakoodin kirjoittamiseen. Etuihin kuuluvat mm. olemassa olevan ohjelmakoodin muotoilu tiettyjen sääntöjen mukaiseksi. Näitä sääntöjä voidaan myös tallentaa IDE-kehitysympäristön omiin konfiguraatiotiedostoihin, jolloin sovitut mallit voidaan jakaa vaikkapa yrityksen tiedostopalvelimen välityksellä ja ottaa nopeasti kaikkien käyttöön.

Meta Group -tutkimusyhtiö jakaa IDE-kehitysympäristöjä J2EE-alustalle (Java 2 Platform, Enterprise Edition) toimittavat yritykset johtajiin, haastajiin ja seuraajiin. Johtajiin lasketaan tavallisimmin Borland JBuilderin, IBM Eclipsen ja WebSpheren sekä Oracle JDeveloper-tuotteensa ansiosta, tässä järjestyksessä. [50] Myös Gartner-tutkimusyhtiön jaottelun mukaan IBM:n ja Borlandin tuotteet kuuluvat johtajiin [51]. Kyseinen johtoasema näkyy jatkuvana kehityksenä ja uusina ominaisuuksina. Uudet versiot JBuilder- ja Eclipse-ohjelmista suorittavat ohjelmakoodin kirjoituksen aikana taustalla käännösprosessia ja raportoivat kehittäjälle käännösvirheistä ja ns. kuolleesta koodista virheilmoitusten ja värikoodien avulla. Tällöin kehittäjä saa käytännössä reaaliaikaista palautetta kirjoittamansa koodin virheettömyydestä. Tämä johtaa laadukkaampaan ohjelmakoodiin, koska jos kehittäjä kirjoittaa paljon ohjelmakoodia, kiinnittyy huomio käännösprosessin aikana ohjelmakoodin kääntämisen onnistumiseen, ei tehtyihin ohjelmointivirheisiin. Tällainen jatkuvan palautteen saaminen on kuin alkeellinen versio XP-menetelmän pariohjelmoinnista eli työskentelytavasta,

jossa varsinaisen ohjelmoijan parina on toinen ohjelmoija joka tarkkailee ensimmäisen työskentelyä ja esittää kommentteja kehitteillä olevasta ohjelmakoodista [36].

Työkaluohjelmille asetettuja vaatimuksia on paitsi integroituminen ja kommunikointi toisten ohjelmistojen kanssa, myös kehittyminen ja sopeutuminen vastaamaan uusia ja odottamattomia asiayhteyksiä. Nämä ovat paitsi teknologiaan liittyviä, myös sosiologisia, kuten uudet toimialat, liiketoimintakäytännöt, prosessit ja säädökset sekä käyttäjät. Käytännössä kaikki tämä on kallista, eikä kenties lainkaan mahdollista, mikä johtuu siitä, että on mahdotonta ennustaa kuinka liiketoimintakäytännöt, prosessit, ihmiset ja tekniikka kehittyvät. [52]

5 TOTEUTUSMENETELMIEN YHTENÄISTÄMINEN

Tässä kappaleessa käsitellään toteutusmenetelmien vaihtelemisen syitä ja esitetään perusteet, miksi toteutusmenetelmät tulisi yhtenäistää. Kappale muodostaa siten teoreettisen pohjan käytännön osuudelle eli selvitystyölle.

5.1 Toteutusmenetelmät ja prosessit

Ideaalitilanteessa organisaation toteutusmenetelmät on kehitetty tukemaan käytettäviä prosesseja. Prosessien ja toteutusmenetelmien välisessä yhteydessä on kuitenkin tavallisesti nähtävissä kaksi näkökulmaa, joiden painotukset ovat erilaiset:

- painotus prosesseissa, toteutusmenetelmät määrittelemättömiä.
- painotus toteutusmenetelmissä, prosessit vapaasti määriteltävissä.

Kun painotus siirtyy prosesseista toteutusmenetelmiin, korostuvat samalla myös ihmiset ja heidän toimintansa. Näin on käynyt ainakin RAD:n ja ketterien menetelmien kohdalla. Erään ketterien menetelmien keskeisen vaikuttajan, Alistair Cockburnin, mukaan menetelmä on joukko sovelluskehitystiimin hyväksymiä käytäntöjä. Ydinidea tässä on, että koska projektitkin vaihtelevat, on menetelmää, eli tiimin omakseen ottamaa käytäntöjoukkoa, kehitettävä. [10]

Perinteiset prosessimallit, sen enempää kuin ketterät menetelmät, eivät kyseenalaista yhtenäisten toteutusmenetelmien tarvetta. Päinvastoin, esimerkiksi XP-menetelmässä pidetään mm. koodauskäytäntöjä välttämättöminä ja XP:n omia käytäntöjä tukevina. [10] Toteutusmenetelmiä siis tarvitaan, oli prosessimalli mikä tahansa.

5.2 Toteutusmenetelmien vaihteleminen

Toteutusmenetelmien vaihteleminen näkyy selvimmin epäyhtenäisenä ohjelmakoodina kappaleessa 4.5 kuvatuin tavoin. Toteutusmenetelmien vaihteleminen ei kuitenkaan rajoitu yksittäisiin ongelmiin, vaan on yhteydessä koko organisaation toimintaan. Epäyhtenäiset toteutusmenetelmät johtavat siihen, että kehittäjät toimivat eri tavoin ja siten toiminta vaihtelee myös projekteissa.

Kappaleessa 2.3 esiteltyyn teoreettiseen viitekehyksen tukeutuen, diplomityössä esitetään, että toteutusmenetelmät voidaan yhtenäistää kahden toimenpiteen avulla. Ensinnäkin organisaatiolla tulisi olla prosessimalli, joka määrittelee organisaation toiminnan päälinjat. Toisekseen organisaatiolla tulisi olla toteutusohjeistus, kuten parhaat käytännöt -dokumentti, joka sisältää suositukset, jotka vaikuttavat yksilötason toimintaan.

Koska diplomityön kohdeorganisaatiolla on prosessimalli, jota kehitetään voimakkaasti, ei diplomityössä käsitellä lainkaan prosessimallin käyttöönottamista, joskin kappaleen 3.6 yhteydessä käsiteltyjen prosessimallien yhteydessä arvioitiin sivumennen kunkin keskeisimmän prosessimallin mahdollista sopivuutta kohdeorganisaatiolle. Sen sijaan toteutusohjeistusta kohdeorganisaatiolla ei ole, minkä vuoksi tässä kappaleessa käsitellään toteutusmenetelmien yhtenäistämistä yksin tästä näkökulmasta. Tämä ei kuitenkaan tarkoita sitä, että organisaatiotaso voitaisiin jättää huomioimatta ja keskittyä vain yksilötasoon.

Koska tähän mennessä diplomityössä on jo käsitelty, mistä toteutusmenetelmien vaihtelemisessa on kyse ja kuinka sen vaikutukset näkyvät, voidaan seuraavaksi siirtyä vaihtelemisen taustoihin.

5.3 Toteutusmenetelmien vaihtelemisen syyt

Prosesseja alemmalla tasolla ei useinkaan ole mitään tyypillisiä toimintatapoja tai toteutusmenetelmiä. Tällöin näyttää siltä, että ohjelmistokehityksen luontevin toimintamalli olisi vakiintumaton toiminta, eli ad hoc -kehitys. Todennäköisempää on kuitenkin, että ihmisten persoonallisuudet, työskentelytyylit ja taustat vaihtelevat suuresti [21].

Tämä varsin yksinkertainen havainto siitä, että pohjimmainen syy toteutusmenetelmien vaihtelemiseen on se, että ohjelmistoyrityksissä kaikkialla maailmassa työntekijöiden työskentelytyylit vaihtelevat – aivan kuten persoonallisuudet ja taustatkin. Tästä syystä voidaan olettaa, ettei ole sellaisia prosesseja tai toteutusmenetelmiä, jotka soveltuisivat kaikille ohjelmistoyrityksille [10] [26]. Mikäli tämä oletus hyväksytään, voidaan kysyä mille tasolle toteutusmenetelmiä olisi yritettävä soveltaa: yrityksen laajuisesti, tiimikohtaisesti vai projektikohtaisesti.

Toteutusmenetelmät voivat muodostua kehittäjäkohtaisiksi, jos organisaatiossa sallitaan yksilölliset menetelmät ja toimintatavat. Kuitenkin jo tiimi- tai projektikohtaisesti vaihtelevat toteutusmenetelmät saattavat johtaa toimintamalliin, joka kehittäjistä vaikuttaa pahimmillaan kaaokselta. Tiettyjä menetelmiä, kuten esimerkiksi koodauskäytäntöjä, voidaan puolestaan soveltaa aina yritystasolle asti. Kaikki toteutusmenetelmät eivät kuitenkaan ole koodauskäytäntöjen tavoin aina samoina pysyviä, joten asiaa ei voi kuitata yksinkertaisilla toteamuksilla kehittäjien kurinalaisuudesta, vaan myös organisaatiotaso on otettava tarkasteluun mukaan.

Ennen huomion siirtämistä organisaatioihin, olivatpa ne tiimejä tai yrityksiä, tarkastellaan yksilöitä, sillä heistä muodostuu se kehittäjäyhteisö joka viime kädessä hyväksyy tai hylkää teknologioita. Jotta voidaan selvittää, millainen tämä joukko on, seuraavaksi tarkastellaan, kuinka sovelluskehittäjiä voidaan jaotella eri kategorioihin.

5.3.1 Kehittäjälähtöiset syyt

Sovelluskehittäjät voidaan jakaa mielenlaadun mukaan kolmeen kategoriaan [53]:

- opportunistinen kehittäjä.
- pragmaattinen kehittäjä.
- perfektionistinen kehittäjä.

Opportunistinen kehittäjä keskittyy tehokkuuteen; hänelle riittävät nopeat ratkaisut, jotka toimivat. Tällainen kehittäjä valitsee ensisijaisesti interaktiiviset kehitysvälineet. Pragmaattinen kehittäjä sen sijaan tavoittelee kestäviä ratkaisuja. Uudelleenkäytettävyys on hänelle tärkein asia, jolloin koodin ymmärtäminen, ongelman analysointi ja laadukkaat ratkaisut ovat keskeisiä. Perfektionistinen kehittäjä puolestaan pyrkii paitsi kooditasolla parhaisiin mahdollisiin ratkaisuihin, keskittyy myös järjestelmätasoon huomioiden suorituskyvyn ja ennustettavuuden. Perfektionisti käyttää ennemmin tavallisia koodieditoreja kuin IDE-kehitysympäristöjä. [53]

Kehittäjän mielenlaatuun vaikuttaa henkilökohtainen tausta eli koulutus, ohjelmointikielet sekä kokemus ohjelmistovirheistä ja työkaluista. Kehittäjät eivät kuitenkaan pysy koko työuraansa samassa kategoriassa, vaan jossain määrin kehittäjiä muokkaa koko ajan heidän käyttämänsä työkalut ja ohjelmointikielet. [53]

Tässä kappaleessa esitelty jaottelu on täysin yksilöllisistä eroista ja mieltymyksistä lähtevä. Sen perusteella on vaikeaa vetää linjoja sen suhteen, millaisia toimenpiteitä tarvitaan koko organisaation tasolla. Johdonmukaisempaa onkin tarkastella, löytyvätkö syyt yksilöllisiinkin eroihin taustaorganisaatiosta – siis yrityksestä itsestään.

5.3.2 Organisaatiolähtöiset syyt

Java-ohjelmistokehitystä tekevä kehittäjäyhteisö voidaan jakaa kolmen eri tyypin yrityksiin [54]:

- A-tyyppi: pioneerit.
- B-tyyppi: valtavirran yritykset.
- C-tyyppi: seuraajat.

