hyväksymispäivä arvosana

arvostelija

Virheiden etsintä: katselmoinnit vai testaaminen?

Timo Usenius

Helsinki 10.11.2009
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion – Faculty/Section	Laitos – Institution – Department	
Matemaattis luonnontieteellinen	Tietojenkäsittelytieteen laitos	
Tekijä – Författare – Author		
Timo Sakari Usenius		
Työn nimi – Arbetets titel – Title		
Virheiden etsintä: katselmoinnit vai testaaminen?		
Oppiaine – Läroämne – Subject		
Tietojenkäsittelytiede		
Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Seminaariesitys	10.11.2009	16

Tiivistelmä – Referat – Abstract

Ohjelmistotuotannon tarkoituksena on kehittää laadukkaita ohjelmistotuotteita, jotka tuottavat lisäarvoa asiakkaille. Laadukkaat ohjelmistot täyttävät niille määritellyt vaatimukset ja toimivat mahdollisimman virheettömästi. Ohjelmistotuotannossa projektin onnistumisen kannalta keskeisessä osassa on virheiden ja puutteiden havaitseminen mahdollisimman aikaisessa vaiheessa, jolloin niistä koituvat kustannukset jäävät mahdollisimman pieniksi. Virheiden ja puutteiden etsintä vaatii kuitenkin suhteellisen paljon resursseja. Niiden etsintätyö on suunniteltava mahdollisimman kustannustehokkaaksi, jotta prosessi olisi kannattava myös tuottajan näkökulmasta. Virheiden ja puutteiden havaitsemiseksi on olemassa monia erilaisia verifiointi ja validointitekniikoita. Niiden hyvyys ja paremmuus ohjelmistotuotannon eri vaiheissa määräytyy sen mukaan miten hyvin virheet ja puutteet löytyvät suhteessa käytettyihin resursseihin. Tässä kirjoituksessa tarkastellaan ja verrataan testausta ja katselmointeja, jotka ovat kaksi yleisimmin käytetyistä verifiointi ja validointitekniikoista, sekä esitellään tutkimuksia, joissa on selvitetty niiden tehokkuutta virheiden löytämisessä. Tutkimuksissa keskeisinä kysymyksinä ovat olleet 1) virheiden ja puutteiden löytyminen verifiointi ja validointiprosesseissa, 2) etsintään käytettyjen resurssien määrä ja 3) virheidenetsintäprosessin parantaminen ja tehostaminen.

ACM Computing Classification System (CCS):
D.2.4 [**Software Engineering**]: Software/Program verification
D.2.5 [**Software Engineering**]: Testing and debugging

Avainsanat – Nyckelord – Keywords

Ohjelmistovirhe, virhe, testaus, katselmointi, tarkastus

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Ohjelmiston elinkaari muodostuu seuraavista vaiheista: tarve ohjelmistolle, vaatimusten määrittely, arkkitehtuurisuunnittelu, yksityiskohtaisempi suunnittelu, toteutus, testaus, asennus toimintaympäristöönsä, käyttökokeet, käyttö, käyttäjien kokemusten kerääminen ohjelmiston jatkokehitys ja niiden dokumentointi. Kaikissa näissä vaiheissa voi tapahtua erilaisia virheitä, jotka vaikuttavat tuotettavan ohjelmiston laatuun, toimivuuteen, tuotanto- ja käyttökustannuksiin.

Virheiden etsinnällä ja korjauksella parannetaan tuotteen ja tuotantoprosessin laatua. Virheet määritellään ohjelmistotuotteen, sen osien tai dokumentoinnin tiloiksi, jotka poikkeavat odotetuista tiloista [IEE93]. Tavoitetilat esitetään muun muassa määrittely- ja suunnitteludokumenteissa sekä käyttö-ohjeissa [IEE93]. Virheitä voi syntyä ja niitä voidaan löytää eri vaiheissa tuotantoprosessia. Virheiden löytämisessä ensi sijalla on nopeus, sillä esimerkiksi virheiden löytäminen ja korjaaminen tuotteesta sen julkaisun jälkeen on usein jopa 100 kertaa kalliimpaa, kuin sen havaitseminen ja korjaaminen tuotteen määrittely tai suunnitteluvaiheessa [BoB01].

Virheiden ehkäisyyn käytetäänkin huomattava osa projektien resursseista, mistä johtuen myös virheiden ehkäisytyön tulisi olla suunnitelmallista ja perustua hyviksi havaittuihin menetelmiin. Virheet voidaan luokitella myös niiden vaikutusten mukaan. Virheet voivat aiheuttaa ohjelman suorituksen aikana häiriöitä tai ne voivat olla kokonaan vaikuttamatta ohjelman suoritukseen. Erään arvion mukaan noin 80 prosenttia ehkäistävissä olevasta korjaustyöstä johtuu vain viidesosasta virheitä [BoB01]. Virheiden etsinnässä ja ehkäisemisessä tulisi ensi sijaisesti kiinnittää huomiota siihen, miten nämä eniten kustannuksia aiheuttavat virheet voitaisiin projektin aikana välttää.

Virheiden löytämiseksi ja projektien laadun parantamiseksi on olemassa monia erilaisia menetelmiä, joiden paremmuutta virheiden etsintäkeinoina ja ohjelmistoprojektin laadun parantajina on tutkimuksilla pyritty selvittämään. Tässä kirjoituksessa verrataan toisiinsa kahta yleisimmin käytettyä menetelmää jotka ovat katselmointi (*review*) ja testaus. Katselmoinnit ovat staattisia virheiden etsintäkeinoja, jotka eivät tarvitse suoritettavaa ohjelmanosaa kuten testaus. Katselmointien osalta kirjoitus keskittyy tutkimuksissa käytettyihin tarkastuksiin (*inspection*) ja läpikäynteihin (*walk-through*), joista on olemassa monia erilaisia muunnelmia. Toinen perinteinen virheiden etsintämenetelmä testaus käsittää toiminnallisen testauksen (*functional testing*) ja rakenteellisen testauksen (*structural testing*).

2 Virheiden etsintä ohjelmistotuotannossa

Virheiden määrällä on suora vaikutus ohjelmiston ja sen tuotantoprosessin laatuun. Virheiden etsintää ja hallintaa suunniteltaessa on olennaista tietää minkälaisia virheitä ohjelmiston tuotannon ja ohjelmiston käytön aikana voi ilmetä. Tässä luvussa kuvataan millaisia virheitä ohjelmiston elinkaaren aikana etsitään ja pyritään välttämään.

