TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietotekniikan tutkinto-ohjelma

Root cause- analyysi ohjelmistuotannossa – Mistä ohjelmistovirheet johtuvat?

Kandidaatintyö

Joni Rannila

Ohjelmistotuotannon ja -liiketoiminnan laboratorio
Espoo 2009

TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä: Joni Rannila

Työn nimi:
Root Cause -analyysi ohjelmistotuotannossa – Mistä ohjelmistovirheet johtuvat?

Päiväys: 26. Huhtikuuta 2009 **Sivumäärä:** 8 + 22
Pääaine: Ohjelmistotuotanto **Koodi:**
Vastuuopettaja: Prof. Lauri Savioja
Työn ohjaaja: TkL Jari Vanhanen

Root Cause -analyysi (RCA) on kokoelma työkaluja sekä prosessin kuvaus, joita käytetään ongelmien ratkaisuun sekä laadun parantamiseen laajalti eri teollisuuden aloilla. RCA:n ideana on etsiä erilaisten tapahtumien, esimerkiksi onnettomuuksien tai tuotteisiin syntyneiden virheiden, perimmäisiä syitä (engl. root cause). Poistamalla perimmäiset syyt voidaan varmistaa, että samoja ongelmia ei tapahdu enää uudelleen ja tällä tavoin nostaa eri prosessien ja tuotteiden laatua.

Root Cause -analyysia on sovellettu myös ohjelmistotuotantoon jo useita vuosikymmeniä ja kirjallisuudesta löytyy joitakin raportteja sen käytännön toteutuksista. Tämän työn tarkoituksena on tehdä yhteenvetoa siitä, miten RCA:ia on sovellettu eri ohjelmistoyrityksissä ohjelmistovirheiden vähentämiseen, minkälaisia vaikutuksia sillä on ollut ohjelmistojen laatuun, minkälaisia kustannuksia sen käyttö aiheutti. Lisäksi on tarkoitus selvittää mistä ohjelmistovirheet johtuvat ja millaisilla keinoilla virheiden määrää on saatu vähennettyä.

Ohjelmistotuotannossa RCA suoritetaan yleensä luokittelemalla ohjelmistovirheet, etsimällä perimmäiset syyt suurimmille luokille, pohtimalla ratkaisu- ja parannusehdotuksia perimmäisille syille ja ottamalla kyseiset ehdotukset käyttöön. Eri Root Cause -analyysien toteutuksista voidaan selvästi erottaa kaksi eri toteutustapaa: ohjelmistoprojektin jälkeinen RCA ja ohjelmistoprojektiin integroitu RCA. Projektiin integroitua RCA:ia käytettiin iteratiivisissa projekteissa, jossa se suoritettiin jokaisen iteraation lopussa. Toteutustavalla ei ollut kuitenkaan suurta vaikutusta RCA:sta saataviin hyötyihin: jokaisessa RCA:n käyttötapauksessa ohjelmistovirheiden määrä laski 15%-54% ja osalla tehokkuutta mitanneista ohjelmistokehityksen tehokkuus nousi 24%-27%. RCA on suhtellisen halpa laadunparannusmenetelmä verrattuna sen hyötyihin. Kustannuksia RCA:sta raportoitiin noin 0.5%-1.5% kehitysbudjetista. Parhaimmillaan koko analyysiprosessi suoritettiin reilussa kahdessa tunnissa eräässä projektiin integroidussa RCA:ssa. Mukaan ei ollut laskettu parannuksiin käytettyä aikaa.

Avainsanat: Root Cause -analyysi, RCA, defect causal analysis, DCA, ohjelmistovirheet, ohjelmistojen laadun parantaminen
Kieli: Suomi

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Degree Program of Computer Science and Engineering

ABSTRACT OF BACHELOR'S THESIS

Author: Joni Rannila

Title of thesis:
Root Cause Analysis in Software engineering – Why does defects occur?

Date:	Apr 26 2009	**Pages:** 8 + 22
Professorship:	Ohjelmistotuotanto	**Code:**
Supervisor:	Professor Lauri Savioja	
Instructor:	Jari Vanhanen, Lic. Sc. (Tech)	

Root Cause Analysis (RCA) is a collection of tools and techniques which are used to solve problems and to improve product quality. It's used widely in different industries. The idea of root cause analysis is to find root causes to different problems and find a way to remove them, so that the original problem will not occur again. This improves the quality of the processes and products.

Root cause analysis is also used in software engineering since the 1970's. There has been some practical case studies over time of implementing the RCA in software companies. The purpose of this study is to make a summary from some of the case studies and to find out how the RCA is used to reduce the number of defects, what kind of effect it has had on software quality and what does the RCA actually cost.

In software engineering the RCA is usually done by first analyzing the defects, then having a causal analysis meeting, where RCA-team finds the root causes for the selected defects and brainstorm improvement suggestions. After the meeting all the improvement suggestions are presented to an action team, who will decide which suggestions are implemented. Based on different case studies of RCA we can see two different types of RCAs taken into action: post-project RCA and a RCA integrated in software development process. Every case study showed a significant reduction of defect rates, all between 15% to 54%. The costs of excecuting the RCA is in average from 0.5% to 1.5% of the development budget, which makes this technique quite affordable way to improve quality.

Keywords: Root Cause Analysis, RCA, Defect Causal Analysis, DCA, defect based process improvement, software quality improvement
Language: Finnish

Alkulause

Työn valmistumisesta haluan kiittää työni ohjaajaa, Jari Vanhasta, hyvistä neuvoista ja ohjeista. Kiitokset myös vaimolleni Elinalle työn oikolukemisesta.