Ensimmäiseen ryhmään kuuluvat yritykset, pioneerit, ovat valmiita ottamaan suuriakin riskejä käyttämällä uusimpia, kenties vielä kypsymättömiä teknologioita, saavuttaakseen kilpailuedun. B-tyypin yritykset edustavat valtavirtaa käyttäen sellaisia teknologioita, jotka ovat osoittautuneet käyttökelpoisiksi ja jotka ovat yleisesti käytettyjä. C-tyypin yritykset puolestaan ovat seuraajia; yrityksiä, jotka välttävät riskejä ja siten myös hyödyntävät uusia teknologioita viimeisinä. [54]

Yritystyypin mukaan myös Java-sovelluskehittäjät voidaan jakaa vastaavasti A-, B- ja C-tyyppeihin. A-tyypin sovelluskehittäjät pitäytyvät perinteisissä koodieditoreissa tai kevyissä IDE-välineissä. He valitsevat avoimen lähdekoodin ratkaisuja, kuten Linux-käyttöjärjestelmän ja Apache- tai Tomcat-palvelinohjelmiston. Heille on tärkeää löytää toimittaja- ja alustariippumattomia ratkaisuja, joten he usein päätyvät vähemmän tunnettuihin, mutta innovatiivisiin toimittajiin. [54]

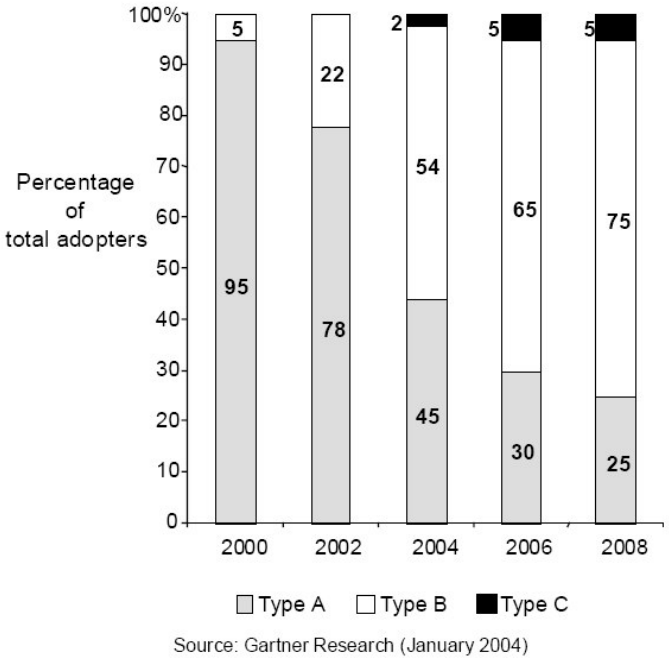
Keskimmäisessä, B-tyypin kehittäjäprofiilissa ovat ne, jotka kyllä ottavat hallittavissa olevia riskejä, mutta samalla luottavat kriittisen massan saavuttaneisiin ratkaisuihin. Tämä kehittäjätyyppi on kiinnostunein mm. RAD-kehitysvälineistä, koodia generoivista mallinnusohjelmistoista ja valmisohjelmistoista. [54]

C-tyypin kehittäjäprofiiliin sijoittuvat ne ohjelmistokehittäjät, jotka eivät halua tai pysty saavuttamaan tavanomaista asiantuntijuutta ohjelmoinnissa ja käyttävät mie-

luiten suunnitteluvetoisia tai muita korkean abstraktiotason välineitä, joiden avulla on mahdollista generoida ohjelmakoodia. [54]

Tällainen jako on luonnollisesti karkea yleistys ja on todettava, etteivät rajat niiden välillä noudata yrityksen rajoja. Suurissa yrityksissä myös yksiköt, osastot ja tiimit eroavat toisistaan. [54] Tämän perusteella voidaan olettaa, että taustalla on muitakin tekijöitä, kuten yrityskulttuuri, jotka vaikuttavat organisaatioihin ja niissä valittuihin toteutusmenetelmiin.

Tutkimusyhtiö Gartnerin tammikuussa 2004 tekemän ennustuksen mukaan suhteet näiden kolmen kehittäjätyypin välillä vaihtuvat vuoteen 2008 mennessä siten, että tyypin A-kehittäjiä on enää 25 %, tyypin B 75 % ja tyypin C-kehittäjiä 5 %, kun lähtötilanne vuonna 2000 oli, että A-tyypin kehittäjiä oli 95 % ja B-tyypin kehittäjiä vain 5 %. Suhteen muutosta esittävän kuvan 5 perusteella voidaan siis todeta, että Java on teknologiana kypsynyt myös kehittäjäyhteisön silmissä ja on käytännössä yksi valtavirtateknologioista. Kokonaisuudessaan Java-sovelluskehittäjien määräksi Gartner on arvioinut 5 miljoonaa kehittäjää. [54]



Kuva 5. Arvio Java-sovelluskehittäjätyyppien suhteista. [54]

5.4 Yhtenäistämisen tarve

Kuvassa 5 esitetty muutos vaikuttaa siten, että organisaation tulisi harkita alihankintaa tai parhaat käytännöt -ohjeistusta sellaisessa tapauksessa, jossa sillä ei ole A-tyypin teknologia-asiantuntijoiden muodostamaa ydinryhmää. [54] Tällainen johtopäätös lähtee kuitenkin oletuksesta, etteivät tyyppi A:n kehittäjät tarvitse noudattaa, tai he eivät halua noudattaa, yhtenäisiä työtapoja ja käyttää samoja toteutusmenetelmiä kuin muut. Tämä on varsin perustavanlainen oletus. Vaikka prosessien merkitys ymmärrettäisiin ja niiden kehitykseen panostettaisiin merkittäviä resursseja, edelleen oletetaan, etteivät sovelluskehittäjät lähtökohtaisesti tahdo noudattaa prosesseja ja toteutusmenetelmiä, vaan he tuntevat sen työn mielenkiintoisuuden laskemisena. Todellisuudessa on havaittu että yrityksissä, joissa on selkeät kehitysprosessit, tiimitasolla selvä organisaatiorakenne ja vakiintuneet käytännöt, sovelluskehittäjien turhautuminen laskee, koska samoja asioita ei tarvitse tehdä uudelleen. Samalla tuotteiden laatu paranee ja niiden markkinoille tuleminen nopeutuu. [21]

Vuonna 1995 tehdyn Standish Groupin tutkimuksen mukaan 84 % kaikista ohjelmistoprojekteista eivät valmistu tavoitellussa budjetissa, aikataulussa tai kaikkine suunniteltuine ominaisuuksineen. Lisäksi yli 30 % kaikista projekteista lakkautettiin ennen valmistumista. Tällaisten epäonnistumisen syiksi nähdään usein ohjelmistojen monimutkaisuus, kehityksen alkuvaiheisiin liittyvän epävarmuuden määrä, väärät suunnitteluvaiheen valinnat, muuttuvat asiakasvaatimukset ja muuttuvat teknologiat, mikä kattaa myös yleisen teknologisen kehityksen. Epärealistiset aikataulut vaikuttavat osan ihmisten motivaatioon ja 40 % kaikista ohjelmistovirheistä aiheutuu stressistä. [21] Samantyyppisiä näkemyksiä esitetään sellaisissa kirjallisuuslähteissä, jotka korostavat yrityskulttuuria tai ohjelmistokehityksen sosiaalisia ulottuvuuksia. Vastaavasti näissä lähteissä korostetaan yhtenäisen yrityskulttuurin ja yhtenäisten menetelmien merkitystä. Toisin kuin perinteisemmissä, teknologiakeskeisissä lähteissä laadun ja tehokkuuden parantamista ei siis lähestytä toteutusvaiheeseen kohdistuvien kehitystoimenpiteiden kautta vaan organisaation kehittämisprosessina. Tässä diplomityössä kuitenkin esitetään, että toteutusmenetelmien yhtenäistämisen kautta voidaan tavoitella etuja, jotka kohdistuvat koko organisaatioon.

5.5 Yhtenäistämisen tavoitteet

Toteutusmenetelmien yhtenäistämisellä voidaan tavoitella:

- Ohjelmistokehityksen tehostamista: ylläpidettävä ohjelmakoodi mahdollistaa jatkokehityksen.
- Ohjelmistojen laadun parantamista: selkeät säännöt kannustavat kurinalaiseen toimintaan.
- Työmotivaation parantamista: yhtenäinen toiminta edistää työssä viihtymistä ja tavoitteiden ymmärtämistä.

Tavoitteet ovat käytännössä samoja kuin kappaleessa 3.5 esitellyt prosessien tavoitteet, mikä on johdonmukainen väite koska diplomityön teoreettisen viitekehyksen mukaan menetelmät vaikuttavat suoraan prosesseihin. Kirjallisuuslähteiden perusteella voidaan todeta, että kaikki edellä mainitut tavoitteet ovat riippuvaisia toisistaan. Väitetään, että kun kaikki organisaation jäsenet voivat ottaa käyttöönsä parhaat saatavilla olevat ohjelmistotuotannon menetelmät, koko organisaation tasolla laatu ja tehokkuus paranevat [16]. Samoin väitetään, että henkilöstöasiat vaikuttavat tuottavuuteen ja ohjelmiston laatuun enemmän kuin mikään muu asia. Kyseiset henkilöstöasiat koostuvat motivaatiosta, tiimityöstä, henkilöstön valinnasta ja koulutuksesta. On kuitenkin huomattava, että ihmiset sitoutuvat ensisijaisesti tiimiin, eivät organisaatioon. Sitoutuminen puolestaan vaatii määriteltyjä tavoitteita, kuten myös tämän diplomityön viitemallissa esitettiin. [55]

5.6 Yhtenäistämisen edellytykset

Toteutusmenetelmien yhtenäistäminen tapahtuu organisaatiossa. Kappaleessa 5.3 esitettiin kysymys siitä, voiko toteutusmenetelmiä soveltaa yrityskohtaisesti, tiimikohtaisesti vai projektikohtaisesti, joista jälkimmäinen rajattiin tarkastelun ulkopuolelle ja siirryttiin tarkastelemaan sovelluskehittäjiä yksilöinä. Havaittiin, että sovelluskehittäjiä voitiin jakaa eri kategorioihin vastaavien yritystyyppien perusteella. Organisaation vaikutus sovelluskehittäjien toimintaan on tästäkin näkökulmasta katsottuna ilmeinen. On siis edelleen suunnattava huomio yrityskulttuuriin. Koska kappaleessa 2.3 esitetyn viitekehyksen mukaan organisaatio ja prosessimalli vaikuttavat toisiinsa, on perusteltua käsitellä organisaatiota osittain prosessimallin

kautta. Toisaalta jo prosessimallin valinta itsessään heijastaa organisaatiossa vallitsevia oletuksia ja uskomuksia siitä, millainen toiminta soveltuu sen yksilöille parhaiten.

Kappaleessa 3.6 käsiteltiin prosessimalleja kohtaan esitettyä kritiikkiä, minkä lisäksi prosessimalleja arvostellaan myös siitä syystä, että ne siirtävät työhön liittyvän päätöksenteon toteuttajilta prosessin kehittäjille. Ohjelmistokehityksen kaltaisessa toiminnassa tämä nähdään pahana siitä syystä, että ohjelmistokehitys on ajatustyötä, ei mekaanista työtä. Ulkoa annetut menetelmät ja toimintatavat on pystyttävä perustelemaan hyvin, koska muutoin niistä ei nähdä olevan hyötyä. [15]

Menetelmien standardoiminen prosessimallissa ei ole järkevää, koska ei ole vain yhtä ainoaa tapaa tehdä asioita, vaan niihin on olemassa aina useampia kilpailevia menetelmiä. Ihmisiä ei myöskään pitäisi pakottaa käyttämään sellaisia menetelmiä, joita he eivät tahdo käyttää. Menetelmien yhtenäistäminen kuitenkin on hyödyllistä ja sen avulla saavutetaan todellista hyötyä. [15] Näiden syiden perusteella toteutusohjeistuksen tulisi olla joukko vapaaehtoisuuteen perustuvia suosituksia. Jotta näillä suosituksilla olisi jotain käytännön merkitystä myös yksilöiden, samoin kuin koko organisaation, on oltava valmis hyväksymään ne.

Kirjallisuudessa esitetään lukuisissa lähteissä, kuinka prosessikehityksen tulee olla asteittaista, jatkuvaa ja toistuvaa. Organisaation ja sen yksilöiden on siis muututtava. Yksilöt kuitenkin muuttuvat vasta, kun heitä kannustetaan siihen. Prosessimuutosten tulisi olla tavoitekeskeisiä, tieto-ohjautuvaa ja uskomuksiin perustumatonta. Kehitysaktiviteetteihin olisi lisäksi suhtauduttava miniprojekteina. [16]

5.6.1 Edellytykset yksilötasolla

Jokainen organisaatio koostuu yksilöistä, jotka vaikuttavat myös sen kehitykseen. Kaikki organisaation jäsenet eivät tahdo muutosta. Tämä voi johtaa organisaation sisäisiin ristiriitoihin, kun osa kannustaa uusia työtapoja ja osa vastustaa. Parhaassa tapauksessa ydinjoukko vetää perässään ne, jotka epäilevät muutoksen tarvetta ja sen odotettuja etuja. Vastustajat puolestaan yrittävät vetää koko tiimin takaisin siihen, mihin he olivat tyytyväisiä tai aikansa valitettuaan lopulta lähtevät organisaatiosta. [16]

Yleisesti ottaen sovelluskehittäjät käyttävät mitä tahansa lähestymistapoja sovellusten suunnittelussa ja toteutuksessa. Osittain tämä johtuu ohjelmistokehityksen uutuudesta ja monimutkaisuudesta, osaltaan taas koska tietojenkäsittelytiedettä opetetaan teoreettisesti. Lisäksi osa kehittäjistä vastustaa mitä tahansa, minkä he tuntevat rajoittavan omaa luovuuttaan. [16]

Yhtenäisiä menetelmiä ei aina siis oteta käyttöön. Kun näin tehdään, menetelmien tulee olla myös tarpeeksi mietittyjä, sillä yksilöt saattavat noudattaa niitä täysin kirjaimellisesti, jolloin menetelmien käyttöönottaminen pahimmassa tapauksessa johtaa vääränlaiseen toimintaan. [15] Parhaassa tapauksessa yhteiset periaatteet ja arvot kuitenkin tarjoavat monia mahdollisuuksia työympäristön ja työn tuloksien parantamiseen [16]. Toteutusmenetelmät ovat prosessien ja tavoitteiden lisäksi konkreettinen ilmentymä organisaatiossa vallitsevista periaatteista ja arvoista.

Dokumentoitu joukko menettelytapoja muodostaa organisaation yhteisen muistin, johon tallennetaan arvokkaimmiksi osoittautuneet tekniikat. Kun nämä menetelmät ovat kehitystiimin rutiinimaisesti käyttämiä, ne määrittävät organisaation ohjelmistokehityskulttuurin näkyvän, eksplisiittisen osan. Menetelmien on kuitenkin oltava sellaisia, joita todella käytetään päivittäisessä työssä. [16]

Huolellisesti valitut ja sovelletut ohjelmistokehityksen menettelytavat ja standardit voivat edistää ohjelmistokehitystä osoittamalla parhaat tunnetut tavat harjoittaa sovelluskehitystä. Muussa tapauksessa nämä standardit koetaan vain pakolliseksi ja tarpeettomiksi. Lopulta, kuten teknologiat, prosessit ja sovellusalueetkin muuttuvat, myös organisaatioiden ohjelmistokehityksen menettelytapojen täytyy kehittyä. [16] Kuten edellisessä kappaleessa mainittiin standardien sijasta toteutusohjeistuksien tulisi olla suosituksia. Tämän oletetaan helpottavan toteutusohjeistuksen hyväksymistä ja käyttöönottamista.