2.1 Virheet ja niiden luokittelu

Tässä kirjoituksessa virheeksi kutsutaan kaikkia mahdollisia ohjelmiston ja dokumenttien virheellisiä toimintoja, poikkeamia ja puutteita, joita ohjelmistotuotteen tuottamisen ja sen elinkaaren aikana voi ilmetä. Virheitä käsittelevissä kirjoituksissa virheitä on luokiteltu eri tavoilla. Yksinkertaisimmillaan virheet voidaan määritellä niin, että jotakin puuttuu (*omission*) tai että jotakin on liikaa tai virheellistä (*commission*) [JuV03].

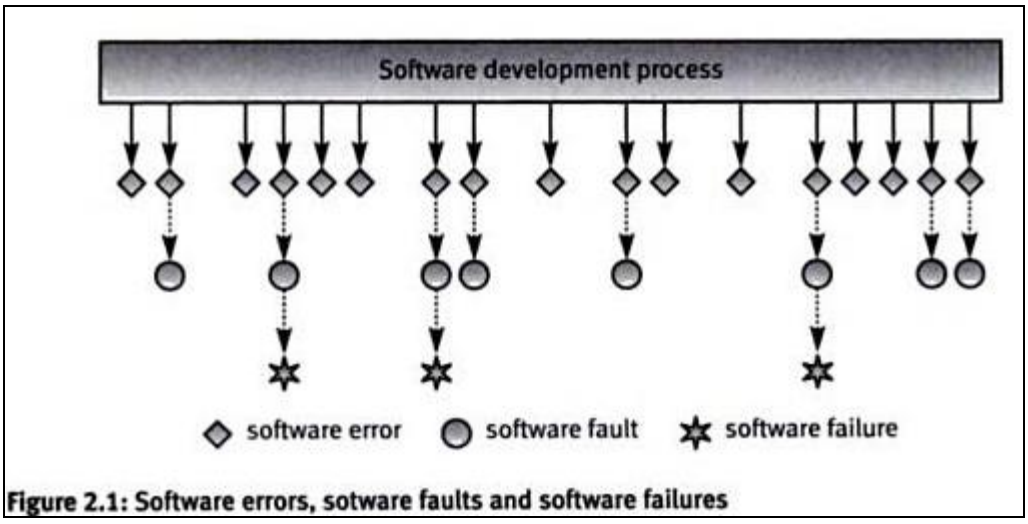
Seuraavassa on esitetty syitä virheiden esiintymiselle ohjelman elinkaaren aikana [Gal04]:

- Väärin määritellyt vaatimukset aiheuttavat suuren osan virheistä. Määrittelyjä puuttuu, ne ovat osiltaan vajaita tai vaatimukset ovat kokonaan tarpeettomia.
- Väärin ymmärrykset kommunikoinnissa asiakkaan ja tuottajan välillä, jolloin tehdään asioita eri tavalla kuin asiakas oli tarkoittanut.
- Tuottaja poikkeaa tarkoituksella alkuperäisistä vaatimuksista ja suunnitelmista. Esimerkiksi komponenttien uudelleenkäytön toimivuus uudessa ohjelmistossa jätetään tarkastamatta eli toteuttaako se kaikki uudet vaatimukset. Aikataulu ja budjetti voi myös painostaa jättämään toteuttamatta osaa vaatimuksista.
- Erilaiset suunnitteluvirheet esimerkiksi ohjelmiston arkkitehtuurin suunnittelussa voi aiheuttaa virheitä.
- Ohjelmoinnin yhteydessä tapahtuvat virheet.
- Dokumentoinnin virheet (kommentoimaton koodi, sekavat dokumentit tai kokonaan puuttuva dokumentointi) vaikeuttavat virheiden korjausta, ylläpitoa ja myöhempää tuotteen kehitystä.
- Testausprosessin puutteellisuus, jonka johdosta ohjelmistoon jää virheitä. Testauksen suunnitelma on tehty puutteellisesti tai sitä ei ole noudatettu.
- Käyttötapausten tulkintavirheet. Käyttötapaus on ymmärretty väärin, jolloin ohjelma toimii eri tavalla kuin oli tarkoitettu.

2.2 Virheiden vaikutus

Virheitä (*error*) voidaan tarkastella niiden vaikutusten perusteella ohjelman suorituksen aikana. Suurin osa ohjelmistovirheistä on sellaisia, etteivät ne mitenkään vaikuta ohjelman toimintaan. Esimerkiksi ne ovat sellaisessa koodin osassa, jota ei koskaan suoriteta. Kun virheellinen ohjelmakohta suoritetaan se saa ohjelman virheelliseen tilaan. Virheellisen kohdan suoritus ei kuitenkaan välttämättä näy käyttäjälle asti virheellisenä tuloksena, sillä jokin muu ohjelman toiminto voi korjata sen [HaM06]. Tällöin on kyseessä ohjelmistovika (*fault*). Vain osa virheistä näkyy käyttäjälle ohjelmiston toimintahäiriönä (*failure*). Virheiden, vikojen ja häiriöiden esiintyminen ohjelmistoprojektin aikana esitetään kuvassa 1 [Gal04].

Nykyisissä ohjelmistoprojekteissa käytetään suuri osa (jopa 50%) resursseista uudelleen tehtävään työhön, joka olisi ollut ennalta ehkäistävissä paljon korjaustyötä vaativien virheiden aiemmalla havaitsemisella ja korjaamisella [BoB01]. Ohjelmistoprojektien kehittämisessä kannattaakin keskittyä tällaisten suurta uudelleen tehtävää työtä vaativien virheiden ennalta ehkäisyyn.



Kuva 1: Virhe (*error*), vika (*fault*) ja häiriö (*failure*) ohjelmistoprosessin aikana [Gal04].

2.3 Virheiden etsimiskeinot

Virheiden jäljittämiseen on olemassa monia erilaisia keinoja. Staattisissa keinoissa virheitä etsitään erilaisista dokumenteista tarkastamalla niitä visuaalisesti jonkin käytännön mukaan. Dynaamisessa virheiden etsinnässä ohjelmakoodia suoritetaan valituilla testitapauksilla ja tuloksia verrataan odotettuihin tuloksiin.