Espoossa 26. huhtikuuta 2009

Joni Rannila

Käytetyt lyhenteet

CSC	Computer Science Corporation; Yhdysvaltalainen yritys, joka otti käyttöön DCA-prosessin.
DCA	Defect Causal Analysis; IBM:n kehittämä prosessi, joka sisältää myös RCA-vaiheen
DIR	Defect Injection Rate; Luku, joka kertoo montako ohjelmistovirhettä koodiin syntyy yhden työtunnin aikana.
HP	Hewlett-Packard; Yksi suurista yrityksistä, joka on ottanut RCA:n käyttöön.
IBM	International Business Machines; Yksi suurista yrityksistä, joka on ottanut RCA:n käyttöön.
IEEE	Institute of Electrical and Electronics Engineering; Eri alan insinöörien yhteenliittymä, joka kehittää standardeja sekä julkaisee alan kirjallisuutta.
RCA	Root Cause Analysis; Root Cause -analyysi.
SIT	Systematic Inventive Thinking; Yleinen ongelmanratkaisutekniikka.
TQM	Total Quality Management; 1980-luvulla suosiota herättänyt laatumenetelmä, joka sisälti myös RCA:n.
TRIZ	The theory of inventive problem solving; Yleinen ongelmanratkaisutekniikka.

Sisältö

Luku 1

Johdanto

1.1 Yleistä

Root cause- analyysi (engl. Root cause analysis, RCA) on prosessi, jonka avulla tutkitaan tapahtumien perimmäisiä syitä (engl. root cause). RCA:ta käytetään laajalti erilaisissa ympäristöissä ongelmanratkaisuun sekä laadun parantamiseen. Virheet eivät tapahdu itsestään, vaan yleensä taustalla on jokin syy. Selvittämällä taustalla olevan syyn, voidaan estää virheen tapahtuminen uudelleen ja tällä tavoin parantaa työn laatua tai esimerkiksi työturvallisuutta. RCA-prosessi on hyvin geneerinen ja muokattavissa lähes mihin toimialaan tahansa, missä laadun parantaminen on tarpeen. RCA:n lopputulos on joukko ideoita ja suosituksia, joiden käyttöönotto johtaa laadun parantumiseen.

Endress oli vuonna 1975 julkaisullaan yksi ensimmäisistä ohjelmistovirheiden analysoijista (Leszak et al., 2000). Tätä ennen tutkimus oli keskittynyt enemmän itse virheisiin kuin siihen, mistä kyseiset virheen johtuvat. Tunnetuimmat julkaisut RCA:n käytöstä ovat kuitenkin ilmestyneet vasta 1990-luvun lopulla, kuten Card (1998), Mays et al. (1990) sekä Leszak et al. (2000). Koska RCA ei ole mitenkään tarkoin määritelty prosessi, vaan enemmänkin joukko ongelmanratkaisumetodeita ja yleinen prosessin kuvaus, kaikki ohjelmistotuotannossa toteutetut RCA-prosessit poikkeavat jonkin verran toisistaan.

Kirjallisuudesta löytyy useita julkaisuja, joissa esitetään, miten RCA:ia on käytetty eri yrityksissä ja minkälaisia tuloksia sen käytöstä on saatu. Koska kirjallisuudesta ei löytynyt julkaisua, jossa olisi tehty yhteenvetoa eri käyttötapauksista sekä niiden tuloksista, tämän työn tarkoituksena on täyttää kyseinen tiedon tyhjiö. Tämä työ on siis kirjallisuuskatsaus RCA:n soveltamisesta ohjelmistotuotanossa. Työssä keskitytään RCA:n sovelluksiin ohjelmistovirheiden vähentämiseksi. Tar-

koituksena on etsiä kirjallisuudesta vastauksia seuraaviin tutkimuskysymyksiin:

1. Miten RCA:ia on sovellettu ohjelmistovirheiden vähentämiseksi?
2. Miten RCA vaikutti ohjelmistojen laatuun ja minkälaisia kustannuksia sen käyttö aiheutti?
3. Mistä ohjelmistovirheet johtuivat ja minkälaisilla keinoilla virheiden määrää saatiin vähennettyä?

Samaa RCA-prosessia voidaan käyttää myös testausmenetelmien parantamiseen analysoimalla virhedataa hieman eri näkökulmasta (Mays et al., 1990). Testiprosessien parantaminen RCA:lla ei kuitenkaan kuulu tämän työn laajuuteen. Kirjallisuus perustuu pääasiassa lehtiartikkeleihin ja alan konferenssijulkaisuihin. Aineiston valinnassa on painoitettu sitä, että RCA:ia on sovellettu käytännössä jossakin yrityksessä. Aineiston läpikäynnissä on käytetty apuna luvussa kaksi esitettyä kysymyslistaa. Aineistoa on haettu seuraavista alan tietokannoista: ACM Digital Library, IEEE Xplore sekä ScienceDirect. Apuna on käytetty myös Scopus-viitetietokantaa. Hakusanoina on käytetty seuraavia sanoja ja niiden yhdistelmiä: root cause analysis, defect causal analysis ja defect classification and analysis.

Luvussa kaksi esitellään yleinen RCA-prosessi ja sen käytetyimmät työkalut. Luvussa kolme esitellään kirjallisuudesta löydettyjä esimerkkejä siitä, kuinka RCA:a on sovellettu ohjelmistotuotannossa ja tarkastellaan niiden yhtäläisyyksiä sekä eroavaisuuksia. Luvussa neljä tarkastellaan luvussa kolme esitettyjen tapausten pohjalta mistä ohjelmistovirheet johtuvat sekä minkälaisia toimenpiteitä on tehty niiden vähentämiseksi. Luvussa viisi tehdään yhteenveto siitä minkälaisia hyötyjä ja kuluja eri RCA:sta on saatu.