Valittujen menetelmien noudattaminen johtaa toivottua ohjelmistokehityskulttuuria kohden. Näiden menetelmien seuraaminen vaatii yksilöiden käyttäytymisen muuttamista. Kun ihmisten tahdotaan muuttavan toimintaansa, on muokattava koko organisaation yhteisiä arvoja ja filosofioita, sillä muutos voimistaa itseään [16].

5.6.2 Edellytykset organisaatiotasolla

Organisaatiokulttuuri, tai yrityskulttuuri, on organisaatiokohtainen joukko yhteisiä uskomuksia, tavoitteita ja arvoja, jotka heijastuvat organisaation prioriteetteihin ja toimenpiteisiin. Yrityskulttuuri vaikuttaa yrityksen tuotteiden laatuun, yrityksen sovelluskehittäjien tuottavuuteen ja tiimityöhön. Ohjelmistotuotantokulttuuri koostuu laatuorientoituneista asenteista, ihmisten kanssakäymisestä ja teknisistä prosesseista. [16]

Yrityksessä on kaksi kulttuuritasoa: eksplisiittinen ja hiljainen. Ohjelmistokehitykseen liittyvät standardit ja menetelmät ovat dokumentoituja tai muutoin sanallisessa muodossa ilmaistuja odotuksia siitä, kuinka sovelluskehittäjien tulisi toimia. Niitä ei välttämättä kuitenkaan noudateta. Hiljaisella tasolla puolestaan viitataan kirjoittamattomaan kulttuuriin eli siihen, kuinka yrityksessä todellisuudessa toimitaan; ”kuinka asiat oikeasti tehdään”. Hiljainen taso perustuu luottamukseen ja keskinäiseen kunnioitukseen organisaation jäsenten kesken ja se ilmenee yhdenmukaisuutena, siinä mitä johtajat ja tiimin jäsenet sanovat ja tekevät. [16]

Nämä kulttuuritasot eivät aina vastaa toisiaan, mikä aiheuttaa epävarmuutta. Ohjelmistokehitykseen liittyy paljon epävarmuutta, jota voidaan vähentää kommunikaation avulla. Ihmisten välinen epämuodollinen kommunikointi on havaittu tehokkaaksi, ei ainoastaan muodollinen dokumentaatio. Koska myös prosessimallia voidaan pitää kommunikoinnin välineenä, väärin valittu prosessimalli voidaan välttää antamalla tiimin itsensä valita prosessinsa. [6] Sama koskee toteutusmenetelmiä.

6 SELVITYSTYÖN TOTEUTUS

Tässä kappaleessa kuvataan diplomityön käytännön osuutta, eli kuinka selvitystyön data kerättiin kohdeorganisaatiosta kyselyn, haastattelujen ja havaintojen avulla ja kuinka data analysoitiin. Lopussa käsitellään tulokset ja suositellut toimenpiteet.

6.1 Nykytilan kuvaus

Kohdeorganisaatiossa korostetaan prosesseja ja niiden kehitykseen panostetaan sekä henkilöresursseja että aikaa. Yksikössä on kuitenkin todettu, ettei mikään käytössä olleista prosessimalleista ole soveltunut pieniin projekteihin, eikä niitä ole voitu soveltaa haltuunottoprojekteissa, eli projekteissa joissa kolmannen osapuolen projekti tuotoksineen otetaan kohdeorganisaation haltuun. Koska prosessimalli ei ole julkinen, sitä ei voida diplomityössä esitellä. Sen sijaan tässä kappaleessa kuvataan kohdeorganisaation nykytilaa yleisellä tasolla, koska se auttaa ymmärtämään datan keräämistä käsitteleviä kappaleita.

Kohdeorganisaation toteuttamien projektien lähdekoodit, tekninen suunnitteludokumentaatio, kaaviot, lähdekoodit, konfiguraatiotiedostot ja kääntämiseen käytetyt tiedostot tallennetaan versionhallintajärjestelmään, joka on Rational ClearCase. Erityistapauksissa lähdekoodeja ylläpidetään ClearCasea edeltäneessä CVS-järjestelmässä (Concurrent Versions System). Muu projektidokumentaatio ja tiedostot tallennetaan projektihakemistoon, joka on vakiohakemistorakenne verkkopalvelimella ja jonne on annettu rajoitetut käyttöoikeudet. Projektihakemistoon tallennetaan projektinhallintaan liittyvä dokumentaatio, kokousmuistiot, raportit, asiakkailta saatu dokumentaatio, projektiin liittyvät standardit ja muut dokumentit. Suunnitteludokumenteista projektihakemistoon tallennetaan dokumentin viimeisin versio ja katselmointivalmiit dokumentit.

Projektiin liittyvien kokousten ja dokumenttien lisäksi projektitiimi vaihtaa tietoa epämuodollisesti sähköpostin ja IRC-ohjelmien (Internet Relay Chat) avulla. IRC-palvelin on kohdeyrityksen omalla palvelimella ylläpitämä ja sinne luodaan projektia varten omat kanavat, joiden välityksellä koko projektiryhmä jakaa tietoa reaaliajassa

keskenään. Tämä on todettu tehokkaaksi tavaksi vaihtaa tietoa, koska projektitiimi voi olla maantieteellisesti erillään toisistaan.

Projekteissa käytetään mallintamiseen UML-kaavioita, jotka tehdään suunnitteluvälineenä käytetyn Rational Rose Modeler Edition -ohjelman avulla. Tietokantasuunnittelussa käytetään puolestaan Enterprise Edition -versiota, jolla suunnitellaan tietokannan taulut ja niiden väliset relaatiot.

Testaajat käyttävät Mercury Test Directory -järjestelmää, jonka avulla kirjatulle ohjelmistovirheelle annetaan prioriteetti ja kuvaus, jonka jälkeen se siirretään oikean kehittäjän vastuulle. Kun virhe on korjattu, kehittäjä kirjaa ongelman syyn, tekemänsä korjaukset, muuttaa ongelman tilan ja siirtää sen testaajan vastuulle, joka suorittaa testit uudelleen.

Näiden ohjelmien lisäksi kohdeorganisaatiossa käytetään lukuisia muita työkaluja, joiden käyttöä ei kuitenkaan diplomityön rajoissa ole perusteltua käsitellä. Niitä ovat projektinhallintaohjelmisto, työryhmäohjelmisto neuvottelujen sopimiseen, joukko toimisto- ja piirto-ohjelmia sekä sisäisiä järjestelmiä, kuten taloushallinnon ohjelmisto ja tuntikirjausjärjestelmä.

6.2 Datan kerääminen

Kappaleessa 2.1 esiteltiin perustelut, joiden mukaisesti datan kerääminen suoritettiin kyselyn, haastattelujen ja havaintojen avulla. Tässä kappaleessa kuvataan, kuinka data kerättiin kohdeorganisaatiosta.

6.2.1 Kysely

Kyselyä varten laadittiin 34 kysymyksestä koostunut kyselylomake (liite 1), joka lähetettiin sähköpostilla kohdeorganisaation työntekijöille 7.2.2005. Kyselyn esittelysivulla kerrottiin, että kyselyn tuloksia käytettäisiin kehitettävässä parhaat käytännöt -dokumentissa ja tässä diplomityössä, ja että vastauksia käsiteltäisiin luottamuksellisesti. Lisäksi vastaajia neuvottiin, kuinka palauttaa vastaukset. Palautusvaihtoehdoiksi annettiin kyselylomakkeen palautus sähköpostilla, tulostettuna paperilla diplomityön tekijän postilokeroon tai palautus kirjepostilla. Vastausaikaa annettiin kaksi

viikkoa. Ensimmäisen viikon jälkeen lähetettiin muistutus, jossa kehotettiin vastaamaan kyselyyn.

Kyselylle asetettiin seuraavat tavoitteet:

- Selvittää, kuinka kohdeorganisaatiossa suhtauduttiin menetelmien yhtenäistämiseen.
- Selvittää, mitä menetelmiä ja työkaluja kohdeorganisaatiossa käytettiin.
- Selvittää, mitä toteutusmenetelmiä ja työkaluja kohdeorganisaation työntekijät suosittelivat.
- Selvittää, kuinka kohdeorganisaatiossa käytettiin suunnittelumalleja, tuotettiin Java-koodia, dokumentoitiin ja testattiin ohjelmistomoduuleja.
- Selvittää, mikä olisi sopiva tapa esitellä yhtenäiset toteutusmenetelmät kohdeorganisaation jäsenille.

Ensimmäinen kysymys koski vastaajan roolia kohdeorganisaatiossa, koska oletettiin että dataa voitaisiin luokitella roolin perusteella. Vastausvaihtoehtoihin ei lisätty mm. esimiesasemassa olevia henkilöitä, sillä kysely oli kohdistettu projektien toteutustason tehtäviin osallistuville. Vastausvaihtoehdoissa käytettiin termejä ”analysti” ja ”arkkitehti”, koska ne olivat vakiintuneita käsitteitä kohdeorganisaatiossa. Vastaavasti myös kysymyksen 11 kohdalla käytettiin suunnittelumallien englanninkielisiä nimiä, koska niiden käyttö osana Java-luokkien nimiä oli vakiintunutta.

Seuraavat kahdeksan kysymystä kartoittivat, pitäisikö suunnittelu-, toteutus- ja testausvaiheen menetelmiä, työkaluja sekä koodauskäytäntöjä käyttää nykyistä enemmän. Näistä viimeinen, kysymys yhdeksän, oli vapaamuotoinen.

Kysymykset 12-21 koskivat työkalujen käyttämistä ja yleisiä ohjelmistokehitykseen liittyviä asioita. Näihin kysymyksiin annetut vastausvaihtoehdot perustuivat diplomityön tekijän kokemuksiin ja oletuksiin kohdeorganisaation toiminnasta. Kysymykset 22-26 puolestaan kartoittivat konkreettisia ohjelmakoodiin liittyviä seikkoja ja kysymykset 27-31 kartoittivat vastaajien asennoitumista dokumentaatioon ja ohjelmakoodin kommentoimiseen. Viimeiset kolme kysymystä, kysymykset 32-34 liittyivät moduulitestaukseen.

Kerätty data tallennettiin Microsoft Excel -taulukkolaskentaohjelmalla, jotta dataa voitiin tarkastella mm. vastaajien roolin perusteella. Kunkin vastaajan valitsemalle vaihtoehdolle annettiin numeroarvoksi yksi, jotta taulukkolaskentaohjelman avulla voitiin laskea eri vastausvaihtoehtojen saamat prosentuaaliset osuudet jokaisen kysymyksen kohdalla. Kysymyksen 11 kohdalla vastausvaihtoehdoille annettiin kuitenkin numeroarvot yhdestä kolmeen, jotta vastauksia voitiin havainnollistaa pylväsdiagrammin avulla. Vapaamuotoiset kommentit ryhmiteltiin kysymyskohtaisesti omaan taulukkoonsa, siten että yhden vastaajan kaikki kommentit olivat yhdellä rivillä.

6.2.2 Haastattelut

Haastatteluja varten laadittiin 16 kysymystä, joita käytettiin haastattelun kulun ohjaamiseen (liite 2). Haastattelut äänitettiin käyttäen mikrofonia ja kannettavaa tietokonetta, johon oli asetettu Audacity-ohjelman versio 1.2.3. Kun kaikki haastattelut oli äänitetty, ne kuunneltiin yksitellen ja kirjoitettiin tekstimuotoon. Haastatteluja analysoitiin luokittelemalla vastauksia Microsoft Excel -ohjelman avulla. Kategorioihin valittiin tapauskohtaisesti joko kokonaisia vastauksia tai vastauksen osia.

Haastattelulle asetettiin seuraavat tavoitteet:

- Selvittää, mitkä olivat toteutusmenetelmien vaihtelemisen syyt kohdeorganisaatiossa.
- Selvittää, noudatettiinko kohdeorganisaatiossa sen omaa prosessimallia vai otettiinko vaikutteita jostain muusta prosessimallista.
- Selvittää, miten prosessimalli vaikutti toteutusmenetelmien vaihtelemiseen.
- Selvittää, miten toteutusmenetelmien vaihteleminen vaikutti prosessimalliin.
- Selvittää, kuinka kohdeorganisaatiossa suhtauduttiin toteutusmenetelmien yhtenäistämiseen.

Haastateltujen valinta perustui kahteen pääkriteeriin. Ensinnäkin kaikilta haastateltavilta edellytettiin vähintään kolmen vuoden työkokemusta kohdeorganisaation palveluksessa. Toinen valintakriteeri oli, että haastateltavien oli oltava edelleen mukana projektien toteutustehtävissä. Näiden valintaperusteiden oletettiin takaavan tarpeeksi kokemukseen perustuvia näkökulmia haastattelun tarpeisiin. Lisäksi asetettiin

lisävaatimus, etteivät valitut henkilöt olleet saaneet toimia yhdessä kuin enintään kahdessa projektissa vuoden 2004 aikana. Tällä varmistettiin, etteivät samoja projekteja koskevat kokemukset ylikorostuneet. Kriteerit täyttäneistä työntekijöistä valittiin haastateltaviksi yhteensä neljä. Näistä yksi toimi projekteissa arkkitehtina, muut sovelluskehittäjinä.