Seuraavien keinojen avulla voidaan virheitä havaita ja ehkäistä [IEE93]:

- Katselmoinnit (monia erilaisia variaatioita perinteisestä tekniikasta)
- Testaus (Rakenteellinen ja toiminnallinen)
- Kääntäjän ilmoittamat virheet
- Ohjelman käytön aikana ilmenevät virheet.
- Analysoimalla edellisten projektien virheitä voidaan niitä ehkäistä tulevissa projekteissa

Tässä kirjoituksessa keskitytään tarkastelemaan virheiden etsintää testauksen ja katselmoinnin avulla, joista on tarkemmin luvuissa 3, 4 ja 5.

3 Katselmoinnit

Katselmoinnilla (*review*) tarkoitetaan prosessia tai tilaisuutta, jossa jokin projektin tuotos esitellään eri sidosryhmille ja se asetetaan tarkastuksen, arvioinnin, kommentoinnin tai hyväksymisen kohteeksi [IEE08]. Katselmointi on staattinen virheiden- ja puutteiden etsintämenetelmä, joka perustuu dokumenttien tarkastamiseen visuaalisesti. Katselmoitavat tuotokset ovat esimerkiksi vaatimusten määrittelydokumentti, suunnitteludokumentti, arkkitehtuurisuunnitelma, testaussuunnitelma, lähdekoodit ja niin edelleen. Katselmoinnin suorittavat henkilöt kuuluvat eri sidosryhmiin, jotka ovat tekemisissä tuotoksen kanssa. Sidosryhmillä tarkoitetaan tässä tapauksessa muun muassa projektihenkilökuntaa, johtoa, asiakkaita, testaajia, käyttäjiä sekä muita henkilöitä joita kyseinen tuotos koskettaa tai kiinnostaa [IEE08].

Seuraavassa luetellaan eri katselmointimenettelyjä [IEE08]:

- Johdon katselmus (*management review*), jossa tarkastetaan ja arvioidaan ohjelmistotuote tai tuotantoprosessin tila
- Tekninen katselmus (*technical review*) on ohjelmistotuotteen systemaattinen arviointi sen sopivuudesta käyttötarkoitukseensa. Katselmoinnin suorittaa siihen pätevöitynyt tiimi.
- Laatujärjestelmän arviointi (*audit*) on yrityksen laatujärjestelmän arviointiprosessi.
- Tarkastus (*inspection*). Tarkastusta käsitellään tarkemmin luvussa 3.1.
- Läpikäynti (*walk-through*) on lähinnä koodin läpikäymistä ryhmissä, joissa koodista etsitään virheitä koodin tekijän johdolla.

Katselmointeja ja testaamista vertaavissa tutkimuksissa on käytetty variaatioita tarkastuksista ja läpikäynneistä, joiden periaatteet käydään läpi tarkemmin.

3.1 Tarkastus

Tarkastus on eräs katselmoinnin tekniikka, jota voidaan käyttää hyvin monessa ohjelmistoprojektin vaiheessa. Sen avulla pyritään etsimään virheitä ja tarkastamaan kohteena olevan tuotoksen laatu systemaattisen tarkastusprosessin avulla. Tarkastusprosessissa löytyneitä virheitä analysoimalla voidaan myös tulevien tuotosten, tarkastusprosessin ja myös koko tuotantoprosessin laatua parantaa [Gil00]. Yleisimmin tarkastuksia käytetään määrittely- ja suunnitteludokumenttien yhteydessä.

Tarkastusprosessille on olemassa erilaisia muunneltuja malleja (kuva 2), jotka perustuvat pääsääntöisesti Michael Faganin 1976 esittämään tarkastusprosessin malliin. Tarkastusta voidaan käyttää monessa vaiheessa ohjelmistoprosessia. Tarkastusten avulla voidaan havaita monia virheitä jo ennen kuin testausta on mahdollista käyttää, esimerkiksi virheellisiä vaatimusmäärittelyitä. Tällöin ne voidaan korjata jo ennen tuotantovaihetta, mikä suoraan alentaa korjauskustannuksia.

Tarkastusprosessi on systemaattinen prosessi, johon osallistuu useampia henkilöitä (4-6), jotka ovat tekemisissä tarkastettavan tuotoksen kanssa. Henkilöillä on määrätyt roolit (puheenjohtaja, sihteeri, tuotoksen tekijä - alustaja, tarkastajat) tarkastuksessa, joiden mukaan he toimivat. Puheenjohtaja johtaa tarkastusta. Sihteeri toimii kirjanpitäjänä. Tuottaja vastaa tarkastettavasta tuotoksesta ja voi vastata sitä koskeviin kysymyksiin. Tarkastajat etsivät tuotoksesta mahdollisia virheitä.

Tarkastusprosesseille on esitelty monia muunnelmia, mutta perusteiltaan ne pohjautuvat Faganin esittämään malliin. Seuraavassa kuvataan tarkastusprosessin keskeiset vaiheet [Gil00]:

1: Tarkastuksen suunnittelu. Puheenjohtaja valitse tarkastusryhmän ja valitsee tarkastettavan materiaalin hyvissä ajoin ennen tarkastustilaisuutta.