Luku 2

Root cause -analyysi

RCA tuli tunnetuksi osana Total Quality Management-menetelmää (TQM) (Andersen ja Fagerhaug, 2006). TQM sai alkunsa 1950-luvulla, mutta herätti suurempaa kiinnostusta vasta 1980-luvulla, jolloin sen ottivat käyttöön Ford Motor Company, Philips Semiconductor, Motorola sekä Toyota Motor Company (Hashmi, 2003). RCA on tärkeä työkalu TQM:n jatkuvan kehityksen ideologiassa. Nykyisin RCA on mukana esimerkiksi SixSigma-menetelmässä sekä TQM:ia muistuttavassa Lean management -ideologiassa.

Andersen ja Fagerhaug (2006) kuvaa perimmäistä syytä seuraavasti:

> *The root cause is "the evil at the bottom"that sets in motion the entire cause-and-effect chain causing the problem(s).*

Ongelmatilanteissa ensimmäiseksi havaitut syyt ongelman syntymiseen ovat yleensä vain välillisiä syitä, oireita perimmäisestä syystä. Ongelman voi yleensä ratkaista poistamalla oireet, mutta jos perimmäistä syytä ei poisteta, se saa aikaiseksi uusia ongelmia ennemmin tai myöhemmin. RCA:in tavoitteena on löytää ja poistaa nämä perimmäiset syyt. RCA:lle ei löydy yleistä määritelmää, vaan se on enemmänkin yhteinen nimitys laajalle joukolle menettelytapoja, työkaluja ja tekniikoita, joilla ongelmia saadaan ratkaistuksi (Andersen ja Fagerhaug, 2006). Vaikka RCA ei ole tarkoin määritelty prosessi voidaan siitä yleisesti tunnistaa seuraavat vaiheet:

1. Ongelman määrittäminen
2. Datan kerääminen
3. Datan analysointi ja kausaalisuuksien tunnistaminen

4. Perimmäisten syiden tunnistaminen
5. Ratkaisut perimmäisten syiden poistamiseen
6. Ratkaisujen käyttöönotto

Ongelman määrittäminen on askel, joka on itse asiassa jo tehty ennen RCA:n aloittamista, sillä tarve RCA:n käyttöön on itsessään jo implikaatio siitä, että on tunnistettu jokin ongelma, joka halutaan ratkaista. Datan kerääminen on ongelmakohtaista ja sitä käsitellään tarkemmin ohjelmistotuotannon näkökulmasta seuraavassa luvussa. Datan analysointi ja kausaalisuuksien tunnistaminen on RCA-prosessin tärkeimpiä vaiheita, sillä tässä vaiheessa tulee datasta saada selville syitä, mitkä johtivat ongelman syntyyn. Tätä varten on olemassa useita eri työkaluja. Andersen ja Fagerhaug (2006) ehdottavat mm. seuraavia työkaluja:

- *Histogrammi.* Helppokäyttöinen kaavio, joka auttaa tunnistamaan poikkeamia ja samaa kaavaa noudattavia elementtejä. Histogrammilla voidaan esittää esimerkiksi löydettyjen ohjelmistovirheiden määrää eri kehitysvaiheissa tai eri komponenteissa.
- *Pareto-kaavio.* Pareto-kaaviolla saadaan selville mitkä syyt vaikuttavat määrällisesti eniten kyseiseen ongelmaan. Pareton ideaa on sovellettu RCA:iin siten, että 20% syistä aiheuttaa noin 80% ongelmista. Ohjelmistovirheitä analysoitaessa käytetään usein Pareto-kaaviota esittämään virheiden jakauma luokittain, kuten esimerkiksi käyttöliittymä, algoritmit, standardit, rajapinnat jne. Tämän jälkeen on helppo valita niin monta luokkaa, että saavutetaan noin 80% määrä virheistä. Valituille luokille pyritään tunnistamaan perimmäiset syyt.
- *XY-kaavio.* XY-kaaviota käytetään etsiessä korrelaatioita eri syiden tai muiden muuttujien välillä. XY-kaaviolla voidaan esimerkiksi tutkia ohjelmiston koon vaikutusta virhemääriin.

RCA:n seuraava vaihe on perimmäisten syiden tunnistaminen. Edellisen vaiheen analyysin perusteella valitaan joukko virheitä, joille voidaan olettaa, että virhe johtuu jostakin tietystä syystä, eikä ole pelkästään satunnainen tapahtuma. Syiden tunnistaminen tapahtuu yleensä muutamasta tunnista muutamaan päivään kestävässä aivoriihessä, riippuen virheiden määrästä. Syiden etsimisen avuksi on monta erilaista työkalua. Andersen ja Fagerhaug (2006) listaa seuraavat työkalut yksinkertaisina ja helppokäyttöisinä:

- *Ishikawa-kaavio.* Ishikawa-kaaviolla, tutummin kalanruotokaaviolla, voidaan helposti listata ja ryhmitellä ongelmaan johtavia syitä eri kategorioihin ja lopulta tunnistaa ongelman perimmäiset syyt. Kalanruotokaavio on helppokäyttöinen ja nopea tapa visualisoida syiden etsintäprosessia. Kaavion ideana on kirjoittaa ongelman kuvaus vaakasuoran nuolen päähän ja nuolesta vedetään viivoja joko ylös- tai alaspäin osoittamaan eri syiden ryhmiä. Tämän jälkeen ongelmaan johtavia syitä kirjoitetaan eri ryhmien alle. Ohjelmistotuotannossa yleisiä pääryhmiä ovat prosessit, ihmiset sekä teknologia.

- *Ongelmamatriisi.* Ongelmamatriisissa etsitään riippuvuuksia eri ongelmien ja ennalta mietittyjen mahdollisten perimmäisten syiden välillä. Jokaisen syyn ja ongelman välille voidaan asettaa joko heikko, keskinkertainen tai vahva riippuvuussuhde, joita merkitään eri symbolein ja jokaisella riippuvuuden tasolla on eri painotuskerroin. Andersen ja Fagerhaug (2006) ehdottavat painoiksi arvoja 1,3 ja 9. Kun jokaisen ongelman ja syyn välinen suhde on analysoitu, lasketaan eri syille asetetut painot yhteen. Mitä suurempi pistemäärä jollakin syyllä on, sitä todennäköisemmin se on perimmäinen syy.