6.2.3 Havainnot

Havainnoille asetettiin seuraavat tavoitteet:

- Selvittää, kuinka toteutusmenetelmät vaihtelivat eri projektien ja eri toteuttajien välillä.
- Selvittää, kuinka toteutusmenetelmien vaihteleminen näkyi projektin tuotoksissa.

Tavoitteisiin perustuen havaintojen kohteiksi valittaville projekteille asetettiin seuraavat vaatimukset:

- Projektien toteuttamisvaiheeseen oli osallistunut vähintään neljä kehittäjää.
- Projektien kehittäjistä ainakin kaksi oli osallistunut kaikkiin valittuihin projekteihin.
- Projekteissa käytettiin ohjelmointikielenä Javaa.
- Projektien toteutusvaihe oli ajoittunut vuoteen 2004.

Vaatimusten perusteena oli oletus, että kun lähdekoodia oli tuotettu runsaasti usean ohjelmoijan toimesta, myös yksilökohtaiset erot tulisivat paremmin esiin. Vaatimus samojen toteuttajien osallistumisesta valittuihin projekteihin oli keskeinen toteutusmenetelmien vaihtelemista silmällä pitäen. Kyseinen vaatimus oli myös merkittävin tekijä, joka rajasi valittavia projekteja. Ohjelmointikielen valinta oli puolestaan diplomityön rajauksen mukainen ja projektien ajoittuminen vuoteen 2004 varmisti, että tarkastelu kohdistuisi kohdeyrityksen nykyiseen tilanteeseen.

Kohdeorganisaation toteuttamista projekteista kaksi täytti asetetut vaatimukset. Koska projektit ovat salaisia, niihin viitataan nimillä ”projekti A” ja ”projekti B”. Projekti A:n päävaihe päättyi vuonna 2004, mutta jatkokehitystä oli tehty myös tammikuussa 2005. Projekti B oli päättynyt kokonaan vuonna 2004. Projektidokumentaatio ja lähdekoodit olivat siis valmiita havaintojen tekohetkellä. Versionhallintaoh-

jelman mukaan projektissa A oli yhteensä 400 java-päätteistä tiedostoa ja projektissa B niitä oli yhteensä 172.

Molemmat projektit olivat jatkokehitysprojekteja, eli ne perustuivat olemassa oleviin järjestelmiin. Projekti A:n kohdalla edellisen version tuotoksia, kuten lähdekoodeja, käytettiin sellaisenaan uuden järjestelmän pohjana. Projektissa B entisiä lähdekoodeja ei käytetty, vaan projektissa vaihdettiin toteutustekniikkaa ja ohjelmointikieltä. Molempien projektien ohjelmointikieli oli Java.

Projektin A toteuttamiseen osallistui neljä kehittäjää, projektissa B kehittäjiä oli yhteensä kuusi. Näistä kolme olivat olleet mukana molempien projektien toteuttamisvaiheessa. Diplomityön tekijä oli yksi kyseisistä kehittäjistä, joten kyseessä oli osallistuva havainnointi, kuten kappaleessa 2.1 todettiin.

Havaintoja tehtiin projekteissa tuotetuista dokumenteista ja Java-lähdekooditiedostoista. Tämän lisäksi käytettiin hyväksi empiirisiä havaintoja. Projektidokumentaatiota ei verrattu prosessimalliin, koska uusi projektimalli ei ollut vielä täysin otettu käyttöön valittujen projektien toteutusvaiheen aikana. Havaintoja ei myöskään tehty projekteihin liittyvistä hallinnollisia asioita käsittelevistä dokumenteista, kuten sopimuksista ja projektisuunnitelmista, koska ne eivät olleet projektin toteutusvaiheen tuotoksia. Myös muu dokumentaatio, jota projektin käytännön toteuttamiseen osallistuneet kehittäjät eivät itse olleet tuottaneet, jätettiin huomioimatta. Kyseinen dokumentaatio oli raportteja, markkinointimateriaalia, esityksiä ja asiakkaiden lähettämiä dokumentteja.

Havaintojen tekemistä vaikeutti merkittävästi se, että projektin B kohdalla suunnitteluvaiheesta oli vastannut eri yksikkö, kuin diplomityön kohdeorganisaatio. Kyseisen yksikön suunnittelumenetelmät poikkesivat täysin kohdeorganisaatiossa käytetyistä.

6.3 Datan analysoiminen

Tässä kappaleessa analysoidaan kyselyn, haastattelujen ja havaintojen avulla kohdeorganisaatiosta kerättyä dataa.

6.3.1 Kysely

Kysely lähetettiin yhteensä 79 henkilölle ja vastauksia saatiin yhteensä 27. Kyselyn vastausprosentti oli siten 34,2. Koska tavoitteeksi asetettiin 40 %, mikä olisi vastannut arviolta kahta kolmasosaa projektien toteutukseen osallistuvista työntekijöistä, asetettua tavoitetta ei kyselyn osalta saavutettu. Koska kyselyn otos oli erittäin pieni, sillä ei myöskään ole tilastollista validiteettia. Näistä syistä johtuen voidaan todeta, ettei kysely tue selvitystyötä tarjoamalla tilastollista dataa. Sen sijaan kyselyn tulosten pääasiallinen rooli oli tukea selvitystyön aikana suoritettuja haastatteluja, minkä katsottiin olevan riittävä syy käsitellä kyselyä ja sen tuloksia tässä diplomityössä.

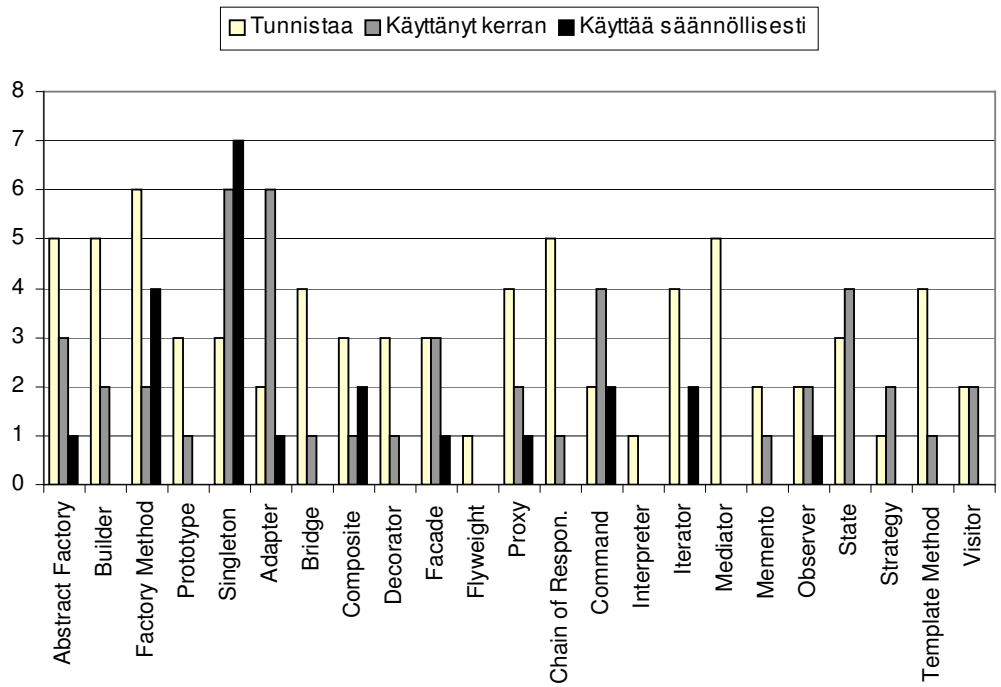
Kyselyyn vastanneista 13,8 % ilmoitti roolikseen analysti, 17,2 % arkkitehti, 27,6 % suunnittelija, 20,7 % ohjelmoija, 6,9 % testaaja ja 13,8 % projektipäällikkö, joista eräs täsmensi olevansa osastopäällikkö. Dataa tarkasteltaessa havaittiin, että myös projektipäälliköt, samoin kuin analystit ja testaajat, olivat usein ilmoittaneet mielipiteensä sellaisiinkin kysymyksiin, jotka eivät liittyneet keskeisesti heidän työhönsä. Tämän oletettiin johtuvan siitä, että useilla oli kokemusta muissakin rooleissa toimimisesta. Tästä syystä, sekä vastausten pienestä määrästä johtuen, dataa ei käsitellä tässä kappaleessa vastaajan roolin perusteella luokiteltuna, sillä keskeisten kysymysten kohdalla kyseisen luokittelun ei havaittu paljastavan datasta uusia tai merkittäviä näkökulmia.

Tulokset olivat poikkeuksellisen yhteneviä kysymysten 2-5 kohdalla. 92-100 % vastanneista ilmoitti, että suunnittelu-, toteutus- ja testausmenetelmiä sekä koodauskäytäntöjä tulisi käyttää nykyistä enemmän. Tästä syystä tulosten osalta oli pakko punnita sitä, että kysymykset olivat joko väärin laadittuja tai kysymysten laatija vaikutti kohderyhmään. Jälkimmäistä vaihtoehtoa ei tutkittu, koska tämän diplomityön tekijällä ei ole pätevyyttä arvioida, kuinka vaikutti se tosiseikka, että osa kyselyyn vastanneista tunsi kyselyn tekijän mielipiteet. Ensimmäinen vaihtoehto puolestaan hylättiin siitä syystä, että vaikka vastausvaihtoehdot olisi voinut laatia eri tavoin, kysymykset itsessään olivat yksiselitteisiä. Kysymyksiin saatiin myös runsaasti vapaamuotoisia kommentteja, jotka rohkaisivat tulkitsemaan, että yhtenäisiä menetelmiä tuettiin laajalti kohdeorganisaatiossa. Esimerkkejä tyypillisistä vastauksista ovat mm. kysymyksessä kaksi ”Standardoiduilla menetelmillä yhteistyö tiimin sisällä helpot-

tuu, voidaan huomioida paremmin käyttäjän tarpeet ja jatkokehitys helpottuu.” tai kysymyksessä kolme ”Käyttämällä standardoituja menetelmiä ei tarvitse keksiä pyörää kovin monta kertaa uudelleen, vaan voidaan keskittyä paremmin todellisten ongelmien ratkaisemiseen.”.

Kysymysten 6-8 kohdalla, jotka kartoittivat työkalujen yhtenäistämistä, oli enemmän hajontaa. Vastaajista 70,4 % toivoi, että yhtenäisiä suunnittelutyökaluja tulisi käyttää nykyistä enemmän, mutta vain 55,6 % toivoi samaa toteutustyökalujen kohdalla. Yhtenäisiä testausvaiheen työkaluja toivoi käytettävän nykyistä enemmän 74,1 % vastaajista.

Kysymyksissä 10 kartoitettiin, kuinka paljon suunnittelumalleja todellisuudessa käytettiin. Vastaajista 53,6 % ilmoitti käyttävänsä suunnittelumalleja työssään, 25,0 % ei käyttänyt niitä lainkaan ja 21,4 % vastasi, ettei asialla ole merkitystä. Kysymyksessä 11 listattiin suunnittelumallit ja annettiin vastausvaihtoehdoiksi ”tunnistan suunnittelumallin”, ”olen käyttänyt suunnittelumallia” ja ”käytän suunnittelumallia säännöllisesti”. Kuvassa 6 esitetään, kuinka kysymykseen vastattiin.



Kuva 6. Suunnittelumallien käyttö kohdeorganisaatiossa.

Kysymykseen 11 vastanneiden mukaan säännöllisesti käytettyjä suunnittelumalleja olivat vain tunnetuimmat suunnittelumallit. Kun dataa tutkittiin tarkemmin, havaittiin että vastaajat jotka tunsivat eniten suunnittelumalleja, myös ilmoittivat käyttävänsä suunnittelumalleja työssään useammin kuin sellaiset vastaajat, jotka ilmoittivat tuntevansa vain vähän suunnittelumalleja.

Kysymykset 12-17 käsittelivät Java-koodin tuottamiseen liittyviä asioita, kuten näkemyksiä siitä, kuinka lähdekoodia tulisi kehittää. Kysymyksessä 12 kysyttiin vastaajan Java-ohjelmointiin ensisijaisesti käyttämää työkalua. Vastaajista 50,0 % käytti IDE-kehitysympäristöä, 10,7 % koodieditoria ja 7,1 % tavallista editoria. Vastaajista 32,3 % ilmoitti ettei asialla ole merkitystä, minkä tulkittiin tarkoittavan, ettei heidän työtehtäviinsä liittynyt Java-ohjelmointia. Kun kysymyksessä 13 kysyttiin, mikä on mieluisin IDE-kehitysympäristö, 21,4 % vastasi käyttävänsä Borlandin JBuilder –kehitysympäristöä, 7,1 % Eclipse-kehitysympäristöä, 17,9 % kertoi käyttävänsä muuta IDE-kehitysympäristöä ja 53,6 % vastasi, ettei asialla ole merkitystä. Muita käytettyjä IDE-kehitysympäristöjä olivat JCreator, IntelliJ Idea, NetBeans ja Sun One Studio. Näistä kukin oli yhden vastaajan käytössä, paitsi IntelliJ, jota käytti kaksi vastaajaa.