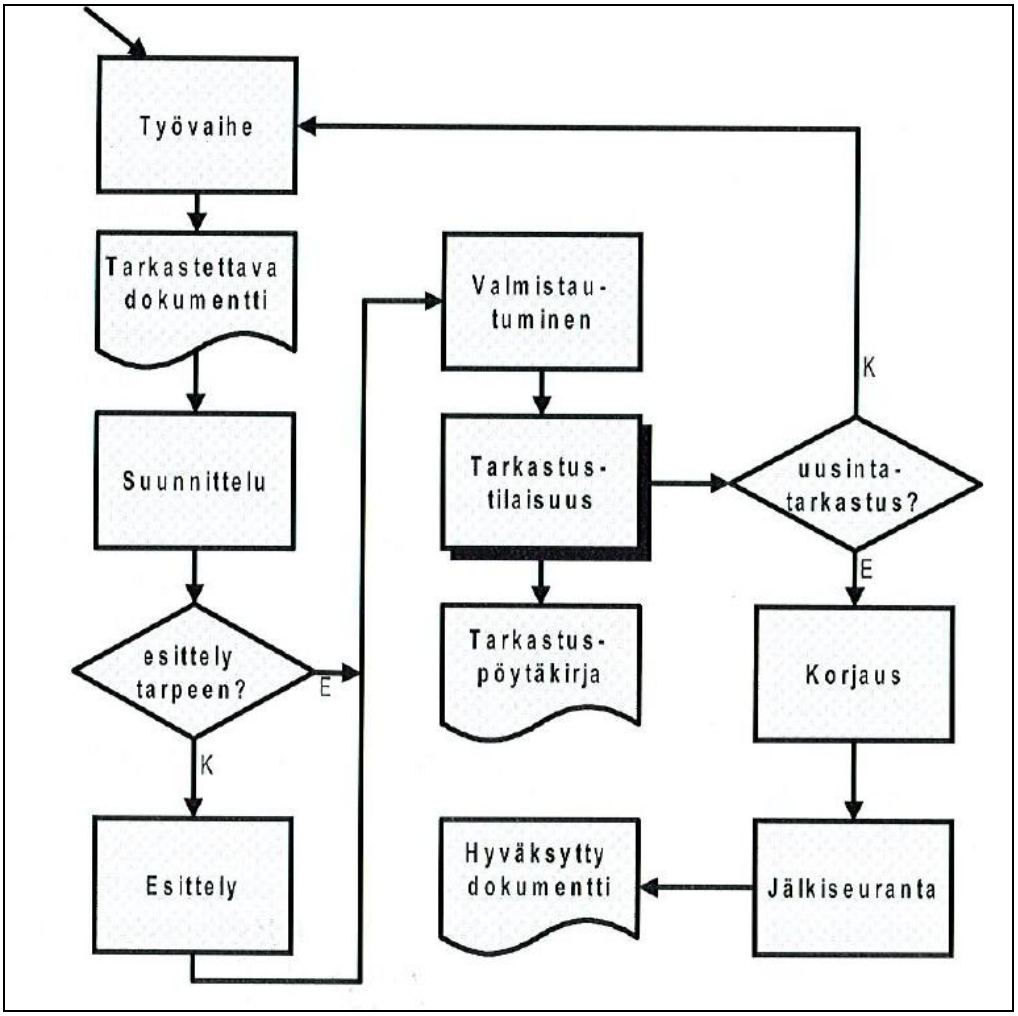
2: Esittelykokous. Tarkastuksen kohteena olevan tuotoksen tuottaja esittelee materiaalin tarkastusryhmälle.

3: Valmistautuminen. Tarkastusryhmän jäsenet tutustuvat materiaaliin ennen tarkastustilaisuutta ja pyrkivät etsimään siitä virheitä.

4: Tarkastustilaisuus. Tarkastustilaisuus on hyvin muodollinen tilaisuus, jotta siinä keskityttäisiin vain olennaiseen asiaan eli kirjaamaan materiaalista löytyneet virheet.

Tarkastustilaisuudessa puheenjohtajan johdolla käydään materiaali läpi sihteerin kirjatessa havaittuja virheitä ylös. Tilaisuus päättyy tuotoksen hyväksymiseen sellaisenaan, hyväksymiseen korjattuna tai hylkäämiseen jolloin järjestetään uusi tarkastus.

5: Jälkitoimet. Vaaditut korjaukset suoritetaan ja järjestetään mahdollinen uusi katselmointi. Analysoimalla tarkastuksessa paljastuneita virheitä voidaan niitä tulevaisuudessa välttää ja näin parantaa ohjelmistoprosessin laatua.



Kuva 2. Tarkastusprosessin vaiheet Haikala et al. mukaan [HaM06].

3.2 Läpikäynti

Läpikäynti ei ole niin muodollinen katselmointikäytäntö kuin tarkastus. Läpikäynnin tarkoituksena on löytää virheitä, parantaa tuotetta, harkita vaihtoehtoisia toteutustapoja, arvioida tuotosta standardeihin ja määrittelyihin tai arvioida käytettävyyttä [IEE08].

Läpikäynnin kohteena voivat olla esimerkiksi vaatimusmäärittelyt, arkkitehtuurikuvaus, ohjelmakoodit, testaussuunnitelma, käyttöohjeet ja niin edelleen. Läpikäynnissä henkilöillä on myös tietyt roolit. Läpikäynnin aikana kohteena olevaa tuotosta tarkastellaan siitä vastaavan tuottajan johdolla. Läpikäynnin puheenjohtaja huolehtii siitä, kaikki tarvittavat osatehtävät tukitaan. Sihteeri kirjaa ilmenneet virheet, korjaus- ja parannusehdotukset. Osallistujat havainnoivat virheitä ja antavat parannusehdotuksia. Puheenjohtaja päättää läpikäynnin.

3.3 Mitä katselmoidaan

Katselmoinnissa voidaan käsitellä mitä tahansa ohjelmistoprosessin tuotoksia. Seuraavassa on lueteltu muutamia katselmoinnin avulla tarkastettavia tuotoksia [IEE08]:

- Vaatimusmäärittelydokumentti
- Suunnitteludokumentti
- Testaussuunnitelma
- Lähdekoodi
- Käyttöohjeet
- Ylläpito-ohjeet
- Arkkitehtuurikuvaukset
- Muut mahdolliset kirjalliset tuotokset

Yrityksissä käytetään katselmointeja hyvinkin erilaisissa laajuuksissa. Mittavassa yrityksiä koskevassa kyselytutkimuksessa paljastui, että useimmissa yrityksissä ei käytetä säännöllisesti katselmointeja. Vaatimusmäärittelydokumentin katselmointi suoritettiin 42% yrityksistä, suunnitteludokumentti katselmoitiin 40% yrityksistä ja koodin katselmointi suoritettiin 28% yrityksistä [Cio03].

3.4 Katselmointien hyödyt

Katselmoinneilla on osoitettu olevan monia hyötyjä osana ohjelmistoprojekteja. Vaikka katselmoinnit vaativat paljon henkilöresursseja on niiden käyttöönotolla todettu saatavan aikaan huomattavia parannuksia prosesseihin ja niiden tuotoksiin. Erään projektin aikana katselmoinnin käyttöön otolla pystyttiin vähentämään virheiden määrää dokumenteissa 20 virheestä 1,5 virheeseen yhtä dokumentin sivua kohden 18 kuukauden aikana [Gil00]. Merkittävää hyötyä tulee myös siitä, että katselmoinneilla voidaan löytää virheitä jo ennen kuin perinteistä testausta on mahdollista käyttää virheiden havaitsemiseen. Esimerkiksi vaatimusten katselmoinnin yhteydessä löytyvän virheen korjaaminen ennen tuotannon aloittamista, tuo merkittävää kustannussäästöä virheen korjauskustannuksissa.

