

Koodikatselmoinnit pienessä ohjelmistoyrityksessä

Nikita Zhuk

Helsinki 14.12.2008

Pro gradu -tutkielma

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen tiedekunta		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Nikita Zhuk			
Työn nimi — Arbetets titel — Title			
Koodikatselmoinnit pienessä ohjelmistoyrityksessä			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Pro gradu -tutkielma		14.12.2008	69 sivua + 4 liitesivua
Tiivistelmä — Referat — Abstract			
<p>Kirjallisuudessa esitettyjen tulosten perusteella katselmoinnit ovat tehokas ohjelmistojen laadunvarmistusmenetelmä, sillä se mahdollistaa ohjelmiston virheiden löytymisen mahdollisimman aikaisessa vaiheessa. Pienet ohjelmistoyritykset kohtaavat ohjelmiston laadunvarmistuksen haasteita, joihin katselmoinnit voisivat vastata. Katselmointien soveltamista pienissä ohjelmistoyrityksissä on käsitelty kirjallisuudessa vähän, ja kvantitatiivista tietoa katselmointien tehokkuudesta pienissä ohjelmistoyrityksissä ei ole juuri saatavilla. Tutkimuksessa selvitettiin, ovatko katselmoinnit tehokas keino parantaa ohjelmistotuotteen laatua pienessä ohjelmistoyrityksessä.</p> <p>Kirjallisuuskatsauksessa esiteltiin katselmointien historiaa, keskeisiä käsitteitä, kirjallisuudessa kuvattuja katselmointimenetelmiä sekä viimeaikaisen katselmointitutkimuksen pääsuuntauksia. Katselmointimenetelmistä tarkasteltiin tarkastusta, ryhmäkatselmoiteja, läpikäyntejä, pariohjelmointia ja kierrätystä. Katselmointitutkimuksen tarkastelu rajattiin lukuteknikoihin, katselmointien tehokkuustekijöihin ja katselmointiprosesseihin.</p> <p>Tutkimusosassa suunniteltiin kirjallisuuskatsauksen tietojen perusteella katselmointiprosessi, josta jätettiin pois pakolliset katselmointikokoukset ja mahdollistettiin seurantakokousten järjestäminen tarpeen mukaan. Katselmointiprosessi otettiin käyttöön pääkaupunkiseudulla sijaitsevassa alle kymmenen hengen ohjelmistoyrityksessä. Tutkimuskysymystä tutkittiin tapaustutkimuksella, jossa selvitettiin, ovatko katselmoinnit tehokas keino parantaa esimerkkirytyksen ohjelmistotuotteen laatua.</p> <p>Tutkimus osoitti, että katselmoinnit olivat tehokkaita ja ne vaikuttivat positiivisesti ohjelmistotuotteen laatuun. Yrityksen työntekijät ja johto pitivät katselmoiteja hyödyllisinä. Katselmoinnit osoittautuivat tehokkaaksi keinoksi levittää tietoa ohjelmistotuotteesta ja ohjelmointikonventioista kehittäjien välillä. Tutkimukseen osallistunut yritys päätti laajentaa katselmointien käyttöä muihin ohjelmistoprojekteihin.</p> <p>ACM Computing Classification System (CCS): D.2.5 [Testing and Debugging] K.6.3 [Software Management]</p>			
Avainsanat — Nyckelord — Keywords			
laadunvarmistus, laatu, katselmointi, pienet ohjelmistoyritykset			
Säilytyspaikka — Förvaringsställe — Where deposited			
Tietojenkäsittelytieteen laitoksen kirjasto, sarjanumero C-			
Muita tietoja — övriga uppgifter — Additional information			

Sisältö

1	Johdanto	1
2	Tarkastus	2
2.1	Historia	4
2.2	Keskeiset käsitteet	5
2.3	Roolit	7
2.4	Tarkastusprosessi	8
2.4.1	Suunnittelu	9
2.4.2	Yleiskatsaus	10
2.4.3	Valmistelu	10
2.4.4	Tarkastuskokous	11
2.4.5	Korjaus	12
2.4.6	Seuranta	12
3	Muut katselmointimenetelmät	13
3.1	Ryhmäkatselmointi	14
3.2	Läpikäynti	16
3.3	Pariohjelmointi	17
3.4	Kierrätys	18
4	Katselmointitutkimus	19
4.1	Lukutekniikat	20
4.1.1	Ad hoc-lukeminen	20
4.1.2	Tarkistuslistat	21
4.1.3	Skenaariopohjaiset lukutekniikat	22
4.1.4	Yksilölliset erot lukutekniikoissa	24
4.2	Kokoukset	24
4.3	Prosessit	26

	iii
5 Tutkimusympäristö	27
5.1 Pienet ohjelmistoyritykset	28
5.2 Yrityksen esittely	30
5.3 Nykyinen laadunvarmistusprosessi	31
5.4 Yrityksen ongelmat	33
6 Tutkimuksen suunnittelu	34
6.1 Tutkimuskysymys	35
6.2 Katselmointiprosessin mittaus	36
6.3 Teemahaastattelu	37
6.4 Tutkimuksen riskit	38
7 Tutkimuksen toteutus	39
7.1 Katselmointiprosessi	39
7.1.1 Roolit	40
7.1.2 Aloitusehdot	41
7.1.3 Tarkistuslista	42
7.1.4 Yleiskatsaus	43
7.1.5 Katselmointi	44
7.1.6 Korjaus	46
7.1.7 Seuranta	46
7.2 Toimitettu materiaali	48
7.3 Haastattelujen järjestelyt	49
8 Tutkimuksen tulokset	49
8.1 Katselmointien tehokkuus	50
8.2 Löytyneet virheet	51
8.3 Vaikutukset ohjelmiston laatuun	54
8.4 Vaikutukset ohjelmiston tuntemukseen	55
8.5 Katselmointiprosessi	55

	iv
8.6 Vaikutukset työkalutukeen	57
8.7 Johtopäätökset tuloksista	58
9 Yhteenveto	59
Lähteet	61
Liitteet	
1 Katselmointiprosessin kuvaus	
2 Haastattelun teemat	
3 Tilastolliset testit	

1 Johdanto

Yksi ohjelmistotuotannon tärkeimmistä osa-alueista on toteutettavan ohjelmiston laadunvarmistus, jolla pyritään varmistamaan, että ohjelmisto täyttää sille asetetut vaatimukset [Joh94]. Ohjelmiston virheiden löytäminen mahdollisimman aikaisessa vaiheessa pienentää ohjelmistojen tuotantokustannuksia, sillä mitä aikaisemmin virheet havaitaan, sitä vähemmän resursseja kuluu niiden korjaamiseen [Fag76, BoB01, SBB02, McC04].

Testaus ja katselmoinnit ovat kaksi tärkeintä menetelmää ohjelmistojen laadunvarmistuksessa [McC96]. Testauksessa valmista ohjelmistoa tai sen osaa suoritetaan antamalla sille testisyötteitä ja varmistetaan ohjelmiston tai sen osan tuottamien tulosteiden oikeellisuus [BaM90]. Katselmoinneissa ohjelmiston dokumentteja tai koodia tarkistetaan manuaalisesti, joten katselmoitavan ohjelmiston tai dokumentin ei tarvitse olla katselmointivaiheessa vielä valmis [McC96, Fag76, Wie01]. Tämä mahdollistaa virheiden löytämisen aikaisemmassa vaiheessa kuin testausta käytettäessä. Katselmointien avulla voidaan löytää myös muita kuin toiminnallisia virheitä, kuten ohjelmiston ylläpidettävyyteen, laajennettavuuteen tai suorituskykyyn liittyviä ongelmia.

Katselmoinnit ovat esiintyneet tieteellisessä kirjallisuudessa 70-luvulta lähtien. Niiden avulla on pystytty löytämään ja korjaamaan 50% – 90% katselmoitavien dokumenttien ja lähdekoodin virheistä [Fag86, ABL89, Gil00]. Grady ja Van Slack raportoivat erään katselmointimenetelmän — tarkastusten — käyttöönoton Hewlett-Packardilla säästäneen 28.5% kustannuksista määrittelyvaiheessa, 33.8% suunnitteluvaiheessa, 37.3% toteutusvaiheessa ja 16.1% testausvaiheessa [GrS94]. Russell arvioi kunkin tarkastustunnin ehkäisevän keskimäärin 33 tuntia ylläpitotyötä Bell-Northern Researchin suuressa, yli 15 miljoonan koodirivin projektissa [Rus91]. Vaikka empiiristen tutkimusten raportoimat tarkastusten ja muiden katselmointimenetelmien hyödyt vaihtelevat merkittävästi, katselmointien käyttöönotto on vaikuttanut lähes jokaisessa tapauksessa positiivisesti [GiG93, Lee97, ABL89, Van99].

Katselmoiteja arvioivat tutkimukset ovat perinteisesti keskittyneet vakiintuneita prosesseja käyttäviin keskisuuriin ja suuriin yrityksiin [Rus91, Iis04]. Katselmoinnit ovat kuitenkin käytössä myös pienissä ohjelmistoyrityksissä [HTH05]. Pienet yritykset eivät kerää katselmointiin liittyviä mittoja yhtä laajasti kuin suuremmat yritykset, joten näiden yritysten katselmoinneista ei ole saatavilla yhtä laajaa kvantitatiivista tietoa.

Ohjelmistojen laadunvarmistuksen haasteet koskevat myös pieniä, alle 10 hengen yrityksiä, joissa yhden projektin parissa saattaa työskennellä 2 – 3 ihmistä [Iis04]. Virheiden löytäminen ajoissa pienten yritysten ohjelmistoprojekteissa on tärkeää, sillä muussa tapauksessa projektin edetessä aikaa käytetään yhä enemmän virheiden paikallistamiseen ja korjaamiseen [Fag02]. Tämä vähentää uusien ominaisuuksien kehittämiseen käytettävää aikaa. Tällaisen trendin jatkuessa kaikki ajalliset resurssit käytetään virheiden paikallistamiseen ja korjaamiseen, eikä asiakkaalle saada tuotettua lisäarvoa uusien ominaisuuksien muodossa.

Tässä pro gradu -tutkielmassa tarkastellaan erilaisia katselmointimenetelmiä sekä esitellään viimeaikaista katselmointitutkimusta. Tutkielma keskittyy lähdekoodin katselmointiin. Tutkielman osana suunnitellaan ja toteutetaan tapaustutkimus eräässä pienessä ohjelmistoyrityksessä. Tavoitteena on selvittää, parantako koodikatselmointi ohjelmistotuotteen laatua. Tutkielma dokumentoi tapaustutkimuksen suunnittelun, valmistelun, toteutuksen, analyysin ja johtopäätökset.

Tutkielman rakenne on seuraava. Luvussa 2 esitellään tarkastus, joka on eniten tieteellistä kiinnostusta herättänyt katselmointimenetelmä. Luvussa 3 tarkastellaan muita yleisessä käytössä olevia katselmointimenetelmiä. Luvussa 4 käsitellään viimeaikaisen katselmointitutkimuksen suuntauksia sekä esitellään alan keskeiset tutkimustulokset. Luvussa 5 esitellään tapaustutkimukseen osallistuva ohjelmistoprojekti, esimerkkiyritys sekä yrityksen nykyinen laadunvarmistusprosessi. Luvussa 6 esitetään tutkimuskysymys ja tutkimuksessa käytetyt tutkimusmenetelmät sekä arvioidaan tutkimuksen riskejä. Luvussa 7 esitellään tapaustutkimuksen suunnittelu, toteutus ja tulokset. Luvussa 8 esitellään tutkimuksen johtopäätökset. Lopuksi koostetaan yhteenveto tutkielmassa käsitellyistä aiheista ja tutkimuksen tuloksista sekä arvioidaan tutkimuksen merkittävyyttä esimerkkiyritykselle luvussa 9.

2 Tarkastus

Tarkastus on IEEE:n määritelmän mukaan staattinen analyysitekniikka, joka perustuu ohjelmistotuotoksen visuaaliseen tarkasteluun. Tarkastelun tavoitteena on löytää virheet, standardien rikkomukset ja muut ongelmat [IEE98]. Tarkastukset ovat olleet historiallisesti tehokkaita ohjelmiston laadunvarmistusmenetelmiä, ja niihin sijoitetun pääoman tuotto prosentti on ollut muita laadunvarmistustekniikoita korkeampi [HTV04]. Dokumenteista ja lähdekoodista on löydetty 50% –

90% virheistä [Fag86, ABL89, Gil00, SBB02], mikä on vähentänyt projektin koko kehityksen vaatimaa aikaa 10% – 30% [GiG93].

Tarkastusprosessista on olemassa eri variaatioita [GiG93, KnM93]. Tässä luvussa keskitytään Michael Faganin [Fag76] alun perin vuonna 1976 kuvaamaan tarkastusmenetelmään. Tämä tarkastusmenetelmä esiintyy kirjallisuudessa myös Fagan-tarkastuksen nimellä.

Faganin alkuperäisessä tarkastuksia esittelevässä artikkelissa [Fag76] tarkastukset nähdään ohjelmistotuotantoprosessin hallinnan välineenä. Faganin mukaan ohjelmistotuotantoprosessi koostuu joukosta peräkkäisiä vaiheita, joista kukin ottaa vastaan tietyn syötteen ja muuntaa sen halutuksi tulosteeksi. Ensimmäisen vaiheen syötteenä toimivat ohjelmiston vaatimukset ja viimeisen vaiheen tulosteena syntyy valmis verifioitu ohjelmistotuote. Kunkin vaiheen tehtävänä on tuottaa tuloste, joka täyttää sille asetetut valmiusehdot (exit criteria). Esimerkiksi toteutusvaiheen valmiusehtona voi olla ohjelmiston lähdekoodin onnistunut käänös, joka ei tuota yhtään kääntäjän varoitusta tai virheilmoitusta.

Koska virheiden korjaamisen siirtyminen prosessin myöhempisiin vaiheisiin nostaa virheiden korjauskustannuksia, kunkin prosessin vaiheen valmiusehdot kannattaa määritellä sellaisiksi, että ne sisältävät kyseisen vaiheen tulosteen tarkastuksen ja tarkastuksen aikana löytyneiden virheiden korjauksen [Fag76]. Näin yhden vaiheen sisällä syntyneet virheet eivät pääse kulkeutumaan seuraavaan prosessin vaiheeseen, mikäli virheet onnistutaan löytämään. Tarkastusten tehtävänä on varmistaa, että kunkin prosessin vaiheen tuloste on valmis toimimaan seuraavan vaiheen syötteenä.

Tarkastus on muodollinen ja taloudellisesti kannattava menetelmä virheiden löytämiseksi [Fag76]. Faganin vuonna 1976 julkaistun artikkelin mukaan tarkastus voidaan suorittaa suunnitteludokumenteille ja ohjelmiston lähdekoodille. Myöhemmin vuonna 1986 julkaisemassaan artikkelissa Fagan huomioi, että tarkastuksia on sovellettu onnistuneesti myös muihin ohjelmistokehityksessä tuotettuihin dokumentteihin kuten arkkitehtuurisuunnitelmiin, vaatimusmäärittelydokumentteihin ja käyttöohjeisiin [Fag86]. Tarkastuksia voidaan soveltaa mihin tahansa ohjelmistokehityksen tuotokseen, mikäli se vain on visuaalisesti tarkasteltavissa.

2.1 Historia

Tarkastusten kehittäjänä pidetään Michael Fagania, joka aloitti työt ohjelmistokehityksen parissa vuonna 1971 [Fag02]. Toimiessaan erään ohjelmistoprojektin projektipäällikkönä hän huomasi, että ohjelmistotuotteet sisälsivät valmistukseen huomattavan määrän virheitä ja niiden paikallistamiseen ja korjaamiseen meni huomattava määrä aikaa. Vaikka korjaustyötä pidettiin välttämättömänä osana ohjelmistojen kehitystä, korjaustyön vaatiman ajan suhdetta koko ohjelmiston kehityksen vaatimaan aikaan ei mitattu. Ilman mitattua tietoa korjaustyön suhteellisen osuuden muutoksiin ei voitu reagoida.

1970-luvulla vesiputousmallia [Roy70] pidettiin tehokkaana tapana tuottaa ohjelmistoja [Fag02]. Käytännön ohjelmistotuotantoprojekteissa ei kuitenkaan ollut selviä valmiusehtoja vesiputousmallin mukaisille prosessin eri vaiheille — määrittely, suunnittelu, toteutus, testaus ja ylläpito. Projektiryhmissä ei ollut yhteistä näkemystä siitä, milloin jokin prosessin työvaihe oli valmis ja milloin projekti oli valmis siirtymään prosessin seuraavaan vaiheeseen. Eriävät näkemykset eri vaiheiden valmiudesta aiheuttivat väärinkäsityksiä ja huomattavan määrän korjaustyötä. Projektien aikataulut oli vaikeaa, koska esimerkiksi testausvaiheen korjaustyön vaatimaa aikaa ei mitattu eikä siihen voitu varautua.

Faganin ammatillinen tausta laitteistoprojektien parissa vaikutti hänen asenteisiin ohjelmistokehityksen korjaustyön kustannuksia kohtaan [Fag02]. Hän päätti soveltaa laitteistoprojekteissa käyttämiään tarkastusmenetelmiä käyttöjärjestelmäprojektin komponentin kehitykseen [Fag76]. Kyseessä oli keskisuuri projekti, jonka suunnitteluun osallistui kolme ohjelmistosuunnittelijaa ja toteutukseen kolmetoista ohjelmoijaa. Projekti suoritettiin vesiputousmallin mukaisesti jakamalla se suunnittelu-, toteutus-, yksikkötestaus- ja integrointivaiheisiin, joiden jälkeen suoritettiin järjestelmätestaus.

Kokeilun aikana komponentin suunnitteludokumentit, lähdekoodi sekä yksikkötestit tarkastettiin manuaalisesti ennen integraatio- ja testausvaihetta [Fag76, Fag02]. Samalla mitattiin tarkastusten vaikutus ohjelmiston integraatio- ja testausvaiheessa löytyneiden virheiden määrään. Myös vaikutus virheiden paikallistamisen ja korjaamisen vaatimaan työmäärään huomioitiin. Faganin mukaan arviot virheiden paikallistamisen ja korjaamisen vaatimasta työmäärästä liikkuvat tuona aikana 30%:n ja 80%:n välillä.

Fagan muotoili tarkastuksille kaksi päätavoitetta — tuotteen parannuksen ja pro-

sessin parannuksen. Tuotteen parannuksen tavoitteena on paikallistaa ja korjata kaikki tuotteen virheet. Prosessin parannuksen tavoitteena on paikallistaa ja korjata ne prosessin syyt, jotka mahdollistivat tuotteen virheiden synnyn. Näin virheiden määrää voidaan alentaa pysyvästi esimerkiksi huomioimalla tarkastuksissa usein löytyvät ohjelmistokoodin virheet ja estämällä niiden syntymisen koodin kirjoitusvaiheessa koulutuksen tai työkalujen avulla [Wie01, SiV01, MVC08]. Samalla tarkastusprosessista kerättyjen mittojen avulla voidaan parantaa tulevien tarkastuskertojen tehokkuutta.

Kokeilun tulokset osoittivat, että ajallinen panostus virheiden paikallistamiseen ja korjaamiseen projektin aikaisessa vaiheessa oli kannattavaa. Kun tarkastuksiin kulunut aika sekä tarkastusten ja testauksen aikana löydettyjen virheiden korjaamiseen kulunut aika huomioitiin, projektin toteutus- ja testausvaiheen tuottavuus parani 23%. Kaikista löydettyistä häiriöistä 82% löydettiin tarkastusten aikana ja loput 18% testausvaiheessa [Fag76]. Saman faktan ovat myöhemmin todenneet lukuisat muut tutkijat ja ohjelmistoalan ammattilaiset [ABL89, Gil00, McC01, BoB01]. Tutkittuaan ja kehitettyään tarkastuksia kolmen ja puolen vuoden ajan Fagan julkaisi artikkelin "Design and code inspections to reduce errors in program development" [Fag76], jossa hän esitteli tarkastusmenetelmänsä ja käytöstä saadut tulokset.

2.2 Keskeiset käsitteet

Käsitteistö ei ole aivan vakiintunutta tarkastuksia ja katselmointeja käsittelevässä kirjallisuudessa. Englanninkielisessä kirjallisuudessa dokumenttien tai lähdekoodin virheistä tai muista parannuksista vaativista kohdista käytetään yleisesti termejä *defect*, *bug*, *error*, *fault* ja *anomaly* [IEE90, FeP98, Kol06]. Termejä käytetään usein synonyymeinä, vaikka niiden välille on määritelty eroja [Kol06, FeP98]. Esimerkiksi *bug*-termillä viitataan lähdekoodissa esiintyviin virheisiin, kun taas *defect*-termiä käytetään ennen lähdekoodin kirjoitusta tehtyihin virheisiin [FeP98]. *Anomaly*-termillä viitataan yleensä virheisiin, jotka eivät aiheuta suoraan ulkoisesti havaittavia häiriöitä ohjelman toiminnassa, mutta jotka voivat aiheuttaa epäsuorasti toimintahäiriöitä jatkossa. *Anomaly*-tyypin virheitä voivat olla esimerkiksi lähdekoodin nimeämiskonventioiden noudattamatta jättäminen tai virheelliset lähdekoodin kommentit.

Tässä tutkielmassa termien *defect*, *bug*, *error*, *fault* ja *anomaly* suomenkielisenä vas-

tineena käytetään sanaa *virhe*. Virheellä tarkoitetaan dokumentin tai ohjelmiston poikkeamaa määrittelystä tai käyttäjävaatimuksista, esimerkiksi väärän loogisen operaattorin käyttöä lähdekoodissa.

Dokumentissa tai koodissa oleva virhe voi aiheuttaa ohjelman toimintaan ulkoisesti havaittavan poikkeaman ohjelman vaaditusta toiminnasta, kuten virheellisen tulosteen tai ohjelman kaatumisen. Tällaisesta poikkeamasta käytetään englanninkielisessä kirjallisuudessa termiä *failure*. Jotkut kirjoittajat käyttävät myös termiä *defect* kuvaamaan sekä ohjelmiston sisäisiä virheitä että ulkoisesti havaittavia poikkeamia ohjelman toiminnassa. Ohjelman kaatumisesta käytetään myös termiä *crash*.

Tässä tutkielmassa termien *failure* ja *crash* suomenkielisenä vastineena käytetään sanaa *häiriö*. Kaikki virheet eivät johda häiriöihin [FeP98]. Virhe ilmenee häiriönä, mikäli joukko ehtoja täyttyy, esimerkiksi käyttäjä käyttää ohjelmistoa tietyllä tavalla tai tietyt ajoitusehdot täyttyvät. Jotkut häiriöt voivat syntyä useamman virheen vaikutuksesta.

Tarkastusten ja katselmointien aikana löydettyjä mahdollisia virheitä ja puutteellisuuksia sekä parannusehdotuksia kutsutaan usein englanninkielisellä termillä *issue* tai *finding*. Näiden termien suomenkielinenä vastineena käytetään sanaa *löydös*. Löydökset eivät ole välttämättä virheitä, koska niiden löytäjät ovat voineet tehdä virheellisiä päätelmiä niiden taustalla olevista tekijöistä. Löydökset todetaan virheiksi tai hylätään väärinä joko ryhmän tai tuotoksen tekijän toimesta.

Tarkastusten ja katselmointien tehokkuuden kuvaamiseen käytetään pääosin kahden eri käsitettä. Englanninkielisillä termeillä *effectiveness* ja *efficacy* tarkoitetaan tarkastuksessa löydettyjen virheiden määrää tuotoksen kokoyksikköä kohden. Yleinen kirjallisuudessa käytetty mittari on löydettyjen virheiden määrä tuhatta koodiriviä kohti. Kommenttirivit sisällytetään koodirivien määrään joissakin kirjallisuudessa esitetyissä tapauksissa. Tässä tutkielmassa näiden termien suomenkielisenä vastineena käytetään sanaa *tuloksellisuus*. Virheiden etsimisen kustannus-hyöty -suhdetta kuvaa taas englanninkielinen termi *efficiency*, jonka yleinen kirjallisuudessa käytetty mittari on löydettyjen virheiden määrä henkilötyötuntia kohti. Tässä tutkielmassa tämän käsitteen suomenkielisenä vastineena käytetään sanaa *kustannustehokkuus*. Tässä tutkielmassa käytetään yleistä termiä *tehokkuus*, kun tarkoitetaan sekä tuloksellisuutta että kustannustehokkuutta.

Tuotoksen tarkastuksessa tai katselmoinnissa tapahtuvan läpikäynnin nopeus on tärkeä työkalu tuloksellisuuden ja kustannustehokkuuden säätelyssä [VVS01].

Mittarina käytetään läpikäydyn tuotoksen kokoa suhteutettuna läpikäynnin vaatimaan aikaan. Lähdekoodin tapauksessa yleinen mittari on läpikäytyjen koodirivien määrä henkilötyötunnissa. Läpikäyntinopeudesta käytetään eri prosesseissa ja prosessin vaiheissa erilaisia termejä.

Tässä tutkielmassa käytetään termiä *katselmointinopeus* (*review rate*) tarkoittamaan tuotoksen läpikäynnin nopeutta. Termiä voidaan käyttää kaikkien katselmointimenetelmien tai katselmointiprosessin vaiheiden yhteydessä. Tarkastusten yhteydessä käytetään seuraavia tarkastuskohtaisia termejä: *valmistelunopeus* (*preparation rate*) ja *tarkastusnopeus* (*inspection rate*). Valmistelunopeus tarkoittaa tarkastuksen valmisteluvaiheessa noudatettua läpikäyntinopeutta (tarkastuksen valmisteluvaihe on kuvattu tarkemmin kohdassa 2.4.3). Tarkastusnopeus tarkoittaa tarkastuskokouksen aikana noudatettua läpikäyntinopeutta (tarkastuskokous on kuvattu tarkemmin kohdassa 2.4.4).