- *5 kertaa "miksi?".* Tämän tekniikan periaatteena on kysyä "miksi?"niin monta kertaa, että uusia vastauksia ei enää tule. Vastausten loppuessa ollaan saavutettu perimmäinen syy. Andersen ja Fagerhaug (2006) mukaan tekniikan nimi tulee siitä, että kysymiskierroksia tulee usein viisi kappaletta.

- *Virhepuuanalyysi (engl. fault tree analysis).* Virhepuuanalyysissä luodaan puumainen rakenne, jonka juurena on alkuperäinen ongelma ja puun lehdissä ovat mahdolliset perimmäiset syyt. Puuta lähdetään rakentamaan etsimällä päällimmäiset syyt, jotka johtivat ongelmaan. Tämän jälkeen analysoidaan syitä ja etsitään niille syitä ja niin edelleen. Puun rakentamisessa käytetään ja- ja tai-operaattoreita, jotka merkitään vastaavasti puoliympyräksi ja kolmioksi. Näillä operaattoreilla saadaan ryhmiteltyä eri syitä ja ylläpidettyä syiden keskinäisiä suhteita.

Jokaiselle löydetylle perimmäiselle syylle yritetään löytää jokin korjaava toimenpide tai ratkaisu, jolla syy saadaan poistettua. Tähän vaiheeseen voidaan soveltaa yleisiä ongelmanratkaisutekniikoita, kuten the six thinking hats, TRIZ (engl. The theory of inventive problem solving) tai SIT (engl. Systematic inventive thinking) (Andersen ja Fagerhaug, 2006). Tämän vaiheen tuloksena on lista parannusehdotuksia, jotka usein priorisoidaan ennen listan julkaisemista. Listasta valitaan parannusehdoituksia resurssien mukaan ja valituille parannusehdotuksille tulisi

asettaa vastuuhenkilö tai -ryhmä, joka pitää huolen siitä että kyseiset parannukset otetaan käyttöön.

Seuraavassa luvussa tarkastellaan kirjallisuudesta löytyneitä käytännön toteutuksia RCA:sta. Tutkimuskysymykset 1 ja 2 on pilkottu useammaksi kysymykseksi, joihin etsitään vastauksia valitusta kirjallisuudesta:

- Minkälaisella henkilöstöllä RCA:a suoritetaan?
- Missä vaiheessa RCA suoritetaan?
- Kuinka paljon ja minkälaisia ohjelmistovirheitä sisällytetään analyysiin?
- Miten ohjelmistovirheet luokitellaan?
- Kuinka suurelle ja minkälaiselle virhejoukolle etsitään perimmäiset syyt?
- Miten perimmäisten syiden etsintä suoritetaan?
- Miten parannus- ja ratkaisuideoita synnytetään?
- Minkälaisia kustannuksia RCA aiheutti?
- Minkälainen vaikutus RCA:lla oli ohjelmistovirhemääriin?

Luku 3

Root Cause -analyysi ohjelmistotuotannossa

3.1 RCA:n soveltaminen ohjelmistotuotantoon

RCA:n käyttö ohjelmistotuotannossa perustuu ohjelmistovirheiden luokitteluun ja analysointiin. Useissa ohjelmistokehitysprojekteissa pidetään jonkinlaista tietokantaa ohjelmistovirheistä. Virhetietokanta on lähtökohtana RCA:n datan keräys ja analyysivaiheille. Moniin muihin toimialoihin verrattuna ohjelmistotuotannon RCA:ssa käsitellään huomattavasti enemmän dataa kuin muualla. Tätä varten on ohjelmistotuotantoon kehitetty ohjelmistovirheiden luokittelujärjestelmiä, kuten esimerkiksi Chillarege et al. (1992):n ortogonaalinen virheiden luokittelu. Muilta osin ohjelmistotuotannon RCA vastaa luvussa kaksi esitettyä yleistä RCA-prosessia. Seuravassa luvussa esitellään kirjallisuudesta löydettyjä RCA:n toteutuksia ohjelmistotuotannosta.

3.2 RCA:n käyttötapauksia

3.2.1 Lucent Technologies

Leszak et al. (2000) kuvailee RCA-projektia Lucent Technologiesin verkkoelementtiprojektissa, jossa tehtiin suurta ohjelmistokomponenttia. Leszak et al. (2000) ei kerro kuinka iso ryhmä oli tekemässä RCA:ta, mutta kuvailee, että ryhmään kuuluu henkilöitä eri osastoilta, kuten ohjelmisto- sekä laittestokehityksen osastoilta, laadunvalvonnasta sekä integraatio-osastolta. RCA suoritettiin ohjelmistokomponentin erään version julkaisun jälkeen.

Koska suurissa projekteissa virhetietokannan koko voi olla valtava, Leszak et al. (2000) esitti toimintamallin, jolla virheiden analysoinnin työmäärä saatiin pidettyä kohtuullisena. Toimintamallin ideana oli valita joukko virheitä jokaisesta ohjelmistokomponentista ja ensimmäiseksi poistaa joukosta kaikki ne virheet, jotka eivät olleet vakavia (kriteerinä oli virheiden näkyminen asiakkaille), eivät aiheuttaneet korjaustoimenpiteitä tai olivat virheitä dokumentoinnissa. Tämän jälkeen joukosta valittiin manuaalisesti 10 vakavinta virhettä ja mukaan arvottiin 40 virhettä. Leszakin ryhmä jakoi virheet kolmeen eri pääluokkaan: toteutus-, rajapinta- ja ulkoiset virheet. Jokainen pääluokka sisältää useita tarkempia virhetyyppejä, yhteensä 21 kappaletta, jotka kuvataan taulukossa 3.1. Toinen teki-