Seuraavassa on lueteltu muutamia syitä suorittaa katselmointeja [Gil00]:

- Virheiden identifiointi, poistaminen ja ennalta ehkäisy
- Laadun lisääminen tuotantoprosessiin ja sen tuotoksiin
- Tuotantoprosessin – tarkastusprosessin jatkuva kehittäminen
- Tuotosten ominaisuuksien mittaaminen ja arviointi (virhemäärä)
- Henkilöiden koulutus työtehtäviin (tiiminvetäjä, tarkastajat)
- Henkilöiden motivointi, yhteistyön kehittäminen, tiimin kokoaminen

3.5 Katselmoinneista koituvat kustannukset

Katselmoinneista koituu aina kustannuksia. Ne voivat vaatia jopa huomattavia määriä henkilöresursseja. Varsinkin jos projekteissa ei ole aiemmin käytetty katselmointeja on niiden tekniikan käyttöönottamiseen varattava riittävästi aikaa. Katselmointien onnistuminen vaatii lisäksi siihen osallistuvien henkilöiden motivaatiota noudattaa tarkasti määrättyjä katselmointikäytäntöjä ja ohjeita.

4 Testaaminen

Tässä luvussa tutustutaan testauksen periaatteisiin, joita on käytetty katselmointeja ja testaamista vertailevissa tutkimuksissa. Testaus on yleisesti dynaaminen menetelmä virheiden etsintään eli testattavaa ohjelmistoa suoritetaan suunnitelluilla syötteillä ja verrataan tuloksia odotettuihin tuloksiin. Vertailututkimukset poikkeavat toisistaan merkittävästi. Tässä luvussa kuvataan yleisellä tasolla periaatteet joita useimmissa tutkimuksissa on testauksen osalla käytetty.

4.1 Testausten periaatteet

Vertailututkimuksissa testaus perustui perinteisiin testausmenetelmiin toiminnalliseen testaukseen ja rakenteelliseen testaukseen. Testaus, toisin kuin katselmoinnit edellyttää olemassa olevaa suoritettavaa ohjelmaa tai sen osia. Testauksen tarkoituksena on löytää kohteena olevasta ohjelmasta virheitä eli poikkeamia odotetuista tuloksista. Testauksessa ohjelmalle annetaan useimmiten suunniteltuja syötteitä ja tuloksia verrataan odotettuihin tuloksiin. Seuraavassa on esitelty keskeiset käsitteet testauksen osalta.

Toiminnallinen testaus - mustalaatikkotestaus (functional testing), on ohjelman toiminnallisuuden testausta, joka perustuu ohjelmasta tehtyihin vaatimusmäärittelyihin. Määrittelyiden pohjalta suunnitellaan testejä, jotka suoritetaan ohjelmalla. Saatua tulosta

verrataan odotettuun tulokseen. Jos tulos poikkeaa odotetusta tuloksesta, tekee ohjelma virheen kyseisellä syötteellä (kun odotettu tulos oikea).

Rakenteellinen testaus – lasilaatikkotestaus (structural testing) , on testausta, jossa testitapausten suunnittelu perustuu ohjelmakoodin rakenteeseen. Rakenteellisen testauksen riittävyyttä kuvataan erilaisilla kattavuusluvuilla *(Coverage – Kattavuus)* [Lai98].

Täydellisessä haaraumakattavuudessa (*Branch coverage*) jokainen ohjelman haarauma käydään läpi tosi ja epätosi vaihtoehdolla. Silmukkakattavuudessa (*Loop coverage*), jokainen silmukka voidaan suorittaa kerran, useammin tai ei kertaakaan. Moniehtokattavuudessa (*Multi-condition coverage*) testataan kaikki rakenteisen ehdon loogiset vaihtoehdot siten, että jokainen looginen tila saa arvon tosi jollakin testillä ja epätosi jollakin muulla testillä. Relaatiokattavuudella (*Relational coverage*) selvitetään relaatio-operaatioiden (<,=,>) oikeellisuus. Esimerkiksi operaation ’>’ (suurempi kuin) käyttö kun pitäisi olla ’>=’ (suurempi ja yhtä suuri kuin).

4.2 Testausten kohteet

Vertailuissa testauksen kohteena olevat ohjelmat olivat hyvin pieniä verrattuna nykyään tehtäviin ohjelmistotuotteisiin. Seuraavassa on lyhyesti esitetty tietoja testauksen kohteena olleista ohjelmista.

- Ohjelmointikielinä eri tutkimuksissa käytettiin muun muassa PL/1, Fortran, Simpl-T, C ja Pascal ohjelmointikieliä.
- Lähdekoodidokumenttien pituudet vaihtelivat 64 ja 2414 rivin välillä.
- Virheettömään (tarkastettuun) lähdekoodiin oli alan ammattilaisten toimesta lisätty erilaisia virheitä.

5 Tuloksia

Tässä luvussa kerrotaan katselmointeja ja testausta vertailevien tutkimusten tuloksista vuosien 1976-2003 välillä. Tutkimukset on tehty pitkällä aikavälillä, jonka aikana menetelmät, ohjelmointikielet ja muu tekniikka on kehittynyt. Tutkimuksissa on käytetty eri ohjelmointikieliä, ohjelmat ovat olleet erilaisia, eivätkä dokumentointitavatkaan ole olleet yhteneväisiä. Muun muassa näiden esitettyjen tekijöiden vuoksi tutkimustulosten vertailu on hankalaa.

5.1 Vaatimusmäärittelyihin liittyvät virheet

Vaatimusmäärittelyistä löytyviä virheitä on tutkittu paljon. Kaikki tutkimukset ovat kuitenkin keskittyneet tutkimaan katselmointimenetelmien erilaisten variaatioiden paremmuutta virheiden havaitsemiseksi. Testaus ei ole ollut mukana missään vaatimuksiin liittyvissä virheiden etsintään liittyvissä tutkimuksissa [RAT06]. Valinta vaatimusmäärittelyjen virheiden etsimismenetelmien välillä onkin ilmeinen jos käytettävissä ovat vaatimusten katselmointi tai järjestelmätestaus. Ei ole järkevää toteuttaa järjestelmää väärien vaatimusten mukaan ja korjata sitä jälkikäteen [RAT06]. Vaatimusmäärittelyjen virheiden etsimiseen kannattaa ehdottomasti käyttää katselmointeja.