2.3 Roolit

Faganin mukaan tarkastusryhmän sopiva koko on neljä jäsentä, mutta ryhmään voi kuulua useampikin jäsen, esimerkiksi jos tuotoksen ymmärtämiseen vaaditaan monen eri ongelma-alueen tuntemusta [Fag76]. Tarkastusryhmään kuuluu tarkastuksen johtaja (moderator), lukija (reader), tarkastettavan tuotoksen tekijä (author), testaaja (tester) sekä kirjuri (recorder, scribe) [Fag86]. Tarkastuksen johtaja voi toimia myös kirjurin roolissa [Fag76, IEE98]. Kaikki tarkastukseen osallistuvat henkilöt toimivat myös tuotoksen tarkastajina.

Tarkastuksen johtajan rooli on keskeinen koko tarkastusprosessin kannalta [Fag76, Fag86]. Hän toimii tarkastusryhmän johtajana ja valmentajana. Johtajan tehtävänä on tunnistaa ja tuoda esille tarkastusryhmän jäsenten henkilökohtaiset vahvuudet ja käyttää niitä tehostamaan tarkastusprosessia [Fag76]. Hän vastaa tarkastusten suunnittelusta, tarkastuskokousten järjestämisestä ja niiden johtamisesta sekä katselmointikokousten yhteenvetoraporttien toimittamisesta tarkastajille.

Tarkastuksen johtaja valitsee lukijan, jonka tehtävänä on lukea ääneen tarkastettava tuotos pieni osa kerrallaan ja selittää oma tulkintansa tuotoksen suunnittelusta tai toiminnallisuudesta [Fag76]. Tavoitteena on käydä läpi kaikki tuotoksen loogiset polut ja niiden haarat. Lukijan tulkinta voi paljastaa tuotoksesta esimerkiksi dokumentoimattomia oletuksia, epätarkkoja ilmaisuja tai kommunikointia

vaikkeuttavia tyyliiongelmia [Wie01]. Lukijan tehtävänä on myös keskittää muiden tarkastajien huomio tarkastettavana olevan tuotoksen tiettyyn kohtaan kerrallaan [ABL89]. Näin voidaan kontrolloida kaikkien tarkastajien tarkastusnopeutta ja pitää se optimaalisella tasolla [Fag76].

Testaajan näkökulma tuo tuotoksen testattavuuden varmistuksen osaksi tarkastusta [Fag86]. Kooditarkastusten tapauksessa testaaja kehittää tuotoksen toiminnolle testitapaukset ja simuloi mielessään toimintojen suorituksen testitapausten avulla [LEH01]. Testaaja löytää virheen, mikäli toiminnon mentaalisen suorituksen tuottama arvo poikkeaa tarkastettavan tuotoksen alkuperäisestä määrittelystä. Testitapausten luomisen vaikeus voi viestiä tuotoksen heikosta testattavuudesta.

Kirjurin tehtävänä on dokumentoida tarkastuskokouksen aikana esille tulleet virheet sekä niiden tyypit ja vakavuusasteet [Fag76]. Mikäli löydetyn virheen ratkaisu on yksinkertainen, se kirjataan virheen yhteyteen. Ratkaisujen kehittäminen löydettyihin virheisiin ei ole katselmointikokouksen tarkoitus, joten keskustelu ratkaisuehdotuksista ei ole sallittua. Faganin tarkastusprosessissa tarkastuksen johtaja toimii myös kirjurina, mutta tätä roolia voi edustaa myös eri henkilö [IEE98].

Tuotoksen tekijän tehtävänä on perehdyttää muut tarkastusryhmän jäsenet tarkastettavan tuotoksen yksityiskohtiin [Fag76, Fag86]. Perehdytys säästää tarkastusryhmän aikaa, sillä tuotoksen tekijä pystyy selvittämään tuotoksen yksityiskohdat ajallisesti tehokkaammin kuin jos kunkin tarkastusryhmän jäsenen pitää selvittää yksityiskohdat itse [ABL89]. Tarkastuksen aikana tekijä voi tehostaa tarkastusprosessia tarjoamalla uusia näkökulmia tuotoksesta muille tarkastajille [Wie01]. Esimerkiksi tarkastajien löytäessä virheitä tekijä voi löytää nopeasti samantyyppisiä virheitä muista tuotoksen osista.

2.4 Tarkastusprosessi

Faganin vuonna 1986 määrittelemä tarkastusprosessi koostuu kuudesta peräkkäisestä vaiheesta, joista jokaisella on oma erillinen tavoite [Fag86]. Nämä vaiheet ovat suunnittelu (planning), yleiskatsaus (overview), valmistelu (preparation), tarkastuskokous (inspection), korjaus (rework) ja seuranta (follow-up). Fagan-tarkastusprosessin vaiheet sekä näihin vaiheisiin liittyvät päätökset on esitetty kuvassa 1.

Johtajan ja tuotoksen tekijän vastuulla on tuotoksen ja tarkastuksessa tarvittavien dokumenttien kerääminen ja jakaminen [IEE98]. Näihin dokumentteihin voi kuulua määrittely- ja suunnitteludokumentteja, tarkistuslistoja, standardeja ja muita dokumentteja, jotka auttavat tarkastettavan tuotoksen ymmärtämisessä [Wie01].

2.4.2 Yleiskatsaus

Yleiskatsausvaiheessa tarkastuksen johtaja jakaa tarkastusryhmän jäsenille roolit, jotka on kuvattu luvussa 2.3, ja perehdyttää jäsenet näiden roolien tehtäviin [Fag76]. Roolien jaon jälkeen tuotoksen tekijä kuvailee tuotoksen ongelma-alueen yleisesti ja esittelee tuotoksen tarkastettavana olevan osan yksityiskohtat [Fag76]. Tällaisia yksityiskohtia ovat esimerkiksi koodissa käytetyt suunnittelumallit, tekijän tekemät oletukset ja riippuvuudet muihin moduuleihin.

Yleiskatsauksen päätehtävänä on kouluttaa tarkastusryhmän jäsenet ymmärtämään tarkastuksen kohteena olevaa tuotosta kunkin jäsenen roolin edellyttämistä näkökulmasta [Fag76]. Mikäli tarkastettava tuotos ei ole ennestään tuttu tarkastusryhmälle, tekijän suorittama tuotoksen esittely voi säästää huomattavasti muiden tarkastajien aikaa [ABL89]. Yleiskatsausvaihe voidaan jättää suorittamatta, mikäli tarkastusryhmä tuntee tuotoksen jo ennestään [Fag76, Fag86].

2.4.3 Valmistelu

Valmisteluvaiheen päätehtävänä on tutustua tarkastettavaan tuotokseen ja valmistautua tarkastuskokoukseen [Fag76, Fag86]. Valmistelun aikana tarkastajat varmistavat, että tarkastettava tuotos vastaa määrittelyään ja noudattaa sille asetettuja standardeja [Wie01]. Tarkastajat dokumentoivat löytämänsä virheet ja tuovat ne esille myöhemmin järjestettävässä tarkastuskokouksessa. Merkityksettömät virheet voidaan kirjata erilliselle listalle (typo list) joka toimitetaan suoraan tuotoksen tekijälle [Gil00, Wie01].

Kukin tarkastaja raportoi valmisteluvaiheessa kuluneen ajan tarkastuksen johtajalle [Wie01]. Valmistelunopeus on tärkeä mittari arvioitaessa tarkastajien valmiutta tarkastuskokoukseen [Fag76, Wie01]. Faganin alkuperäisessä artikkelissa suositeltu valmistelunopeus käyttöjärjestelmän komponentin lähdekoodin osalta on 125 koodiriviä tunnissa. Fagan kuitenkin huomioi, että kyseinen esimerkiksi on konservatiivinen ja pätee vain käyttöjärjestelmäkoodin tapauksessa. Sovellusohjelmien lähdekoodin osalta tyypillinen kirjallisuudessa esiintyvä val-

mistautumisnopeus on 125 – 250 koodiriviä tunnissa, kun kommenttirivejä ja tyhjiä rivejä ei huomioida [Fag76, ABL89, BaP94, Wie01, RDN04]. Mikäli yrityksellä ei ole aikaisemmista tarkastuksista kerättyjä mittoja, tarkastuksen johtaja voi käyttää kirjallisuudesta saatuja arvoja lähtökohtana valmistelun ja tarkastuskokouksen suunnittelussa. Arvoja voidaan myöhemmin tarkentaa yrityksen omista tarkastuksista saaduilla mitoilla [Fag76]. Valmistelun tehokkuutta mitataan sekä tuloksellisuuden että kustannustehokkuuden mittareilla. Valmistelunopeus vaikuttaa suoraan valmistelun tuloksellisuuteen [VVS01, Wie01].

2.4.4 Tarkastuskokous

Tarkastuskokouksen aikana tarkastajat etsivät virheitä tuotoksesta [Fag76, Wie01]. Ennen kokouksen aloittamista tarkastuksen johtaja varmistaa, että tarkastajat ovat riittävän valmistautuneita tehokkaaseen tarkastuskokoukseen. Kokouksen aikana lukija valitsee tuotoksesta käsiteltäväksi yhden loogisen kokonaisuuden kerrallaan, kooditarkastusten tapauksessa yleensä 5 – 10 koodiriviä, ja selittää sen toiminnan muille tarkastajille [Fag76, Wie01]. Tarkastajat raportoivat ne virheet, jotka löytyivät kyseisestä kokonaisuudesta valmisteluvaiheessa ja pyrkivät löytämään samalla uusia virheitä.

Tarkastuksen johtajan vastuulla on pitää keskustelu löydöksistä konstruktiiivisena ja lyhyenä. Tarkastuskokouksessa ei tule keskustella löydettyjen virheiden ratkaisusta. Mikäli löydettyyn virheeseen on tiedossa triviaali ratkaisu, se voidaan kuitenkin dokumentoida. Suositeltu tarkastuskokouksen maksimipituus on kaksi tuntia, ja kuhunkin ongelmaan liittyvään keskusteluun tulee käyttää korkeintaan minuutti [Wie01]. Joissakin tapauksissa laajemman keskustelun salliminen on johtanut uusien virheiden löytämiseen [Doo92].

Tarkastuksen johtaja pyrkii pitämään tarkastusnopeuden sopivalla tasolla varmistukseen tarkastuksen tehokkuuden [Wie01, Fag76, VVS01]. Kirjallisuudessa suositellut tarkastusnopeudet ovat 95 – 300 kommentoimatonta koodiriviä tunnissa [Fag86, ABL89].

Kirjurin suorittaman kirjauksen yhteydessä virheet luokitellaan eri kategorioihin, tyypeihin ja vakavuusasteisiin [Fag76]. Virhekatgoria ilmaisee, millaisesta virheestä on kysymys, esimerkiksi looginen virhe, suorituskykyongelma tai luettavuusongelma. Virhetyyppi voi olla esimerkiksi puuttuva (missing), väärä (wrong) tai ylimääräinen (extra). Vakavuusasteiksi Fagan ehdottaa vakavaa (ma-

lor) ja vähäistä (minor), missä vakava määritellään virheeksi, joka jäädessään tuotokseen aiheuttaa todennäköisen häiriön tuotoksen toiminnassa. Vähäinen virhe ei aiheuta häiriötä, mutta voi heikentää esimerkiksi tuotoksen ylläpidettävyyttä tai testattavuutta.

Tarkastuksesta tehdään tarkastusraportti, johon liitetään yhteenveto kaikista tuotoksen osista löydetyistä virheistä, tarkastetun tuotoksen koosta sekä valmisteluun ja tarkastuskokoukseen käytetystä ajasta. Kokouksen päättymisen jälkeen tarkastuksen johtajan vastuulla on toimittaa tarkastusraportti sekä tarkastuksessa löydettyjen virheiden listaukset tuotoksen tekijälle [Fag76].

2.4.5 Korjaus

Korjausvaiheessa tarkastettavan tuotoksen tekijä käy virhelistat läpi ja tekee tarvittavat korjaukset tuotokseen [Fag76]. Tekijä voi muuttaa virheiden luokittelua tai vakavuusastetta, mikäli ne ovat tekijän mielestä virheellisiä. Tekijä voi jättää virheitä korjaamatta, mikäli hän ei näe niiden korjausta tarpeeksi tärkeänä tai niiden korjaus ei onnistu aikataulusyistä [Wie01]. Virheiden korjaamatta jättäminen aikataulusyistä on riskialtista, sillä korjauskustannukset nousevat ajan myötä. Virheiden korjauksen siirtäminen myöhemmälle ajankohdalle voi olla esimerkiksi perusteltua, jos ohjelmisto on pakko julkaista pikaisesti ja virheen ei oleteta aiheuttavan häiriötä ohjelmiston toiminnassa. Yksi tällainen virhe on lähdekoodin puutteelliset kommentit. Kaikki korjaamatta jääneet virheet sekä syyt niiden korjaamatta jättämiseen tulee dokumentoida, jotta näiden virheiden olemassaolosta tiedetään projektin myöhemmissä vaiheissa.

2.4.6 Seuranta

Seurantavaiheen tehtävänä on varmistaa, että tarkastuksessa löydetyt virheet on korjattu ja korjaukset eivät aiheuta uusia virheitä [Fag76]. Tarkastuksen johtaja varmistaa, että tekijä on käsitellyt kaikki löydetyt virheet joko tekemällä tarvittavat korjaukset tai esittämällä pätevät perustelut korjaamatta jättämiselle [Fag76, Wie01].

Mikäli korjaustyön seurauksena tuotoksesta muuttuu enemmän kuin 5%, tuotokselle tulee järjestää kokonaan uusi tarkastus [Fag76]. Pienemmissä muutoksissa joko tarkastuksen johtaja tai tarkastusryhmän muut jäsenet voivat tarkastaa tehdyt muutokset.

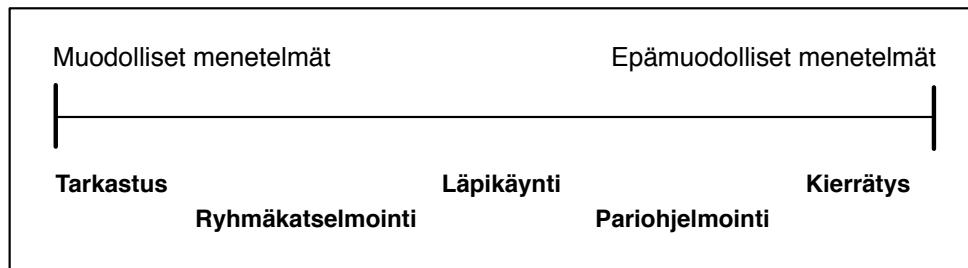
3 Muut katselmointimenetelmät

Luvussa 2 esitetty tarkastus edustaa yhtä mahdollista tapaa ohjelmistotuotteen katselmoimiseksi. Tarkastusten avulla voidaan löytää kustannustehokkaasti virheitä tarkastettavasta tuotoksesta. Tieteellisen kirjallisuuden mukaan tarkastusten avulla pystytään löytämään muita katselmointimenetelmiä enemmän virheitä [ABL89, Wie01, Fag02]. Tarkastusprosessiin liittyy samalla lukuisia vaatimuksia, joiden tulee täytyä, jotta katselmointimenetelmä voidaan pitää tarkastuksena. Näihin vaatimuksiin kuuluvat muun muassa dokumentoitu prosessi, eksplisiittiset aloitusehdot, hyvin määritellyt osallistujien roolit ja vastuut, koulutetun tarkastuksen johtajan vetovastuu, katselmointiryhmän jäsenten koulutus ja muodollisten kokousten pitäminen [Wie01]. Tällainen muodollisuuden ja kurinalaisuuden aste ei ole tarpeellinen kaikille ohjelmistoprojekteille, eikä se sovi kaikkien yritysten yrityskulttuuriin [TIH98].

Tarkastusten lisäksi on olemassa muita katselmointimenetelmiä, joiden vaatima kurinalaisuus ja tarjoama joustavuus eroavat huomattavasti tarkastuksista [Wie01]. Eri katselmointimenetelmiä voidaan luokitella sijoittamalla ne suoralle kunkin menetelmän muodollisuuden asteen perusteella. Tässä luvussa käsiteltävät katselmointimenetelmät on sijoitettu menetelmien muodollisuutta kuvaavalle akselille kuvassa 2.

IEEE:n ohjelmistokatselmointistandardi Std 1028–1997 [IEE98] määrittelee katselmoinnin (review) prosessiksi tai kokoukseksi, jonka aikana ohjelmistotuotos esitellään projektin henkilöstölle, johtajille, käyttäjille, asiakkaille, käyttäjien edustajille tai muille kiinnostuneille osapuolille kommentoitavaksi tai hyväksyttäväksi. Katselmointi voidaan nähdä verifiointimenetelmänä, jonka tavoitteena on varmistaa, että katselmoitava tuotos täyttää sille asetetut vaatimukset sekä noudattaa sille asetettuja määritelmiä ja standardeja [Wie01]. Katselmoitavana voi olla mikä tahansa ohjelmistokehityksen aikana syntyvä tuotos, kuten määrittely- tai suunnitteludokumentti, lähdekoodi tai testitapaus [PVB95, Fag02, IEE98, Doo92]. IEEE:n ohjelmistokatselmointistandardi Std 1028–1997 [IEE98] listaa 37 eri ohjelmistokehityksen tuotosta, jotka voidaan ottaa katselmoinnin kohteeksi.

Kaikille eri katselmointimenetelmille on yhteistä se, että tuotoksen katselmoi joku muu kuin tuotoksen alkuperäinen tekijä, mutta tuotoksen tekijä on kuitenkin usein mukana katselmointiprosessissa [KnM93, BMW94, Wie01]. IEEE:n ohjelmistokatselmointistandardi Std 1028–1997 [IEE98] listaa viisi eri katselmointi-



Kuva 2: Katselmointimenetelmien muodollisuus [Wie01].

menetelmää: johdon suorittama katselmointi (management review), ryhmäkatselmointi (team review), tarkastus (inspection), läpikäynti (walkthrough) ja auditointi (audit) [IEE98]. Näiden katselmointimenetelmien lisäksi myös pariohjelmointi (pair programming) [Bec99], parikatselmointi (pair deskcheck) sekä kierrätys (passaround) ovat standardin määritelmän mukaisia katselmointimenetelmiä [Wie01].

Katselmointimenetelmät eroavat toisistaan muodollisuus-, kurinalaisuus- ja joustavuusaspektiensa lisäksi tavoitteissaan [Wie01, Fag02]. Vaikka virheiden löytäminen katselmoitavasta tuotoksesta on osana kaikkien katselmointimenetelmien tavoitteita, sen painoarvo vaihtelee kunkin menetelmän sisällä. Esimerkiksi läpikäynnin päätavoite on usein tuotoksen esittely siitä kiinnostuneille ihmisille, ja virheiden löytäminen on toissijaista. Jotkut menetelmät taas eivät tavoittele prosessinparannusta, joten katselmoinnin tuloksellisuuteen, kustannustehokkuuteen ja nopeuteen liittyvien mittojen keräys ei ole osana näitä menetelmiä.

3.1 Ryhmäkatselmointi

Ryhmäkatselmointi (team review, myös technical review) on eräänlainen tarkastusten kevennetty versio [Wie01]. IEEE:n ohjelmistokatselmointistandardi Std 1028–1997 [IEE98] määrittelee ryhmäkatselmoinnin systemaattiseksi asiantuntijoista koostuvan ryhmän suorittamaksi ohjelmatuotoksen evaluoinniksi, joka tarkastelee tuotoksen soveltuvuutta sen aiottuun käyttöön sekä identifioi poikkeamat sen määrittelystä ja standardeista. Ryhmäkatselmointien aikana voidaan käsitellä myös vaihtoehtoisia ratkaisutapoja. Ryhmäkatselmoinnin tavoitteena on parantaa tuotosta, kerätä ja vertailla toteutusnäkökulmia, etsiä ja korjata tuotoksesta löytyneet virheet sekä kouluttaa ryhmän jäseniä [Fag02].

Ohjelmistokatselmointistandardin [IEE98] mukaan ryhmäkatselmoinnin roolit ovat päätöksentekijä, katselmointijohtaja, kirjuri sekä teknisistä asiantuntijoista koostuva ryhmä. Päätöksentekijänä toimii se henkilö, jota varten ryhmäkatselmointi järjestetään, käytännössä usein tuotoksen tekijä [Wie01]. Katselmointijohtaja vastaa katselmoinnin järjestämisestä ja suorituksesta. Kirjurin ja asiantuntijaryhmän tehtävät ovat samoja kuin tarkastuksissa.

Ryhmäkatselmoinnin suoritus etenee pitkälti samalla tavalla kuin tarkastuksen tapauksessa [Wie01, IEE98]. Katselmoinnin johtaja valitsee katselmointiryhmän, suunnittelee aikataulun ja varaa tilat kokousta varten. Tuotoksen tekijä kerää yhdessä katselmoinnin johtajan kanssa katselmoinnin aikana tarvittavat dokumentit, kuten tarkistuslistat tai suunnitteludokumentit, ja jakaa ne yhdessä katselmoitavan tuotoksen kanssa osallistujille. Johtaja järjestää tarvittaessa yleiskatsauskokouksen joko ennen katselmointikokousta tai osana sitä, mutta yleiskatsausvaihe on ryhmäkatselmoinneissa harvinainen [Wie01, Van99].

Kukin osallistuja tutustuu itsenäisesti tuotokseen ja dokumentoi kaikki löytämänsä virheet [Wie01, IEE98]. Katselmoijat pitävät kirjaa valmisteluun kuluneesta ajasta. Tämän jälkeen järjestetään katselmointikokous, jossa kirjuri dokumentoi löytyneet virheet ja koostaa kokouksesta yhteenvetoraportin, joka toimitetaan tuotoksen tekijälle. Kokouksen aikana sallitaan keskustelu löytyneiden virheiden ratkaisuehdotuksista tai tuotoksen osien vaihtoehtoisista toteutustavoista.

Katselmointijohtaja voi järjestää seurantakokouksen virheiden korjausten varmistusta varten. Kokouksessa varmistetaan, että kaikki löydetty virheet on korjattu. Seurantakokous on kuitenkin yleiskatsausvaiheen tavoin harvinainen [Wie01, Van99].

Ryhmäkatselmoinnin suurin ero tarkastukseen verrattuna on tuotoksen tekijän aktiivinen rooli katselmointiprosessissa. Tekijä voi toimia samalla katselmoinnin johtajan roolissa [Wie01]. Katselmointikokouksessa tuotoksen esittelyyn ei käytetä erillistä lukijaa, vaan katselmoinnin johtaja käy tuotoksen läpi esimerkiksi sivu kerrallaan ja kerää osallistujien kommentit. Erillisen lukijan puute, yleiskatsaus- ja seurantakokousten puuttuminen sekä ratkaisuvaihtoehtojen evaluoinnin salliminen tekevät ryhmäkatselmoinnista tarkastusta kevyemmän ja vähemmän muodollisen katselmointimenetelmän. Ryhmäkatselmointien tuloksellisuuden on todettu olevan tarkastuksia heikompi [Van99, Wie01, Fag02].

3.2 Läpikäynti

Ohjelmistokatselmointistandardin [IEE98] mukaan läpikäynti (walkthrough) on staattinen analyysitekniikka, jossa suunnittelija tai ohjelmoija esittelee tuotoksen kehittäjäryhmälle ja muille kiinnostuneille osapuolille ja osallistujat esittävät tuotokseen liittyviä kysymyksiä sekä kommentoivat mahdollisia virheitä, standardien rikkomuksia ja muita ongelmia. Läpikäynnissä tekijällä on johtava rooli. Hänen vastuullaan on läpikäynnin suunnittelu, läpikäyntikokouksen johtaminen ja tuotoksen esittely [Wie01]. Muita erityisiä rooleja ei yleensä käytetä, mutta kirjurin ja erillisen läpikäyntikokouksen johtajan käyttäminen on mahdollista [IEE98]. Läpikäyntiprosessin alussa tuotoksen tekijä määrittelee tavoitteet, jotka pyritään saavuttamaan läpikäynnin avulla [IEE98]. Katselmointistandardi Std 1028–1997 [IEE98] määrittelee neljä läpikäyntien päätavoitetta: virheiden löytäminen, tuotoksen laadun parantaminen, vaihtoehtoisten ratkaisujen evaluointi sekä standardien ja määritelmien mukaisuuden tarkistus. Tekijä voi hyödyntää läpikäyntiä esimerkiksi esitellessään mahdollista toteutusratkaisua johonkin ongelmaan [Wie01]. Läpikäynnit voivat tällöin stimuloida muita osallistujia kehittämään vaihtoehtoisia, mahdollisesti parempia toteutusratkaisuja.

Läpikäynnissä tekijä valitsee osallistujat, suunnittelee aikataulun ja varaa tilat läpikäyntikokousta varten [IEE98, Wie01]. Tekijän vastuulla on tuotoksen ja siihen liittyvien dokumenttien jakelu osallistujille. Tarkistuslistoja ei yleensä käytetä. Kukin osallistuja tutustuu itsenäisesti tuotokseen ja dokumentoi löydetyt virheet ja muut ongelmat ennen läpikäyntikokousta. Kokouksen aikana tekijä esittelee tuotosta ja kirjuri dokumentoi esille tuodut asiat, kuten löydetyt virheet tai vaihtoehtoiset ratkaisuehdotukset. Kokouksen jälkeen tekijän vastuulla on kokouksessa ilmenneiden asioiden käsittely ja toteutus, esimerkiksi virheiden korjaus tai vaihtoehtoisten ratkaisujen käyttöönotto.