Kysymyksessä 14 tiedusteltiin, mikä on Java-koodin tärkein ominaisuus. Vastaajista 24,1 % valitsi ohjelmistoteknisen laadun, 13,8 % uudelleenkäytettävyyden, 48,3 % ratkaisun toimivuuden ja 13,8 % valitsi vaihtoehdon ”asialla ei ole merkitystä”.

Luokan toteutukseen tulisi kysymyksen 15 perusteella 55,6 % mielestä osallistua vain yhden ohjelmoijan, 18,5 % mielestä koko projektitiimin ohjelmoijien ja 25,9 % vastasi ettei asialla ole merkitystä.

Kysymyksen 16 perusteella 70,4 % oli sitä mieltä, että projektin tuotosten tulisi olla kaikkien kohdeorganisaation työntekijöiden nähtävillä. Vastaajista 11,1 % ilmoitti, että lähdekoodien tulisi olla vain projektitiimin nähtävillä ja 18,5 % ilmoitti ettei asialla ole merkitystä. Kysymyksen 17 mukaan 44,4 % kannatti, että lähdekoodeihin tehtäisiin muutoksia vain projektin aikana. Vastaajista 37,0 % oli puolestaan sitä mieltä, että lähdekoodeja tulisi voida muuttaa myös projektin päättymisen jälkeen. Loput 18,5 % ei ottanut kantaa asiaan.

Kysymyksissä 18-20 kartoitettiin mielipiteitä koodauskäytännöistä. Vastauksen 18 kohdalla mieluisimmaksi koodauskäytännöksi valitsi Sunin koodauskäytännön 25,9 % vastaajista ja jonkin muun 11,1 %. Muiksi koodauskäytännöiksi ehdotettiin Scott W. Amblerin kehittämää AmbySoft-yrityksen koodauskäytäntöä sekä Jon S. Stevensin ja Jason van Zylin Jakarta-projekteissa määriteltyä koodauskäytäntöä. Yksi vastaajista kannatti edellisessä yrityksessä käyttämäänsä koodauskäytäntöä, jonka tärkeimmät kohdat hän esitti vapaamuotoisissa kommenteissa. Peräti 63,0 % vastaajista kuitenkin valitsi, ettei asialla ole merkitystä. Monet täsmensivät myös vapaamuotoisesti, ettei valittu koodauskäytäntö ole tärkeä, vaan se että kaikki toimivat koodauskäytännön mukaisesti.

Kysymyksen 20 kohdalla vastaajista 40,7 % päätyivät valitsemaan useamman kuin yhden vastausvaihtoehdon. Vaikka tätä ei kyselyn tekijä ollut tarkoittanut, kysymyksen sanamuodosta ei käynyt ilmi, että toivottiin vain yhtä vastausvaihtoehtoa. Tästä syystä kaikkiin vastausvaihtoehtoihin annetut merkinnät laskettiin siten, että painokerroin oli muiden kysymysten tavoin yksi jokaista valintaa kohden. Kysymyksen mukaan 30,2 % kannatti koodauskäytäntöjen esittelemistä koodipohjatiedoston avulla, 37,2 %, referenssiprojektin lähdekoodin avulla ja 27,9 % toivoi koulutustilaisuutta. Vaihtoehdon ”asialla ei ole merkitystä” valitsi vain 4,7 % vastanneista.

Kysymykset 21-26 keskittyivät ohjelmakoodin tuottamiseen liittyviin käytännön asioihin. Kysymyksessä 21 vastaajista 63,0 % oli sitä mieltä, että koodin muotoiluun liittyvät säännöt kuuluvat yksikölle ja 18,5 % mukaan asia oli projektitiimin sisäinen. Kunkin ohjelmoijan päätettäväksi asian jättäisi 14,8 % vastaajista. Vain 3,7 % valitsi vaihtoehdon ”asialla ei ole merkitystä”.

Kysymyksen 22 perusteella Java-koodin sisennyksiin tulisi käyttää tabulaattoreita 18,5 % mielestä, kun puolestaan 37,5 % kannatti välilyöntejä ja 44,4 % oli sitä mieltä, ettei asialla ole merkitystä. Kysymyksessä 23 vastaajista 63,0 % ilmoitti ettei sisennyksen pituudella ole merkitystä. Vastaajista 11,1 % kannatti kahden merkin mittaisia sisennyksiä, 3,7 % verran kannatusta keräsivät sekä kolme ja kahdeksan merkkiä. Kysymykseen vastanneista 18,5 % valitsi neljä merkkiä.

Kysymykseen 24 valitsi 29,6 % vaihtoehdon a ja saman verran vaihtoehdon b. Vaihtoehdon c valitsi 3,7 %. Loput 37,0 % ei ottanut kysymykseen kantaa. Luokkien ni-

meämisestä oltiin yksimielisempiä kysymyksessä 25, jonka kohdalla vastaajista 46,4 % valitsi vaihtoehdon a ja 28,6 % vaihtoehdon b. Muut 25,0 % eivät ottaneet asiaan kantaa. Kukaan ei valinnut vaihtoehtoa c. Samoin tapahtui kysymyksen 26 kohdalla, eli attribuuttien nimeämiseen liittyen vaihtoehtoa c ei valinnut kukaan, kun vaihtoehdon a valitsi 11,1 % ja vaihtoehtoon b päätyi 55,6 % vastaajista. Jäljelle jäänyt 33,3 % ei ottanut kysymykseen kantaa.

Kysymykset 27-31 keskittyivät dokumentaation ja lähdekoodien väliseen yhteyteen sekä lähdekoodien kommentoimiseen. Kysymyksessä 27 tiedusteltiin, pitäisikö lähdekoodien kommentointia harrastaa nykyistä enemmän. Vastaajista peräti 81,5 % vastasi kyllä ja loput 18,5 % valitsi, ettei asialla ollut merkitystä. Kysymyksessä 28 pyydettiin ilmoittamaan, missä dokumentoinnin tulisi ensisijaisesti olla. Kaaviot olivat oikea paikka 23,3 % mielestä, 33,3 % vastasi dokumentit ja 30,3 % oli sitä mieltä, että lähdekoodien kommentit olivat ensisijainen paikka. Vastaajista vain 13,3 % vastasi, ettei asialla ole merkitystä.

Kysymyksen 29 mukaan kukaan vastaajista ei päivittänyt kaavioita muutoksia toteuttaessaan. Projektin päättyessä ne päivitti 40,7 % vastanneista, satunnaisesti 37,0 % ja 22,2 % valitsi ettei asialla ole merkitystä. Sen sijaan kysymyksen 30 mukaan dokumenttien, kaavioiden ja Java-koodien tulisi vastata toisiaan kaikissa tilanteissa 77,8 % mielestä, kun 11,1 % oli päinvastaista mieltä ja samoin 11,1 % oli sitä mieltä, ettei asialla ollut merkitystä.

Vastaajista enemmistö vastusti omia koodauskäytäntöjä kysymyksen 18 kohdalla, mutta kysymyksessä 31 peräti 55,6 % kannatti omien merkkikoodien käyttämistä lähdekoodien kommenteissa. Kyseisistä merkkikoodeista ”@bug” ja ”@history” olivat AmbySoftin kehittämän koodauskäytännön mukaisia, ”@test” kyselyn tekijän itse keksimä. Vain 14,8 % vastusti niitä ja 29,6 % vastasi ettei asialla ole merkitystä.

Kysymykset 32-34 keskittyivät moduulitestaukseen. Kysymykseen 32 vastasi peräti 81,5 % että moduulitestausta pitäisi tehdä nykyistä enemmän ja vain 3,7 % oli asiasta päinvastaista mieltä. Loput 14,8 % valitsi, ettei asialla ole merkitystä. Kysymyksen 33 perusteella vastuu moduulitestauksesta kuuluu 63,0 % mukaan koodin tuottaneelle ohjelmoijalle. Jaettua vastuuta projektitiimin ohjelmoijien kesken kannatti 29,6 %

vastaajista. Testaajalle moduulitestauksen jättäisi 3,7 % ja saman verran valitsi, ettei asialla ole merkitystä.

Kysymyksen 34 mukaan vastaajien moduulitestausta suoritti testisuunnitelman ja testitapauksien mukaisesti ainoastaan 3,7 % vastaajista, kun puolestaan 37,0 % teki sitä oman suunnitelmansa pohjalta, 22,2 % suunnittelemattomasti ja 11,1 % ei tehnyt lainkaan. Vastaajista 25,9 % vastasi, ettei asialla ollut merkitystä, mikä tulkittiin tässä tapauksessa, ettei se kuulunut vastaajan työtehtäviin.

6.3.2 Haastattelut

Haastattelun kaksi ensimmäistä kysymystä koskivat toteutusmenetelmien vaihtelemista ja vaihtelemisen syitä. Vastaukset kategorisoitiin kolmeen pääluokkaan, jotka olivat projektikohtaiset syyt, kehittäjäkohtaiset syyt ja asiakaskohtaiset syyt. Haastateltavien vastauksia analysoimalla todettiin, että toteutusmenetelmien vaihtelemiseen vaikuttavista syistä mainittiin vähiten kehittäjäkohtaisiksi luokiteltavia syitä ja eniten asiakaskohtaisia syitä. On myös huomattavaa, että asiakaskohtaiset syyt ainoana luokkana esiintyivät jokaisen haastatellun vastauksissa.

Haastatellut mainitsivat asiakaskohtaisiksi syiksi asiakkaan vaatimukset, joihin saattoi kuulua tietyn prosessimallin noudattaminen tai että asiakkaat odottivat kohdeorganisaation jäsenten muulla tavoin mukautuvan asiakkaiden oman organisaation toimintaan. Tämä nähtiin kuitenkin käytännössä kohdeorganisaation joustavuutena asiakasta kohtaan, ei asiakkaan sanelemana pakkona.

Projektikohtaisiksi syiksi haastatellut mainitsivat prosessin soveltamisen ja projektien erilaisuuden. Jälkimmäiseen vaikuttivat projektin laajuus ja luonne, eli tehtiinkö kokonaan uutta ohjelmistoa vai paranneltiinko olemassa olevaa järjestelmää. Kehittäjäkohtaisista syistä haastatellut mainitsivat henkilökohtaiset mieltymykset työkalujen käyttämiseen ja tottumisen tiettyihin toimintatapoihin.

Toteutusmenetelmien vaihtelemisesta haastatellut näkivät olevan hyötyä siinä mielessä, että he saattoivat näin tutustua uusiin toteutusmenetelmiin ja ottaa niitä kokemuksistaan riippuen omaan käyttöön. Muita konkreettisia etuja eivät haastateltavat pystyneet nimeämään, mutta he korostivat usein, ettei toteutusmenetelmien vaihtele-

minen ollut ainoastaan negatiivinen asia. Haittoja toteutusmenetelmien vaihtelemisesta haastatellut puolestaan nimesivät useita. Niihin kuuluivat uudelleenkäytettävyyden vaikeus, dokumentaation niukkuus, toiminnan vaihteleminen ja siitä johtuva toiminnan tehottomuus sekä laadun vaihteleminen. Haastatellut käsittelivät vastauksissaan haittapuolia yksityiskohtaisemmin ja laajemmin kuin etuja.

Haastateltujen vastauksista ilmeni, että kohdeorganisaatiossa pyrittiin jossain määrin muuttamaan suunnittelu-, testaus- ja toteutusvaiheiden painotusta, pääasiassa prosessimallia kehittämällä. Muita keinoja he eivät pystyneet nimeämään. Sen sijaan useimmat haastatellut olivat sitä mieltä, että kohdeorganisaation toimintaan vaikuttivat yleisesti tunnetut prosessimallit ja menetelmät. Prosessimalleja ei kukaan haastatelluista kuitenkaan maininnut nimeltä. Menetelmistä nimettiin vain suunnittelumallit.

Kysyttäessä, miten kohdeorganisaation prosessimalli soveltui sen toimintaan, vastaukset kategorisoitiin kolmeen luokkaan sen mukaan, miten haastatellut näkivät prosessimallia sovellettavan. Prosessimallia sovellettiin joko sellaisenaan, asiakkaasta riippuen tai projektista riippuen. Haastateltujen mukaan kohdeorganisaation prosessimallia sovellettiin valitsemalla sopiviksi katsottuja osioita projektin käyttöön. Asiakkaan prosessimalleja noudatettaessa voitiin mahdollisuuksien mukaan soveltaa dokumenttipohjia.

Kun haastateltavilta kysyttiin poikettiinko prosessimallista, vastausten perusteella poikkeamiin vaikuttaneet syyt olivat käytännössä täysin asiakkaista johtuvia. Nämä syyt olivat haastateltavien vastausten perusteella merkittävimpiä prosessimallista poikkeamiseen vaikuttaneista syistä. Konkreettisiksi syiksi haastatellut nimesivät asiakkaiden odotukset mm. nopeista tuloksista ja sen etteivät asiakkaat aina ymmärrä niitä etuja, joita prosessimallin käyttäminen toisi mukanaan. Haastatellut eivät kuitenkaan pitäneet prosessimallin poikkeamia yksistään haittana, vaan näkivät niistä olevan hyötyä. Hyödyillä haastatellut viittasivat ohjelmistokehityksen nopeuteen, joka saavutettiin sillä, ettei prosessimallin noudattamiseen käytetty aikaa. Vastaavasti asiakkaan prosessimallien käyttämistä pidettiin hyvänä siitä syystä, että niitä pidettiin iteratiivisempina ja erikoistuneempina johonkin tiettyyn kohteeseen kuin kohdeorganisaation omaa prosessimallia.