Taulukko 3.1: Ohjelmistovirheiden luokittelu Leszak et al. (2000) mukaan

Implementation	Interface	External
1: data design/usage	9: data design/usage	16: development environment
2: resource allocation/usage	10: functionality design/usage	17: test environment (tools/infrastructure)
3: exception handling	11: communication protocol	18: test environment (test cases/suites)
4: algorithm	12: process coordination	19: concurrent work (other releases)
5: functionality	13: unexpected interactions	20: previous (inherited from prev. releases)
6: performance	14: change coordination	21: other
7: language pitfalls	15: other	
8: other		

jä virheiden luokittelussa oli projektin vaihe, jossa virhe on huomattu. Leszakin ryhmän tavoitteena oli löytää kaksi eri virhejoukkoa analysoitavaksi. Toinen joukoista oli tuotteiden julkaisujen jälkeen löytyneet virheet (engl Post-GA defects) ja toinen joukko etsittiin tekemällä tilastollinen analyysi virheiden joukolle, joka määriteltiin prosessin alussa. Analyysissä käytettiin pääasiassa Pareto-kaavioita, jotka havainnollistavat hyvin virheiden jakauman eri virhetyypeissä ja havaitsemisvaiheissa. Toisen virhejoukon tulisi sisältää virheet, joiden korjaaminen vaatii paljon työtä ja jotka ovat löydetty myöhäisessä vaiheessa kehitysprosessia.

Koska perimmäisiä syitä virheille on usein enemmän kuin yksi, Leszakin ryhmä luokitteli perimmäiset syyt neljään eri luokkaan: eri kehitysvaiheista johtuvat syyt, ihmisistä johtuvat syyt, projektista johtuvat syyt ja katselmuksista johtuvat syyt. Nämä luokat virittävät neliulotteisen perimmäisten syiden avaruuden,

jossa jokainen perimmäinen syy voidaan määritellä asettamalla sille jokin arvo jokaisessa ulottuvuudessa. Kaikissa edellä mainituissa luokissa on oletusarvona *ei soveltuva*, koska harvoista syistä saa johdettua neljää tai useampaa perimmäistä syytä. Jokaiseen luokkaan määriteltiin valmiiksi kaikista yleisimmät perimmäiset syyt helpottamaan virheiden analyysivaihetta. Virhejoukkojen perimmäisten syiden analysointia varten järjestettiin kahden päivän mittainen työpaja (engl. workshop), jonka lopputuloksena oli priorisoitu lista toimenpiteistä, joilla perimmäiset syyt saadaan poistettua. Tämä lista esitettiin ylemmälle johdolle sekä projektiryhmille.

Leszak et al. (2000) arvioi, että esittämillään viidellä eri toimenpiteellä saavutetaan noin 53% säästöt kustannuksissa ja parannetaan tehokkuutta noin 24%. Itse RCA:n kustannuksia ei mainita.

3.2.2 Computer Science Corporation

Card (1998) kuvailee Computer Science Corporationissa käyttöön ottamansa IBM:llä (International Business Machines) kehitetyn Defect Causal Analysis -prosessin (DCA). DCA-prosessi integroituu ohjelmistokehitykseen siten, että jokaisen ohjelmiston testausvaiheen jälkeen, kun ohjelmiston virheet on saatu virhetietokantaan, tehdään virheille RCA kausaalisuuksien analyysipalaverissa (engl. Causal analysis meeting). RCA:n tuloksena saadut parannusehdoitukset käsitellään toimenpideryhmän tapaamisessa (engl. Action team meeting). Tapaamisessa päätetään, otetaanko parannusehdoitukset käyttöön pelkästään meneillä olevassa projektissa, vai ovatko ne hyödyllisiä koko organisaatiolle ja tuleville projekteille. Näin tehtävät jakautuvat lyhyen aikavälin toimenpiteisiin (engl. short-term actions) sekä pitkän aikavälin toimenpiteisiin (engl. long-term actions).

Kausaalisuuksien analyysipalavereita varten muodostettiin ryhmä, joka koostui pääasiassa ohjelmistokehittäjistä ja muista henkilöistä, joilla on kokemusta kyseisen ohjelmiston tuottamisesta. Ryhmässä tulee olla yksi henkilö, joka ohjaa ryhmän toimintaa ja pitää huolen siitä, että ryhmä pysyy agendassa. Kokenut ja kunnioitettu ohjelmistokehittäjä on huomattavasti parempi valinta ryhmän ohjaajaksi, kuin projektipäällikkö tai laadunvalvonnan henkilö, koska auktoriteetit yleensä haittaavat ryhmän jäsenten välistä keskustelua (Card, 1998). Tehokkaan ryhmän koko on alle 25 henkilöä, joten suurissa projekteissa kannattaa projekti jakaa loogisiin komponentteihin ja muodostaa jokaiselle komponentille oma ryhmänsä. Kausaalisuuksien analyysipalaveri tulee pitää säännöllisin väliajoin jokaisen testaus- ja kehitysvaiheen jälkeen sekä 3 kuukautta jokaisen tuotteen julkaisun jälkeen. Palaveri noudattaa seuraavaa kaavaa: valitaan joukko virheitä, luokitellaan virheet, tunnistetaan luokitelluista virhejoukoista systemaattiset vir-

heet, etsitään systemaattisten virheiden perimmäiset syyt, kehitetään toimenpiteet perimmäisten syiden poistamiseksi ja dokumentoidaan tulokset. Card (1998) suosittelee valitsemaan enintään 20 virhettä yhteen palaveriin. Ajan säästämiseksi virheet kannattaa valita jo valmiiksi ennen palaveria. Virheiden luokittelussa Card (1998) suosittelee kolmiulotteista luokittelua: vaihe, jossa virhe oli tapahtunut, virheen havaitsemisvaihe sekä virheen tyyppi. Löytyneille systemaattisille virheille tehdään perimmäisten syiden analyysi. Perimmäiset syyt löytyvät usein helposti jo ongelman kuvauksesta, mutta joskus syitä ei löydetäkään niin helposti. Tällöin ongelmasta kannattaa tehdä kalanruoto-kaavio. Card (1998) suosittelee käyttämään analyysin apuna neljää eri perimmäisten syiden kategoriaa, johon kuuluu suurin osa löydetyistä perimmäisistä syistä:

- *työskentelytavat* (engl. methods), jotka voivat olla vääriä, moniselitteisiä, vapaaehtoisia tai ei loppuun asti kehitettyjä
- *työkalut ja ympäristö*, jotka voivat olla huonoja ja tehottomia
- *ihmiset*, joilla voi olla matala osaamistaso tai ymmärrys projektista
- *vaatimukset ja palaute*, jotka voivat olla vajaata tai moniselitteistä

Kun korjaavat toimenpiteet perimmäisiin syihin on dokumentoitu, toimenpideryhmä käsittelee sen omassa palverissaan. Toimenpideryhmä priorisoi korjausehdoitukset ja suunnittelee ja tekee aikataulun niiden käyttöönotolle. Se myös allokoi resurssit ja asettaa vastuuhenkilöt toimenpiteiden toteutukselle. Toimenpideryhmän tehtäviin kuuluu myös seurata toteutusten tilannetta ja toimenpiteiden tehokkuutta sekä kommunikoida tilanteesta takaisin analyysiryhmälle tai -ryhmille. Toimenpideryhmässä tulisi olla ainakin yksi johdon edustaja, joka mahdollistaa toimenpiteiden nopean hyväksymisen sekä käyttöönoton. Paikka toimenpideryhmässä on hyvä tapa sitouttaa ylintä johtoa DCA-prosessiin.

RCA:n käyttö laski virheiden määrää keskimäärin 50% kahden vuoden aikana ja säästi merkittävän summan rahaa. RCA:n kustannukset ovat 1.5% ohjelmistokehityksen budjetista. Kustannuksiin ei sisälly alkuvaiheen RCA:n koulutusta, virheluokkajärjestelmien määrittelyä sekä muiden RCA-proseduurien määrittelyjä.

3.2.3 Infosys

Jalote ja Agrawal (2005) kuvaa RCA-prosessin käyttöä iteratiivisia ja ketteriä ohjelmistokehitysmenetelmiä käyttävässä Infosys-yrityksessä. Projektiliiketoiminnassa RCA:n suorittava henkilöstö tulee valita projektiryhmästä projektin suunnitteluvaiheessa, jotta henkilöt ehditään kouluttamaan tarpeen mukaan. Ryhmän

kokoon Jalote ja Agrawal (2005) eivät ottaneet kantaa. Ennen ensimmäistä iteraatiota RCA-ryhmä pitää aloitustapaamisen, jossa tarkastellaan edellisten projektien RCA:n tuloksia ja ratkaisuja ilmenneisiin ongelmiin. Iteratiivisissa projekteissa RCA suoritetaan jokaisen iteraation lopussa, jolloin kerätään ja analysoidaan iteraation aikana löytyneet ohjelmistovirheet. Ohjelmistovirhejoukolle tehdään Pareto-analyysi, jossa virheet luokitellaan niiden tyypin mukaan. Eri virhekategoriat tulisi sopia yhteisesti joko projektin alussa tai yritystasolla, jolloin eri projektien välinen vertailu on helpompaa. Virheiden luokittelussa Jalote ja Agrawal (2005) suosittelevat avuksi IEEE standardia (IEEE Std. 1044-1993) tai ortogonaalista virheiden luokittelumenetelmää (Chillarege et al., 1992). Erään kolmen iteraation mittaisen projektin ensimmäisen iteraation jälkeen ohjelmistovirheitä tuli analysoitavaksi 57 kappaletta. Ohjelmistovirheet luokiteltiin seuraavasti:

- loogiset virheet
- standardit
- redundanttia koodia
- käyttöliittymä
- arkkitehtuuri

Virheiden analyysipalaverissa päätettiin vähentää virheitä suurella kädellä, ja 3 suurinta virheryhmää Pareto-kaaviosta valittiin RCA:n kohteeksi. Nämä kolme ryhmää olivat loogiset virheet, standardit sekä redundantti koodi, jotka sisältivät noin 80% kaikista virheistä. Jalote ja Agrawal (2005) lähtivät ratkaisemaan perimmäisiä syitä kalanruotokaavion ja 5 kertaa "miksi?"kaltaisen prosessin yhdistelmän avulla. Kalanruotokaavion nuolen päähän asetettiin ongelmaksi "liian monta virhettä ryhmässä X". Kun kalanruotokaavio oli saatu valmiiksi, siitä etsittiin kaikista tärkeimmät perimmäiset syyt, joiden eliminointi vaikutti eniten virheiden määrän vähenemiseen.

Infosys:ssä perimmäisten syiden etsiminen sekä ratkaisut syiden eliminoimiseksi tehtiin yhdessä aivoriihessä. Kun jokin ratkaisu löydettiin, ratkaisun toteuttamiselle asetettiin samalla vastuuhenkilö ja ratkaisusta tehtiin tehtävä, joka laitettiin tehtävienhallintajärjestelmään ohjelmointitehtävien rinnalle, jos se vain oli mahdollista. Aivoriihen lopputuloksena oli taulukko johon oli listattuna perimmäiset syyt, toimenpiteet syiden poistamiseksi, toteutuksen aikataulu sekä vastuuhenkilö. Kaikki korjaavat toimenpiteet otettiin käyttöön ennen seuraavan iteraation alkua.