Yksi tärkeä läpikäyntien sovellusalue ovat globaalit ohjelmistoprojektit [Paa05]. Maantieteellisesti hajautettujen ohjelmistoprojektien yleistyessä tärkeäksi haasteeksi nousee sekä yritysten sisällä olevien projektiryhmien että eri yritysten välinen kommunikaatio. Läpikäyntien avulla voidaan varmistaa mahdollisimman aikaisessa vaiheessa, että osapuolet ovat ymmärtäneet jonkin ohjelmistotuotoksen vaatimukset oikein. Tähän voidaan käyttää perinteisten kokousten lisäksi myös virtuaalisia läpikäyntejä Internetin tai puhelimen välityksellä. Yritysten välistä kommunikaatiota käsittelevässä väitöskirjassaan Maria Paasivaara tutki suomalaisten ohjelmistoyritysten globaaleissa projekteissa käyttämiä kommunikoin-

titapoja [Paa05]. Seitsemässä projektissa kymmenestä läpikäyntejä käytettiin suunnitteludokumenttien ja lähdekoodin yhteydessä. Projekteihin osallistuneet kehittäjät pitivät läpikäyntien tarjoamaa palautetta erityisen tärkeänä silloin, kun se saatiin aikaisessa vaiheessa — ennen kuin suunnittelusta tai koodista oli valmiina merkittäviä osuuksia.

Läpikäynnit eivät ole yleensä dokumentoituja menetelmiä. Läpikäyntien aikana ei kerätä mittaustietoa niiden tuloksellisuudesta tai tehokkuudesta. Mittaustiedon puuttuminen vaikeuttaa läpikäyntien tutkimista ja niiden tuloksellisuuden tai tehokkuuden vertaamista muihin katselmointimenetelmiin [Fag86]. Eräässä Faganin tutkimuksessa tarkastusprosessin avulla katselmoitu ohjelmistokomponentti sisälsi testausvaiheessa 38% vähemmän virheitä kuin vastaava komponentti, jonka katselmointiin sovellettiin läpikäyntiä [Fag76].

3.3 Pariohjelmointi

Pariohjelmointi (pair programming) on menetelmä, jossa kaksi kehittäjää istuu fyysisesti saman koneen ääressä ja jakaa hiiren ja näppäimistön [Bec99]. Kehittäjillä on kaksi roolia — ohjaaja (controller, myös driver) ja tarkkailija (observer, myös navigator) [Bec99, ChH07]. Ohjaajalla on hallussaan tietokoneen syöttölaitteet, ja hänen tehtävänä on suunnitella ja tuottaa lähdekoodia. Tarkkailijan tehtävänä on tarkkailla ohjaajan työskentelyä lukemalla ja tarkistamalla lähdekoodia. Virheiden etsimisen lisäksi tarkkailija voi kysyä ohjaajalta selventäviä koodiin tai suunnitteluun liittyviä kysymyksiä. Kehittäjät vaihtavat rooleja keskenään säännöllisin väliajoin.

Pariohjelmointi voidaan nähdä katselmointimenetelmänä, koska tarkkailijan roolissa oleva kehittäjä suorittaa jatkuvaa reaaliaikaista koodikatselmointia koodin ilmestyessä tietokoneen näytölle [Wie01]. Se on hyvin epämuodollinen menetelmä, koska siihen ei liity erillistä suunnitteluvaihetta, prosessinkuvausta tai dokumentointia. Löytyneitä virheitä ei dokumentoida, jolloin virheiden jatkoanalysointi ja prosessinparannus eivät ole mahdollisia samalla tavalla kuin muodollisissa katselmointimenetelmissä.

Virheiden löytäminen reaaliaikaisen katselmoinnin avulla ei ole ainoa pariohjelmoinnin tavoite. Koska tuotos syntyy kahden kehittäjän voimin, tiedot tuotoksen suunnitteluperiaatteista ja toteutuksen yksityiskohdista eivät ole vain tuotoksen kehittäjän tiedossa [Bec99]. Pariohjelmoinnissa kehittäjien välinen kommunika-

tio on jatkuvaa ja kohdistuu kehitettävään tuotokseen, mikä mahdollistaa pariohjelmoinnin käyttämisen koulutuksen välineenä esimerkiksi muodostamalla parit kokemustasoltaan erilaisista kehittäjistä [Bec99, Wie01, ChH07].

Kahden kehittäjän käyttäminen kehitysvaiheessa vaatii kaksinkertaista työmäärää, mikä tuplaa kehitysvaiheen kustannukset [Mul04]. Pariohjelmoinnin kehittäjän Kent Beckin mukaan kehitysvaiheen korkeammat kustannukset ovat perusteltuja, sillä pariohjelmoinnin tuloksena syntyvä tuote syntyy nopeammin ja sisältää vähemmän virheitä kuin tuote, jonka kehitykseen ei käytetä pariohjelmointia [Bec99]. Virheiden pienempi määrä vähentää korjaustyön tarvetta projektin myöhemmissä vaiheissa, jolloin pariohjelmoinnin käyttäminen johtaa nettosäästöihin. Tämä väite on saanut osakseen sekä tukea [MuP03, HeI06] että kritiikkiä [HuA05].

3.4 Kierrätys

Kierrätys (passaround, myös virtual peer review) on katselmointimenetelmä, jossa tuotoksen tekijä valitsee yhden tai useamman katselmoijan ja toimittaa heille tuotoksensa katselmoitavaksi [Wie01]. Tekijän ja katselmoijien lisäksi kierrätyksessä ei käytetä muita rooleja. Kukin katselmoija tarkastelee tuotosta itsenäisesti ja raportoi löydetyt virheet ja muut ongelmat tekijälle. Tekijä koostaa osallistujilta saamansa palautteen ja tekee tuotokseen tarvittavat muutokset. Tekijä voi halutessaan tarkentaa osallistujien kommentteja keskustelemalla heidän kanssaan [IiT98, TIH98, MFR94].

Yksi kierrätyksen erikoistapaus on parikatselmointi (peer deskcheck, pair review), jossa tuotoksen tekijä pyytää yhtä henkilöä katselmoimaan tuotosta [Wie01, IiT98]. Mikäli tuotoksen tekijä ja katselmoija muodostavat kiinteän työparin, he sitoutuvat tuotokseen vahvemmin kuin useamman osallistujan toteuttamassa katselmoinnissa ja suhtautuvat suuremmalla todennäköisyydellä katselmointiin vakavasti [IiT98]. Parikatselmointi on halpa katselmointimenetelmä, sillä tekijän lisäksi tarvitaan vain yhden henkilön työpanos. Tämä mahdollistaa parikatselmoinnin käytön pienissä ohjelmistoyrityksissä, joissa työvoiman saatavuus on rajallista [ITH99]. Yhden katselmoijan käyttäminen on kuitenkin riskialtista, sillä katselmoinnin tuloksella ja yhden henkilön tiedoilla ja virheenlöytökyvyillä on suora riippuvuus [Wie01].

Valtaosa parikatselmoinnin ja pariohjelmoinnin tavoitteista on samoja — kah-

den kehittäjän käyttäminen tuotteen laadun parannukseen ja tietojen jakamiseen [Mul04]. Eräässä parikatselmoinnin ja pariohjelmoinnin tehokkuutta ja tuloksellisuutta vertailevassa tutkimuksessa parikatselmoinnilla saavutettiin keskimäärin sama ohjelmiston luotettavuustaso kuin pariohjelmoinnin avulla [Mul04]. Parikatselmointi vaati pariohjelmointiin verrattuna hieman vähemmän työaikaa.

Kierrätyksen muodollisuus vaihtelee sen mukaan, millaisia menetelmiä osallistujat käyttävät [Wie01]. Kierrätykselle ei ole olemassa virallista prosessikuvausta, joka määrittelee prosessin vaiheet ja kerättävät mitat. Parikatselmoinnissa työparin väliset katselmointikäytännöt kehittyvät yleensä itsestään tekijän ja tarkastajan keskustelujen aikana, eikä näiden keskustelujen aikana löytyneiden virheiden dokumentointi ole pakollista [ITH99]. Samoin kuin pariohjelmoinnissa, virheiden jatkoanalysointi ja prosessin parannus eivät ole mahdollisia samalla tavalla kuin muodollisissa katselmointimenetelmissä.

4 Katselmointitutkimus

Faganin artikkelin [Fag76] julkaisun jälkeen on julkaistu useita katselmointien ja erityisesti tarkastusten hyötyjä tutkivia artikkeleita [Mye78, ABL89, Wel93, Ber98, BoB01]. Katselmoinneista julkaistujen artikkeleiden kirjoittajien konsensus on, että katselmoinnit yleisesti ja tarkastukset erityisesti parantavat ohjelmiston laatua. Katselmointi on kustannustehokas menetelmä, sillä virheiden korjaaminen on kannattavaa mahdollisimman aikaisessa projektin vaiheessa. Katselmointeja on sovellettu onnistuneesti muun muassa avoimen lähdekoodin projekteissa [Lus04], yliopistoissa [BiL89] sekä pienten ja suurten yritysten projekteissa [HTH05, Rus91].

Myönteisen konsensuksen muodostuttua luonnollinen suuntaus jatkotutkimukselle on ollut katselmointiprosessin tehokkuuden parantaminen [Kol06]. Viimeisen kahdenkymmenen vuoden aikana katselmointeja käsittelevän tutkimuksen määrä on kasvanut jatkuvasti. Kollanus [Kol06] luokittelee vuosina 1990 – 2004 katselmoinneista kirjoitetut artikkelit viiteen eri näkökulmaan artikkeleiden tutkimusongelmien mukaisesti: tekninen näkökulma, johdon näkökulma, työkalunäkökulma, virheiden määrän arviointi sekä kokonaisvaltainen katsaus alaan. Tekninen näkökulma jakautuu neljään teemaan: lukutekniikat, tehokkuustekijät katselmoinneissa, prosessit sekä sisältöön liittyvät erityiskysymykset. Johdon näkökulman osana ovat katselmoinnin merkitys organisaatiolle sekä muut johdon

näkökulmaan kuuluvat aiheet, kuten kustannuslaskenta ja katselmointien aikataulutus.

Tässä luvussa tarkastettava tutkimus rajataan tekniseen näkökulmaan. Tarkastettavan tutkimuksen rajaus on tehty kyseisen tutkielman pääteeman sekä tutkielman osana tehtävän tapaustutkimuksen tavoitteiden perusteella. Koska pääpaino on katselmointien käytännönläheisessä toteutuksessa pienessä ohjelmistoyrityksessä, kyseiset rajauksen ulkopuolelle jätetyt neljä näkökulmaa eivät liity suoraan kyseiseen tutkielmaan. Tätä tutkielmaa varten laajennetaan Kollanuksen [Kol06] katselmointitutkimuksen luokittelua pienten ohjelmistoyritysten näkökulmalla sekä kasvatetaan tarkasteltavien artikkeleiden aikaväliä vuosille 1990 – 2008.

4.1 Lukutekniikat

Lukutekniikoilla tarkoitetaan niitä menetelmiä ja prosesseja, joiden avulla pyritään maksimoimaan katselmoijien kykyä löytää virheitä tai muita puutteita katselmoitavasta tuotoksesta. Yleisimmät käytössä olevat katselmointien lukutekniikat ovat *ad hoc* -lukeminen (*ad-hoc reading*) sekä tarkistuslistojen käyttö (*checklist-based reading*) [PVB95], mutta niiden lisäksi on kehitetty lukuisia systemaattisempia tekniikoita. Tässä kohdassa esitellään pääosin vuosina 1990 – 2008 tehtyä lukutekniikoita käsittelevää katselmointitutkimusta.

4.1.1 Ad hoc-lukeminen

Ad hoc -lukemisella tarkoitetaan lukutekniikkaa, jossa kaikilla osallistujilla on samat vastuualueet ja tuotoksen lukemistekniikka jätetään määrittelemättä [PoV97, LEH01]. Tässä lukutekniikassa osallistujille ei anneta erityistä ohjeistusta tehokkaan lukemisen varmistamiseksi. Katselmoitava tuotos jaetaan kaikille osallistujille, ja kunkin osallistujan vastuulla on päättää, millä lukumenetelmällä tuotoksesta saa parhaan mahdollisen ymmärryksen ja millaisia virheitä kunkin osallistujan kannattaa tuotoksesta etsiä [LaD00].

Ad hoc -lukeminen on edullista, sillä se ei vaadi työntekijöiden koulutusta tai apumateriaalin, kuten tarkistuslistojen, ylläpitoa. Lukemisen tehokkuus määräytyy osallistujien taitojen, tietojen sekä aikaisemman katselmointikokemuksen perusteella [LaD00]. Tämän johdosta ad hoc -lukemisen tehokkuuden vertaaminen

muiden lukutekniikoiden tehokkuuteen on vaikeaa.

4.1.2 Tarkistuslistat

Tarkistuslistat ovat ad hoc -lukemista systemaattisempi menetelmä, jossa osallistujien lukemista ohjataan etukäteen suunniteltujen kysymysten avulla. Tarkistuslistat sisältävät joukon kysymyksiä, joihin osallistujien on vastattava katselmoimissaan tuotosta. Perinteisessä Faganin esittämässä mallissa kaikki osallistujat käyttävät samoja tarkistuslistoja [Fag76], mitä on myöhemmin kritisoitu muun muassa Parnasin ja Weissin [PaW87] sekä Porterin ja kumppaneiden toimesta [PVB95]. Myöhemmin Fagan itse suositteli niiden käytöstä luopumista kehittyneiden automaattisten tarkistustyökalujen vuoksi [Fag02].

Tarkistuslistojen tarkoituksena on kiinnittää tarkastajien huomio sellaisiin virhetyyppihin, joiden löytäminen on kulloinkin suoritettavan tarkastuksen tavoitteena [ABL89]. Listojen sisältö voi muuttua tarkastuksen tavoitteen mukaan. Mikäli päätavoitteena on esimerkiksi varmistaa tuotoksen suorituskyky, tarkastajien huomio ohjataan suorituskykykriittisiin ongelmiin. Kysymysten valinnassa tulee ottaa huomioon sen yrityksen kulttuuri ja työtavat, jossa kysymyksiä tullaan käyttämään [Kok06]. Kokkonien mukaan tarkistuslistat tulee luoda yhteistyössä yrityksen omien, kokeneiden työntekijöiden kanssa, eikä kirjallisuudessa julkaistuja tai konsulttien tekemiä valmiita tarkistuslistoja tule käyttää sellaisenaan. Sen jälkeen kun kysymykset on luotu, listojen sisältöä päivitetään aina, kun tarkastuksissa tulee esille uusia virhetyyppisiä tai kun vanhat virhetyyppit eivät ole enää ajankohtaisia [Fag76, Wie01, Bry99].

Bryczynski esittää yhteenvedon 127:sta tarkistuslistasta 24:stä eri lähteestä artikkelissaan "A Survey of Software Inspection Checklists" [Bry99]. Suurin osa mainituista tarkistuslistoista on suunniteltu kooditarkastuksia varten. Tyypillinen kooditarkastuksen tarkistuslista sisältää ohjelmointikielikohtaisia kysymyksiä, kuten C-kieleen liittyvä "onko sizeof-operaattorin argumentti oikeaa tyyppiä?", tai kieliriippumattomia kysymyksiä, kuten "voiko jako-operaation jakaja olla nolla?". Jotkut tarkistuslistat sisältävät myös koodin tyylin liittyviä kysymyksiä, kuten "käytetäänkö vakioden nimissä vain isoja kirjaimia?" [DCB03].

Laitenbergerin ja DeBaudin kirjallisuudesta keräämän kritiikin mukaan tarkistuslistojen kysymykset on usein esitetty liian yleisellä tasolla, konkreettinen ohjeistus tarkistuslistojen käytöstä on usein puutteellista ja tarkistuslistojen kysy-

mysten avulla pyritään usein löytämään vain tiettyntyyppisiä virheitä, jolloin muut virheet jäävät löytymättä [LaD00]. Koska tarkistuslistat perustuvat aikaisemmin havaittuihin virhetyyppeihin [Fag76, Wie01, Bry99], niin täysin uudentyypiset virheet saattavat jäädä tarkistuslistoja käytettäessä huomaamatta [LaD00]. Eräässä tarkastuksen lukutekniikoiden tehokkuutta tutkivassa kokeessa tarkistuslistapohjaisen lukemisen avulla pystyttiin löytämään virheitä yhtä tehokkaasti kuin ad hoc -lukemisen avulla [PVB95]. Tarkistuslistat olivat myös abstraktio- ja käytötapausmenetelmiä tehokkaampi tarkastuskeino olio-ohjelmien lähdekoodin tarkastuksessa [DRW03].

Monet yleiset lähdekoodissa esiintyvät ongelmat eivät ole löydettävissä automaattisesti. Esimerkiksi olioiden säieturvallinen käsittely, tietyn kirjaston rajapinnan käyttö dokumentoidulla tavalla tai konventiot tiettyjen tehtävien toteutukseen ovat asioita, jotka on tarkastettava manuaalisesti. Kokeneet kehittäjät osavat usein tunnistaa nämä ongelmakohdat, jolloin dokumentoimalla ohjeet näiden ongelmakohdientunnistamiseksi yksittäisen kehittäjän kokemus muuttuu kokemukseräiseksi tiedoksi [Kok06]. Kun dokumentoidut ohjeet tulevat osaksi tarkistuslistaa, tarkistuslistoista tulee koko yrityksen kokemukseräisen tiedon hallinnan välineitä.

4.1.3 Skenaariopohjaiset lukutekniikat

Basilin artikkelissa [Bas97] esitellään skenaariopohjainen lukeminen (*scenario-based reading*), jonka tavoitteena on käyttää skenaarioita ohjeistamaan katselmoijia katselmointiprosessissa. Skenaariot muodostetaan valitsemalla arvot lukutekniikoiden eri ulottuvuuksista, kuten katselmoitava tuotos, lukijan rooli, lukemisen tavoitteet tai tavoiteltavat tuotoksen laatuattribuutit. Näin skenaarioita voidaan muodostaa vastaamaan eri tilanteiden tarpeisiin, ja ne toimivat yksityiskohtaisina ohjeina osallistujille katselmoinnin tavoitteista ja toteutustavoista. Kukin osallistuja voi käyttää eri skenaariota, jolloin eri osallistujat keskittyvät erityyppisten virheiden etsimiseen erilaisista näkökulmista, mikä voi tehostaa osallistujien virheidenlöytötehokkuutta ryhmänä [LaD00].

Skenaariopohjainen lukeminen on monta eri lukutekniikkaa yhdistävä yläkäsite, johon kuuluu useita konkreettisia lukutekniikoita. Ensimmäisessä vaatimusdokumenttien tarkastukseen kehitetty virhelähtöinen lukeminen (*defect-based reading*) on skenaariopohjainen lukutekniikka, jossa kukin tarkastaja keskittyy etsimään vaatimusdokumentista tiettyyn virheluokkaan kuuluvia virheitä [PVB95,

LaD00]. Tätä varten kullekin tarkastajille jaetaan lista kysymyksiä, jotka auttavat kyseisen virhetyypin virheiden löytämisessä. Kysymyslistan kattama ongelma-alue on suppeampi kuin tarkistuslistoja käytettäessä, ja kullakin tarkastajalla on käytössään eri kysymykset [PVB95]. Virhelähtöinen lukutekniikka on osoittautunut kahdessa eri kokeessa noin 35% tehokkaammaksi tarkistuslistapohjaiseen tai ad hoc -lukemiseen verrattuna [PoV94, PVB95]. Porter ja kumppanit tekivät näistä kokeista johtopäätöksen, jonka mukaan tarkastusten tehokkuutta voidaan parantaa, kun kukin tarkastaja käyttää systemaattista lähestymistapaa pienen virhejoukon etsimiseen [PVB95]. Samoja ajatuksia esittivät aikaisemmin Parnas ja Weiss [PaW87].

Faganin alkuperäinen kuvaus tarkastusprosessista sisälsi perspektiiviaspektin, jonka mukaan kunkin tarkastajan on tarkastettava tuotosta omasta, erillisestä perspektiivistään, kuten ohjelmistoarkkitehdin tai testaajan näkökulmasta. Perspektiivipohjainen lukutekniikka (*perspective-based reading*) pohjautuu samoille ajatuksille, ja käyttää hyväksi skenaariopohjaisessa lukemisessa esitettyjä ideoita [LaD97]. Kullekin tarkastajalle luodaan yksilöllinen, toistettavissa oleva skenaario, joka kuvaa kaikki toiminnot ja keskittymistä vaativat asiat, jotka ovat kyseisen perspektiivin kannalta oleellisia. Esimerkiksi testaajan tehtävänä on luoda tarkastettavan koodin pohjalta testitapaukset, suorittaa ne mielessään ja tarkastaa samalla tuotosta vastaamalla testauksen kannalta oleellisiin kysymyksiin.

Laitenbergerin ja kumppaneiden suorittamassa kokeessa perspektiivipohjainen lukutekniikka osoittautui hieman tehokkaammaksi kuin tarkistuslistapohjainen lukutekniikka [LEH01]. Tarkastusten kustannukset olivat sen sijaan huomattavasti alhaisemmat kuin tarkistuslistapohjaista lukutekniikkaa käytettäessä. Toisaalta Hallingin ja kumppaneiden [HBG01] suorittamassa kahdessa kokeessa tarkistuslistapohjaisen lukemisen todettiin olevan perspektiivipohjaista lukemista tehokkaampi. Nämä kaksi tutkimusta suoritettiin erilaisissa tutkimusympäristöissä. Laitenbergerin ja kumppaneiden tutkimuksessa katselmoinnit tapahtuivat suuressa yrityksessä ja tutkimukseen osallistuneet ohjelmistokehittäjät olivat kokeneita ammattilaisia [LEH01]. Hallingin ja kumppaneiden tutkimus suoritettiin yliopistossa ja tutkimukseen osallistui opiskelijoita, joiden osaamisen taso vaihteli merkittävästi [HBG01]. Tutkimukseen osallistuneiden kehittäjien kokemuksella ja osaamisen tasolla voi olla vaikutuksia siihen, kumpi lukutekniikka soveltuu paremmin katselmoointeihin.

Erilaisten skenaariopohjaisten lukutekniikoiden tehokkuuseroja tarkistuslistapoh-

jaiseen lukemiseen tai ad hoc -lukemiseen on tutkittu, ja vaikka tutkimustulokset ovat lupaavia, ne eivät ole yhdenmukaisia ja sisältävät myös ristiriitaisia tuloksia [SBK98, PFV04]. Skenaariopohjaisia lukutekniikoita ei ole juuri verrattu toisiinsa [Kol06]. Myöskään yksittäisten katselmoijien käyttämien lukutekniikoiden välisiä tehokkuuseroja ei ole juuri tutkittu [UNM06].

4.1.4 Yksilölliset erot lukutekniikoissa

Lukutekniikoiden välisistä tehokkuuseroista huolimatta katselmoijien yksilöllinen suoritus vaikuttaa lukemisen tehokkuuteen enemmän kuin käytetty lukutekniikka [UNM06]. Yksittäisten katselmoijien väliset tehokkuuserot ovat eri lukutekniikoita tarkastelevissa tutkimuksissa olleet usein suurempia kuin lukutekniikoiden väliset erot.

Uwano ja kumppanit suorittivat kokeen, jossa he pyrkivät selittämään katselmoijien välisiä tehokkuuseroja tarkkailemalla heidän silmiensä liikkeitä lähdekoodin katselmoinnin aikana. He onnistuivat löytämään eri koehenkilöiden välillä toistuvan silmäliikemallin, jossa katselmoija "skannaa" (*scan*) lähdekoodin kokonaisuudessaan, ennen kuin hän tutkii tarkemmin koodin eri osia. Tämä skannausvaihe kesti keskimäärin 30% koko katselmoinnin ajasta, ja sen aikana osallistujat lukivat keskimäärin 72,8% katselmoitavasta lähdekoodista. Mitä enemmän aikaa kului skannausvaiheessa, sitä nopeammin katselmoijat pystyivät löytämään virheitä koodista katselmoinnin myöhemmissä vaiheissa. Tutkimuksen mukaan skannausvaiheen pituudella ja lukemisen tehokkuudella näyttää olevan suora riippuvuus.

4.2 Kokoukset

Kokoukset ovat perinteisesti olleet kiinteä osa tarkastuksia ja ryhmäkatselmointeja [Fag76, Wie01]. Faganin alkuperäisessä tarkastusprosessin kuvauksessa virheiden etsiminen tapahtuu pääosin tarkastuskokouksen aikana [Fag76, Fag86]. Gilbin tarkastusprosessissa kokouksen ensisijaisena tavoitteena on kirjata valmisteluvaiheessa löytyneet virheet, mutta kokouksessa pyritään samalla löytämään uusia virheitä [GiG93].

Kokousten kustannukset ovat suuria, sillä ne sitovat monen ihmisen työpanoksen. Mitä enemmän osallistujia kokoukseen osallistuu, sitä vaikeampi on sovitt-

taa yhteen monen osallistujan aikatauluja. Tästä aiheutuu ajanhukkaa, joka on ollut joissakin tutkimuksissa jopa 20% projektin vaatimusmäärittely- ja suunnitteluvaiheisiin kuluneesta ajasta [LGV93]. Myös kokousten aikana syntyy ajanhukkaa, koska yleensä vain kaksi henkilöä voi osallistua keskusteluun samanaikaisesti, jolloin muut osallistujat joutuvat odottamaan vuoroaan. Muita lisäkustannuksia syntyy virheiden kirjaamisesta kahdesti — ensin valmisteluvaiheessa ja myöhemmin tarkastuskokouksen aikana [RDN04]. Tästä voi aiheutua tietohävikkiä, koska osallistujat eivät ole välttämättä halukkaita tuomaan julkisesti esille huomautuksia, joista he eivät ole täysin varmoja. Kokouksen aikana joitakin esille tuotuja virheitä voidaan luokitella virheellisesti vääriksi, jolloin tuotoksen tekijä ei saa tietoonsa kaikkia virheitä ja virheet pääsevät läpi tarkastusprosessista ohjelmistotuotantoprosessin myöhempisiin vaiheisiin [LGV93, RDN04].