Prosessimallin poikkeamien haittapuolia käsitellessään haastatellut nostivat esiin riskit, joiksi nimettiin tuotosten epäyhtenäisyys, kommunikaation vaikeus eri sidosryhmien välillä ja uusien toimintatapojen opetteluun käytetyn ajan.

Haastateltavat olivat lähes yhtä mieltä sen suhteen, etteivät prosessimallin poikkeamat vaikuttaneet toteutusmenetelmiin. Heidän vastauksissaan toistui ajatus siitä, että menetelmät ja työkalut, sekä jossain määrin myös totutut toimintamallit eri työtehtävien suhteen, pysyivät samoina. Haastatellut mainitsivat myös, etteivät toteutusmenetelmät olleet riippuvaisia prosessimallista ja että menetelmät yleisesti ottaen olivat kehittäjäkohtaisia asioita.

Kun haastateltavilta kysyttiin samaa asiaa päinvastaiseksi käännettynä, eli vaikuttavatko toteutusmenetelmät prosessimallin poikkeamiin, yhtä lukuun ottamatta haastateltavat kielsivät tällaisen yhteyden olemassa olon. Vastauksissa haastattelut mainitsivat pitävänsä toteutusmenetelmiä asioina, jotka eivät kuuluneet prosessimalliin. Tätä lukuun ottamatta haastatellut eivät pystyneet täysin perustelemaan mielipidettään. Edes eri mieltä ollut haastateltava ei pystynyt sanomaan, kuinka toteutusmenetelmät olisivat vaikuttaneet prosessimallin poikkeamiin.

Kaikkien haastateltujen mielestä kohdeorganisaation tulisi toimia yhtenäisemmin. Vastausten perusteella yhtenäisen toiminnan edut voidaan jakaa täsmällisiin ja yleisluontoisiin etuihin. Ensimmäisiin kuului ensisijaisesti uudelleenkäytettävyys ja jatkokehityksen helpottuminen. Jälkimmäiseen luokkaan puolestaan tulkittiin mm. vastaus, joka käsitteli yhtenäisen toiminnan liittyvän pitkäkestoisempaan näkemykseen ohjelmistokehityksestä.

Haastateltavat olivat myös yksimielisiä siitä, että yhtenäiset toteutusmenetelmät tulisi koota parhaat käytännöt -dokumenttiin. Tätä haastatellut perustelivat sillä, että se edistäisi kommunikointia eri sidosryhmiä kohtaan. Toisaalta vastauksissa mainittiin myös, että dokumentoiduista menetelmistä tulisi tiedottaa riittävästi ja se olisi saatava organisaation kehittäjien käyttöön. Eräs haastateltu korosti erityisesti myös sitä, että tällaisen dokumentin tulisi olla jatkuvasti kehittyvä ja ajan tasalla oleva. Hän jatkoi myös, että dokumentoituihin menetelmiin tulisi suhtautua kriittisesti ja poistaa siitä sellaisia, jotka todettaisiin toimimattomiksi.

Yhtä lailla haastatellut olivat samaa mieltä siitä, että yhtenäisten toteutusmenetelmien käyttämistä tulisi valvoa. Syyt jaettiin kahteen luokkaan: toteutusmenetelmien käyttöönottamisen edistäminen ja toteutusmenetelmien käyttökelpoisuuden selvittäminen. Ensimmäiseen luokkaan jaotelluissa vastauksissa korostettiin, että seurannan avulla voitaisiin saada ihmiset tottumaan yhtenäisten toteutusmenetelmien käyttöön tai motivoida heitä ilman sanktioita. Jälkimmäiseen ryhmään kuuluneissa vastauksissa mainittiin katselmointeihin pohjautuva valvonta.

Lopulta haastateltavilta kysyttiin, olisiko heillä sellaisia toteutusmenetelmiä, ratkaisumalleja tai toimintatapoja jotka voisivat soveltua muille. Yllättävää vastauksissa oli, etteivät haastateltavat nimenneet konkreettisia asioita, vaan viittasivat eri organisaatioiden käyttämiin malleihin. Haastateltavat myös mainitsivat kaipaavansa vinkkejä muilta.

6.3.3 Havainnot

Projektidokumentaatiosta tehdyt havainnot olivat seuraavat:

- Projektissa A oli määritelty, että sisennyksen pituuden tulisi olla neljä välilyöntiä.
- Projektissa B ei ollut käytännössä lainkaan suunnitteludokumentaatiota tai UML-kaavioita.
- Projektissa B annettiin ohjelmoijille koodipohjatiedostoja, joiden mukaisesti tietyt ohjelmistomoduulit tuli toteuttaa. Tämä ei kuitenkaan ollut kohdeorganisaation käyttämä menetelmä, vaan kyseiset tiedostot oli tuottanut toinen yksikkö, johon viitattiin kappaleessa 6.2.3.
- Projektissa B dokumenttipohjia sovellettiin kirjoittajakohtaisesti. Dokumenttipohjien kirjoittamatta jätetyt kappaleet joko poistettiin kokonaan tai jätettiin tyhjiksi.

Työkalujen käytöstä tehtiin seuraavat havainnot:

- Molemmissa projekteissa kehittäjät käyttivät itse valitsemiaan työkaluja.
- Projektissa A ohjelmakoodit käännettiin projektin kehitysympäristöön kuuluvalla koneella, projektissa B kääntäminen tehtiin kunkin kehittäjän omalla työasemalla.

Projektien lähdekooditiedostoja tutkimalla tehtiin seuraavat yleiset havainnot:

- Yhteistä koodauskäytäntöä ei ollut käytössä kummassakaan projektissa.
- Molemmissa projekteissa nimeämiskäytännöt olivat yhtenevät ja johdonmukaiset.
- Kehittäjästä riippuen eri lähdekooditiedostot olivat muotoiluiltaan erilaisia.
- Mikäli yhtä Java-luokkaa oli ollut toteuttamassa useampi kuin yksi kehittäjä, kyseisessä luokassa oli kehittäjäkohtaisia eroja koodin muotoilussa. Yleensä yksi muotoilutapa oli kuitenkin luokassa hallitseva.
- Jos kehittäjä oli tehnyt uuden metodin toisen kehittäjän luokkaan, kehittäjäkohtaisesti henkilökohtaiset käytännöt näkyivät johdonmukaisesti kaikissa muotoiluun liittyvissä asioissa, kuten ohjelmalohkosulkujen käytössä ja sisennyksissä. Kaikkien kehittäjien kohdalla näin ei kuitenkaan ollut.
- Jos kehittäjä teki muutoksen keskelle toisen kehittäjän ohjelmakoodia, kehittäjäkohtaisia eroja ei yleensä ollut, vaan muutoksen tekijä oli pyrkinyt pääsääntöisesti noudattamaan luokassa hallitsevaa muotoilua.
- Versionhallintaohjelman avulla lähdekoodien vanhoja versioita tarkkailtaessa voitiin havaita, että kehittäjäkohtaisesti osa tiedostoista oli muotoiltu kokonaan uusiksi. Asiaa selvitettäessä todettiin, että osa toteuttajista käytti sellaisia työkaluja, jotka automaattisesti muotoilivat lähdekooditiedoston työkaluun määriteltyjen sääntöjen mukaan.

Lähdekooditiedostoja tutkimalla havaittiin, että ohjelmakoodin muotoilu oli erilaista seuraavilla tavoilla:

- Sisennykseen käytettiin pääsääntöisesti välilyöntejä, toisinaan tabulaattoreja ja osassa lähdekooditiedostoja molempia.
- Sisennysten pituus oli merkkeinä laskettuina kaksi, kolme, neljä, seitsemän tai kahdeksan merkkiä.
- Ohjelmalohkon aloittava sulku oli mm. if-haaran kanssa joko samalla rivillä tai vasta seuraavalla.
- Toisinaan ohjelmalohkon sulut puuttuivat jos ohjelmalohko koostui vain yhdestä koodirivistä, toisinaan ne olivat mukana.
- Metodien parametrit erotettiin joko pilkulla tai pilkulla ja välilyönnillä.
- Ohjelmakoodirivin maksimipituus vaihteli.
- Osa luokista, niiden attribuuteista ja metodeista oli kommentoitu, osa ei.

– Toisten metodien kommenteissa merkkikoodit ”@return”, ”@param” ja ”@throws” oli merkitty, kun niitä tarvittiin, toisissa niitä ei ollut.

Koska muotoilussa oli havaittavissa selviä tekijäkohtaisia eroja, perehdyttiin eroihin tarkemmin ja kysyttiin toteuttajilta itseltään, olivatko tehdyt oletukset erojen syistä oikeita.

Kuten yllä todettiin, projektissa A oli määritelty sisennyksen pituus. Sisennyksien pituus kuitenkin vaihteli projektin Java-lähdekooditiedostoissa. Tämä johtui siitä, että osa projektitiimin toteuttajista tuli mukaan toisia myöhemmin, eivätkä tienneet asiasta siirtyessään projektiin.

Kun projektien toteuttajilta kysyttiin asiasta, sisennysten pituuden vaihteleminen johtui henkilökohtaisista mieltymyksistä. Versionhallintajärjestelmän avulla tehtiin lisäksi havainto, että eräs projektin A kehittäjä oli muotoillut jossain vaiheessa koko lähdekooditiedoston uudella tavalla. Tämän oletettiin johtuvan käytetyn työkalun ominaisuudesta muotoilla koodi automaattisesti uusiksi. Kehittäjä vahvisti, että asia johtui emacs-ohjelman ominaisuudesta, jota kehittäjä oli tarkoituksella käyttänyt. Hän kuitenkin huomautti, että koko lähdekooditiedoston muotoileminen uusiksi aiheutti sen, ettei kyseistä tiedostoa voitu enää verrata versionhallintajärjestelmän avulla edelliseen versioon, joten hän oli tietoinen myös toimintansa aiheuttamasta haitasta.

Projektissa A vain yksi kehittäjä oli käyttänyt seitsemän merkkiä pitkiä sisennyksiä. Kehittäjän mukaan sisennyksen pituus johtui käytetyn työkalun automaattisesta muotoilusta, eikä hän itse ollut havainnut sisennyksen pituuden olleen seitsemän merkkiä, vaan tarkoitus oli käyttää kahdeksan merkkiä pitkiä sisennyksiä.

Kun välilyöntien ja tabulaattoreiden suhdetta lähdekooditiedostoissa selvitettiin, havaittiin että projektin A lähdekooditiedostoissa, joissa oli sekä välilyöntejä että tabulaattoreja, tabulaattoreita tavattiin vain sellaisissa metodeissa, jotka oli kopioitu sellaisenaan toisista projekteista. Projektissa A myös osassa tiedostoja kaikki tabulaattorit olivat koodikommenteissa, eivät itse koodissa. Sen sijaan projektissa B samassa tilanteessa tabulaattoreja löydettiin vain tiedoston alkuun liitetystä tekijänoikeustekstistä, jonka oletettiin olevan samalla tavoin kopioitu toisaalta ilman muutoksia.

Ohjelmalohkon sulkujen käyttöön liittyvät erot todettiin täysin kehittäjän henkilökohtaisista mieltymyksistä riippuviksi seikoiksi, samoin lähdekoodikommentteihin liittyvät erot ja rivin maksimipituus. Tosin jälkimmäiseen vaikutti myös käytetty editori, mikäli se esimerkiksi ilmoitti maksimipituuden ylittymisestä jollain tavoin.

Versionhallintajärjestelmän avulla tehtiin lisäksi yllättävä havainto. Sama kehittäjä oli saattanut muotoilla tuottamansa ohjelmakoodin eri luokissa hieman eri tavoin. Koska jokaisella projektien toteutukseen osallistuneella kehittäjällä oli vuosien ohjelmointikokemus, pidettiin epätodennäköisenä että kyseisten henkilöiden ohjelmointityyli muuttui projektien aikana. Projektien kehittäjät myönsivät mukautuvansa tuottamansa koodin muiden toteuttajien käyttämään tyyliin. On kuitenkin mahdollista, että kehittäjät alitajuisesti mukautuivat projekteissa käytettyihin erilaisiin koodin muotoiluihin vielä tätäkin voimakkaammin ja käyttivät muiden toteuttajien tyyliä, vaikka olivatkin siirtyneet kirjoittamaan kokonaan itse kehittämiään luokkia.

6.3.4 Yhteenveto

Selvitystyön aikana tehtyjen havaintojen perusteella todettiin, että toteutusmenetelmien vaihtelemiseen vaikuttivat projektikohtaiset syyt, kehittäjäkohtaiset mieltymykset ja työkalut, jotka mahdollistivat mm. lähdekoodin automaattisen muotoilun.

Merkittävin selvitystyön aikana tehty huomio on, että kohdeorganisaatio uskoo nykyisen, tai sitä vastaavan, prosessimallin olevan sille sopivin mahdollinen, mutta yksilöinä monet toimivat sen vastaisesti. Käytännössä yksilökohtaiset erot näkyvät siten, että prosessimallia sovelletaan erittäin vapaasti. Mitään yhteyttä ei prosessimallin vastaisesti toimineiden sovelluskehittäjien välillä voitu vetää. Monet kohdeorganisaation työntekijät toivoivat dokumentaation vähentämistä ja ohjelmistokehityksen muuttamista kevyemmäksi. Ristiriitaisesti samat henkilöt silti pitävät nykyistä prosessimallia hyvänä ja yksikölleen sopivana.