5.2 Tuloksia koodiin keskittyneistä tutkimuksista

Tässä luvussa tutustutaan tuloksiin, joita on saatu kun pienistä eri ohjelmointikielisistä (muun muassa Pascal, Fortran ja C) ohjelmista (64-2414 riviä) on etsitty virheitä. Menetelminä on käytetty katselmointia, toiminnallista testausta ja rakenteellista testausta. Tavoitteena on ollut selvittää virheiden löytymisen osuudet kaikista virheistä, sekä virheiden löytämisnopeudet eri menetelmillä. Koska etenkin katselmoinnin (luku 3) osalta tutkimuksissa on käytetty toisistaan selvästi poikkeavia menetelmiä, joko versioita tarkastuksista tai läpikäynneistä, ei näitä tutkimuksia ole mahdollista käydä yksityiskohtaisesti läpi. Myös rakenteellisen testaamisen osalta tuloksiin vaikuttavat esimerkiksi erilaisten kattavuuksien ja niiden riittävyysehtojen määrittelyt (luku 4). Lisäksi tutkimuksissa käytettyjen henkilöiden kokemus alalta vaihtelee opiskelijoista ammattilaisiin, joten tutkimustuloksiin liittyy paljon tulkinnan varaa.

Tutkimuksissa pääperiaatteet noudattavat kuitenkin seuraavanlaista toimintamallia [Sun05]:

1. **Tutkimukseen osallistuvat henkilöt**. Useimmiten käytettiin alalla olevia opiskelijoita, joista kootaan ryhmiä eri tekniikoiden vertailuja varten. Myös ammattilaisista koottuja ryhmiä on käytetty.
2. **Virheet ja koodi**. Ammattilaisten suunnittelemat ohjelmat (erikieliset lähdekoodit), joihin on tarkoituksella lisätty erilaisia virheitä.
3. **Katselmointi.** Menetelmänä käytetään jotakin katselmointitekniikkaa (luku 3), jossa virheitä etsitään koodia suorittamatta.
4. **Toiminnallinen testaus**. Suoritetaan vaatimuksista laaditut testitapaukset.
5. **Rakenteellinen testaus**. Koodin tarkastajat pyrkivät laatimaan 100 % kattavuuden antavat testit jollekin tai useammalle kattavuudelle (luku 4).

6. **Ryhmät ja niiden toiminta**. Osallistujat jaettiin satunnaisesti ryhmiin, jotka käyttivät 1 vaiheessa eri virheiden etsintämenetelmiä eri ohjelmiin sovitun ajan. 2 vaiheessa samat ryhmät käyttivät eri menetelmää toiseen ohjelmaan. Tilastollisen tutkimuksen nollahypoteesina oli yleisesti, että menetelmillä ei ole eroa virheiden etsinnässä.

Taulukossa 1 on esitetty yhteenveto tutkimusten tuloksista, joissa vertailtiin katselmoinnin, toiminnallisen testauksen ja rakenteellisen testaamisen hyödyllisyyttä pienten ohjelmien virheiden löytämisessä.

Tutkimus	Virheiden löytämisen osuus eri menetelmillä (%)			Virheiden löytämisnopeus (virhettä/tunti)			Erilaisia
	Katselmointi	Toiminnallinen Testaus	Rakenteellinen Testaus	Katselmointi	Toiminnallinen Testaus	Rakenteellinen Testaus	virheitä löytyi
1. Herzel	37.3	47.7	46.7				
2. Myers	38.0	30.0	36.0	0.80	1.62	2.07	Kyllä
3. Basili and Selby	54.1	54.6	41.2				Kyllä
4. Kamstiens and Lott 1	43.5	47.5	47.4	2.11	4.69	2.92	Osittain
5. Kamstiens and Lott 2	50.3	60.7	52.8	1.52	3.07	1.92	Osittain
6. Roper et al.	32.1	55.2	57.5	1.06	2.47	2.20	Kyllä
7. Laitenberger	38.0						Ei
8. So et al.	17.9	43.0		0.16	0.34		Kyllä
9. So et al. 2	34.6	43.0		0.26	0.34		Kyllä
10. Runeson and Andrews	27.5	37.5		1.49	1.80		Kyllä
11. Juristo and Vegas 1	20.0	37.7	35.5				Osittain
12. Juristo and Vegas 2		75.8	71.4				Osittain

Taulukko 1: Tutkimuksissa paljastuneiden virheiden suhteelliset osuudet ja niiden löytämisnopeudet katselmointeja, toiminnallista testausta ja rakenteellista testausta verranneissa tutkimuksissa [RAT03].

Seuraavassa esitetään yhteenveto eri testeistä [RAT03]:

1. Herzel. 1976 julkaistussa tutkimuksessa todettiin, että testaamalla löydetään suurempi osa virheistä kuin katselmoinnilla. Tutkimukseen osallistui 39 opiskelijaa. Tarkoituksena oli löytää virheitä 3 ohjelmasta, jotka sisälsivät 64-170 lausetta.
2. Myers. 1978 julkaistussa tutkimuksessa todettiin, että testaamisella ja katselmoinneilla löydetään suhteellisesti yhtä paljon virheitä, mutta virheet ovat osaltaan erityyppisiä. Tutkimuksessa 59 ammattilaista etsi virheitä (15 kappaletta) 63 lausetta sisältävästä ohjelmakoodista.
3. Basili and Selby. 1987 julkaistu tutkimus totesi, että virheiden suhteellinen löytyminen ja virheiden löytämisnopeus katselmointien ja testaamisen avulla riippuvat ohjelmatyypistä. Tutkimukseen osallistui 32 ammattilaista ja 42 opintojen loppuvaiheessa olevaa opiskelijaa, jotka etsivät virheitä neljästä eri ohjelmointikielillä kirjoitetusta ohjelmasta.

4 ja 5. Kamsties and Lott. 1995 julkaistussa tutkimuksessa todettiin, että eri menetelmillä löydetään virheitä suhteellisesti yhtä paljon, mutta aikayksikköä kohden testaamalla löytyy enemmän virheitä. Kahteen toistotutkimukseen osallistui 27 ja 15 opiskelijaa, jotka etsivät virheitä kolmesta C-kielisestä noin 200 rivin ohjelmasta. Tekniikoilla löytyi osittain erilaisia virheitä.

6. Roper et al. 1997 julkaistussa tutkimuksessa testaus oli katselmointia tehokkaampi virheiden löytämisessä käytettyä aikayksikköä kohden, muuten menetelmillä ei ollut merkittävää eroa (yhdistelmä paras). Tutkimuksessa 47 opiskelijaa etsi virheitä kolmesta ohjelmasta, joissa virheiden määrät olivat 8, 9 ja 8.