Votta kyseenalaisti ensimmäisenä kokousten tarpeellisuuden [LGV93]. Hänen mukaansa kokousten vaikutusta koko projektin kustannuksiin ei ole otettu huomioon katselmointeja tarkastelevassa kirjallisuudessa. Myös synergiaedut, joilla kokousten tarpeellisuutta perustellaan, ovat hänen mukaansa pieniä tai olemattomia. Votta ja kumppanit tutkivat kokousten tarpeellisuutta kokeessa, jossa verrattiin kolmea tapaa järjestää katselmointeja [MPS96]. Tutkimuksessa ensimmäinen ryhmä käytti Faganin menetelmää, jossa valmisteluvaiheen ensisijaisena tarkoituksena oli tuotokseen tutustuminen, ja kokouksen ensisijaisena tarkoituksena virheiden löytäminen [MPS96]. Toinen ryhmä käytti Gilbin menetelmää, jossa virheitä pyrittiin löytämään mahdollisimman paljon valmisteluvaiheessa, ja kokouksen ensisijaisena tarkoituksena oli virheiden kirjaus. Kolmannen ryhmän jäsenet tarkastivat tuotoksen itsenäisesti kaksi kertaa, eivätkä pitäneet kokousta lainkaan. Kokeen tulosten mukaan kolmas ryhmä löysi enemmän virheitä samassa ajassa kuin ensimmäinen tai toinen ryhmä. Samansuuntaisia tuloksia on saatu muiden tutkijoiden suorittamista kokeista [PoJ97, MWR98].

Synergia on tärkein syy kokousten järjestämiselle [LGV93]. Katselmointien yhteydessä synergialla tarkoitetaan tilannetta, jossa kokoukseen osallistuva ryhmä saa aikaan parempia tuloksia kuin jos saman ryhmän jäsenet tekisivät työnsä yksin. Vahvasta synergiasta kirjoittaessaan Fagan kuvasi tätä tilannetta haamutarkastajan käsitteellä (*phantom inspector*), joka ilmeni tarkastusryhmän keskuudessa olevana tunteena, että jokin ulkopuolinen voima paransi koko ryhmän työskentelytehokkuutta [Fag86].

Sauer ja kumppanit tutkivat kokousten tehokkuutta käyttäytymistieteellisen teo-

rian pohjalta [SjL00]. Heidän mukaansa koko ryhmän tehokkuus määräytyy pääosin yksittäisten jäsenten asiantuntemuksen perusteella, eikä ryhmäaktiiviteetin seurauksena synny synergiaa. Ryhmät eivät siis pysty itse tekemään merkittäviä määriä uusia löydöksiä, vaan kokousten aikana ryhmä käsittelee kunkin yksilön itsenäisen työskentelyvaiheen aikana löytämiä asioita. Ryhmäaktiiviteettien tärkein tehtävä on synergian sijasta yhteinen päätöksenteko — tarkastuskokousten tapauksessa esimerkiksi päätettäessä löydöksen luokittelusta virheeksi.

Ryhmät pystyvät eliminoimaan tehokkaasti vääriä virheitä (*false positives*) sekä duplikaattivirheitä. Mikäli väärrien virheiden tai duplikaattivirheiden määrä on pieni tai niiden eliminoiminen on yksinkertaista, yksittäisten tarkastajien käyttö on kannattavampaa kuin kokousten järjestäminen [PJS97]. Votton mukaan kokous ei ole lainkaan tehokas menetelmä väärrien virheiden eliminoimiseksi [LGV93].

Kokouksen tehokkuutta kyseenalaistavat teoreettiset ajatukset ovat saaneet tukea empiiriseltä tutkimukselta, jonka tulosten mukaan kokousten eliminoiminen voi olla perusteltua [LGV93, PoJ97, MWR98]. Esimerkiksi Votton tutkimuksessa [LGV93] kokouksessa löydettyjen virheiden osuus kaikista virheistä oli vain 4%, kun taas Gilbin mukaan [GiG93] se on noin 15%. Kokousten tehokkuudesta on saatu myös päinvastaisia tuloksia. Esimerkiksi erään suuren ohjelmistoprojektin tarkastuskokouksissa löydettiin 33% kaikista tarkastuksissa löytyneistä virheistä [PST97].

4.3 Prosessit

Luvuissa 2 ja 3 esitellyille katselmointimenetelmille on kehitetty joitakin variaatioita, joilla pyritään tehostamaan katselmoinnin tehokkuutta. Parnasin ja Weisin mukaan perinteiset katselmointiprosessit ovat ongelmallisia, sillä katselmoijien tehtävät eivät ole tarkasti rajattuja [PaW87]. Selkeän tehtävärajauksen puute johtaa joko katselmoijien ylikuormittumiseen heidän yrittäessään ymmärtää koko tarkasteltavaa tuotosta tai liian pinnalliseen tarkastukseen. Myös kokoukset ovat heidän mukaansa tehottomia, mikäli niihin osallistuu paljon osallistujia.

Parnas ja Weiss esittivät suunnitteludokumenttien katselmointiin tarkoitettua aktiivisen suunnittelukatselmointimenetelmän (*active design review*), jossa katselmointiprosessi jaetaan peräkkäisiin vaiheisiin [PaW87]. Kussakin vaiheessa tuotoksesta pyritään löytämään tietyn tyyppisiä virheitä, jolloin voidaan käyttää eri ihmisten ammattitaitoa tarpeen mukaan. Tuotos jaetaan osiin ja yhdessä katselmoin-

tiprosessin vaiheessa tarkastetaan yksi tuotoksen osa kerrallaan. Ennen katselmoinnin alkamista tuotoksen tekijä tuottaa kullekin katselmoijalle joukon tuotokseen liittyviä kysymyksiä, joihin katselmoijien on pystyttävä vastaamaan. Kokoukset korvataan suppeammilla tapaamisilla, joihin osallistuvat tuotoksen tekijä sekä kyseiseen katselmointiprosessin vaiheeseen osallistuvat katselmoijat.

Knight ja Myers kehittivät aktiivisen suunnittelukatselmointimenetelmän pohjalta vaiheistetun tarkastuksen (*phased inspection*), joka laajensi alkuperäistä menetelmää muiden kuin suunnitelmadokumenttien kanssa käytettäväksi [KnM93]. Vaiheistetussa tarkastuksessa kussakin vaiheessa pyritään tarkastamaan koko tuotos tietyn tyyppisten virheiden varalta.

Thelin ja kumppanit [TPR04] suosittelivat erillistä esitarkastusvaihetta, jossa tuotoksesta tarkastetaan noin 20%-30%:n otos. Otoksen perusteella tuotoksen osien sisältämien virheiden määrät arvioidaan ja osat lajitellaan virhemäärien perusteella. Varsinaisen tarkastusvaiheen aikana tarkastetaan vain eniten arvioituja virheitä sisältävät tuotoksen osat. Gilbin mukaan otosten käyttö on perusteltua suurten tarkastettavien tuotosten kohdalla [Gil00]. Mikäli otoksen perusteella koko tuotoksen laatu on hyväksyttävällä tasolla, tuotoksen tarkastaminen voidaan jättää suorittamatta. Myös kokouksessa tapahtuva virheiden etsiminen voidaan jättää pois, mikäli tuotoksen arvioitu laatu on riittävän korkea.

Eräässä empiirisessä tutkimuksessa tutkittiin otoksen esitarkastuksen vaikutusta koko katselmointiprosessin tehokkuuteen, kun katselmointiin käytettävissä olevaa aikaa rajoitettiin [KKM96]. Tutkimuksessa ohjelmiston lähdekoodi jaettiin osiin, ja kustakin osasta esitarkastettiin yksi otos. Esitarkastuksessa löytyneiden virheiden määrien sekä osien koon perusteella laskettiin kunkin osan tarkastukseen käytettävä aika. Tutkimuksessa otosten käytön avulla löydettiin 43% enemmän virheitä kuin ilman otosten käyttöä.

5 Tutkimusympäristö

Tässä kohdassa kuvataan ympäristöä, jossa tutkimus suoritettiin. Tutkimusympäristön kuvaaminen on tärkeää tapaustutkimuksen kannalta, sillä sen avulla tutkimuksen tulokset voidaan asettaa oikeaan kontekstiin. Tutkimusympäristö kuvataan yrityksen, ohjelmistotuotteen sekä yrityksen laadunvarmistusprosessin näkökulmasta. Laadunvarmistusprosessin kuvauksessa kuvataan prosessi, joka

on ollut käytössä yrityksessä ennen kyseisen tapaustutkimuksen aloittamista.

5.1 Pienet ohjelmistoyritykset

Pienet ohjelmistoyritykset muodostavat monessa maassa valtaosan kaikista ohjelmistoyrityksistä [RiG07]. Esimerkiksi Kiinassa, USA:ssa, Suomessa, Irlannissa ja Unkarissa alle 50 työntekijän ohjelmistoyritysten osuus on jopa 85%. Pienten ohjelmistoyritysten suuresta osuudesta huolimatta ohjelmistoteknisiä prosesseja ja ratkaisuja kuten katselmointeja käsittelevä tieteellinen kirjallisuus keskittyy tutkimaan usein huomattavasti suurempien yritysten ongelmia ja etsimään niihin ratkaisuja suurten yritysten näkökulmasta. Kirjallisuudessa esitetyt prosessit ja menetelmät nähdään usein liian työläinä ja kalliina pienissä yrityksissä sovellettaviksi [RiG07, TIH98].

Vaikka pienten ohjelmistoyritysten haasteet ovat pitkältä samoja kuin suurissakin yrityksissä, kuten prosessien kehittäminen ja tuotteen laadunvarmistus, mahdolliset ratkaisut näihin ongelmiin eivät ole samoja erilaisista liiketoimintamalleista, markkina-alueesta, yritysten koosta ja resurssien rajallisuuden johdosta [RiG07]. Pienet yritykset eivät ole suurten yritysten pienoismalleja — niillä on usein matala organisaation hierarkia, johtajat osallistuvat usein toteuttavaan työhön ja yrityksissä vallitsee vahva innovaatioita suosiva yrittäjähenki. Näissä yrityksissä ohjelmistotuotteiden kehitys on pääsijalla, eikä yrityksellä ole resursseja monimutkaisten aputoimintojen tai työvälineiden kehittämiseen tuotekehityksen rinnalla.

Pienet yritykset eroavat isoista myös yrityskulttuuriltaan [KeC99]. Usein pienet ohjelmistoyritykset ovat syntyneet yhden tai muutaman henkilön työn seurauksena, jolloin perustajajäsenet ovat tottuneet osallistumaan aktiivisesti ohjelmistokehityksen kaikkiin vaiheisiin. Tämä kulttuuri jää usein elämään, jolloin yrityksen kaikki työntekijät olettavat pystyvänsä vaikuttamaan yrityksessä tehtäviin päätöksiin. Tällaisessa yrityskulttuurissa katselmointien tai minkä tahansa muun prosessin käyttöönotto vaatii tiivistä yhteistyötä työntekijöiden kanssa. Muussa tapauksessa prosessi voidaan nähdä esteenä yrityksessä tapahtuvalle luovuudelle ja innovaatioille.

Faganin mukaan [Fag02] yleinen ongelma pienissä ohjelmistoyrityksissä on julkaistuista tuoteversioista löytyneiden virheiden suuri määrä, joita joudutaan etsimään ja korjaamaan silloin, kun seuraava tuoteversio on jo kehitteillä. Tämä ai-

heuttaa viivästyksiä seuraavan version kehitykseen, mikä voi johtaa siihen, että joitakin ominaisuuksia jätetään toteuttamatta. Kun julkaistujen, tukea vaativien tuoteversioiden määrä sekä ohjelmistotuotteen koko kasvaa, ongelma pahenee entisestään. Mikäli tuotejulkaisujen välillä ei tehdä parannuksia ohjelmistotuotantoprosessiin, yrityksen epäonnistumisen riski kasvaa suureksi. Fagan kutsui tätä tilannetta "brute force" -kehitykseksi [Fag02].

Yksi ratkaisu pienten yritysten yleiseen ongelmaan — pulaan saatavilla olevasta työvoimasta — on katselmointiin osallistuvien henkilöiden määrän pienentäminen. Sauerin ja kumppaneiden mukaan [SJL00] katselmoinnissa, jossa osallistujilla on vahva katselmoitavaan tuotokseen liittyvä asiantuntemus, ryhmän koon kasvattaminen yli kahden ei tuota merkittäviä etuja. Kahden ihmisen suorittamaa katselmointia on tutkittu erässä yliopiston opiskelijoilla suoritetussa kokeessa, ja tulokset ovat olleet positiivisia [BiL89]. Katselmointien avulla pystyttiin löytämään virheitä nopeammin ja ne paransivat kokeeseen osallistuneiden kehittäjien tuottavuutta.

Oulun yliopistossa toimii tarkastusmenetelmiä ja -työkaluja kehittävä ja tutkiva i3GO-tutkimusryhmä [HTV04], joka on tuottanut runsaasti tarkastuksia käsitteleviä artikkeleita. Pienten ohjelmistoyritysten suuri osuus kaikista ohjelmistoyrityksistä Suomessa näkyy myös näiden artikkeleiden aihepiireissä. Harjumaan ja kumppanit [HTH05] tutkivat pienehköjen, pääosin 18 – 75 ihmistä työllistävien ohjelmistoyritysten käyttämiä katselmointimenetelmiä. Heidän tutkimuksen mukaan yritysten käytössä olleet katselmointimenetelmät olivat epämuodollisempia kuin kirjallisuudessa esitetyt mallit. Muodollinen tarkastusprosessi oli kaikista katselmointimenetelmistä vähiten tunnettu.

Harjumaan ja kumppaneiden tutkimuksen mukaan tutkituissa yrityksissä käytössä olleet katselmoinnit jakautuivat pääosin kahteen vaiheeseen: osallistujien itseenäiseen valmistautumiseen sekä löydettyjen virheiden korjausvaiheeseen [HTH05]. Tuotoksen tekijä aloitti katselmointiprosessin, ja projektipäällikkö vastasi katselmoinnin järjestämisestä. Käytetyin työkalu katselmointien järjestämisessä oli sähköposti, ja tarkistuslistoja käytettiin harvoin. Katselmointikokouksia järjestettiin lähes aina noin puolessa tutkituista yrityksistä. Yhdessäkään tutkituista yrityksistä mittausta tai sen mahdollistamaa prosessinparannusta ei pidetty tärkeänä tavoitteena, ja katselmoinneista kerättiin hyvin vähän mittoja. Suurimmat esteet katselmointien järjestämiselle näissä yrityksissä olivat ajan sekä työvoiman puute ja katselmointiprosessin työläys.

Resurssien rajallisuuden vuoksi prosessinparannushankkeet nähdään pienissä yrityksissä usein kalliina ja niiden toteutus nähdään siten riskinä aktiivisessa kehityksessä oleville projekteille [HTV04]. Laajojen prosessinparannushankkeiden toteutus ei siksi usein onnistu. Harjumaa ja kumppanit [HTV04] näkevät tarkastuksen sopivana alkuaskeleena pienten yritysten prosessinparannukselle. Tarkastus on selkeästi rajattu menetelmä, jolla on korkea sijoitetun pääoman tuotto ja jonka hyödyt ovat helposti mitattavissa. Tarkastuksella on myös positiivisia sivuvaikutuksia, kuten aktiivisempi kommunikointi sekä tiedon välitys yrityksen työntekijöiden kesken. Harjumaa ja kumppanit ovat kehittäneet katselmointiprosessin arvioimiseksi i3GO-kypsyysmallin sekä joukon siihen liittyviä toimintamalleja katselmointiprosessin parannukseen. Mallia on sovellettu viidessä pienessä ohjelmistoyrityksessä ja se on osoittautunut toimivaksi ja hyödylliseksi.

Katselmoiteja voidaan soveltaa myös erityisesti pienille projektiryhmille suunniteltujen ketterien prosessimallien mukaisissa projekteissa [HeI06, Amb08]. Hedberg ja Iisakka [HeI06] tutkivat ryhmäkatselmointien käyttöä eräässä ohjelmistoprojektissa, joka noudatti ketterää Mobile-D -menetelmää. Projektissa katselmoiteja suoritettiin positiivisin tuloksin hyväksymistestien laadunvarmistuksessa. Ohjelmiston koodi tuotettiin osittain pariohjelmoinnin avulla, joka vaikutti myönteisesti koodin laatuun. Pariohjelmointia katselmointien näkökulmasta on esitelty tarkemmin kohdassa 3.3. Tutkimuksen mukaan katselmoineista on hyötyä myös ketteriä prosessimalleja noudattavissa projekteissa.

5.2 Yrityksen esittely

Tapaustutkimuksen kohteeksi otettiin pääkaupunkiseudulla sijaitseva alle kymmenen hengen ohjelmistoyritys. Yrityksen työntekijät työskentelivät pääosin yrityksen toimipisteessä, mutta myös etätyöskentely oli sallittua. Kyseisessä tutkielmassa yritykseen viitataan termillä esimerkkiyritys.

Esimerkkiyrityksen liiketoiminta koostui sekä omasta tuotekehityksestä että toimitusprojekteista asiakkaille. Yrityksen toteuttamien ohjelmistojen keskimääräinen koko oli alle 100 000 koodiriviä. Yhden ohjelmistotuotteen tuottamiseen osallistui 1 – 5 työntekijää.

Esimerkkiyrityksen yrityskulttuuri oli tyypillinen pienen ohjelmistoyrityksen kulttuuri, jota on esitelty kohdassa 5.1. Yrityksen organisaatiohierarkia oli matala ja rooli- tai tehtäväjako ei ollut juuri käytössä. Kaikki työntekijät olivat tottuneet

osallistumaan aktiivisesti ohjelmistokehityksen kaikkiin vaiheisiin. Työntekijöillä oli suhteellisen vapaat kädet tehdä omaa työtään koskevia päätöksiä, esimerkiksi yrityksen ohjelmistotuotantoprosessista tai lähdekoodin kirjoituksessa käytetyistä konventioista ei ollut olemassa kirjallisia kuvauksia.

Työntekijöiden välinen epämuodollinen kommunikaatio oli tärkein väline yrityksen sisäisessä tiedonvälityksessä. Muodolliset kokoukset tai työmenetelmien muodollinen dokumentointi eivät kuuluneet esimerkkiyrityksen yrityskulttuuriin. Työntekijät kommunikoivat epämuodollisesti keskenään kasvokkain yrityksen toimitiloissa sekä yrityksen sisäisessä reaaliaikaisessa viestintäjärjestelmässä (chat). Yrityksen chat-järjestelmä oli etätyöskentelyn kriittinen osa.

Yrityksen johto vastasi pääosin ohjelmistojen vaatimusmäärittelystä, ohjelmistojen julkaisuaikatauluista sekä yhteyksistä yrityksen asiakkaisiin. Johto ei juuri puuttunut työntekijöiden työmenetelmiin. Yrityksen laajuiset päätökset esimerkiksi työkalujen tai työmenetelmien käytöstä syntyivät pääosin työntekijöiden välisinä demokraattisina päätöksinä. Eri työmenetelmien vaikutusta työn tuottavuuteen tai ohjelmiston laatuun ei mitattu, joten käytettävät työmenetelmät tulivat valituksi pääosin työntekijöiden omien, subjektiivisten näkemysten perusteella. Yrityksellä ei ollut erikseen määriteltyä laatustrategiaa tai -suunnitelmaa, vaan kukin työntekijä pyrki itsenäisesti parhaaseen mahdolliseen lopputulokseen.

5.3 Nykyinen laadunvarmistusprosessi

Tapaustutkimus suoritettiin esimerkkiyrityksen erään ohjelmistoprojektin yhteydessä. Projektiin kuului ohjelmistotuotteen kehitys- ja ylläpitotehtäviä. Ohjelmiston koodin koko oli alle 100 000 koodiriviä. Tutkielmassa ohjelmistoon viitataan nimellä Beta.

Beta-ohjelmiston kehitykseen ja ylläpitoon osallistui useampia esimerkkiyrityksen työntekijöitä. Työntekijöiden välillä ei juuri ollut roolijakoa kohdassa 5.2 esitellyn yrityskulttuurin mukaisesti. Kukin projektiin osallistunut työntekijä vastasi yhden tai useamman ohjelmiston osan kehityksestä ja ylläpidosta. Ohjelmiston eri osat integroitiin kokonaiseksi ohjelmistoksi päivittäin.

Beta-ohjelmiston kehityksessä käytetty ohjelmistotuotantoprosessi oli hyvin epämuodollinen ja iteratiivinen. Ohjelmistosta julkaistiin vuoden aikana useita uusia versioita. Uusien versioiden julkaisuväli vaihteli muutamasta viikosta muuta-

miin kuukausiin. Uusiin versioihin toteutettiin uusia ominaisuuksia sekä korjattiin vanhojen versioiden virheitä.

Ohjelmiston kehitykseen liittyvää työtä hallittiin yrityksen tietojärjestelmässä, jossa kukin erillinen työtehtävä muodosti yhdelle työntekijälle osoitetun työmääräimen. Tietojärjestelmä toimi priorisoituna listana, jossa listattiin projektiin kuuluvat työtehtävät. Kunkin työmääräimen sisältönä saattoi olla uuden ominaisuuden toteutus tai ohjelmiston häiriön taustalla olevan virheen selvitys ja korjaus. Työmääräimeen voitiin myös tallentaa kyseisen tehtävän suoritukseen kulunut työaika, mutta tätä ominaisuutta ei käytetty.

Ohjelmistotuotantoprosessista ei ollut olemassa kirjallista kuvausta, vaan ohjelmiston kehitys tapahtui epämuodollisesti ohjelmiston julkaisujen välillä. Pääosa työajasta kului ohjelmiston uusien ominaisuuksien toteutukseen, olemassa olevan koodin refaktorointiin sekä ohjelmiston häiriöiden syynä olevien virheiden etsimiseen ja korjaamiseen. Ohjelmiston suunnittelu tapahtui pääosin pienissä osissa ennen kunkin uuden ominaisuuden toteutusta. Määrittely- ja suunnittelu-dokumentteja ei juuri käytetty.

Beta-ohjelmiston kehityksessä käytettiin yksikkötestausta, mutta sen käyttö oli suppeaa. Yksikkötestien lausekattavuus oli alle 30%. Pääosa testauksesta oli järjestelmätestausta, jossa koko ohjelmistoa testattiin käyttäjän näkökulmasta. Kunkin uuden ominaisuuden toteutuksen jälkeen ohjelmistoa järjestelmätestattiin keskittyen uuden ominaisuuden toimintoihin.

Kaikki ohjelmistoon tehdyt koodimuutokset tallennettiin yrityksen versionhallintajärjestelmään. Ohjelmiston kehitys tapahtui pääosin yhdessä versionhallintajärjestelmän haarassa, johon tallennettiin useita muutoksia päivittäin. Koodimuutoksia ei tallennettu versionhallintajärjestelmään, mikäli ne aiheuttaisivat virheitä tai varoituksia ohjelmiston käynnöksen aikana.

Yrityksen käytössä oli myös staattisen analyysin työkalu, joka pystyi löytämään automaattisesti joitakin virhetyyppejä, joita ei voitu löytää käänös vaiheessa. Staattisen analyysin työkalun käyttö oli vapaaehtoista ja epäsäännöllistä. Työkalu mahdollisti myös uusien tarkastusten määrittelyn, mutta tätä ominaisuutta ei käytetty.

Yrityksen käytössä oli eräs kohdassa 3.4 esitellyn katselmointimenetelmän muoto — sähköpostikierrätys. Palvelin lähetti sähköpostiviestin jokaisesta ohjelmiston lähdekoodiin tehdystä muutoksesta jokaiselle ohjelmiston kehitykseen osal-

listuvalle työntekijälle. Viestin sisältönä oli muutosten tekijän nimi, muutettujen lähdekooditiedostojen nimet sekä kaikki lisätyt, muutetut tai poistetut koodirivit.

Sähköpostikierrätyksen ansiosta kaikki ohjelmistoon tehdyt muutokset olivat kaikkien projektiryhmän jäsenten nähtävissä. Muutosten katselmointi oli kuitenkin vapaaehtoista ja epäsäännöllistä. Mikäli koodimuutosten katselmointi tehtiin, sen suoritti lähes aina yksi työntekijä. Virheen löytänyt työntekijä saattoi kertoa siitä koodimuutoksen tekijälle, jonka vastuulla oli tehdä tarvittavat korjaukset. Korjausten oikeellisuuden tarkistus oli myös vapaaehtoista.

Sähköpostikierrätyksen aikana löytyneiden virheiden määrää ei laskettu ja virheitä ei luokiteltu. Virheiden taustalla olevia syitä tai mahdollisia keinoja virheiden ehkäisyyn jatkossa ei selvitetty. Katselmoinnin tehokkuudesta tai tuloksellisuudesta ei ollut saatavilla mitattua tietoa menetelmän epämuodollisuudesta ja epäsäännöllisyydestä johtuen. Sähköpostikierrätyksellä ei ollut myöskään merkittävää vaikutusta virheiden ennaltaehkäisyyn, sillä kierrätyksen aikana löytyneiden virheiden perusteella ei kehitetty yrityksen käytössä olevia konventioita tai työkaluja.

5.4 Yrityksen ongelmat

Esimerkkiyrityksen Beta-ohjelmistoon liittyvien ongelmien kartoitusta varten käytiin keskustelu yrityksen työntekijöiden kanssa. Keskustelun perusteella analysoitiin yrityksen tietojärjestelmään tallennettua aineistoa ohjelmistossa ilmenneistä häiriöistä. Keskustelujen perusteella selvisi, mitkä olivat ohjelmiston laatuun liittyvät ongelmat yrityksen työntekijöiden näkökulmasta. Yrityksen tietojärjestelmään tallennetun aineiston analysointi tarkensi ja laajensi tätä näkökulmaa tarjoamalla konkreettista, määrällistä tietoa tuotteen laadusta.