Haastattelujen aikana tuli ilmi, että suunnitteluvaiheeseen olisi haastateltavien mielestä käytettävä enemmän aikaa, jotta toteutusvaiheessa ei tarvitsisi tehdä enää yhtä paljon suunnitteluun liittyviä ratkaisuja tai ettei UML-kaavioita tarvitsisi muuttaa merkittävässä määrin enää toteutusvaiheen aikana. Tämä tukee sitä oletusta, että kohdeorganisaatio valitsee itselleen mieluiten nykyisen kaltaisen prosessimallin, ei esi-

merkiksi jotain ketterää menetelmää. Yhtäläisyydet kohdeorganisaation yksilöiden toiminnan ja ketterien menetelmien korostamien periaatteiden välillä ovat siis satunnaisia. Silti voidaan sanoa, että kun kohdeorganisaatiota tarkastellaan yksilötasolla, ei ketterien menetelmien soveltaminen ole mahdoton ajatus. Kehitettiin prosesseja jatkossa mihin suuntaan tahansa, tulisi prosessikehityksen suhteen punnita, kuinka sitä voitaisiin tehdä entistä enemmän kehittäjien ehdoilla.

Merkittävä huomio on, että kappaleessa 4.6.2 esitellyn ryhmittelyn mukaisesti, monet kohdeorganisaation yksilöistä voitaisiin luokitella opportunistisiksi sovelluskehittäjiksi. Ratkaisun toimivuus ja sen kautta saavutettava asiakastyytyväisyys on monelle tärkeää. Silti nämäkin yksilöt ovat yhtä mieltä muiden sovelluskehittäjien kanssa sen suhteen, että tietty laatutaso on saavutettava. Ohjelmistotekninen laatu kuitenkin merkitsee hieman eri asioita eri kehittäjille. Puheen kääntyessä nykytilaan ja siihen, mihin suuntaan toimintaa olisi kehitettävä, toiset korostavat mm. lähdekoodien kommentointia ja suunnittelumallien käyttämistä, toisten mainitessa uudelleenkäytettävyyden parantamisen.

Kappaleessa 4.7.2 IDE-kehitysympäristöjen toimittajat jaoteltiin johtajiin, haastajiin ja seuraajiin. Kohdeorganisaatiossa käytetyt IDE-ohjelmat kuuluivat kyseisen luokittelun mukaan johtajiin. On kuitenkin huomattavaa, että joidenkin arvioiden mukaan noin 40 % Java-sovelluskehittäjistä eivät käytä IDE-kehitysympäristöjä, vaan tavallisia koodieditoreja. Tämän lisäksi kokemuksen karttuessa sovelluskehittäjät siirtyvät vähitellen automatisoitavista kehitysympäristöistä kohti muokattavampia vaihtoehtoja. [50] Kysely ei täysin tue tätä väitettä, koska IDE-kehitysympäristöjä käytettiin eniten. Tämä on vielä suhteutettava siihen, että kohdeyrityksessä ei viime vuosina ole juurikaan rekrytoitu uutta henkilöstöä, joten useimmilla yksikön sovelluskehittäjillä on Java-teknologian ikään suhteutettuna pitkä kokemus Java-ohjelmistojen kehittämisestä.

7 JOHTOPÄÄTÖKSET

Tässä diplomityössä saatiin selville, että kohdeorganisaatiossa suhtaudutaan myönteisesti toteutusmenetelmien yhtenäistämiseen ja toteutusohjeistukseen, joka on parhaat käytännöt -dokumentti. Diplomityön teoriaosuuden ja selvitystyön perusteella todettiin, että toteutusmenetelmät voidaan yhtenäistää kohdeorganisaatiossa parhaat käytännöt -dokumentin avulla. Selvitystyön perusteella kohdeorganisaatiossa on myös tarvetta parhaat käytännöt -dokumentille, koska kyselyn, haastattelujen ja havaintojen avulla saatiin selville, että toteutusmenetelmät vaihtelivat mm. ohjelmointiin käytettyjen työkalujen ja koodauskäytäntöjen osalta. Vaihtelua esiintyi sekä projektien että sovelluskehittäjien välillä. Vaihteleminen näkyi käytännössä tuotosten, kuten ohjelmakoodin ja dokumenttien epäyhtenäisyytenä. Vaihtelemisen syiksi todettiin asiakkaasta johtuvat vaatimukset, projektien vaihteleminen ja kehittäjäkohtaiset toimintatavat.

Tärkein yhtenäistämistä vaativa kohde ovat koodauskäytännöt, koska yhtenäisten koodauskäytäntöjen puute nähdään kohdeorganisaatiossa ohjelmiston laatuun negatiivisesti vaikuttavana tekijänä. Koodauskäytäntöjen puutteesta on myös konkreettista haittaa, koska kehittäjän käyttäessä automaattista ohjelmakoodin muotoilua, versionhallintaohjelman avulla ei ole enää mahdollista havaita tai jäljittää uuteen versioon tehtyjä muutoksia.

Koska kohdeorganisaatio on asiantuntijaorganisaatio, sen jäsenet kykenevät tunnistamaan omaan työhönsä kohdistuvat kehitystoimenpiteet ja analysoimaan niitä. Tästä syystä olisi merkityksetöntä tuoda yleispäteviä, oppilauseisiin perustuvia toteutusmenetelmiä, koska ne eivät toisi todellisia ratkaisuja jokapäiväiseen työhön. Kun vielä otetaan huomioon, että asiantuntijoita on vaikea pakottaa toimimaan sellaisella tavalla, jota he eivät tunne hyödylliseksi, on syytä harkita tarkkaan parhaat käytännöt -dokumentin kattavuutta.

Selvitystyön tulosten perusteella suositellaan, että Implementation Policy -dokumentti jaetaan kahdeksi dokumentiksi siten, että kyseinen dokumentti kattaa vain konkreettisimmat toteutusvaiheeseen liittyvät suositukset. Lisäksi suositellaan, että kyseistä toteutusohjeistusta täydentää toinen, esimerkiksi vastaava dokumentti, jonka

nimi olisi esimerkiksi ”Design Policy”. Kyseinen dokumentti keskittyisi suunnitteluvaiheen menetelmiin ja toimintatapoihin liittyviin suosituksiin.

Ehdotuksen mukaisesti Implementation Policy -dokumentin ensimmäisen version tulisi kattaa seuraavat asiat:

- Suositukset koodauskäytännöistä.
- Suositukset työkaluista.
- Ohjeistukset jotka liittyvät versionhallintajärjestelmän käyttöön.
- Ohjeistukset moduulitestauksesta.

Edelleen tässä diplomityössä suositellaan, että Design Policy -dokumentti sisältäisi:

- Konsernisuositukset alustoista ja teknologioista.
- Ohjeistukset suunnittelumallien käytöstä.
- Ohjeistukset moduulisuunnittelusta.
- Ohjeistukset suunnitteludokumenttien toteutuksesta.

LÄHTEET

[1] Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto. Talentum Media Oy. 10. painos. ISBN 952-14-0850-2.

[2] Järvinen, P. & Järvinen, A. 2004. Tutkimustyön metodeista. Opinpajan kirja. ISBN 952-99233-2-5.

[3] Yin, R. 1994. Case Study Research: Design and Methods Second Edition. Sage Publications.

[4] Hirsjärvi, S. Remes, P. & Sajavaara, P. 2004. Tutki ja kirjoita. Kirjayhtymä Oy. 10. painos. ISBN 951-26-5113-0.

[5] IEEE Std 610.12-1990. 1990. IEEE standard glossary of software engineering terminology.

[6] Pressman, R. 2000. Software Engineering: A Practitioner's Approach: European Adaptation Fifth Edition. McGraw-Hill. ISBN 0 07 709677 0.

[7] Tietotekniikan liitto. 2003. ATK-sanakirja. Talentum Media Oy. 12. painos. ISBN 951-762-831-5.

[8] Jaakohuhta, H. 2001. IT Ensyklopedia. IT Press. ISBN 951-826-573-9.

[9] Avison, D. & Fitzgerald, G. 2003. Where Now for Development Methodologies?. Communications of the ACM, Volume 46 Number 1, January 2003. s. 79-82.

[10] Highsmith, J. 2002. Agile Software Development Ecosystems. Addison-Wesley. ISBN 0-201-76043-6.

[11] Kitchenham, B. 1996. Evaluating Software Engineering Methods and Tool: Part 1: The Evaluation Context and Evaluation Methods. ACM SIGSOFT Software Engineering Notes, Volume 21 Issue 1, January 1996. s. 11-14.

[12] Kazi, I. Chen, H. Stanley, B. & Lilja, D. 2000. Techniques for Obtaining High Performance in Java Programs. ACM Computing Surveys, Volume 32 Issue 3, September 2000. s. 213-240.

[13] Bosch, J. 2000. Design and use of software architectures: Adopting and evolving a product-line approach. Addison-Wesley. ISBN 0-201-67494-7.

[14] Howard, M. & Leblanc, D. 2004. Ohjelmoijan tietoturvaopas. IT Press. ISBN 951-826-663-8.

[15] DeMarco, T. & Lister, T. 1999. Peopleware: Productive Projects and Teams. 2. painos. Dorset House. ISBN 0-932633-43-9.

[16] Wiegers, K.1996. Creating a Software Engineering Culture. Dorset House Publishing. ISBN 0-932633-33-1.

[17] Ståhle, P. & Grönroos, M. 2000. Dynamic intellectual capital: Knowledge management in theory and practice. WSOY. ISBN 951-0-25286-7.

[18] Fuggetta, A. 2000. Software Process: A Roadmap. Proceedings of the Conference on The Future of Software Engineering, May 2000. s. 25-34.

[19] Mahoney, M. 2004. Finding a History for Software Engineering. IEEE Annals of the History of Computing, Volume 26 Issue 1, January-March 2004. s. 8-19.

[20] Lycett, M. Macredie, R. Patel, C. & Paul, R. 2003. Migrating Agile Methods to Standardized Development Practice. Computer, Volume 36 Issue 6, June 2003, s. 79-85.

[21] Hoch, D. Roeding, C. Purkert, G. Lindner, S. & Müller, R. 1999. Secrets of Software Success: Management Insights from 100 Software Firms around the World. Harvard Business School Press. ISBN 1578511054.

[22] Lindvall, M. Muthig, D. Dagnino, A. Wallin, C. Stupperich, M. Kiefer, D. May, J. & Kähkönen, T. 2004. Agile Software Development in Large Organizations. Computer, Volume 37 Issue 12, December 2004. s. 26-34.

[23] Bosch, F. Ellis, J. Freeman, P. Johnson, L. McClure, C. Robinson, D. Scacchi, W. Scheff, B. & Staa, A. 1982. Evaluation of Software Development Life Cycle: Methodology Implementation. Software Engineering Notes, Volume 7 Number 1, January 1982. s. 45-60.

[24] Kokkarinen, I. 2004. Java, Prolog ja Python. IT Press. ISBN 951-826-778-2.

[25] Armano, G. & Marchesi, M. 2000. A Rapid Development Process with UML. ACM SIGAPP Applied Computing Review. Volume 8 Issue 1, April 2000. s. 4-11.

[26] Booch, G. Rumbaugh, J. & Jacobson, I. 2000. The Unified Modeling Language User Guide. 7. painos. Addison-Wesley. ISBN 0-201-57168-4.

[27] Larman, C. & Basili, V. 2003. Iterative and Incremental Development: A Brief History. Computer, Volume 36 Issue 6, June 2003. s 47-56.

[28] Henderson-Sellers, B. Collins, G. Graham, I. 2001. UML-Compatible Process. Proceedings of the 34th Annual Hawaii International Conference on System Sciences. January 2001. s. 1-10.

[29] Abrahamsson, P. Salo, O. Ronkainen, J. & Warsta, J. 2002. Agile software development methods: Review and analysis. VTT. ISBN 951-38-6010-8.

[30] Gornik, D. 2001. IBM Rational Unified Process: Best Practices for Software Development Teams [pdf-tallenne]. IBM:n verkkosivut [viitattu 23.12.2004]. Saatavissa: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/ 2003/rup_bestpractices.pdf

[31] Murphy, T. 2004. Borland: Pulling IT All Together [verkkoartikkeli]. Meta Group [viitattu 23.12.2004]. Saatavissa: http://www.metagroup.com

[32] Howard, A. 2002. Rapid Application Development: Rough and Dirty or Value-for-Money Engineering? Communications of the ACM, Volume 45 Number 10, October 2002. s. 27-29.

[33] Agarwal, R. Prasad, J. Tanniru, M. & Lynch, J. 2000. Risks of Rapid Application Development. Communications of the ACM, Volume 43 Issue 11es. November 2000. s. 177-188.

[34] Reilly, J. & Carmel E. 1995. Does RAD Live Up to the Hype? IEEE Software, Volume 12 Issue 5, September 1995. s. 24-26.

[35] Abrahamsson, P. Warsta, J. Siponen, M.& Ronkainen, J. 2003. New Directions on Agile Methods: A Comparative Analysis. 25th International Conference on Software Engineering, 2003. Proceedings. 3-10, May 2003. s. 244-254.

[36] Beck, K. 1999. Embracing Change with Extreme Programming. Computer, Volume 32 Issue 10, October 1999. s. 70-77.

[37] English, A. 2002. Extreme Programming: It's Worth a Look. IT Professional, Volume 4 Issue 3, May-June 2002. s. 48-50.

[38] Neill, C. 2003. The Extreme Programming Bandwagon: Revolution or Just Revolting? IT Professional, Volume 5 Issue 5, September-October 2003. s. 62-64.

[39] Feller, J. & Fitzgerald, B. 2000. A Framework Analysis of the open Source Software Development Paradigm. Proceedings of the twenty first international conference on Information systems, December 2000. s. 58-69.