Infosys:ssa mitataan RCA:n vaikutuksia ohjelmistovirheisiin DIR-arvon avulla (Defect Injection Rate), joka kertoo montako virhettä koodiin syntyy yhden työtunnin aikana. Ensimmäisen iteraation DIR-arvo oli 0.33. Toisen iteraation DIR-arvo oli vain 0.1 ja kolmannen noin 0.09. Huomattavaa on ensimmäisen ja toisen iteraation välinen DIR-arvojen erotus verrattuna toisen ja kolmannen iteraation väliseen erotukseen. Jalote ja Agrawal (2005) eivät maininneet suoraan RCA:sta aiheutuvia kustannuksia, mutta kyseisestä prosessista voidaan havaita kolme eri vaihetta, joihin vaaditaan resursseja: iteraation virhedatan keräys ja lajittelu, tapaaminen, jossa tehtiin RCA ja ideoitiin korjaavat toimenpiteet sekä pohdittiin toimenpiteiden käyttöönottoa.

3.2.4 IBM

Mays et al. (1990) esittää IBM:ssä (International Business Machines) käyttöönotetun RCA-menetelmän, joka integroituu ohjelmistokehitysprosessiin. RCA-prosessi koostuu neljästä eri elementistä: kausaalisuuksien analysointipalaverista, toimenpideryhmän palaverista, kickoff-tapaamisista sekä datan keräämisestä ja seurannasta. Kausaalisuuksien analysointipalaveriin osallistuu koko projektin ohjelmistokehittäjät ja palaveria ohjaa kokenut ohjelmistokehittäjä, joka on koulutettu kyseiseen tehtävään. Toimenpideryhmän koko on pienissä organisaatioissa kolmesta neljään henkilöä ja suuremmissa organisaatioissa noin kahdeksasta kymmeneen henkilöä. Isossa, useita ohjelmistoprojektiryhmiä sisältävässä organisaatiossa yksi toimenpideryhmä voi käsitellä usean kehitysryhmän toimenpideesitykset. Kickoff-tapaamisia järjestetään jokaisen kehitysvaiheen alussa. Tapaamisessa toimenpideryhmä esittelee ohjelmistokehitysryhmälle käyttöön ottamansa laatua parantavat menetelmät sekä käy läpi aikaisempia saman vaiheen ongelmia ja ratkaisuja.

Kausaalisuuksien analysointipalaveriin otetaan mukaan kuluneen vaiheen korjatut ohjelmistovirheet. Mays et al. (1990) mukaan noin kolmasosa virheistä otettiin mukaan analysointipalaveriin. Sopivien virheiden valinta on analysointipalaverin ohjaajan vastuulla. Virheet jaettiin palaverissa neljään eri syyluokkaan: kommunikaatio, huolimattomuus/ylenkatsominen, koulutus sekä transkriptiot. Samalla selvitetään kuinka ja missä vaiheessa virhe oli syntynyt. Virheitä analysoitaessa tulee lopuksi pohtia, mitä pitää tehdä, jotta virhe ei toistu sekä kuinka vastaavat virheet voidaan löytää muualta tuotteesta. Kun virheet oli analysoitu yksilötasolla, niitä tutkittiin joukkona. Joukosta yritettiin löytää systemaattisia virheitä, jotka viittaavat laajempiin ongelmiin. Analysointipalaverin ohjaaja tekee yhteenvedon käsitellyistä virheistä, niiden syistä sekä ratkaisuehdotuksista, joka esitetään toimenpideryhmälle. Toimenpideryhmä priorisoi ratkaisuehdoitukset ja päättää mitkä ehdotuksista otetaan käyttöön. Toimenpideryhmä toteuttaa käyt-

töönoton ja seuraa sen vaikutuksia ohjelmistovirhemääriin.

RCA:n kustannuksille oli IBM:ssä asetettu ylärajaksi 0.5% kehitysbudjetista. Kickoff- ja kausaalisuuksien analyysipalavereille oli varattu kaksi tuntia. Toimenpideryhmän jäsenet käyttivät keskimäärin 10% työajastaan toimenpideryhmän tehtäviin. RCA:n vaikutus pitkällä tähtäimellä on ollut merkittävä: virheiden määrä oli tuotteen kahdeksannen RCA:ta käyttävän julkaisun jälkeen vähentynyt 54%. RCA-prosessiin oli käytetty resursseja keskimäärin 0.85 henkilötyövuotta vuodessa ja kokonaissäästöksi laskettiin 2.2 henkilötyövuotta vuodessa. Säästöt muodostuivat, kun virheiden korjaamiseen, tutkimiseen sekä testaukseen käytetty aika vähentyi virheiden määrän vähentyessä.

3.2.5 Hewlett-Packard

Grady (1996) esittää RCA:n suorittamista eräässä HP:n (Hewlett-Packard) ohjelmistokehityksen osastossa. RCA:n suorittavaan ryhmään kuului 12 henkilöä, suurin osa ohjelmistokehittäjiä, mutta myös johtotason henkilöitä. Ryhmässä tulee olla yksi henkilö, joka toimii fasilitoijana ja prosessin ohjaajana. Tämän henkilön vastuulla on myös loppuryhmän kouluttaminen ennen analyysin aloittamista.

Grady (1996) määrittelee kolme erilaista RCA-prosessia eri tilanteisiin: kertaluontoinen- (one-shot), projektinjälkeinen- (post-project) sekä jatkuvan kehityksen (continuous improvement) prosessi. Kertaluontoinen prosessi on tarkoitettu organisaatioille, jotka eivät ennen tehneet RCA:ta ja luokitelleet ohjelmistovirheitään. Kertaluontoinen RCA sopii myös sellaisille organisaatioille, jotka haluavat minimoida RCA:n käytettävän työn määrää ja palkata ulkopuolisen fasilitoijan. Projektinjälkeinen RCA-prosessi vastaa suurimmalta osin kertaluonteista prosessia, mutta on huomattavasti nopeampi, koska virheiden luokittelua ja valmista RCA-dataa on jo olemassa. Jatkuvan kehityksen prosessimallissa RCA on integroitu ohjelmistokehitysprosesseihin ja sitä suoritetaan tarpeen mukaan eri ohjelmistokehityksen vaiheissa. Grady (1996) esittämässä tapauksessa RCA suoritettiin projektinjälkeisenä prosessina.