Esimerkkiyrityksessä koettiin, että Beta-ohjelmiston julkaisun jälkeen ilmenneiden häiriöiden määrä oli huomattava. Häiriöiden määrä ei myöskään pienentynyt ohjelmiston uusien versioiden myötä. Julkaistun ohjelmistoversion häiriöiden syynä olevien virheiden etsimiselle ja korjaamiselle löytyi harvoin tarpeeksi aikaa, koska Beta-ohjelmistoprojektin käytettävissä olevat työvoimaresurssit olivat rajallisia. Suurin osa saatavilla olevista resursseista pyrittiin käyttämään seuraavan ohjelmistoversion ominaisuuksien kehittämiseen. Koko ohjelmiston laadunvarmistus koettiin haasteelliseksi, koska uusien ominaisuuksien myötä ohjelmistoon syntyi uusia virheitä, jotka aiheuttivat uusia häiriöitä.

Häiriöiden määrän lisäksi myös häiriöiden taustalla olevien virheiden löydettävyys koettiin ongelmaksi. Käyttäjien löytämistä häiriöistä huomattava osa oli sellaisia, joiden voitiin päätellä johtuvan yhdestä tai muutamasta samantyyppisestä ohjelmistossa olevasta virheestä. Nämä virheet ilmenivät kuitenkin erilaisina häiriöinä johtuen ohjelmiston syötteiden tai käytön ajoituksen vaihteluista. Usein näiden käyttäjille näkyvien häiriöiden ja ohjelmistossa olevien virheiden välistä riippuvuutta oli mahdotonta löytää häiriöraporttien perusteella. Tämän johdosta häiriöiden taustalla olevia virheitä ei voitu korjata luotettavasti. Tämä vaikeutti ohjelmiston laadun tarkkailua, sillä Beta-ohjelmiston uuden version julkaisun yhteydessä projektiryhmä ei pystynyt listaamaan kaikkia niitä häiriöitä, joiden syinä olevat virheet oli korjattu. Korjattujen häiriöiden lista oli kuitenkin tärkeä ohjelmiston loppukäyttäjille.

Esimerkkiyrityksen ensisijaiseksi tavoitteeksi muodostui ohjelmiston lähdekoodissa olevien virheiden mahdollisimman tehokas löytäminen ja korjaaminen ennen ohjelmiston julkaisua. Katselmointi nähtiin yhtenä mahdollisena välineenä virheiden löytämisessä. Yritys tuki katselmointien käyttöönottoa, koska julkaisun jälkeen ilmenneiden häiriöiden syinä olevien virheiden etsiminen ja korjaaminen tuntui vievän huomattavasti aikaa. Yrityksen käytössä olevan ohjelmistotuotantoprosessin ainoa säännöllisesti syntyvä tuotos oli ohjelmiston lähdekoodi, joten katselmoinnit nähtiin ensisijaisesti lähdekoodin laadunvarmistuksen välineenä.

6 Tutkimuksen suunnittelu

Tutkimus toteutetaan toimintatutkimuksena, jossa tutkimuksen tekijä suunnittelee yrityksen katselmointiprosessin sekä osallistuu itse aktiivisesti katselmointeihin [Met06]. Tutkimuksen kesto tulee olemaan noin kaksi kuukautta. Tutkimuksen toteutuksen aikana otetaan käyttöön kohdassa 7.1 kuvattu katselmointiprosessi, jonka avulla katselmoidaan vähintään 5% Beta-ohjelmiston lähdekoodista. Tutkimuksen loppuvaiheessa suoritetaan kohdassa 6.3 kuvatut haastattelut, joissa haastatellaan katselmointeihin osallistuneita työntekijöitä.

Tutkimus toteutetaan hypoteesittomana tutkimuksena. Tutkimus selvittää haastattelun keinoin, kuinka hyödylliseksi pienen ohjelmistoyrityksen työntekijät kokevat lähdekoodin katselmoinnit ja mitkä katselmointiprosessiin liittyvät tekniset ja sosiaaliset seikat vaikuttavat työntekijöiden asenteisiin ja mielipiteisiin.

6.1 Tutkimuskysymys

Tutkimuskysymyksenä on, ovatko katselmoinnit tehokas keino parantaa Beta-ohjelmiston lähdekoodin laatua esimerkkiyrityksessä. Katselmointiprosessi, jolla pyritään vaikuttamaan ohjelmiston laatuun, on määritelty tarkemmin kohdassa 7.1. Beta-ohjelmiston lähdekoodin laadun parannus määritellään lähdekoodista löytyneiden ja korjattujen virheiden määränä. Tutkimuskysymykseen pyritään vastaamaan sekä kvantitatiivista että kvalitatiivista analyysiä käyttäen.

Tutkimuksen kvantitatiivinen osuus keskittyy katselmointiprosessin tehokkuusmittoihin, jotka määritellään tarkemmin kohdassa 6.2. Katselmointiprosessin tehokkuusmittojen keräys tapahtuu tutkimuksen tekijän toimesta koko tutkimuksen toteutuksen ajan. Kerätyt mitat analysoidaan ja niitä verrataan kirjallisuudessa esitettyihin arvoihin.

Tutkimuksen kvalitatiivinen osuus tutkii pienen ohjelmistoyrityksen työntekijöiden käsityksiä lähdekoodin katselmointien vaikutuksesta ohjelmiston laatuun. Haastatteleamalla työntekijöitä selvitetään myös katselmointiprosessin sopivuutta esimerkkiyrityksen yrityskulttuuriin ja yrityksen toimintatapoihin. Kvalitatiivisen osuuden aineiston hankintamenetelmänä käytetään teemahaastattelua, jossa kaikkien haastateltavien kanssa keskustellaan samoista teemoista. Haastattelumenetelmä on esitelty tarkemmin kohdassa 6.3.

Katselmointien ensisijaisena tavoitteena on ohjelmiston lähdekoodissa olevien virheiden määrän pienentäminen. Löydettyjen virheiden määrän analysointia ei voida tehdä luotettavasti pelkästään kvantitatiivisilla menetelmillä, sillä saatuja tuloksia ei voida verrata yrityksen muihin mittareihin. Yrityksellä ei ole esimerkiksi historiallista tietoa löydettyjen virheiden etsimiseen ja korjaamiseen menneestä ajasta ohjelmistotuotantoprosessin muissa vaiheissa. Ainoa saatavilla oleva vertailukohta on kirjallisuudessa esitetyt tulokset katselmointien aikana löydettyjen virheiden määrästä. Kirjallisuudessa esitetyt tulokset on kuitenkin saatu usein joko suurissa ohjelmistoyrityksissä tai yliopistoissa järjestetyissä katselmoinneissa, ei pienissä ohjelmistoyrityksissä. Kvantitatiivisen analyysin rajoitusten johdosta katselmointien vaikutusta ohjelmiston laatuun tutkitaan pääosin kvalitatiivisen analyysin avulla.

6.2 Katselmointiprosessin mittaus

Tutkimuksen aikana tullaan keräämään mittoja katselmointiprosessista, joka on määritelty kohdassa 7.1. Katselmointiprosessiin liittyvien mittojen kerääminen on edellytys käytetyn katselmointiprosessin tehokkuuden arvioinnille ja jatkokehitykselle. Katselmointiprosessin tehokkuuden seuranta otetaan osaksi kyseistä tapaustutkimusta, mutta prosessin jatkokehitys rajataan tutkimuksen ulkopuolelle.

Tutkimuksen aikana lasketaan järjestetyt katselmointikerrat sekä katselmointeihin kulunut työaika henkilötyötunteina. Katselmointeihin kulunut työaika eritellään katselmointiprosessin eri vaiheiden mukaan. Katselmointiprosessin työvaiheet määritellään tarkemmin kohdassa 7.1. Katselmointien aikana löydettyjen virheiden korjaamiseen kuluva aikaa ei tulla laskemaan osaksi katselmointeja, koska tämä työvaihe on jo osa yrityksen nykyistä prosessia. Virheiden korjaamisen vaatimalla työmäärällä ei ole myöskään suoraa riippuvuutta itse katselmointiprosessin tehokkuuteen.

Katselmointien aikana löydetty virheet lasketaan ja luokitellaan käyttäen kaksiasiateista minor/major -asteikkoa. Kirjallisuudessa näillä virheasteilla ei ole vakiintunutta määritelmää, vaan määritelmät vaihtelevat katselmointien tavoitteen mukaan [RAV96, Van99, Gil00]. Tämän tapaustutkimuksen ensisijainen tavoite on ohjelmiston käyttäjälle näkyvän laadun parantaminen, joten virheasteiden määrittely perustuu käyttäjän näkökulmaan. Tämän tutkimuksen puitteissa major-virhe määritellään virheeksi, joka jäädessään ohjelmistoon aiheuttaa ohjelmiston käyttäjälle näkyvän toimintahäiriön. Tällaisia virheitä ovat esimerkiksi virheelliset loogiset ehdot, virheellisiä tuloksia tuottavat algoritmit, vakavat suorituskykyongelmat tai ohjelmiston kaatumisen aiheuttavat virheet.

Minor-virhe on virhe, jolla ei ole suoraa yhteyttä yhteenkään toimintahäiriöön, mutta joka ohjelmistoon jäädessään heikentää ohjelmiston muita laadullisia attribuutteja, kuten lähdekoodin ymmärrettävyyttä tai ylläpidettävyyttä. Tällaisia virheitä ovat esimerkiksi virheelliset lähdekoodin kommentit, nimeämiskonventioiden noudattamatta jättäminen tai monitasoiset sisäkkäiset koodilohkot. Minor-virhe voi mahdollistaa uuden major-virheen syntymisen myöhemmin esimerkiksi antamalla lähdekoodin lukijalle väärä oletuksia koodin toiminnasta.

Minor/major -luokittelun perusteella varmistetaan, että katselmoinnin aikana pyritään löytämään mahdollisimman paljon major-virheitä. Mikäli virheiden luo-

kittelun perusteella huomataan, että minor-virheiden suhteellinen osuus kaikista löydettyistä virheistä on suuri, voidaan soveltaa toimenpiteitä minor-virheiden osuuden pienentämiseksi [Gil00]. Keskittymällä major-tason virheisiin voidaan vaikuttaa tehokkaimmin ohjelmiston käyttäjälle näkyvään laatuun.

Jokaisesta katselmoinnista tuotetaan raportti, johon kirjataan katselmoitavan tuotoksen koko, katselmoijien määrä, katselmoijien käyttämä työaika työvaiheittain sekä katselmoinnin aikana löydettyjen virheiden määrät minor/major-asteikolla luokiteltuna. Tämän lisäksi luokitellaan katselmoinnin aikana löytyneet virheet virhetyypin mukaisiin luokkiin. Tällaisia luokkia ovat muun muassa nimeämiskonventiot, toiminnalliset virheet, muistinhallinnan virheet ja puutteellinen poikkeustilanteiden käsittely. Virhetyypin mukaisen luokittelun tarkoituksena on selvittää minor/major-luokittelua yksityiskohtaisemmin katselmointien löytämien virheiden tyypit. Tutkimuksen lopussa kaikkien katselmointien raporteista luodaan yhteenveto, jossa tarkastellaan käytettyä työaika, löytyneiden virheiden määriä ja virheiden jakautumista virhetyyppeihin ja minor/major-virheisiin. Yhteenveto katselmointien tehokkuusmitoista esitetään kohdassa 8.1.

6.3 Teemahaastattelu

Tutkimuksessa käytetään haastattelua selvittämään työntekijöiden käsityksiä katselmointien tehokkuudesta ja hyödyllisyydestä sekä arvioimaan katselmointiprosessin soveltuvuutta esimerkkiyrityksen yrityskulttuuriin. Haastattelumenetelmänä käytetään teemahaastattelua, joka esiintyy kirjallisuudessa myös nimillä puolistrukturoitu haastattelu, puolistandardoitu haastattelu tai kohdennettu haastattelu [Met06, HiH08].

Teemahaastattelu on haastattelumenetelmä, jossa haastattelun aihepiirit eli teemat ovat kaikille haastateltaville samat [HiH08]. Teemahaastattelu suoritetaan ympäristössä, jossa haastateltavien tiedetään kokeneen tietyn tilanteen. Tutkijalla on tiettyjä oletuksia tilanteen määräävien piirteiden seurauksista haastateltaville. Näiden oletusten perusteella tutkija suunnittelee haastattelurungon, joka suunnataan tutkittavien henkilöiden subjektiivisiin kokemuksiin, jotka ovat syntyneet haastateltavien kokemassa tilanteessa.

Teemahaastattelussa ei ole tarkasti määritelty kysymysten muotoa tai esittämisjärjestystä, vaan haastattelu etenee tiettyjen keskeisten teemojen varassa [HiH08]. Haastateltavilla on vapaus pohtia haastattelun teemoihin liittyviä asioita laajasti

oman kokemuksensa pohjalta.

Haastattelut suoritetaan tutkimuksen loppuvaiheessa sen jälkeen, kun kaikki tutkimukseen liittyvät katselmoinnit ovat suoritettu ja tutkimukseen osallistuvat työntekijät ovat voineet muodostaa mielipiteen katselmoinneista. Haastattelujen keskeiset teemat on esitetty liitteessä 2. Haastateltavina ovat kaikki ne työntekijät, jotka ovat osallistuneet katselmointeihin joko tuotoksen tekijän tai katselmoijan roolissa.

6.4 Tutkimuksen riskit

Tutkimuksessa pyritään löytämään ja korjaamaan mahdollisimman paljon virheitä ohjelmiston lähdekoodista. Yrityksen alkuperäinen ongelma on kuitenkin ohjelmiston julkaisun jälkeen löytyneiden häiriöiden suuri määrä, ei lähdekoodin virheiden määrä. Vaikka ohjelmiston häiriöiden ja lähdekoodissa olevien virheiden välillä on yhteys, virheiden määrän pienentäminen ei takaa häiriöiden määrän pienenemistä. Katselmointien aikana löydettyjen virheiden määrällä ei siis voida sanoa olevan suoraa riippuvuutta ohjelmiston julkaisun jälkeen ilmenneiden häiriöiden määrään. Tämän johdosta katselmointien vaikutusta ohjelmiston julkaisun jälkeiseen laatuun tullaan arvioimaan sekä löytyneiden virheiden määrän että haastatteluista saatujen kvalitatiivisten tulosten perusteella.

Yrityksessä ei ole aikaisempaa kokemusta lähdekoodin katselmoinneista, joten oikeiden katselmointitekniikoiden oppiminen tapahtuu tutkimuksen aikana. Tämä saattaa vääristää katselmoinnin tehokkuusmittareiden arvoja, sillä katselmoinnin oppimisvaiheessa tehokkuus on heikompi kuin vakiintuneen katselmointiprosessin tehokkuus. Tätä riskiä pyritään pienentämään järjestämällä useampia katselmointikertoja ja muokkaamalla katselmointiprosessia jatkuvasti osallistujilta saadun palautteen mukaisesti.

Minkä tahansa uuden menetelmän adoptio epäonnistuu, mikäli menetelmää ei nähdä hyödyllisenä tai sen käyttöönotto on vaikeaa [Dav89]. Katselmointiin osallistuvien työntekijöiden motivaatiolla on suora riippuvuus katselmointiprosessin tehokkuuteen. Mikäli katselmointiin osallistuvat työntekijät eivät usko katselmointien parantavan ohjelmiston laatua, heidän motivaationsa virheiden etsimiseen laskee. Heikko motivaatio vaikuttaa negatiivisesti katselmointien tehokkuuteen ja pienentää katselmointien vaikutusta ohjelmiston laatuun. Mikäli työntekijät eivät saa omakohtaisia, positiivisia kokemuksia katselmoinneista, heidän

epäilykset katselmointien hyödyistä vahvistuvat.

Tutkimuksen aikana pyritään pitämään työntekijöiden motivaatiota yllä keskittämällä katselmoinnit pääosin koodiin, johon on tehty viime aikoina muutoksia tai jota ei ole julkaistu yhdessäkään Beta-ohjelmiston versiossa. Aikaisemmin testatun ja julkaisun koodin katselmointi on tehottomampaa kuin uuden tai muuttuneen koodin katselmointi [Gil00]. Testatusta ja julkaistusta koodista oletetaan löytyvän vähemmän virheitä kuin uudesta tai muuttuneesta koodista, mikä laskee osallistujien motivaatiota katselmoida koodia. Virheiden löytäminen taas vahvistaa työntekijöiden motivaatiota ja parantaa seuraavien katselmointien tehokkuutta. Katselmointiprosessi pyritään myös suunnittelemaan niin yksinkertaiseksi, että sen käyttöönotto on helppoa eikä se vaadi työntekijöiltä suurta ajallista panostusta.

7 Tutkimuksen toteutus

Tässä kohdassa kuvataan tutkimuksen toteutusta yrityksessä. Tutkimuksen aikana otetaan käyttöön katselmointiprosessi, joka esitellään kohdassa 7.1. Katselmointiprosessia varten yritykselle toimitetaan kohdassa 7.2 kuvattu materiaali. Tutkimuksen loppuvaiheessa yrityksessä suoritetaan haastattelututkimus, joka on kuvattu kohdassa 7.3.

7.1 Katselmointiprosessi

Tapaustutkimuksen ensisijaisena tavoitteena on vastata kohdassa 6.1 esitettyyn tutkimuskysymykseen. Tämän lisäksi tutkimuksen tavoitteena on ottaa käyttöön katselmointimenetelmä, joka huomioi pienen yrityksen yrityskulttuurin ja resursien rajallisuuden. Pienten ohjelmistoyritysten katselmointiprosesseja tutkivaa kirjallisuutta käsiteltiin tarkemmin kohdassa 5.1. Käytettävän katselmointimenetelmän tulee olla aloituskustannuksiltaan edullinen, sillä esimerkkiyrityksen resurssit ovat rajallisia eikä suuria taloudellisia tai työvoimallisia panostuksia katselmointeihin voida tehdä. Katselmointimenetelmän tulee mahdollistaa myös Internetin yli tapahtuvat hajautetut katselmoinnit, sillä esimerkkiyrityksen työntekijöillä on mahdollisuus etätyöskentelyyn. Tässä kohdassa kuvataan esimerkkiyrityksessä käyttöön otettava katselmointiprosessi.

7.1.1 Roolit

Esimerkkiyrityksen katselmointiprosessissa otetaan käyttöön samat roolit kuin kohdassa 3.1 kuvatussa ryhmäkatselmoinnissa, mutta kirjurin rooli jätetään pois. Erillisen kirjurin käyttö esimerkkiyrityksen katselmoinneissa ei ole perusteltua, sillä katselmointikokouksia ei järjestetä. Katselmointiryhmän sisäinen kommunikatio tapahtuu yrityksen sisäisessä reaaliaikaisessa viestintäjärjestelmässä (chat) sekä sähköpostin välityksellä. Tämän johdosta kommunikaation tallennus on automaattista.

Katselmointiin osallistuu tuotoksen tekijä, katselmointivastaava sekä mahdollisesti muita teknisiä asiantuntijoita. Sekä tuotoksen tekijä että katselmointivastaava osallistuvat omien tehtäviensä lisäksi myös tuotoksen katselmointiin muiden katselmoijien mukana. Tuotoksen tekijä on henkilö, joka on kirjoittanut katselmoitavan koodin tai merkittävän osan siitä. Katselmointi järjestetään tekijän omasta toivomuksesta. Tekijä perehdyttää muut katselmoijat katselmoitavan koodin taustalla oleviin suunnitteluratkaisuihin sekä vastaa katselmoinnin aikana esille tulleisiin kysymyksiin. Tekijän vastuulla on kaikkien katselmoinnin aikana löydettyjen virheiden korjaaminen.

Katselmointivastaavan tehtävänä on katselmointiin liittyvien järjestelyjen tekeminen, katselmointiprosessin tarkkailu ja raportointi. Järjestelyihin kuuluu katselmoijien valinta, työmääräimien luominen kullekin katselmoijalle, mahdollisen seurantakokouksen päätösten kirjaaminen sekä tarkistuslistojen ylläpito katselmointien löydösten perusteella. Katselmointivastaava tarkkailee katselmointiprosessia ja pyrkii reagoimaan siinä havaittuihin poikkeamiin. Tällaisia poikkeamia voivat olla esimerkiksi liian lyhyt katselmointiaika tai löydettyjen virheiden ratkaisuvaihtoehtojen miettiminen.

Esimerkkiyrityksessä sekä tuotoksen tekijä että katselmointivastaavat ovat teknisiä asiantuntijoita, joilla on hyvät edellytykset ymmärtää katselmoitavaa tuotosta. Heidän lisäksi katselmointiin voi osallistua myös muita teknisiä asiantuntijoita, jotka eivät suorita katselmoinnin ohella muita tehtäviä. Esimerkiksi Beta-ohjelmiston muiden osien parissa työskentelevät työntekijät voivat olla kiinnostuneita tuntemaan syvällisemmin ohjelmiston katselmoitavaa osaa, jolloin he voisivat osallistua katselmointiin. Katselmointien avulla voidaan myös kouluttaa muista yrityksistä tai muiden projektien parista siirtyneitä työntekijöitä esimerkiksi yrityksen lähdekoodin kirjoittamiseen liittyviin konventioihin.

Yrityksen pienen koon takia katselmointiryhmän koko tulee käytännössä olemaan 2 – 3 henkilöä: tekijä, katselmointivastaava sekä mahdollisesti yksi katselmoija. Mikäli tuotoksen tekijä pyytää tuotokselleen katselmointia ja kukaan yrityksen muista työntekijöistä ei ehdi hoitamaan katselmointivastaavan rooliin kuuluvia tehtäviä, tekijä voi ottaa vastuulleen myös katselmointivastaavan tehtävät.

7.1.2 Aloitusehdot

Esimerkkiyrityksen katselmointiprosessin aloitusehtoja on kolme: tuotoksen onnistunut ja virheetön käänös, tuotoksen staattisen analyysin virheettömyys sekä katselmoitavan tuotoksen looginen rajaus. Aloitusehtojen tarkoituksena on varmistaa katselmoitavaksi ehdotetun tuotoksen riittävän korkea laatu, jotta sen katselmointi on mahdollisimman tehokasta. Katselmointivastaavan tehtävänä on varmistaa, että tuotos täyttää aloitusehdot. Mikäli tuotos ei täytä aloitusehtoja, sen katselmointia ei aloiteta, vaan tuotoksen tekijää pyydetään tekemään tarpeelliset muutokset, jotta aloitusehdot täyttyisivät.

Beta-ohjelmiston toteutuksessa käytetään pääosin käännettäviä ohjelmointikieliä, joten yksi aloitusehto on katselmoitavan tuotoksen käänöksen onnistuminen. Kääntäjä pystyy tekemään käänöksen aikana myös joitakin lähdekoodin tarkistuksia, kuten raportoida käyttämättömistä muuttujista. Kaikki kääntäjän antamat varoitukset tulee korjata ennen kuin tuotos hyväksytään katselmoitavaksi.

Kääntäjän lisäksi esimerkkiyritys käyttää staattisen analyysin työkaluja lähdekoodin virheiden löytämiseksi. Staattisen analyysin avulla voidaan löytää automaattisesti osa niistä virheistä, joita katselmoijat joutuisivat muuten etsimään manuaalisten katselmointien aikana. Työkalujen käyttö virheiden etsimisessä vapauttaa katselmoijat mekaanisesta koodin lukemisesta ja mahdollistaa katselmoijien keskittymisen ohjelman toimintaan liittyvien virheiden etsimiseen [Fag02]. Tämän johdosta yhtenä aloitusehtona on staattisen analyysin työkalujen löytämien virheiden korjaus katselmoitavaksi ehdotetusta tuotoksesta.

Katselmoitavan kokonaisuuden on oltava loogisesti rajattu, jotta sen katselmointi on tehokasta. Käytännössä jokaista katselmointikertaa varten pyritään valitsemaan joukko yhteen kuuluvia luokkia, joiden koodin yhteenlaskettu koko ei ylitä kirjallisuudessa suositeltua 200 – 300 koodirivin määrää. Keskimääräisellä katsel-

mointinopeudella 150 – 200 koodiriviä tunnissa tällaisen tuotoksen katselmointi vaatii alle kaksi tuntia aikaa, mikä takaa tehokkaan katselmoinnin.

Mikäli tuotos täyttää yllä mainitut aloitusehdot, katselmointivastaava luo yrityksen tietojärjestelmään uuden työmääräimen tuotoksen katselmoinnista. Työmääräimeen tallennetaan katselmoitavan tuotoksen rajausta listaamalla kaikkien katselmoitavien luokkien, metodien tai lähdekooditiedostojen nimet ja versiot. Mikäli katselmoitavaan tuotokseen liittyy joitakin erityisen huolellista katselmointia vaativia kohtia, kuten monimutkaisia algoritmeja tai säikeistettyä koodia, niin nämä kohdat identifioidaan ja merkitään työmääräimeen. Työmääräimeen merkitään myös koodin koko koodiriveinä sekä arvio työajasta, joka vaaditaan tuotoksen katselmointiin. Aika-arvio perustuu aikaisemmista katselmoineista saattuihin tuloksiin. Ensimmäiset aika-arviot johdetaan suoraan katselmoitavan koodin koosta käyttäen kirjallisuudessa suositeltua katselmointinopeutta 200 – 300 koodiriviä tunnissa. Työmääräimeen merkitään myös katselmoinnin takaraja, johon menneessä kunkin katselmoijan on suoritettava tuotoksen katselmointi ja raportoitava tulokset.

7.1.3 Tarkistuslista

Esimerkkiyrityksen katselmointien lukumenetelmänä käytetään tarkistuslistapohjaista lukumenetelmää, joka on kuvattu tarkemmin kohdassa 4.1.2. Tarkistuslistan tarkoituksena on kohdistaa katselmoijien huomio yleisimpiin lähdekoodin virhetyyppeihin.

Katselmointivastaava suunnittelee ensimmäisen version tarkistuslistasta, jota tullaan käyttämään Beta-ohjelmiston ensimmäisessä katselmoinnissa. Hän analysoi Beta-ohjelmistoon tehtyjä virheiden korjauksia viimeisen kuuden kuukauden aikana ja listaa ne virheet, joiden syntyminen voidaan välttää katselmointien avulla. Virheistä johdetaan joukko sääntöjä, joita noudattamalla virheiden synty voidaan estää.