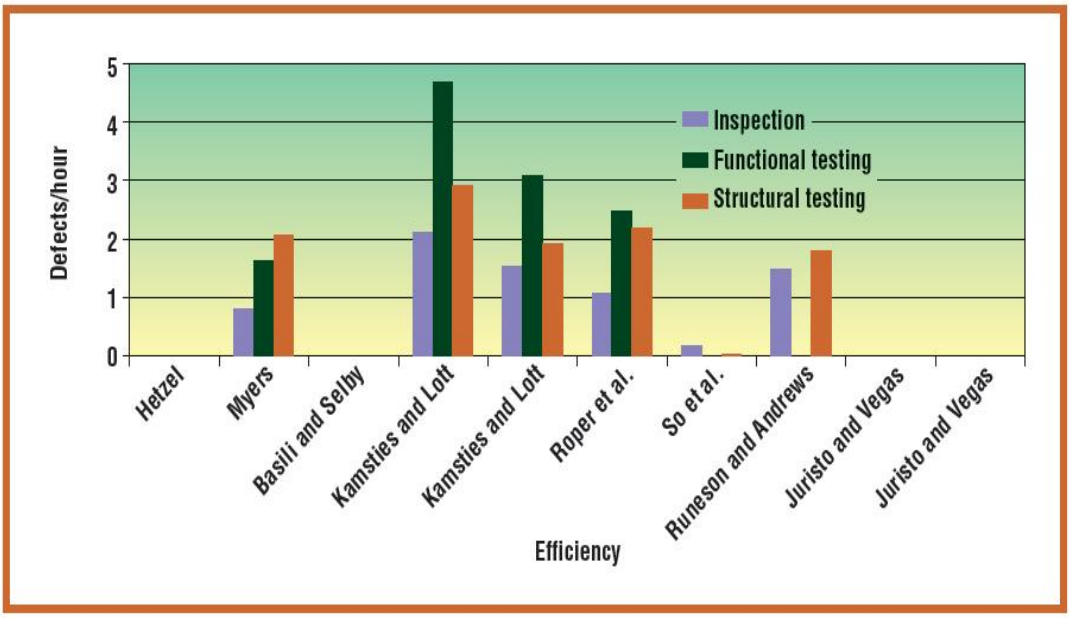
7. Laitenberger. 1998 julkaistussa tutkimuksessa 20 opiskelijaa suorittivat C-kieliselle 262 rivin ohjelmalle katselmoinnin ja sen jälkeen rakenteellisen testauksen. Tutkimuksen tuloksena saatiin, että katselmointitiimit löysivät keskimäärin 52 % virheistä, minkä jälkeen testaustiimit löysivät lopuista 42 % virheistä vain 17 % [Lai98].

8. ja 9. So et al. 2002 julkaistussa tutkimuksessa saatiin tulos jossa katselmointi perinteisellä Faganin menetelmällä löysi testausta paremmin virheitä aikayksikköä kohden. Testauksella kuitenkin löydettiin määrällisesti enemmän virheitä. Kahdessa kokeessa opiskelijat etsivät virheitä eri menetelmillä kahdeksasta Pascal-kielisestä ohjelmasta, joiden pituudet vaihtelivat1200 ja 2400 rivin välillä.

10. Runeson and Andrew. 2003 julkaistussa tutkimuksessa virheiden löytämisessä testaus voitti katselmoinnin, mutta virheiden eristämisessä katselmointi oli parempi. Tutkimukseen osallistui 30 opiskelijaa tarkoituksena verrata rakenteellista testausta ja katselmointia. Kohteena oli kaksi C-kielistä ohjelmaa jotka olivat 190 ja 208 ohjelmariviä pitkiä.

11 ja 12. Juristo and Vegas. 2003 julkaistussa tutkimuksessa, jossa selvitettiin katselmoinnin, toiminnallisen ja rakenteellisen testauksen tehokkuutta, tultiin johtopäätökseen, että eri menetelmät sopivat erilaisiin virheisiin. Ensimmäisessä tutkimuksessa 196 opiskelijaa etsi virheitä uudesta ohjelmasta ja ohjelmakoodista. Toisessa tutkimuksessa 46 opiskelijaa etsi virheitä ohjelmakoodista.

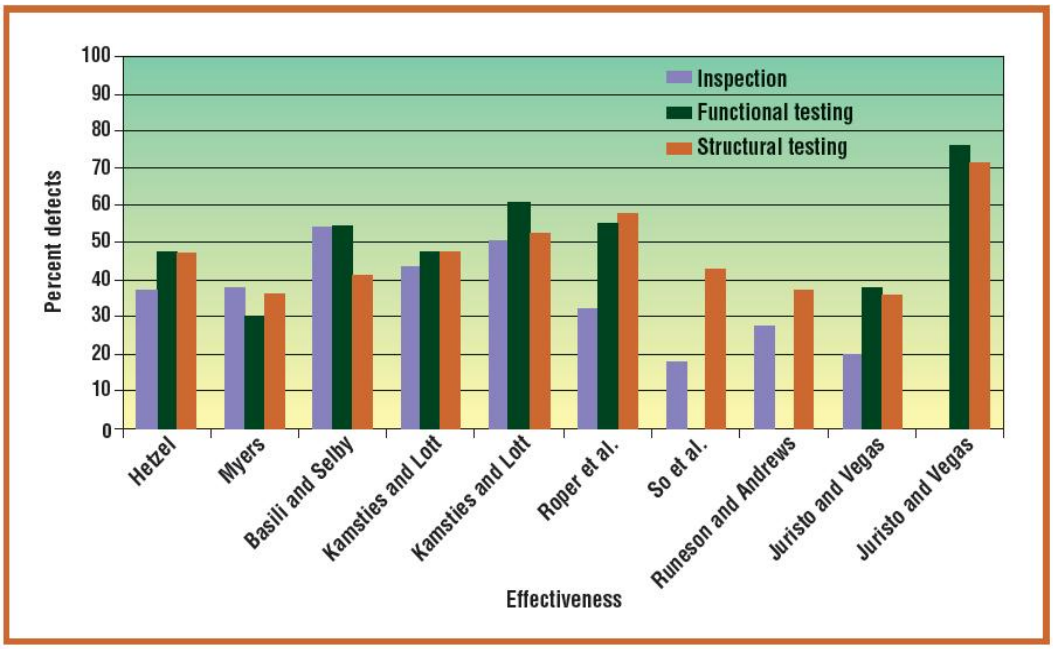
Osassa tutkimuksista tutkittiin miten paljon eri tekniikoilla löydetään virheitä aikayksikköä kohden. Kuvassa 3 esitetään katselmointeja ja testausta vertailevien tutkimusten tulokset graafisesti. Tutkimuksissa päädyttiin hyvin erilaisiin tuloksiin, sillä jokainen edellä mainituista menetelmistä on selviytynyt parhaaksi ainakin yhdessä tutkimuksessa.