[40] Spinellis, D. 2003. Code Reading: The Open Source Perspective. Addison-Wesley. ISBN 0-201-79940-5.

[41] Grand, M. 1998. Design Patterns in Java, Volume 1: A Catalog of Reusable Design Patterns Illustrated with UML. John Wiley & Sons. ISBN 0-471-25839-3.

[42] Gamma, E. Helm, R. Johnson, R. & Vlissides, J. 2001. Olio-ohjelmointi: Suunnittelumallit: Design Patterns. IT Press. ISBN 951-826-428-7.

[43] Grenning, J. 2001. Launching Extreme Programming at a Process-Intensive Company. IEEE Software, Volume 18 Issue 6. November-December 2001. s. 27-33.

[44] Brooks, F. 1998. The Mythical Man-Month: Essays on Software Engineering. 8. painos. ISBN 0-201-83595-9.

[45] Reiss, S. 1996. Software Tools and Environments. ACM Computing Surveys, Volume 28 Number 1, March 1996. s. 281-284.

[46] Kramer, D. 1999. API Documentation from Source Code Comments: A Case Study of Javadoc. Proceedings of the 17th annual international conference on Computer documentation. s. 147-153.

[47] Cox, B. & Novobilski A. 1991. Object-Oriented Programming: An Evolutionary Approach. 2nd Edition. Addison-Wesley. ISBN 0-201-54834-8.

[48] Groth, R. 2004. Is the Software Industry's Productivity Declining? IEEE Software. Volume 21 Issue 6, November-December 2004 s. 92-94.

[49] Mäntylahti, P. 2004. Oliot ojennukseen. Tietokone 13/2004. s. 58-65.

[50] Murphy, T. 2003. Assessing Java Integrated Development Environments: Part 1-3 [verkkoartikkeli, vaatii salasanan]. Meta Group [viitattu 20.12.2004]. Saatavissa: http://www.metagroup.de

[51] Blechar, M. 2004. Magic Quadrant for OOA&D Tools, Update for 2005. [verkkoartikkeli, vaatii salasanan]. Gartner Research [viitattu 20.12.2004]. Saatavissa: http://www3.gartner.com/Init

[52] Harrison, W. Ossher, H. & Tarr, P. 2000. Software Engineering Tools and Environments: A Roadmap. Proceedings of the Conference on The Future of Software Engineering, May 2000. s. 261-277.

[53] Murphy, T. 2002. Developer Matching [verkkoartikkeli, vaatii salasanan]. Meta Group [viitattu 20.12.2004]. Saatavissa: http://www.metagroup.com

[54] Driver, M. 2004. CIO Update: The Java Development Community Is Changing [verkkoartikkeli, vaatii salasanan]. Gartner Research [viitattu 20.12.2004]. Saatavissa: http://www3.gartner.com/Init

[55] McConnell, S. 2002. Ohjelmistotuotannon hallinta: ohjelmistoprojektien aikataulut kuriin! Microsoft Press. ISBN 951-826-194-6.

Liite 1. Kyselylomake

Implementation Policy on SDI-yksikön parhaat käytännöt -dokumentti (best practices), johon kootaan suosituksia SDI:n käyttämistä ja hyväksi havaitsemista menetelmistä, käytännöistä ja työkaluista. Tämän kyselyn tavoitteena on kerätä tietoa näistä SDI:n käyttämistä menetelmistä, käytännöistä ja työkaluista. Kysely on lähetetty koko SDI:lle. Kaikkien mielipiteet ovat mukana vaikuttamassa ja niitä arvostetaan!

Tuloksia hyödynnetään SDI:n Implementation Policy -dokumentissa sekä Jarkko Sikiön diplomityössä. Kaikki vastaukset käsitellään luottamuksellisesti. Vastausaika päättyy 18.2.2005.

Palautus (nimettömänä):
- sähköpostilla: jarkko.sikio@teliasonera.com
- paperitulosteena: postilokero (Jarkko Sikiö), Laserkatu 8 aula
- paperitulosteena osoitteeseen:

  Jarkko Sikiö
  TeliaSonera Finland Oyj
  Laserkatu 8
  53850 Lappeenranta

Vastaa merkitsemällä X sopivan vaihtoehdon kohdalle. Vaihtoehto "asialla ei ole merkitystä" on tarkoitettu käytettäväksi silloin, kun kysymys ei ole tärkeä tai se ei liity omaan työhön. Kohta "kommentit/perustelut" on tarkoitettu vastauksen vapaamuotoiseen täsmentämiseen.

Lisätietoa, selvennyksiä kysymyksiin ja termeihin antaa Jarkko Sikiö (jarkko.sikio@teliasonera.com).

(jatkuu)

Liite 1. jatkuu

Yleistä

1. Ensisijainen roolini (ei ammattinimike)
   analysti:
   arkkitehti:
   suunnittelija:
   ohjelmoija:
   testaaja:
   projektipäällikkö:
   muu, mikä: _____

2. Yhtenäisiä suunnitteluvaiheen menetelmiä tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:
   kommentit/perustelut:
   _____
   _____

3. Yhtenäisiä toteutusvaiheen menetelmiä tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:
   kommentit/perustelut:
   _____
   _____

4. Yhtenäisiä testausvaiheen menetelmiä tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:
   kommentit/perustelut:
   _____
   _____

5. Yhtenäisiä koodauskäytäntöjä (coding conventions) tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:
   kommentit/perustelut:
   _____
   _____

6. Yhteisesti valittuja suunnitteluvaiheen työkaluja tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:

(jatkuu)

Liite 1. jatkuu

7. Yhteisesti valittuja toteutusvaiheen työkaluja tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:

8. Yhteisesti valittuja testausvaiheen työkaluja tulisi käyttää nykyistä enemmän
   kyllä:
   ei:
   asialla ei ole merkitystä:

9. Ehdotuksiani menetelmistä, käytännöistä ja työkaluista
   _____
   _____

Suunnittelu

10. Käytän työssäni suunnittelumalleja (design patterns)
    kyllä:
    ei:
    asialla ei ole merkitystä:
    kommentit/perustelut:
    _____
    _____

11. Olen käyttänyt työssäni seuraavia suunnittelumalleja (X = tunnistan suunnittelumallin, 1 = olen käyttänyt suunnittelumallia, N = käytän suunnittelumallia säännöllisesti)
    Creational patterns
    - Abstract Factory:
    - Builder:
    - Factory Method:
    - Prototype:
    - Singleton:

    Structural patterns
    - Adapter:
    - Bridge:
    - Composite:
    - Decorator:
    - Facade:
    - Flyweight:
    - Proxy:

    Behavioral patterns
    - Chain of responsibility:
    - Command:
    - Interpreter:
    - Iterator:
    - Mediator:
    - Memento:
    - Observer:

(jatkuu)

Liite 1. jatkuu

- State:
- Strategy:
- Template Method:
- Visitor:

Java-ohjelmointi

12. Java-ohjelmointiin käytän ensisijaisesti
tavallista editoria:
koodieditoria:
IDE-kehitysympäristöä:
muu, mikä: _____
asialla ei ole merkitystä:

13. Mieluisin IDE-kehitysympäristö on
Borland JBuilder:
Eclipse:
muu, mikä: _____
asialla ei ole merkitystä:

14. Java-koodin tärkein ominaisuus on
ohjelmistotekninen laatu:
uudelleenkäytettävyys:
ratkaisun toimivuus:
asialla ei ole merkitystä:
kommentit/perustelut:
_____
_____

15. Projektin aikana yhden luokan toteutukseen tulisi osallistua
vain yhden ohjelmoijan:
kaikkien projektitiimin jäsenten:
asialla ei ole merkitystä:

16. Projektin tuotosten, kuten kaavioiden ja lähdekoodien, tulisi olla mm. arviontia varten
vain projektitiimin nähtävissä:
kaikkien yksikön työntekijöiden nähtävissä:
asialla ei ole merkitystä:

17. Lähdekoodia tulisi voida muuttaa
vain projektin aikana:
milloin tahansa projektin päättymisen jälkeen:
asialla ei ole merkitystä:

Java-koodauskäytännöt

18. Olemassa olevien koodauskäytäntöjen sijaan tulisi kehittää yksikön omat koodauskäytännöt

(jatkuu)

kyllä:
ei:
asialla ei ole merkitystä:
kommentit/perustelut:
_____
_____

19. Mikäli valitsit edelliseen kysymykseen "ei", mieluisin koodauskäytäntö on Sunin kehittämä:
muu, mikä: _____
asialla ei ole merkitystä:
kommentit/perustelut:
_____
_____

20. Yhteiset koodauskäytännöt tulisi esitellä yksikössä
templaatin avulla:
referenssiprojektin ohjelmakoodien avulla:
koulutustilaisuuden avulla:
muu, mikä: _____
asialla ei ole merkitystä:

21. Koodauskäytäntöjen yksityiskohtien (esim. koodin muotoilu) tulisi olla
yksikön päätettävissä (Implementation Policy):
projektitiimin päätettävissä:
ohjelmoijan päätettävissä:
asialla ei ole merkitystä:

22. Java-koodissa tulisi käyttää sisennykseen
tabulaattoreita:
välilyöntejä:
asialla ei ole merkitystä:

23. Yhden sisennyksen pituus merkeissä laskettuna
___ merkkiä.
asialla ei ole merkitystä:

24. Sulkujen ja rivinvaihtojen käyttö (kirjoita tähän paras vaihtoehto): _____

a)
```
if (disabled)
{
    enable();
}
```

b)
```
if (disabled) {
    enable();
}
```

(jatkuu)

Liite 1. jatkuu

c) if (disabled)
enable();

d) if (disabled) enable();

e) asialla ei ole merkitystä

25. Nimeämiskäytännöt, luokat (kirjoita tähän paras vaihtoehto): _____
a) public class FTPSender
b) public class FtpSender
c) public class Ftpsender
d) asialla ei ole merkitystä:
26. Nimeämiskäytännöt, muuttujat (valitse paras vaihtoehto): _____
a) boolean sent = true;
b) boolean isSent = true;
c) boolean issent = true;
d) asialla ei ole merkitystä:

Kommentit

27. Lähdekoodien kommentointia tulisi tehdä nykyistä enemmän
kyllä:
ei:
asialla ei ole merkitystä:

28. Dokumentoinnin tulisi olla ensisijaisesti
kaavioissa:
dokumenteissa:
lähdekoodien kommenteissa:
asialla ei ole merkitystä:

29. Päivitän kaaviot ja dokumentit
aina muutoksia toteuttaessani:
projektin lopussa:
satunnaisesti:
asialla ei ole merkitystä:

30. Kaavioiden, dokumenttien ja Java-koodien tulisi vastata toisiaan kaikissa olosuhteissa
kyllä:
ei:
asialla ei ole merkitystä:

31. Java-koodissa tulisi olla vakiotägien lisäksi yhteisesti määritteltyjä tägejä, kuten "@history" (muutoksille), "@test" (onko uusin versio testattu) ja "@bug" (tunnetuille ongelmille)
kyllä:
ei:
asialla ei ole merkitystä:

(jatkuu)

Liite 1. jatkuu

Moduulitestaus

32. Moduulitestausta tulisi tehdä nykyistä enemmän
    kyllä:
    ei:
    asialla ei ole merkitystä:

33. Moduulitestaus kuuluu
    koodin tuottaneelle ohjelmoijalle:
    kenelle tahansa projektitiimiin kuuluvalle ohjelmoijalle:
    testaajalle:
    asialla ei ole merkitystä:

34. Teen moduulitestausta
    testisuunnitelman ja testitapauksien perusteella:
    oman suunnitelmani perusteella:
    suunnittelemattomasti:
    en ollenkaan:
    asialla ei ole merkitystä:

Liite 2. Haastattelulomake

Toteutusmenetelmät ja toiminnan nykytila

1. Vaihtelevatko toteutusmenetelmät esim. moduulisuunnittelun, ohjelmakoodin kirjoittamisen, dokumentoinnin ja moduulitestauksen osalta eri projekteissa?
2. Miksi toteutusmenetelmät vaihtelevat?
3. Onko toteutusmenetelmien vaihtelemisesta hyötyä/haittaa?
4. Esiintyykö SDI:ssä pyrkimyksiä muuttaa nykyisten suunnittelu-, toteutus- ja testausvaiheiden painotusta?
5. Vaikuttavatko SDI:n toimintaan yleisesti tunnetut prosessimallit tai menetelmät?

Prosessimalli

6. Soveltuuko SDI:n nykyinen prosessimalli SDI:n toimintaan?
7. Poiketaanko prosessimallista SDI:n projekteissa?
8. Miksi prosessimallista poiketaan?
9. Onko prosessimallin poikkeamista hyötyä/haittaa?
10. Vaikuttavatko prosessimallin poikkeamat toteutusmenetelmiin?
11. Vaikuttavatko vaihtelevat toteutusmenetelmät prosessimallin poikkeamiin?

Yhtenäinen toiminta

12. Pitäisikö SDI:n pyrkiä toimimaan yhtenäisemmin toteutusmenetelmien osalta?
13. Pitäisikö yhtenäiset toteutusmenetelmät koota parhaat käytännöt -dokumenttiin?
14. Pitäisikö yhtenäisten toteutusmenetelmien käyttämistä valvoa?
15. Onko Sinulla sellaisia toteutusmenetelmiä, ratkaisumalleja tai toimintatapoja jotka voisivat soveltua muillekin?
16. Mitä nämä toteutusmenetelmät, ratkaisumallit tai toimintatavat ovat?