Ensimmäinen tarkistuslistan versio sisältää noin 20 sääntöä. Koska tarkistuslista on johdettu Beta-ohjelmiston aikaisemmista virheistä, sen sisältämät säännöt ovat konkreettisia ja käytännönläheisiä. Ne sisältävät mm. ohjelmointikielikohaisia tai kirjastokohtaisia sääntöjä, jotka ovat relevantteja Beta-ohjelmiston kehityksessä. Johtamalla säännöt ohjelmiston omasta muutoshistoriasta pyritään välttämään Laitenbergerin ja DeBaudin [LaD00] kuvaamia liian yleisellä tasol-

la esitettyjä kysymyksiä ja sääntöjä.

Tarkistuslistaan tehdään muutoksia ja lisäyksiä aina, kun katselmointien aikana löydetään uusia virhetyyppejä. Mikäli jotkut tarkistuslistan kysymykset tai säännöt osoittautuvat epärelevantteiksi esimerkiksi muuttuneiden kirjastojen tai uusien ohjelmointikonventioiden takia, ne poistetaan listasta. Tarkistuslista pyritään pitämään korkeintaan 20 kohdan pituisena, sillä liian pitkien tarkistuslistojen käyttö on tehotonta [CTB06].

Beta-ohjelmistolle tyypillisten virheiden löytämiseksi tarkoitettujen kysymysten ja sääntöjen lisäksi tarkistuslistaa käytetään prosessin parannusvälineenä. Joidenkin tarkistuslistassa olevien kysymysten ja sääntöjen tarkistus voidaan automatisoida esimerkiksi kehittämällä staattisen analyysin keinoin automaattisesti suoritettavia tarkistuksia. Mikäli jotkut tarkistuslistan kohdista ovat automatisoitavissa ja automatisointi on kustannustehokasta, ne automatisoidaan ja poistetaan listasta. Näin tarkistuslista pysyy lyhyenä ja katselmoijat välttyvät mekaanisilta tarkistuksilta [Gil00]. Samalla parannetaan ohjelmistotuotantoprosessia sekä sen tuottaman ohjelmiston lähdekoodin laatua.

7.1.4 Yleiskatsaus

Aloitusehtojen varmistuksen jälkeen katselmointivastaava ja tuotoksen tekijä valitsevat muut mahdolliset katselmointiin osallistuvat asiantuntijat. Tämän jälkeen katselmointivastaava järjestää yleiskatsausvaiheen, jossa katselmoitavan tuotoksen tekijä esittelee tuotoksen muille katselmoijille.

Yleiskatsausvaiheen tarkoituksena on tutustuttaa katselmointiin osallistuvat henkilöt katselmoitavaan tuotokseen. Tuotoksen tekijä kuvailee tuotoksen ongelma-alueita sekä tuotoksen suhdetta muihin ohjelmiston osiin. Tekijä myös esittelee tuotoksen suunnittelun ja toteutuksen taustalla olevia suunnittelumalleja sekä relevantteja oletuksia, jotka katselmoijien on tunnettava ymmärtääkseen tuotoksen lähdekoodia.

Yleiskatsaus voidaan suorittaa joko kokouksena tai virtuaalisesti yrityksen chatissa. Mikäli kaikki katselmoijat ovat samassa fyysisessä tilassa, kokouksen järjestäminen voi olla tehokkaampaa kuin kommunikointi chatin kautta, koska tällöin voidaan käyttää luontevasti muita esittämisen apuvälineitä, kuten paperipiirroksia tai fläppitaulua.

Katselmointivastaava varmistaa, että yleiskatsauksen aikana käydyn keskustelun

aikana esitellään katselmoitavaa tuotosta ja esitetään sen suunnitteluun ja toteutukseen liittyviä kysymyksiä. Keskustelun aikana voi selvitä, että tekijä on ymmärtänyt ohjelmiston vaatimukset väärin ja toteuttanut tuotoksen, joka ei vastaa ohjelmiston oikeita vaatimuksia. Tällöin katselmointivastaava voi lopettaa katselmointiprosessin ja pyytää tekijää muuttamaan tuotosta ohjelmiston vaatimusten mukaiseksi.

Yleiskatsauksen kesto pidetään alle tunnin pituisena. Mikäli katselmoijat tuntevat katselmoitavan tuotoksen hyvin, yleiskatsausvaihe voidaan jättää suorittamatta ja siirtyä suoraan kohdassa 7.1.5 esitettyyn katselmointivaiheeseen.

7.1.5 Katselmointi

Kukin katselmoija päättää itsenäisesti sen ajankohdan, jolloin katselmoinnin suoritus sopii hänelle parhaiten. Ajankohdan valinnassa noudatetaan katselmoinnin työmääräimessä mainittua takarajaa. Katselmoinnin alkaessa yrityksen tietojärjestelmään merkitään katselmoinnin aloitusaika työmääräimen yhteyteen. Tämän jälkeen katselmoija varmistaa, että hänellä on paikallinen kopio katselmoitavan tuotoksen työmääräimeen merkitystä versiosta. Käytännössä yleensä katselmoidaan tuotoksen viimeisintä versiota.

Katselmointi suoritetaan ensisijaisesti tietokoneen ruudulla samassa ohjelmointiympäristössä, missä ohjelmiston normaali kehitys tapahtuu. Katselmoijat voivat halutessaan tulostaa lähdekoodin tai sen osia paperille, mikäli se helpottaa koodin lukemista. Katselmoinnin aikana voidaan käyttää katselmoitavaan koodiin liittyvää dokumentaatiota, lukea muuta tuotokseen liittyvää koodia tai hakea tietoa esimerkiksi ohjelmistossa käytettyjen kirjastojen dokumentaatiosta, sähköpostilistoilta tai keskustelufoorumeilta. Katselmoinnin aikana tehdyt löydökset merkitään suoraan lähdekoodin sekaan kommentteina, joilla on tietty ennalta määrätty muoto. Kommentin alkuun tulee teksti "Review finding", joka mahdollistaa kaikkien löydösten listaamisen ohjelmointiympäristössä yksinkertaisen hakutoiminnon avulla. Tämän jälkeen kirjoitetaan löydöksen kuvaus yhdellä tai kahdella virkkeellä, esimerkiksi "metodikutsu ei toimi oikein, jos olio on NULL".

Katselmoinnin aikana voidaan myös käydä lyhyitä keskusteluja yrityksen chatissa, jos katselmoijalla on esimerkiksi tarvetta selventää joitakin yksityiskohdita tuotoksen tekijältä. Keskustelut pidetään kuitenkin lyhyinä, enintään kaksi- kolme minuuttia per löydös. Mikäli asia ei selviä muutamassa minuutissa, kyse

voi olla yleiskatsausvaiheen puutteellisuudesta tai lähdekoodin heikosta ymmärrettävyydestä. Yhden löydöksen pohtimiseen ei kuitenkaan jäädä kiinni pitkäksi aikaa, vaan mahdollinen löydös merkitään muistiin ja jatketaan katselmointia eteenpäin. Laajemmat keskustelut mahdollisista löydöksistä voidaan käydä katselmoinnin jälkeen.

Ohjelmointiympäristön lisäksi katselmoijat voivat käyttää erilaisia työkaluja analysoidessaan koodin oikeellisuutta. Esimerkiksi tietyt staattisen analyysin työkalujen avulla lähdekoodista voidaan löytää keskimääräistä monimutkaisempia rakenteita. Monimutkainen lähdekoodi voi viitata joko monimutkaisiin algoritmeihin tai heikosti ymmärrettävään koodiin. Katselmoijat voivat tällöin kohdistaa huomionsa koodin monimutkaisempiin osiin, sillä virheiden esiintymisen todennäköisyys on näissä kohdissa yleensä keskimääräistä suurempi. Koodin luetuutta haittaavat erittäin korkeat kompleksisuudet voidaan merkitä suoraan löydöksiksi.

Katselmoijat pyrkivät löytämään lähdekoodista virheitä, ei niiden ratkaisuja. Löydettyjen virheiden triviaalit ratkaisut voidaan kirjata ylös löydöksen kommentin yhteyteen, mutta ratkaisun toteutus jätetään aina tuotoksen tekijän vastuulle. Sääntö koskee myös työkalujen avulla löytyneitä löydöksiä. Tämä rajoitus on perusteltu, sillä tuotoksen katselmoijalla ei ole välttämättä kaikkea sitä taustatietoa, mikä vaaditaan virheen oikeellisen korjauksen tekemiseksi. Koodissa voi olla tiettyjä oletuksia, jotka eivät ole katselmoijalle selviä. Mikäli katselmoija tekee korjauksen itse, tuotoksen tekijän pitää joka tapauksessa varmistaa korjauksen oikeellisuus. Katselmointiprosessin jakaminen erillisiin katselmointi- ja korjausvaiheisiin helpottaa myös prosessin seuranta ja jatkokehitystä.

Kun koko tuotos on katselmoitu, katselmoija tallentaa muuttuneet lähdekooditiedostot löydöksineen yrityksen versionhallintajärjestelmään. Mikäli samoista lähdekoodin osista on tehty keskenään konfliktioivia löydöksiä, versionhallintajärjestelmä ei suorita tallennusta ennen kuin löydösten tekijät ratkaisevat konfliktit. Kun kaikki muutokset on tallennettu onnistuneesti, katselmoija merkitsee yrityksen tietojärjestelmään katselmoinnin lopetusajan työmääräimen yhteyteen. Katselmointivaihe päättyy, kun kaikki katselmoijat ovat tallentaneet muuttuneet lähdekooditiedostot löydöksineen yrityksen versionhallintajärjestelmään.

7.1.6 Korjaus

Katselmointivaiheen jälkeen katselmoinnin työmääräin siirtyy tuotoksen tekijälle, jonka vastuulla on ottaa kantaa jokaiseen katselmointivaiheen aikana tehtyyn löydökseen. Tekijä tekee löydösten perusteella korjauksia lähdekoodiin ja hylkää väärät löydökset. Tuotoksen tekijä ei tallenna korjaustyön aloitus- ja lopetusajan kohtaa, sillä korjaustyötä ei lasketa mukaan katselmoinnin työmäärään.

Mikäli löydös osoittautuu oikeaksi virheeksi ja virheen korjaus voidaan toteuttaa heti, tekijä tekee tarvittavat muutokset koodiin. Joissakin tilanteissa löydettyjen virheiden korjaus saattaa vaatia laajojakin muutoksia, esimerkiksi jos tekijän oletukset ohjelmiston vaatimuksista ovat olleet väärää alusta lähtien. Laajojen muutosten toteutusta voidaan joutua lykkäämään aikataulullisista syistä, jolloin yrityksen tietojärjestelmään luodaan uusi työmääräin, joka koskee näitä muutoksia. Tällöin tekijä merkitsee lähdekoodiin löydöksen tilalle viitteen muutoksia koskevaan työmääräimeen ja jatkaa muiden löydösten käsittelyä.

Mikäli tekijä hylkää löydöksen, hän esittää hylkäämiselleen perustelut joko yrityksen chatissa tai lähdekoodin kommentteissa. Löydöksen hylkääminen voi johtua katselmoijan väärinkäsityksestä tai koodin taustalla olevista oletuksista, joista katselmoija ei ollut tietoinen katselmoinnin aikana. Tekijä voi tällöin harkita esimerkiksi ohjelmiston dokumentaation täydentämistä tai koodin luettavuuden parantamista.

Korjausten aikana tuotoksen tekijä voi tarkentaa löydöksiä kysymällä katselmoijilta lisätietoja löydöksistä yrityksen chatissa. Katselmoija voi esimerkiksi tarkentaa, millaisissa olosuhteissa hänen löytämänsä virhe esiintyy ohjelmiston toiminnan aikana.

Korjausvaihe päättyy, kun tuotoksen tekijä on ottanut kantaa jokaiseen löydökseen. Tämän jälkeen tekijä tallentaa muuttuneet lähdekooditiedostot yrityksen versionhallintajärjestelmään ja siirtää katselmoinnin työmääräimen katselmointivastaavalle.

7.1.7 Seuranta

Seurantavaiheessa katselmointivastaava tarkistaa, että tuotos täyttää sille asetetut valmiusehdot. Valmiusehtoja on neljä: tuotoksen tekijä on ottanut kantaa jokaiseen löydökseen, tuotos kääntyy virheettömästi, tuotoksen staattinen analyysi

si ei tuota virheitä ja katselmointivastaava on katselmoinut jokaisen tekijän tekemän muutoksen.

Katselmointivastaava tarkistaa, että tekijä on ottanut kantaa jokaiseen löydöksen, jonka jälkeen katselmoitava tuotos käännetään ja sille suoritetaan staattinen analyysi. Näiden työvaiheiden tarkoituksena on varmistaa, että tekijä ei ole jättänyt katselmoijien löydöksiä huomioimatta ja että tehdyt muutokset eivät aiheuttaneet uusia, automaattisesti löydettävissä olevia virheitä. Katselmointivastaava palauttaa tuotoksen tekijälle lisäkorjauksia varten, mikäli joihinkin löydöksiin ei ole reagoitu tai automaattiset tarkistukset tuottavat virheitä.

Kun tuotos on läpäissyt kolme ensimmäistä valmiusehtoa, katselmointivastaava katselmoi tekijän tekemät muutokset, jotta ne eivät aiheuta uusia virheitä tuotokseen. Uutta katselmointia ei järjestetä samalle tuotokselle. Tämä käytäntö eroaa Faganin suosituksesta, jonka mukaan tuotos tulee katselmoida uudestaan, mikäli korjaustoimenpiteet muuttavat enemmän kuin 5% tuotoksesta [Fag76]. Faganin säännön noudattaminen vaatii liikaa resursseja, sillä tuotoksen koon ollessa 200 – 300 koodiriviä uusi katselmointi tulee järjestää aina, kun tuotoksesta muuttuu enemmän kuin 10 – 15 koodiriviä. Mikäli katselmoitavan tuotoksen koko pidetään pienenä, tuotokseen tehtävät laajatkin muutokset ovat yhden henkilön katselmoitavissa.

Valmiusehtojen tarkistamisen jälkeen katselmointivastaava päättää yhdessä tuotoksen tekijän kanssa seurantakokouksen järjestämisestä. Seurantakokoukseen osallistuvat tuotoksen tekijä, katselmointivastaava sekä muut katselmoinnissa mukana olleet katselmoijat. Kaikkien katselmoijien ei ole kuitenkaan pakko osallistua kokoukseen. Kokouksen tarkoituksena on analysoida katselmoinnin aikana löytyneitä virheitä ja keskustella näiden virheiden ehkäisystä esimerkiksi kehittämällä ohjelmointikonventioita tai lisäämällä automaattisia tarkistuksia staattisen analyysin työkalun avulla. Seurantakokous voidaan järjestää joko virtuaalisena yrityksen chatissa tai fyysisenä yrityksen tiloissa. Kokousta ei pidetä, mikäli katselmoinnin aikana ei löytynyt virheitä tai löydettyjen virheiden ehkäisymahdollisuudet on käsitelty jo aikaisemmissa kokouksissa.

Seurantavaiheen päätteeksi katselmointivastaava tuottaa katselmointiraportin, johon kirjataan tiedot katselmoitavasta tuotoksesta, katselmoijien käyttämästä työajasta sekä katselmoinnin aikana löydettyistä virheistä minor/major -asteikolla luokiteltuna. Virheiden luokittelu on katselmointivastaavan vastuulla.

7.2 Toimitettu materiaali

Ennen katselmointien aloittamista yritykselle toimitettiin ensimmäinen tarkistuslistan versio sekä käyttöön otettavan katselmointiprosessin yksityiskohtainen kuvaus. Ensimmäinen tarkistuslista tehtiin analysoimalla vuoden 2008 aikana Beta-ohjelmistoon tehtyjä lähdekoodin muutoksia, jotka liittyivät virheiden korjauksiin. Korjausten taustalla olevat virheet ja niiden syyt selvitettiin. Mikäli virheen syy oli korjattavissa käyttäen tiettyä sääntöä katselmoinnin aikana, tämä sääntö lisättiin tarkistuslistaan.

Muutosten analysoinnin jälkeen tarkistuslistan säännöt järjestettiin aihealueen mukaan 12 aiheeseen ja kunkin aiheen sääntölista rajoitettiin muutamaa tärkeimpään sääntöön. Tarkistuslistan ensimmäisessä versiossa oli 24 sääntöä, jotka mahtuivat tulostettuna kahdelle A4-arkille. Säännöt olivat hyvin käytännönläheisiä ja ne liittyivät Beta-ohjelmistossa käytettävään ohjelmointikieleen sekä ohjelmiston käyttämiin kirjastoihin. Tarkistuslista tallennettiin tekstitiedostona yrityksen versionhallintajärjestelmään.

Katselmointiprosessin kuvaus sisälsi numeroidun listan kohdassa 7.1 esitellyn prosessin työvaiheista sekä niihin kuuluvista työtehtävistä ja roolijaosta. Toisin kuin kohdassa 7.1, prosessi kuvattiin yritysriippuvaisella tavalla käyttäen yrityksessä käytössä olevien järjestelmien nimiä sekä yrityksen sisällä tunnettuja termejä. Prosessikuvauksen kohdat pidettiin lyhyinä, yhden – kolmen virkkeen pituisina. Lopullinen prosessikuvaus koostui 23 numeroidusta kohdasta, ja sen kokonaispituus tulostettuna oli yksi A4-arkki. Prosessikuvaus on tutkielman liitteenä 1.

Tarkistuslista koettiin yrityksessä heti hyödylliseksi, sillä se nähtiin tapana tallentaa ja ylläpitää yrityksessä syntyvää kokemusperäistä tietoa. Eräs yrityksen työntekijä kertoi, että hänelle tulee ohjelmointityön aikana mieleen tiettyjä sääntöjä ja konventioita, joita hän kokee hyödyllisiksi. Näitä sääntöjä ja konventioita ei kuitenkaan tallenneta mihinkään, jolloin ne unohtuvat helposti. Tarkistuslistan avulla näitä työn ohessa syntyviä ajatuksia voi tallentaa helposti. Tarkistuslista nähtiin myös hyödyllisenä välineenä uusien työntekijöiden koulutuksessa, sillä se tarjoaa kokeneiden kehittäjien hyväksi havaitsemia sääntöjä, joita noudattamalla uudet kehittäjät voivat välttää useita virhetilanteita.

7.3 Haastattelujen järjestelyt

Haastattelut järjestettiin esimerkkiyrityksessä viiden katselmointikerran suorituksen jälkeen, joissa katselmoitiin yhteensä noin 1600 kommentoimatonta riviä lähdekoodia. Haastattelijana toimi tutkija itse ja haastateltavina olivat yrityksen kolme työntekijää. Haastateltavista yksi työntekijä osallistui katselmointeihin vain tuotoksen tekijän roolissa, yksi työntekijä vain katselmoijan roolissa ja yksi työntekijä sekä katselmoijan että tuotoksen tekijän roolissa. Haastattelija itse osallistui katselmointeihin sekä katselmoijan että tuotoksen tekijän roolissa.

Haastateltaville oli ehtinyt kertymään katselmointien aikana kokemusta katselmoinneista ja he pystyivät muodostamaan niihin liittyviä mielipiteitä. Lisäkatselmointien järjestäminen olisi vahvistanut haastateltavien käsitystä katselmoinneista, mutta katselmointikerrat jouduttiin rajoittamaan viiteen kappaleeseen ajanpuutteen vuoksi. Kuitenkin jo viiden katselmointikerran jälkeen sekä ohjelmistotuotteessa että esimerkkiyrityksen ohjelmistotuotantoprosessissa näkyi selviä muutoksia, joten haastattelujen järjestäminen oli perusteltua.

Jokaista haastatteluun osallistunutta työntekijää haastateltiin henkilökohtaisesti. Haastattelujen rakenne oli kaikille haastateltaville sama, ja jokaisen työntekijän kanssa käsiteltiin liitteessä 2 esitettyjä teemoja. Haastattelut nauhoitettiin, mikä mahdollisti luontevan keskustelun haastattelujen aikana. Haastattelut kestivät 24 – 32 minuuttia. Haastattelut suoritettiin yrityksen neuvotteluhuoneessa 16.10.2008.

Haastateltavat suhtautuivat tutkimukseen ja haastatteluihin poikkeuksetta myönteisesti. Haastattelun teemoihin liittyvä keskustelu oli vapaata ja sujuvaa. Keskustelu siirtyi luontevasti aihepiiristä toiseen, joskus jopa ilman haastattelijan esittämiä kysymyksiä. Haastateltavat myös kertoivat aikaisemmista katselmointeihin tai yleisesti laaduntarkkailuun liittyvistä kokemuksistaan ja vertailivat niitä tutkimuksen aikana saatuihin kokemuksiin.

8 Tutkimuksen tulokset

Tässä kohdassa esitellään tutkimuksesta saadut tulokset. Kohdassa 8.1 tarkastellaan katselmointien tehokkuutta tutkimuksen aikana kerätyn määrällisen aineiston pohjalta. Löydettyjä virheitä analysoidaan luokittelemalla ne tyypeittäin kohdassa 8.2.

Määrällisen aineiston tarkastelun jälkeen kuvataan teemahaastattelun avulla saatuja laadullisia tietoja. Katselmointien vaikutusta ohjelmiston laatuun tarkastellaan kohdassa 8.3 ja vaikutuksia ohjelmiston osien tuntemukseen kohdassa 8.4.

Kohdassa 8.5 esitellään työntekijöiden näkemyksiä käytetystä katselmointiprosessista. Kohdassa 8.6 arvioidaan katselmointiprosessin aikana käytettyjä työkaluja ja esitellään työkalutuessa tapahtunutta kehitystä. Lopuksi tuloksista johdetaan johtopäätökset, jotka esitetään kohdassa 8.7.

8.1 Katselmointien tehokkuus

Tutkimuksen aikana suoritettiin viisi lähdekoodin katselmointia, joissa katselmoitiin yhteensä 1594 kommentoimatonta koodiriviä. Katselmointien aikana löydettiin yhteensä 196 virhettä. Näistä virheistä 25 % olivat major-tason virheitä ja loput 75 % minor-tason virheitä. Katselmointeihin käytettiin 14 tuntia 20 minuuttia ja kokouksiin 3 tuntia. Yhteensä työaikaä käytettiin 17 tuntia 20 minuuttia. Katselmoitavien tuotosten koot, katselmoinnin vaiheiden vaatima työaika sekä löydettyjen virheiden määrät on esitetty taulukossa 8.1.

Katselmointikerta	1	2	3	4	5
Tuotoksen koko	224	343	365	461	201
Katselmoijien määrä	2	2	2	2	1
Yleiskatsaus	-	2h	-	-	-
Katselmointi	2h 20min	2h 30min	2h 20min	3h 30min	3h 45min
Seurantakokous	1h	-	-	-	-
Major-virheet	16	1	10	6	16
Minor-virheet	40	21	19	37	30
Major-virheiden osuus	29 %	5 %	34 %	14 %	35 %
Minor-virheiden osuus	71 %	95 %	66 %	86 %	65 %

Taulukko 1: Katselmointien ajankäyttö ja löydettyjen virheiden määrät. Kestot on ilmoitettu henkilötyötunteina.

Tutkimuksessa saatujen tulosten perusteella katselmointien tuloksellisuus oli 123 virhettä tuhatta koodiriviä kohden. Tuloksellisuus oli huomattavasti suurempi kuin kirjallisuudessa esitetyt arvot, jotka ovat 8 – 42 virhettä tuhatta koodiriviä kohden [ABL89, Wie01]. On huomattava, että 75% löydettyistä virheistä oli minor-

tason virheitä. Gilbin mukaan [Gil00] minor-tason virheet voidaan jättää pois katselmointien tehokkuutta tarkasteltaessa. Mikäli vain major-tason virheet otetaan huomioon, katselmointien tuloksellisuus oli 31 virhettä tuhatta koodiriviä kohden, mikä on kirjallisuudessa esitettyjen arvojen mukainen tulos.

Katselmointien kustannustehokkuus oli 14 virhettä henkilötyötuntia kohden. Kustannustehokkuus oli huomattavasti suurempi kuin kirjallisuudessa esitetyt arvot, jotka ovat 0,2 – 1,5 virhettä henkilötyötuntia kohden. Mikäli vain major-tason virheet otetaan huomioon, katselmointien kustannustehokkuus oli 3,4 virhettä henkilötyötuntia kohden.

Katselmoijien keskimääräinen katselmointinopeus oli 221 kommentoimatonta koodiriviä tunnissa. Katselmointinopeus oli kirjallisuuden suositeltujen arvojen mukainen [Fag76, ABL89, BaP94, VVS01, Wie01, RDN04]. Yksittäisten katselmoijien katselmointinopeudet vaihtelivat huomattavasti tutkimuksen aikana. Tutkimuksen aikana katselmoijan roolissa toimi kaksi henkilöä. Ensimmäisen katselmoijan keskimääräinen katselmointinopeus oli 146 koodiriviä tunnissa ja toisen katselmoijan keskimääräinen katselmointinopeus oli 687 koodiriviä tunnissa.