Kuva 3: Virheiden keskimääräinen löytymisnopeus eri tutkimuskissa [RAT06].

Virheiden löytymisnopeus vaihtelee varsin huomattavasti eri tutkimusten välillä ja myös eri menetelmien välillä. Parhaimmillaan virheitä on paljastunut lähes 5 kappaletta tunnissa. Huonoimmillaan tunnissa ei ole pystytty paljastamaan yhtään ainoata virhettä. Tutkimusten tulosten perusteella näyttää siltä, että katselmoinneilla paljastuu 1…2 virhettä tunnissa. Vastaavat arvot toiminnallisessa testauksessa ovat 2…5 virhettä ja rakenteellisessa testauksessa 2…3 virhettä.

Kuvasta 4 nähdään, että löytyneiden virheiden osuus vaihtelee paljon eri tutkimusten välillä. Parhaimmillaan virheistä on löytynyt 75 prosenttia ja huonoimmillaan noin 20 prosenttia. Tehokkaimmaksi menetelmäksi näyttää osoittautuvan toiminnallinen testaus. Rakenteellinen testaus on seuraavaksi paras ja huonoimmaksi osoittautuu tarkastus.



Kuva 4: Tutkimuksissa löytyneiden virheiden osuudet kaikista virheistä [RAT06].

Tutkimuksien välillä on hyvin paljon eroja. Ensimmäiseksi osallistuvien henkilöiden osaamistaso ja lukumäärä vaihtelevat tutkimusten välillä. Toiseksi tutkimuksen kohteena olevat ohjelmat ovat hyvin erilaisia ja kokoisia. Kolmanneksi tutkimusten järjestelyt poikkeavat jonkin verran toisistaan muun muassa katselmointi on järjestetty eri tavoilla. Tutkimusten tuloksista voi kuitenkin vetää sen johtopäätöksen, että virheiden etsinnässä ohjelmakoodista testaus näyttää olevan katselmointia parempi menetelmä.

5.3 Tuloksia suunnitteluvirheiden etsinnästä

Ohjelmistojen suunnitteluvirheiden löytämisestä ei ole olemassa monia tutkimuksia, jotka vertailevat katselmointeja ja testaamista. Kolme tutkimusta on tutkinut suunnitteludokumenttien tarkastamista ja verrannut siinä löytyneiden virheiden määrää toteutettujen funktioiden testaamiseen. Ensimmäisessä tutkimuksessa löytyi virheitä tarkastamalla 53,5 % ja testaamalla 41,8 % kaikista virheistä. Virheiden löytämisnopeus oli tarkastuksissa 5 ja testauksella alle 3 virhettä tunnissa [RAT06]. Toinen teollisuuden piirissä tehty koe varmisti edellisen tuloksia. Tarkastuksilla löydettiin 0,68 virhettä tuntia kohden kun taas testaamalla löydettiin vain 0,1 virhettä tunnissa. Kolmas koe vahvisti edelleen tuloksia. Katselmoimalla löydettiin 0,82 virhettä ja testaamalla vain 0,013 virhettä tuntia kohden [RAT06]. Suunnitteludokumentin katselmointi on siis kannattavampaa kuin toiminnallinen testaus. Edelleen toiminnallisessa testauksessa löydetty virhe tietää lisätyötä suunnittelun ja toteutuksen uudelleensuorittamisessa.

6 Yhteenveto

Ohjelmistotuotteiden ja prosessien laadun parantamisessa keskeisenä tekijänä on virheiden määrän vähentäminen lopputuotteista. Virheiden hallintaan käytettävät kustannukset ovat suuria, jopa puolet koko projektin kustannuksista. Tämän vuoksi virheiden hallintaan käytettävät menetelmät kannattaa optimoida mahdollisimman hyvin käytettävien tekniikoiden kesken. Katselmoinnit ja testaus ovat kaksi yleisimmin käytettyä menetelmää virheiden hallitsemiseksi. Yleisesti on tiedossa, että mitä aikaisemmassa vaiheessa virhe löydetään, sitä vähemmän siitä koituu korjauskustannuksia. Virheitä on monentyyppisiä ja niitä syntyy tuotantoprosessin eri vaiheissa. Tällöin on hyvä käyttää myös eri menetelmiä virheiden hallintaan. Katselmointien etuna on muun muassa se, että niitä voidaan käyttää jo ennen kuin on mitään testattavaa, sekä kehittää tulevia projekteja analysoimalla katselmoinneilla löytyneitä virheitä. Testaamalla voidaan taas tutkia sellaisia tapauksia, joita dokumentteja lukemalla on vaikea, jopa mahdoton havaita. Tutkimustuloksiin viitaten onkin järkevää käyttää ohjelmistoprosessin aikana sekä katselmointeja, että testausta sopivasti. Tulosten mukaan katselmointeja kannattaa käyttää vaatimus- ja suunnitteludokumentteihin ja testausta lähdekoodivirheiden etsintään. Tieteellistä tutkimusta tarvittaisiin kuitenkin vielä lisää asian varmistamiseksi.

Lähteet

BoB Boehm B, Basili V,. Software Defect Reduction Top 10 List, *Computer* 34,1(2001), sivut 135-137.

Gal04 Galin D., *Software Quality Assurance*: *From Theory to Implementation.* Pearson Education Limited, England, 2004.

Gil00 Gilb T., Planning To Get the Most Out of Inspection, Software Quality Professional, 2,2,(2001).

HaM06 Haikala I., Merijärvi J., *Ohjelmistotuotanto.* Talentum media Oy, Jyväskylä, 2006.

IEE93 IEEE. *1044-1993 Standard Classification for Software Anomalies, 1993.*

IEE08 IEEE. *1028-2008 Standard for Software Reviews and Audits, 2008.*

Lai98 Laitenberger O., Studying the Effects of Code inspection and Structural Testing on Software Quality. *Proceedings of the 9th International Symposium n Sofwarereliability Engineering*, Paderborn, Germany, 1998, sivut 237-246.

RAT06 Runeson P., Andersson C., Thelin T. Andrews A., Berling T., What Do We Know about Defect Detection Methods?, *IEEE SOFTWARE*, 23,3(2006), sivut 82-90.

JuV03 Juristo N., Vegas S., Functional Testing, Structural Testing, and Code Reading What Fault Type Do They Each Detect? *Lecture Notes in Computer Science.* 2003, sivut 208-232.

-