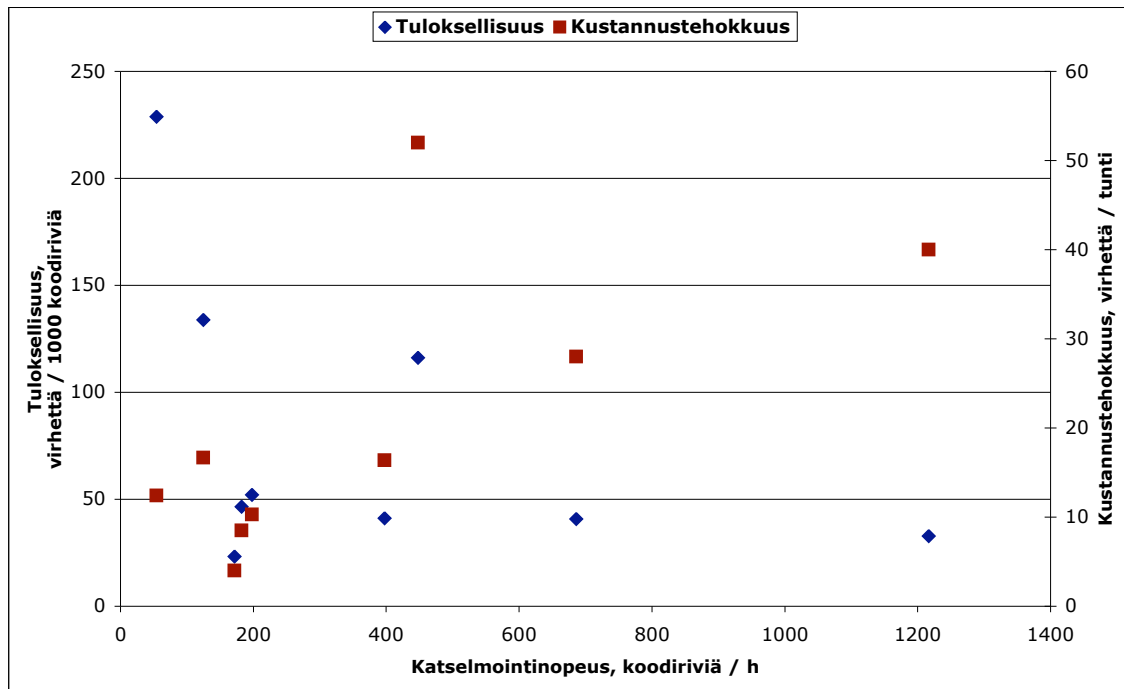
Katselmointinopeudella ja katselmoinnin tuloksellisuudella on katselmointikirjallisuuden mukaan negatiivinen riippuvuus [VVS01, Wie01]. Tapaustutkimuksen aikana todettiin kyseisen riippuvuuden olemassaolo. Yksittäisen katselmoijan katselmointinopeuden ja katselmoinnin tuloksellisuuden välinen korrelaatiokerroin oli -0.44. Havaintojen rajallisen lukumäärän vuoksi korrelaatio ei ole tilastollisesti merkitsevä ($p = 0.23$).

Yksittäisen katselmoijan katselmointinopeuden ja kustannustehokkuuden välillä oli positiivinen riippuvuus, jonka korrelaatiokerroin oli 0.68. Korrelaatio on tilastollisesti merkitsevä ($p = 0.04$). Tilastolliset testit on esitetty liitteessä 3.

Korkeampi katselmointinopeus mahdollisti suuremman virhemäärän löytämisen samassa ajassa, mutta jätti tuotokseen enemmän virheitä kuin hitaampaa katselmointinopeutta käytettäessä. Katselmointinopeuden vaikutukset tuloksellisuuteen ja kustannustehokkuuteen on esitetty hajontakaaviona kuvassa 3.

8.2 Löytyneet virheet

Katselmoinneissa löytyneiden virheiden analyysiä varten virheet luokiteltiin viiteen eri luokkaan, joiden suhteelliset osuudet on esitetty kuvassa 4. Nämä luokat olivat puutteet virheiden ja poikkeustilanteiden käsittelyssä, heikko koodin luet-



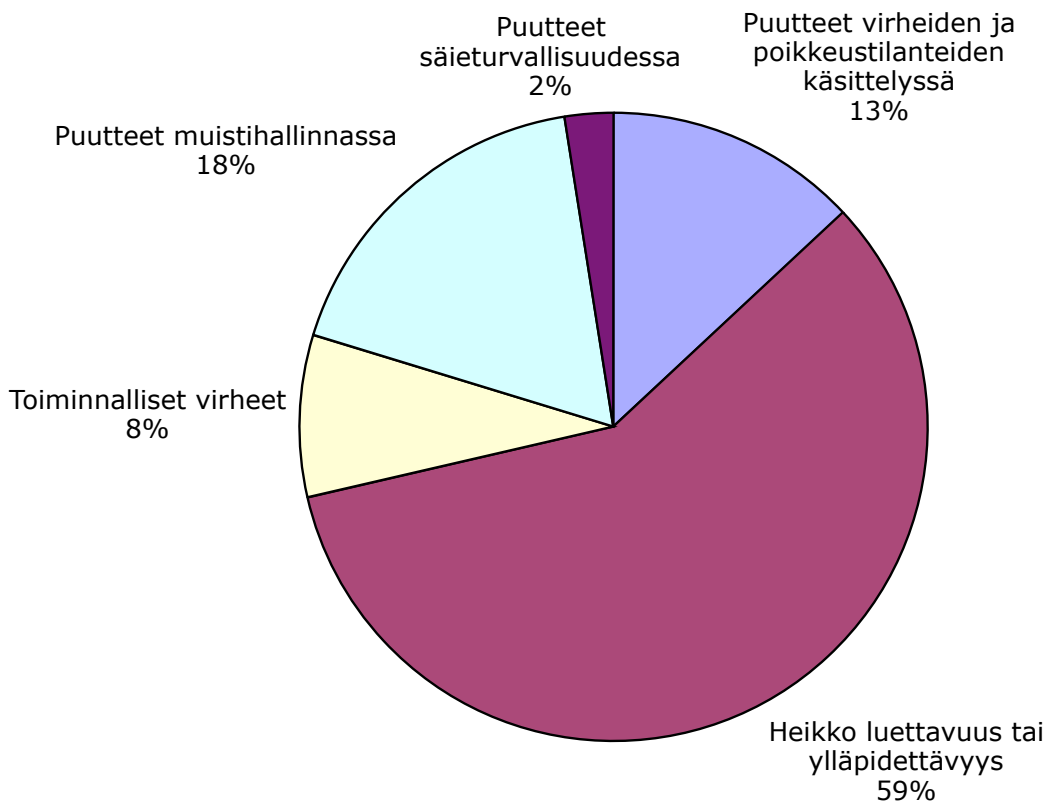
Kuva 3: Katselmointinopeuden ja tuloksellisuuden välinen riippuvuus.

tavuus tai ylläpidettävyys, puutteet muistinhallinnassa, puutteet säieturvallisuudessa sekä toiminnalliset virheet.

Löydetyistä virheistä suurin osa, 59%, liittyi koodin heikkoon luettavuuteen tai ylläpidettävyYTEEN. Nämä virheet olivat poikkeuksetta minor-tason virheitä, jotka heikensivät ohjelmiston sisäistä laatua. Osa luettavuuteen liittyvistä virheistä johtui ohjelmointikielen ja käytettyjen ohjelmointikirjastojen konventioiden noudattamatta jättämisestä.

Toiseksi suurin virhetyyppiluokka oli muistinhallinnan puutteet, joita oli 18% löydetyistä virheistä. Tämä virhetyyppi sisälsi virheitä, jotka olisivat voineet aiheuttaa ohjelmiston muistivuotoja tai vapautetun muistin virheellistä käyttöä. Näistä virheistä suurin osa oli major-tason virheitä. Pieni osa muistinhallinnan virheistä luokiteltiin minor-virheiksi esimerkiksi tilanteissa, joissa muistivuoto oli pieni ja tapahtui vain kerran ohjelmiston käytön aikana.

Puutteet virheiden ja poikkeustilanteiden hallinnassa muodostivat 13% löydetyistä virheistä. Nämä puutteet liittyivät virhe- ja poikkeustilanteiden huomaamiseen ja käsittelyyn. Tässä virhetyypissä oli sekä major- että minor-tason virheitä. Esimerkiksi ohjelmiston virhetilanteen käsittelyn suorittamatta jättäminen



Kuva 4: Virheiden tyyppiluokittelu.

johtaa usein häiriöön ja se on tällöin major-tason virhe. Virhetilanteen puutteellinen raportointi ohjelmiston lokitiedostoon on esimerkki minor-virheestä.

Käyttäjälle näkyviä häiriöitä aiheuttaneet toiminnalliset virheet muodostivat 8% löydetyistä virheistä. Näihin virheisiin kuuluivat muun muassa virheelliset loogiset vertailut ja rajapintojen virheellinen käyttö. Kaikki toiminnalliset virheet olivat major-tason virheitä.

Pienimmän osuuden muodostivat säieturvallisuuteen liittyvät puutteet, joita oli 2% löydetyistä virheistä. Kaikki säieturvallisuuden puutteet olivat kriittisiä major-tason virheitä, sillä ne aiheuttivat häiriöitä ohjelmiston toiminnassa.

8.3 Vaikutukset ohjelmiston laatuun

Haastatteluun osallistuneilta työntekijöiltä kysyttiin mielipidettä katselmointien vaikutuksesta ohjelmiston sisäiseen ja ulkoiseen laatuun. Ohjelmiston sisäisellä laadulla tarkoitetaan lähdekoodissa olleita virheitä, joita katselmointien aikana löydettiin ja korjattiin. Ohjelmiston ulkoisella laadulla tarkoitetaan ohjelmiston käyttäjälle näkyvien toimintahäiriöiden määrää. Haastateltaville esitettiin sisäisen ja ulkoisen laadun käsitteiden määrittelyt ja pyydettiin pohtimaan katselmointien vaikutusta ohjelmiston laatuun määritelmien mukaisista näkökulmista. Haastateltavilla ei ollut käytössään katselmointien aikana kerättyjä tehokkuusmittoja.

Yksi haastateltava kolmesta totesi, että katselmoinnit olivat ehdottomasti parantaneet ohjelmiston sisäistä laatua. Hänen mukaansa katselmointien käyttäminen osana ohjelmiston laadunvarmistusta esti virheiden syntymistä. Kahden muun haastateltavan mukaan vaikutus ohjelmiston sisäiseen laatuun ei ollut kovin merkittävä tai vaikutusta oli vaikea tunnistaa tutkimuksen aikana. Yksi haastateltava piti vaikutuksia ohjelmiston sisäiseen laatuun hyödyllisinä pitkällä tähtäimellä, sillä katselmointien johdosta ohjelmiston lähdekoodista tuli tyyllisemmin yhtenäisempää kuin ennen katselmoiteja. Toinen haastateltava piti lähdekoodin nimeämiskonventioiden käyttöä hieman kömpelönä, mutta ei nähnyt niiden käyttöä kovin negatiivisena asiana. Osa nimeämiskonventioista nähtiin puhtaasti makuasioina, joilla ei ollut merkittävää vaikutusta ohjelmiston sisäiseen laatuun.

Yksi haastateltava kolmesta piti katselmointien vaikutusta ohjelmiston ulkoiseen laatuun hyvin merkittävänä, yksi piti vaikutusta positiivisena pitemmällä tähtäimellä ohjelmiston sisäisen laadun parannusten takia ja yksi totesi, että vaikutusta on vaikea arvioida. Yksi haastateltava totesi, että vaikka katselmointien aikana saatiin korjattua joitakin häiriöihin johtaneita virheitä, osa muutoksista aiheutti uusia häiriöitä. Kaikki uudet häiriöt eivät olleet virheellisten korjausten tulosta, vaan osa uusista häiriöistä syntyi, kun katselmoinneissa löydettyjen virheiden korjaukset paljastivat uusia ohjelmiston virheitä.

Haastattelujen aikana tuli esille myös ajatus katselmointien soveltamisesta ohjelmistosuunnittelussa. Lähdekoodin katselmointien aikana tuli esille puutteita ohjelmiston rakenteessa, joihin haastateltavien mielestä olisi ollut hyödyllistä puuttua aikaisemmin katselmointien avulla. Suunnittelukatselmointien haasteina todettiin muun muassa ohjelmistosuunnittelun heikko työkalutuki ja suunnittelukatselmointien vaatimat ajalliset resurssit.

8.4 Vaikutukset ohjelmiston tuntemukseen

Kaikki haastateltavat näkivät katselmointien vaikuttaneen positiivisesti katselmoijien tuntemukseen ohjelmiston komponenttien ja ohjelmointikirjastojen toiminnasta. Haastateltavien mukaan katselmointien aikana levisi runsaasti hyväksittyjä koettuja ohjelmointikonventioita kehittäjien välillä. Katselmointiin osallistuneiden kehittäjien välinen kommunikointi ohjelmiston komponenttien toiminnasta sekä ohjelmiston ongelma-alueesta nähtiin hyvin tärkeänä, sillä ohjelmistosta ei juuri ollut olemassa kehittäjille suunnattua dokumentaatiota. Kehittäjien välinen kommunikointi katselmointien aikana korvasi osittain dokumentaation puutteen.

Kaikki haastateltavat pitivät katselmointeja hyödyllisinä usein esiintyvien virheiden ja niiden ratkaisuvaihtoehtojen tuntemuksen leviämässä. Yksi haastateltava piti katselmointeja erityisen tehokkaana keinona kehittäjien tutustuttamisessa uuteen ohjelmointikieleen ja ohjelmointiympäristöön. Hänen mukaansa kehittäjien tutustuttaminen tulee aloittaa mahdollisimman aikaisessa vaiheessa, jotta virheiden syntyminen voidaan estää. Toinen haastateltava totesi, että katselmoinnit toivat esille ohjelmointisääntöjä, jotka eivät olleet ennestään tuntemattomia, mutta jotka olivat olleet unohtuneet aikojen kuluessa. Hän näki katselmoinnit tehokkaana kollektiivisena muistinvirkistäjänä.

8.5 Katselmointiprosessi

Kaikki haastatellut työntekijät pitivät tutkimuksen aikana käytetyn katselmointiprosessin muodollisuusastetta sopivana. Yksi haastateltava kertoi kokemuksistaan entisessä työpaikassaan, jossa käytettiin perinteisiä Fagan-tarkastuksia. Hänen mukaansa tarkastukset koettiin raskaiksi ja tämän johdosta niitä ei järjestetty tarpeeksi usein. Tutkimuksen aikana käyttöön otettu katselmointiprosessi oli hänen mukaansa huomattavasti Fagan-tarkastuksia kevyempi prosessi, minkä johdosta katselmointeja voitiin järjestää useammin.

Työntekijöiltä kysyttiin mielipiteitä katselmoitujen tuotosten kokojen sopivuudesta sekä katselmointien vaatimasta ajasta. Kaikkien haastateltujen mielestä katselmoitujen tuotokset olivat sopivan kokoisia. Yksi haastateltava totesi, että koodirivien määrä ei ole kovin hyvä mittari katselmoitavan tuotoksen koolle, koska katselmoinnin vaatimalla ajalla ja koodin monimutkaisuudella on vahvempi riippuvuus kuin katselmoinnin vaatimalla ajalla ja koodirivien määrällä. Koodin

monimutkaisuudella tarkoitetaan tässä sekä koodin rakenteellista monimutkaisuutta että koodin toteuttamien ohjelmiston toimintojen monimutkaisuutta. Toinen haastateltava korosti katselmoitavan tuotoksen loogisen yhtenäisyyden tärkeyttä koodirivien määrän sijasta. Hänen mukaansa katselmoinnissa tulisi aina katselmoida kerralla yksi looginen kokonaisuus, kuten luokka tai joukko toisiinsa liittyviä luokan metodeita.

Kaikki työntekijät pitivät katselmointeihin käytettyä aikaa hyvin käytettynä ja määrältään sopivana. Yksi haastateltavista totesi, että katselmointien vaatimaa aikaa käytettiin hänen mielestään huomattavasti kustannustehokkaammin kuin hänen edellisessä työpaikassaan, jossa sovellettiin Fagan-tarkastuksia. Katselmointiprosessin koettiin sopivan hyvin yrityskulttuuriin.

Kolmesta haastateltavasta kaksi toimi tutkimuksen aikana katselmoijan roolissa. Molemmat katselmoijat käyttivät ad hoc -lukutekniikkaa, vaikka tarkistuslista oli saatavilla. Lukutekniikasta kysyttäessä haastateltavat kertoivat lukeneensa lähdekooditiedostoja pääosin yhden kerran ylhäältä alas. Katselmoijat rajasivat lukemansa koodin pääosin katselmoitavaan tuotokseen ja seurasivat harvoin esimerkiksi katselmoitavan tuotoksen ulkopuolelle suuntautuvia metodikutsuja. Mikäli metodikutsuja lähdettiin seuraamaan, niitä seurattiin yhden metodikutsun etäisyydelle. Katselmoijat olivat kiinnostuneita oppimaan systemaattisempia ja tehokkaampia lähdekoodin lukutekniikoita.

Yksi haastateltava piti tarkistuslistan merkitystä yrityksen kokemusperäisen tiedon hallintavälineenä suurempana kuin katselmointien lukutekniikkaan vaikuttavana välineenä. Toinen haastateltava totesi, että yrityksen kokemusperäistä tietoa on tarkoituksenmukaisempaa kerätä yrityksen tietojärjestelmään, jotta tarkistuslista pysyy sopivan lyhyenä katselmointeja varten. Haastateltavan mukaan tarkistuslistan tulee mahtua kokonaisuudessaan tietokoneen näytölle.

Kaikki haastateltavat pitivät oman työn ehdottamista katselmoitavaksi hyödyllisenä. Yksikään haastateltava ei pitänyt muiden antamaa kritiikkiä omasta työstään häiritsevänä, mikäli se kohdistui todellisiin virheisiin. Yhden haastateltavan mukaan tuotoksen tekijä saattoi nähdä löydöksiin liitetyt kommentit tylyinä, koska kommentit pyrittiin pitämään lyhyinä. Tuotoksen tekijä saattoi myös kokea katselmointien kohdistuvan häneen henkilökohtaisesti, mikäli katselmointeja järjestettiin pääosin vain hänen tuotoksilleen. Haastateltavan mukaan katselmoinnit tulisikin järjestää tasapuolisesti kaikkien kehittäjien tuotoksille.

8.6 Vaikutukset työkalutukeen

Haastattelujen aikana työntekijöiltä kysyttiin mielipiteitä katselmointien aikana käytetyistä työkaluista. Katselmointien koodin lukemisessa ja löydösten kirjauksessa käytettiin samaa ohjelmointiympäristöä kuin päivittäisessä ohjelmistokehityksessä. Kaikki työntekijät pitivät katselmoinneissa käytettyä ohjelmointiympäristöä riittävänä työkalutukena. Ohjelmointiympäristö oli kaikille kehittäjille ennestään tuttu ja se tarjosi koodin lukemista helpottavia ominaisuuksia, kuten koodin syntaksiväriytyksen ja helpon tavan liikkua koodin eri osien välillä.

Löydösten merkitseminen lähdekoodin sekaan kommentteina oli kaikkien haastateltavien mielestä yksinkertainen ja toimiva tekniikka. Ohjelmointiympäristö ei tarjonnut tukea löydösten merkitsemiselle, joten katselmoijat merkitsivät löydökset joko kirjoittamalla kommentit käsin tai kopioimalla kommentteja muista löydöksistä ja muuttamalla niiden sisältöä. Yksi haastateltava totesi, että löydösten merkitseminen tuntui joskus työläältä. Haastateltavat pohtivat löydösten esittämistä erillään koodista, mutta löydösten ja lähdekoodin välisten viittausten eheyden ylläpitäminen nähtiin haasteellisena.

Muutaman löydöksen kohdalla katselmoijat kävivät keskustelua löydöksestä kirjoittamalla puheenvuoronsa lähdekoodin kommentteihin. Keskustelu koodin kommenttien kautta oli kohdassa 7.1 esitetyn katselmointiprosessin vastaista. Prosessikuvauksen mukaan keskustelut olisi pitänyt käydä yrityksen chatissa. Yksi haastateltava totesikin, että ohjelmointiympäristö ei tarjonnut sopivaa tukea keskusteluun löydöksistä.

Katselmointien aikana löydetty virheet analysoitiin ja uudet virhetyypit kirjattiin tarkistuslistaan. Löydettyjen virheiden perusteella toteutettiin joukko staattisia analyysijä avoimen lähdekoodin Clang-työkaluun [Cla08]. Tutkimuksen aikana kehitetyt analyysit ja Clang-työkalu yhdistettiin AnalysisTool-työkaluksi, joka otettiin käyttöön yrityksen sisällä. AnalysisTool-työkalu julkaistiin ilmaiseksi myös muiden kehittäjien käytettäväksi [Zhu08]. Kaikki haastateltavat pitivät AnalysisTool-työkalua hyödyllisenä. Tutkimuksen aikana saatiin AnalysisTool-työkaluun liittyvää palautetta myös muilta kehittäjiltä. Kaikki palaute on ollut positiivista. Osa tutkimuksen aikana kehitetyistä staattisista analyyseistä siirrettiin Clang-työkalun viralliseen versioon.

8.7 Johtopäätökset tuloksista

Valittu katselmointiprosessi oli tehokas keino parantaa esimerkkiyrityksen ohjelmistotuotteen lähdekoodin laatua. Katselmointien tehokkuutta voidaan perustella sekä määrällisten että laadullisten tulosten avulla. Määrällisten tulosten vertailussa tulee ottaa huomioon, että kirjallisuudessa esitetyt tulokset on usein saatu joko suurissa ohjelmistoyrityksissä tai yliopistoissa järjestetyissä katselmoinneissa, minkä johdosta ne eivät ole suoraan vertailukelpoisia tutkimuksen aikana saatujen tulosten kanssa. Kirjallisuudessa esitettyjä tuloksia käytetään määrällisten tulosten vertailussa, koska ne ovat ainoa saatavilla oleva vertailukohta tutkimuksen tuloksille.

Mikäli katselmointien tehokkuudessa laskemisessa huomioidaan vain major-tason virheet, katselmointien tuloksellisuus noudatti kirjallisuudessa esitettyjä arvoja ja kustannustehokkuus ylitti arvot yli kaksinkertaisesti. Mikäli tehokkuudessa huomioidaan myös minor-tason virheet, tuloksellisuus ylitti kirjallisuudessa esitetyt arvot kolminkertaisesti ja kustannustehokkuus yhdeksänkertaisesti.

Kaikki katselmointeihin osallistuneet esimerkkiyrityksen työntekijät pitivät katselmointeja hyödyllisinä ja katselmointeihin käytettyä työaika perusteltuna. Katselmointien nähtiin parantavan ohjelmiston laatua pitkällä tähtäimellä, sillä katselmoinnit levittivät ohjelmistoon liittyvää tietoa kehittäjien keskuudessa. Kehittäjien mukaan katselmoinnit levittivät tehokkaasti tietoa ohjelmiston eri komponenttien toiminnasta, usein esiintyvistä virheistä ja niiden ratkaisuvaihtoehdoista sekä hyviksi havaituista ohjelmointikonventioista.

Valittu katselmointiprosessi sopi hyvin pienen ohjelmistoyrityksen yrityskulttuuriin. Katselmointeihin osallistuneet työntekijät pitivät katselmointien muodollisuusastetta sopivana. Erityisesti työntekijät arvostivat pakollisten kokousten puutetta.

Katselmoinneissa havaitut ongelmat olivat minor-tason virheiden suuri osuus kaikista löydetyistä virheistä sekä tarkistuslistan vähäinen merkitys katselmoinneissa. Lähes kaikki minor-tason virheet ovat löydettävissä automaattisesti staattisen analyysin työkalun avulla, mikä johdosta minor-tason virheiden osuus kaikista löydetyistä virheistä pienenee, jos yritys jatkaa staattisen analyysin työkalun kehitystä. Yrityksen tulee integroida staattisen analyysin työkalun käyttö kiinteäksi osaksi ohjelmistotuotantoprosessia, jotta työkalun löytämät virheet huomataan ja korjataan ennen katselmointeja.

Tarkistuslistan käyttö jäi katselmoinneissa vähäiseksi. Tarkistuslistaa käytettiin pääosin katselmoinneissa löydettyjen virheiden tyyppien kirjaamiseen. Listan perusteella kehitettiin staattisen analyysin työkalua, jonka avulla automatisoitiin osa katselmoinneissa tehtävistä tarkistuksista. Kaikki katselmointeihin osallistuneet työntekijät ja yrityksen johto pitivät katselmointien tulosten perusteella kehitettyä automaattista tarkistusta hyödyllisenä ja tehokkaana keinona parantaa ohjelmiston laatua.

Yrityksen kokemukseräisen tiedon tallennus tarkistuslistaan on ongelmallista, mikäli lista kasvaa tämän johdosta liian pitkäksi. Yrityksen kokemukseräinen tieto tulee siirtää tarkistuslistasta yrityksen tietojärjestelmään. Tietojärjestelmän avulla voidaan koostaa lyhyitä, tietokoneen näytölle kerrallaan mahtuvia tarkistuslistoja, jotka pitävät sisällään kullekin katselmoijalle sopivia kysymyksiä. Kysymysten valinnassa voidaan käyttää skenaariopohjaisissa lukutekniikoissa käytettäviä ulottuvuuksia.

Tutkimus osoitti, että lähdekoodin katselmointi on tehokas keino ohjelmiston laadunvarmistuksessa pienessä ohjelmistoyrityksessä. Esimerkkiyrityksen johto päätti laajentaa katselmointien käyttöä muihin yrityksen projekteihin. Lähdekoodin katselmoinnit otettiin käyttöön kiinteäksi osaksi yrityksen ohjelmistotuotantoprosessia.

9 Yhteenveto

Katselmointeja on tutkittu tieteellisessä kirjallisuudessa 70-luvulta lähtien. Kirjallisuudessa esitettyjen tulosten perusteella katselmoinnit ovat tehokas ohjelmistojen laadunvarmistusmenetelmä, sillä se mahdollistaa ohjelmiston virheiden löytymisen mahdollisimman aikaisessa vaiheessa. Katselmointeja arvioivat tutkimukset ovat kuitenkin keskittyneet joko yliopistoissa suoritettuihin projekteihin tai vakiintuneita prosesseja käyttäviin keskisuuriin ja suuriin yrityksiin. Pienet ohjelmistoyritykset kohtaavat ohjelmiston laadunvarmistuksen haasteita, joihin katselmoinnit voisivat vastata. Katselmointien soveltamista pienissä ohjelmistoyrityksissä on käsitelty kirjallisuudessa vain vähän, ja kvantitatiivista tietoa katselmointien tehokkuudesta pienissä ohjelmistoyrityksissä ei ole juuri saatavilla. Tutkimuksen tavoitteena oli selvittää, ovatko katselmoinnit tehokas keino parantaa ohjelmistotuotteen laatua pienessä ohjelmistoyrityksessä.

Kirjallisuuskatsauksessa esiteltiin katselmointien historiaa, keskeisiä käsitteitä, kirjallisuudessa kuvattuja katselmointimenetelmiä sekä viimeaikaisen katselmointitutkimuksen pääsuuntauksia. Michael Faganin 70-luvulla kuvaama tarkastus on katselmointimenetelmistä tutkituin ja muodollisin. Tarkastuksissa painotetaan osallistujien roolijaon ja kokousten tärkeyttä tarkastusten tehokkuuden takaamiseksi. Tarkastusten lisäksi on olemassa myös epämuodollisempia katselmointimenetelmiä, joissa roolijaon ja kokousten merkitys on pienempi kuin tarkastuksissa.

Katselmointitutkimuksessa katselmointien teknisen näkökulman tarkastelu on jakautunut neljään teemaan: lukutekniikoihin, katselmointien tehokkuustekijöihin, katselmointiprosesseihin sekä sisältöön liittyviin erityiskysymyksiin. Katselmointitutkimuksessa on erityisesti kyseenalaistettu kokousten asemaa katselmointien tehokkuuden takaajina.

Tutkimusosassa suunniteltiin kirjallisuuskatsauksen tietojen perusteella katselmointiprosessi, josta jätettiin pois pakolliset katselmointikokoukset ja mahdollistettiin seurantakokousten järjestäminen tarpeen mukaan. Katselmointiprosessi otettiin käyttöön pääkaupunkiseudulla sijaitsevassa alle kymmenen hengen ohjelmistoyrityksessä. Tutkimuskysymystä tutkittiin tapaustutkimuksella, jossa selvitettiin, ovatko katselmoinnit tehokas keino parantaa esimerkkiyrityksen ohjelmistotuotteen laatua.

Tutkimus osoitti, että katselmoinnit olivat tehokkaita ja ne vaikuttivat positiivisesti ohjelmistotuotteen laatuun. Yrityksen työntekijät ja johto pitivät katselmointeja hyödyllisinä. Katselmoinnit osoittautuivat tehokkaaksi keinoksi levittää tietoa ohjelmistotuotteesta ja ohjelmointikonventioista kehittäjien välillä. Tutkimukseen osallistunut yritys päätti laajentaa katselmointien käyttöä muihin ohjelmistoprojekteihin.

Tapaustutkimuksen aikana suoritettiin viisi katselmointia. Katselmointikertojen pieni määrä rajaa tutkimustuloksista johdettavia johtopäätöksiä. Yksittäisessä pienessä ohjelmistoyrityksessä havaittua katselmointien vaikutusta ei voida yleistää kaikkiin pieniin ohjelmistoyrityksiin. Mahdollinen aihe jatkotutkimukselle on katselmointikertojen määrän lisääminen ja tutkimuksen toistaminen muissa pienissä ohjelmistoyrityksissä. Tapaustutkimuksessa käytössä ollut katselmointiprosessia voi myös kehittää edelleen ja tutkia muutosten vaikutuksia ohjelmiston laatuun sekä työntekijöiden asenteisiin katselmointeja kohtaan.

Lähteet

- ABL89 Ackerman, A., Buchwald, L. ja Lewski, F., Software inspections: an effective verification process. *IEEE Software*, 6,3(1989), sivut 31–36.
- Amb08 Ambler, S. W., Agile adoption rate survey: February 2008, 2008. [Myös <http://www.ambysoft.com/surveys/agileFebruary2008.html>, 2.7.2008].
- Bas97 Basili, V. R., Evolving and packaging reading technologies. *Journal of Systems and Software*, 38,1(1997), sivut 3–12.
- BoB01 Boehm, B. ja Basili, V., Software defect reduction top 10 list. *Computer*, 34,1(2001), sivut 135–137.
- Bec99 Beck, K., *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, October 1999.
- Ber98 Berman, E., Software inspections at Fermilab - use and experience. *IEEE Transactions on Nuclear Science*, 45,4(1998), sivut 1937–1941.
- BiL89 Bisant, D. ja Lyle, J., A two-person inspection method to improve programming productivity. *IEEE Transactions on Software Engineering*, 15,10(1989), sivut 1294–1304.
- BaM90 Bache, R. ja Mullerburg, M., Measures of testability as a basis for quality assurance. *Software Engineering Journal*, 5,2(1990), sivut 86–92.
- BMW94 Brykczynski, B., Meeson, R. ja Wheeler, D. A., Software inspection: Eliminating software defects. *Proceedings of the Sixth Annual Software Technology Conference*, Alexandria, VA, USA, May 1994, Software Technology Center: Conference and Institute Division.
- BaP94 Barnard, J. ja Price, A., Managing code inspection information. *IEEE Software*, 11,2(1994), sivut 59–69.
- Bry99 Brykczynski, B., A survey of software inspection checklists. *Software Engineering Notes*, 24,1(1999), sivu 82.

- ChH07 Chong, J. ja Hurlbutt, T., The social dynamics of pair programming. *Proceedings of the 29th International Conference on Software Engineering*, Minneapolis, MN, USA, May 2007, IEEE Computer Society, sivut 354–363.
- Cla08 Clang: a C language family frontend for LLVM. <http://clang.llvm.org/>, 29.10.2008.
- CTB06 Cohen, J., Teleki, S. ja Brown, E., *Best Kept Secrets of Peer Code Review*. Smartbearssoftware.com, 2006.
- DCB03 De Almeida Jr, J., Camargo Jr, J., Basseto, B. ja Paz, S., Best practices in code inspection for safety-critical software. *IEEE Software*, 20,3(2003), sivut 56–63.
- Dav89 Davis, F. D., Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 3,13(1989), sivut 319–340.
- Doo92 Doolan, E. P., Experience with Fagan’s inspection method. *Software - Practice and Experience*, 22,2(1992), sivut 173–182.
- DRW03 Dunsmore, A., Roper, M. ja Wood, M., Practical code inspection techniques for object-oriented systems: an experimental comparison. *IEEE Software*, 20,4(2003), sivut 21–29.
- Fag76 Fagan, M. E., Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15,3(1976), sivut 182–211.
- Fag86 Fagan, M. E., Advances in software inspections. *IEEE Transactions on Software Engineering*, 12,7(1986), sivut 744–751.
- Fag02 Fagan, M., A history of software inspections. Teoksessa *Software Pioneers: Contributions to Software Engineering*, Springer-Verlag, New York, NY, USA, 2002, sivut 562–573. [Myös http://www.mfagan.com/software_pioneers.pdf, 22.5.2008].
- FeP98 Fenton, N. E. ja Pfleeger, S. L., *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 1998.
- GiG93 Gilb, T. ja Graham, D., *Software Inspection*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

- Gil00 Gilb, T., Planning to get the most out of inspections. *Software Quality Professional*, 2,2(2000), sivut 7–19.
- GrS94 Grady, R. ja Slack, T., Key lessons in achieving widespread inspection use. *IEEE Software*, 11,4(1994), sivut 46–57.
- HuA05 Hulkko, H. ja Abrahamsson, P., A multiple case study on the impact of pair programming on product quality. *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MO, USA, May 2005, ACM Press, sivut 495–504.
- HBG01 Halling, M., Biffl, S., Grechenig, T. ja Kohle, M., Using reading techniques to focus inspection performance. *Proceedings of the 27th Euro-micro Conference*, Warsaw, Poland, September 2001, IEEE Computer Society, sivut 248–257.
- HiH08 Hirsjärvi, S. ja Hurme, H., *Tutkimushaastattelu*. Gaudeamus Helsinki University Press, 2008.
- HeI06 Hedberg, H. ja Iisakka, J., Technical reviews in agile development: case Mobile-D. *Proceedings of the Sixth International Conference on Quality Software*, Beijing, China, October 2006, IEEE Computer Society Press, sivut 347–353.
- HTH05 Harjumaa, L., Tervonen, I. ja Huttunen, A., Peer reviews in real life - motivators and demotivators. *Proceedings of the Fifth International Conference on Quality Software*, Melbourne, Australia, September 2005, IEEE Computer Society, sivut 29–36.
- HTV04 Harjumaa, L., Tervonen, I. ja Vuorio, P., Using software inspection as a catalyst for SPI in a small company. *Proceedings of the 5th International Conference on Product Focused Software Process Improvement*, Kausai Science City, Japan, April 2004, Springer, sivut 62–75.
- IEE90 IEEE, *IEEE standard glossary of software engineering terminology*, 610.12-1990. IEEE, 1990.
- IEE98 IEEE Computer Society, *IEEE Standard for Software Reviews*, 1028-1997. IEEE Computer Society Press, Los Alamitos, CA, USA, 1998.

- Iis04 Iisakka, J., Inspections in small projects. *Proceedings of SOQUA 2004 (First International Workshop on Software Quality) and TECOS 2004 (Workshop Testing Component-Based Systems)*, Erfurt, Germany, September 2004, Gesellschaft für Informatik, sivut 151–156.
- IiT98 Iisakka, J. ja Tervonen, I., Painless improvements to the review process. *Software Quality Control*, 7,1(1998), sivut 11–20.
- ITH99 Iisakka, J., Tervonen, I. ja Harjumaa, L., Experiences of painless improvements in software inspection. *Proceedings of the 10th European Software Control and Metrics Conference*, Herstmonceux Castle, United Kingdom, April 1999, Shaker Publishing B.V., sivut 321–327.
- Joh94 Johnson, P., An instrumented approach to improving software quality through formal technical review. *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, IEEE Computer Society Press, sivut 113–122.
- KeC99 Kelly, D. ja Culleton, B., Process improvement for small organizations. *Computer*, 32,10(1999), sivut 41–47.
- KKM96 Kusumoto, S., Kikuno, T., ichi Matsumoto, K. ja Torii, K., Experimental evaluation of time allocation procedure for technical reviews. *Journal of Systems and Software*, 35,2(1996), sivut 119–126.
- KnM93 Knight, J. C. ja Myers, E. A., An improved inspection technique. *Communications of the ACM*, 36,11(1993), sivut 51–61.
- Kok06 Kokkonen, J., Experiences from generating checklists. *Proceedings of the Fourth IASTED International Conference on Knowledge Sharing and Collaborative Engineering*, St. Thomas, US Virgin Islands, November/December 2006, IASTED, ACTA Press, sivut 51–62.
- Kol06 Kollanus, S., *Ohjelmistojen tarkastuskäytänteiden puutteet ja ongelmat teoriassa ja käytännössä*. Väitöskirja, Jyväskylän yliopisto, 2006.
- LaD97 Laitenberger, O. ja DeBaud, J.-M., Perspective-based reading of code documents at Robert Bosch GmbH. *Information and Software Technology*, 39,11(1997), sivut 781–791.

- LaD00 Laitenberger, O. ja DeBaud, J.-M., An encompassing life cycle centric survey of software inspection. *The Journal of Systems and Software*, 50,1(2000), sivut 5–31.
- Lee97 Lee, E., Software inspections: How to diagnose problems and improve the odds of organizational acceptance. *Crosstalk*, 10,8(1997), sivut 10–13.
- LEH01 Laitenberger, O., El Emam, K. ja Harbich, T., An internally replicated quasi-experimental comparison of checklist and perspective based reading of code documents. *IEEE Transactions on Software Engineering*, 27,5(2001), sivut 387–421.
- LGV93 Lawrence G. Votta, J., Does every inspection need a meeting? *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, Los Angeles, CA, USA, December 1993, ACM, sivut 107–114.
- Lus04 Lussier, S., New tricks: How open source changed the way my team works. *IEEE Software*, 21,1(2004), sivut 68–72.
- Mul04 Müller, M., Are reviews an alternative to pair programming? *Empirical Software Engineering*, 9,4(2004), sivut 335–351.
- McC96 McConnell, S., Software quality at top speed. *Software Development*, 4,8(1996), sivut 38–42.
- McC01 McConnell, S., An ounce of prevention. *IEEE Software*, 18,3(2001), sivut 5–7.
- McC04 McConnell, S., *Code Complete*. Microsoft Press, Redmond, WA, USA, 2004.
- Met06 Metsämuuronen, J., *Tutkimuksen tekemisen perusteet ihmistieteissä*. International Methelp Ky, 2006.
- MFR94 Mashayekhi, V., Feulner, C. ja Riedl, J., CAIS: Collaborative Asynchronous Inspection of Software. *Software Engineering Notes*, 19,5(1994), sivut 21–34.
- MuP03 Müller, M. M. ja Padberg, F., On the economic evaluation of XP projects. *Software Engineering Notes*, 28,5(2003), sivut 168–177.

- MPS96 McCarthy, P., Porter, A., Siy, H. ja Votta, L.G., J., An experiment to assess cost-benefits of inspection meetings and their alternatives: a pilot study. *Proceedings of the 3rd International Software Metrics Symposium*, Berlin, Germany, March 1996, IEEE Computer Society, sivut 100–111.
- MVC08 McMeekin, D. A., Von Kinsky, B. R., Chang, E. ja Cooper, D. J. A., Checklist based reading's influence on a developer's understanding. *Proceedings of the 19th Australian Conference on Software Engineering*, Perth, Australia, March 2008, IEEE Computer Society, sivut 489–496.
- MWR98 Miller, J., Wood, M. ja Roper, M., Further experiences with scenarios and checklists. *Empirical Software Engineering*, 3,1(1998), sivut 37–64.
- Mye78 Myers, G. J., A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM*, 21,9(1978), sivut 760–768.
- Paa05 Paasivaara, M., *Communication Practices in Inter-Organisational Product Development*. Väitöskirja, Helsinki University of Technology, 2005.
- PFV04 Pierfrancesco Fusaro, F. L. ja Visaggio, G., A replicated experiment to assess requirements inspection techniques. *Empirical Software Engineering*, 2,1(2004), sivut 39–57.
- PoJ97 Porter, A. A. ja Johnson, P. M., Assessing software review meetings: Results of a comparative analysis of two experimental studies. *Software Engineering*, 23,3(1997), sivut 129–145.
- PST97 Porter, A., Siy, H., Toman, C. A. ja Votta, L. G., An experiment to assess the cost-benefits of code inspections in large scale software development. *IEEE Transactions on Software Engineering*, 23,6(1997), sivut 329–346.
- PoV94 Porter, A. ja Votta, L., An experiment to assess different defect detection methods for software requirements inspections. *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, IEEE Computer Society, sivut 103–112.

- PoV97 Porter, A. ja Votta, L., What makes inspections work? *IEEE Software*, 14,6(1997), sivut 99–102.
- PVB95 Porter, A., Votta, L.G., J. ja Basili, V., Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering*, 21,6(1995), sivut 563–575.
- PaW87 Parnas, D. L. ja Weiss, D. M., Active design reviews: principles and practices. *Journal of Systems and Software*, 7,4(1987), sivut 259 – 265.
- PJS97 Pek Wee Land, L., Jeffery, R. ja Sauer, C., Validating the defect detection performance advantage of group designs for software reviews: report of a replicated experiment. *Proceedings of the Australian Software Engineering Conference*, Sydney, NSW, Australia, October 1997, IEEE Computer Society, sivut 17–26.
- RAV96 Rooijmans, J., Aerts, H. ja van Genuchten, M., Software quality in consumer electronics products. *IEEE Software*, 13,1(1996), sivut 55–64.
- RDN04 Rodgers, T., Dean, D. ja Nunamaker, J.F., J., Increasing inspection efficiency through group support systems. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, USA, January 2004, IEEE Computer Society, sivut 1–10.
- RiG07 Richardson, I. ja Gresse von Wangenheim, C., Why are small software organizations different? *IEEE Software*, 24,1(2007), sivut 18–22.
- Roy70 Royce, W. W., Managing the development of large software systems: Concepts and techniques. Tekninen raportti, Western Electronic Show and Convention (WESCON), Los Angeles, August 1970.
- Rus91 Russell, G., Experience with inspection in ultralarge-scale development. *IEEE Software*, 8,1(1991), sivut 25–31.
- SBB02 Shull, F., Basili, V., Boehm, B., Brown, A., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R. ja Zelkowitz, M., What we have learned about fighting defects. *Proceedings of the Eighth IEEE Symposium on Software Metrics*, Ottawa, Canada, June 2002, IEEE Computer Society, sivut 249–258.

- SBK98 Sandahl, K., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M. ja Ohlsson, N., An extended replication of an experiment for assessing methods for software requirements inspections. *Empirical Software Engineering*, 3,4(1998), sivut 327–354.
- SJL00 Sauer, C., Jeffery, D., Land, L. ja Yetton, P., The effectiveness of software development technical reviews: a behaviorally motivated program of research. *IEEE Transactions on Software Engineering*, 26,1(2000), sivut 1–14.
- SiV01 Siy, H. ja Votta, L., Does the modern code inspection have value? *Proceedings of the IEEE International Conference on Software Maintenance*, Florence, Italy, November 2001, IEEE Computer Society, sivut 281–289.
- TIH98 Tervonen, I., Iisakka, J. ja Harjumaa, L., Software inspection – a blend of discipline and flexibility. *Proceedings of Fifth Conference for the European Network of Clubs for Reliability and Safety of Software*, Rome, Italy, May 1998, Shaker Publishing B.V., sivut 157–166.
- TPR04 Thelin, T., Petersson, H., Runeson, P. ja Wohlin, C., Applying sampling to improve software inspections. *Journal of Systems and Software*, 73,2(2004), sivut 257–269.
- UNM06 Uwano, H., Nakamura, M., Monden, A. ja Matsumoto, K., Analyzing individual performance of source code review using reviewers' eye movement. *Proceedings of the 2006 symposium on Eye tracking research & applications*, San Diego, CA, USA, March 2006, ACM, sivut 133–140.
- VVS01 Van Genuchten, M., Van Dijk, C., Scholten, H. ja Vogel, D., Using group support systems for software inspections. *IEEE Software*, 18,3(2001), sivut 60–65.
- Van99 Van Veenendaal, E. P., Practical quality assurance for embedded software. *Software Quality Professional*, 1,3(1999).
- Wel93 Weller, E., Lessons from three years of inspection data. *IEEE Software*, 10,5(1993), sivut 38–45.

- Wie01 Wiegers, K. E., *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- Zhu08 Zhuk, N., AnalysisTool: a GUI wrapper for LLVM/Clang static analyzer. <http://www.karppinen.fi/analysistool/>, 29.10.2008.

Liite 1. Katselmointiprosessin kuvaus

Alla on esimerkkiyritykselle toimitettu prosessikuvaus. Prosessikuvauksesta on poistettu yritykseen, sen tuottamiin ohjelmistoihin tai sen käyttämiin työkaluihin viittaavat nimet.

1. Roolit: tuotoksen tekijä, katselmointivastaava (KV), muut katselmoijat.
2. Tekijä ja/tai KV päättää, mitä katselmoidaan. Kerralla max. 300 ELOC.
3. Tuotoksen on täytettävä esiehdot (koodi kääntyy, staattisen analyysin virheet korjattu). Jos ei täytä, katselmointia ei aloiteta. KV varmistaa tämän.
4. KV ja tekijä valitsevat katselmoijat (KV + tekijä + muut).
5. KV tekee katselmointityömääräimet (yleinen työmääräin sekä henkilökohtaiset työmääräimet). Henkilökohtaisiin työmääräimiin merkitään deadline.
6. Katselmoija aloittaa katselmoinnin silloin, kun hänelle sopii, kuitenkin ennen deadlineä. Katselmoija avaa henkilökohtaisen työmääräimensä ja aloittaa työajan laskennan aktivoimalla työmääräimen työaikalaskurin.
7. Katselmoija hankkii puhtaan työkopion versiohallinnasta.
8. Katselmoija suorittaa katselmoinnin itsenäisesti IDEssä. Löydökset merkitään suoraan lähdekoodin yhteyteen kommentteina, joiden tekeminen kannattaa tallentaa IDE:n makrokksi.
9. Voi käyttää muuta koodia, dokumentaatiota, työkaluja, Googlea jne.
10. Löydöksistä voi käydä keskustelua chatissa, max. muutama min / löydös.
11. Keskitytään virheisiin tai epäselviin ilmauksiin.
12. Ei ehdoteta vaihtoehtoisia toteutustapoja, elleivät ne paranna selvästi koodin luettavuutta tai ylläpidettävyyttä.
13. Etsitään virheitä, ei niiden ratkaisuja. Jos ratkaisu on triviaali, merkitään se löydöksen yhteyteen. Koodia ei muuteta, jätetään se tekijän vastuulle.

14. Katselmoinnin jälkeen katselmoija tallentaa muuttuneet tiedostot versiohallintaan. Jos tallennuksen aikana tulee konflikti, se resolvataan yhdessä konfliktivien löydösten kirjoittajien kanssa.
15. Tallennuksen jälkeen katselmoija lopettaa työmääräimen ajanlaskennan. Työmääräin suljetaan statuksella Completed.
16. Kun KV on saanut kaikkien katselmoijien kommentit (kaikkien status Completed), niin hän assignaa yleisen katselmointityömääräimen tuotoksen tekijälle, jonka vastuulla on ottaa kantaa jokaiseen löydökseen (esim. hylätä ja kertoa miksi hylkäsi tai korjata löydökset). Korjaukset voi tehdä joko heti tai (jos kyse on isommasta virheestä) luoda niistä uudet työmääräimet ja merkitä viitteet niihin löydösten kommenttien paikalle.
17. Tekijä voi kysyä löydöksistä katselmoijilta chatissa. Jos kyse on esim. väärin ymmärretystä koodista, tuotoksen tekijä voi harkita parantavansa koodin luettavuutta tai dokumentointia.
18. Tekijä korjaa löydetyt virheet ja tallentaa uuden version.
19. Kun tekijä on valmis, hän assignaa työmääräimen takaisin KV:lle.
20. KV tarkistaa, että kaikkiin löydöksiin on reagoitu.
21. KV ja tekijä päättää muiden katselmoijien kanssa, tarvitaanko seurantakokousta. Kaikkien ei ole pakko osallistua.
22. Jos halutaan pitää kokous, se voidaan pitää chatissä tai fyysisesti.
23. Kokouksessa käydään läpi löydetyt virheet ja mietitään, voitaisiinko niiden syntymistä estää esim. muuttamalla konventioita tai tekemällä tarkistuksista automaattisia. Päivitetään tarvittaessa tarkistuslistaa.

Liite 2. Haastattelun teemat

Tässä esitellään teemahaastattelun keskeiset teemat. Haastattelun aikana käsiteltiin myös muita teemoja, jotka nousivat keskustelussa esille haastateltavien toimesta.

1. Vaikutus lähdekoodin virheisiin.
2. Vaikutus ohjelmiston häiriöihin.
3. Vaikutus ohjelmiston komponenttien toiminnan tuntemukseen.
4. Vaikutus yleisten lähdekoodin virheiden ja niiden ratkaisujen tuntemukseen.
5. Vaikutus virheiden ennaltaehkäisyyn.
6. Katselmointien vaatima aika.
7. Kerralla katselmoitavan tuotoksen koko.
8. Tarkistuslistan merkitys katselmointien aikana sekä yrityksen kokemuspärisen tiedon hallinnassa.
9. Katselmointien aikana käytetty lukutekniikka.
10. Työkalutuki.
11. Oman työn ehdottaminen katselmoitavaksi, työylpeys, ego.
12. Katselmointiprosessin muodollisuus.
13. Katselmoinnit osana ohjelmistotuotantoprosessia.
14. Havaitut ongelmat ja parannusehdotukset.

Liite 3. Tilastolliset testit

Katselmointinopeus ja tuloksellisuus

Nollahypoteesi H_0 : Katselmointinopeuden ja katselmoinnin tuloksellisuuden välinen korrelaatiokerroin on 0.

Vaihtoehtoinen hypoteesi H_1 : Katselmointinopeuden ja katselmoinnin tuloksellisuuden välinen korrelaatiokerroin on erisuuri kuin 0.

Testin merkitsevyytaso: 0.05.

Otoksen havaintojen lukumäärä $N = 9$.

Otoksen korrelaatiokerroin $r = -0.44$.

Testisuure: $t = [-0.44 * \sqrt{9 - 2}] / (1 - (-0.44)^2)$

$t = -1.3057$

$P = 0.2329$

Kaksisuuntaisen t-testin P-arvo on 0.2329. Otoksessa huomattu katselmointinopeuden ja katselmoinnin tuloksellisuuden välinen korrelaatio ei ole tilastollisesti merkitsevä, koska testin P-arvo on suurempi kuin valittu merkitsevyytaso 0.05. Nollahypoteesi jää voimaan.

Katselmointinopeus ja kustannustehokkuus

Nollahypoteesi H_0 : Katselmointinopeuden ja katselmoinnin kustannustehokkuuden välinen korrelaatiokerroin on 0.

Vaihtoehtoinen hypoteesi H_1 : Katselmointinopeuden ja katselmoinnin kustannustehokkuuden välinen korrelaatiokerroin on erisuuri kuin 0.

Testin merkitsevyytaso: 0.05.

Otoksen havaintojen lukumäärä $N = 9$.

Otoksen korrelaatiokerroin $r = 0.68$.

Testisuure: $t = [0.68 * \sqrt{9 - 2}] / (1 - (0.68)^2)$

$t = 2.4399$

$P = 0.0448$

Kaksisuuntaisen t-testin P-arvo on 0.0448. Otoksessa huomattu katselmointinopeuden ja katselmoinnin kustannustehokkuuden välinen korrelaatio on tilastollisesti merkitsevä, koska testin P-arvo on pienempi kuin valittu merkitsevyytaso 0.05. Nollahypoteesi hylätään ja vaihtoehtoinen hypoteesi hyväksytään.