Ohjelmistovirheitä analysoitiin 476 kpl. Virheet luokiteltiin virheen todennäköisen syyn perusteella. Luokista muodostettiin sektoridiagrammi sekä painoitettu sektoridiagrammi, jossa luokkiin laskettiin painoituskertoimet virheiden korjaamiseen arvoitujen työmäärien perusteella. Perimmäiset syyt päätettiin selvittää kahdesta suurimmasta luokasta, jotka olivat spesifikaatiot (64 virhettä) ja käyttöliittymän virheet (110 virhettä). Tämän jälkeen ohjelmistokehittäjiä ohjeistettiin valitsemaan kaksi itsensä tekemää, valittuihin luokkiin kuuluvaa virhettä sekä miettimään miten virheet olisi voitu estää tai havaita aikaisemmin. Kehittäjät esittelivät virheet ja ratkaisut tapaamisessa, jossa keskustellaan virheistä ja ideoi-

daan niille mahdollisia perimmäisiä syitä. Perimmäisiä syitä etsittiin tekemällä kaksi kalanruoto-diagrammia, käyttöliittymävirheistä sekä spesifikaatiovirheistä. Koska edellisen RCA:n ryhmä oli jo tekemässä parannuksia spesifikaatiovirheisiin, tapaamisessa päätettiin keskittyä käyttöliittymävirheiden syiden ratkaisujen löytämiseen. Ratkaisuja ongelmiin etsittiin samassa tapaamisessa ja samalla tehtiin myös parannussuunnitelma, johon listattiin parannusehdoitukset sekä niille vastuuhenkilöt ja aikataulut. Tapaamisen jälkeen RCA:n fasilitoija koostaa analyysitapaamisesta saamansa datan esityskuntoiseksi ja dokumentti lähetetään kaikille RCA:iin osallistuneille.

RCA:n vaikutus ohjelmistovirhemääriin oli huomattava. Aikaisemmin käyttöliittymävirheiden osuus oli ollut noin 20% kaikista virheistä. RCA:n jälkeen virheiden osuus putosi seuraavassa projektissa 5%:iin. Vaikka seurava projekti oli koodimäärältään 34% suurempi, testaukseen kului 27% vähemmän aikaa. Grady (1996) mainitsee myös muidenkin HP:n projektiryhmien saavuttaneen vastaavanlaisia tuloksia. Grady (1996) ei mainitse paljonko aikaa kaikenkaikkiaan koko RCA-prosessiin käytettiin, mutta on budjetoinut koko ryhmän tapaamiselle yhden tunnin ja 20 minuuttia. Kertaluonteiselle RCA-prosessille Grady (1996) on antanut arvioksi noin 7 tuntia ryhmätyötä.

3.3 Yhteenvetoa RCA:n sovelluksista

Esitetyistä RCA-prosesseista voi erottaa kaksi eri suoritustapaa: joko projektin jälkeinen RCA tai jatkuvan kehityksen RCA. Projektien jälkeiset RCA:t ovat virhemääriltään isompia kuin jatkuvan kehityksen RCA:t, koska jatkuvan kehityksen RCA suoritetaan, joko kerran iteraatiossa (Infosys) tai kehitys- tai testivaiheen jälkeen (CSC). Taulukossa 3.2 on tehty yhteenvetoa RCA:n suoritustavasta sekä ohjelmistovirheiden määrästä ja niiden luokittelusta. RCA:n suorittavissa ryhmissä on eroja jatkuvan kehityksen- sekä projektien jälkeisen RCA:n välillä. Jokaisessa jatkuvan kehityksen RCA:ssa virheitä tutkiva ryhmä koostui projektin ohjelmistokehittäjistä, kun taas esimerkiksi Lucent Technologiesin tapauksessa ryhmä oli kasattu eri osastoilta ja osaamisalueilta. Yksi huomattava ero oli myös se, että jatkuvan kehityksen RCA-prosesseissa oli nimetty toimenpideryhmä, joka käsitteli RCA:n tuloksia ja vastasi ratkaisujen käyttöönotosta. Esimerkiksi Lucent Technologiesin tapauksessa RCA:n tulokset priorisoitiin ja esitettiin yrityksen johdolle sekä ohjelmistokehittäjille ilman sitoutumista mihinkään korjaavaan toimenpiteeseen. Erot analysoitavien ohjelmistovirheiden määrässä oli huomattavat riippuen siitä, toteutettiinko RCA omana projektinaan vai integroituna kehitysprosessiin.

Virheet oli luokiteltu joko todennäköisten syiden mukaan tai virheiden sijain-

Taulukko 3.2: RCA-prosessit lyhesti kuvattuna.

Yritys	RCA:n tekijät	Missä vaiheessa?	Analysoitujen virheiden määrä?	Virheiden luokittelu
Lucent Tech.	Erillinen projektiryhmä eri alan osaajia, kokoa ei mainita	Tuotteen julkaisun jälkeen omana projektinaan.	ei mainita.	3 pääluokkaa: toteutus, rajapinta ja ulkoiset. Pääluokilla yht. 21 alaluokkaa.
CSC	Ryhmä ohjelmistokehittäjiä, alle 25 henkilöä. Toimenpideryhmä, kokoa ei mainita.	Integroitu kehitysprosessiin. Säännöllisin väliajoin testaus- tai kehitysvaiheen jälkeen	enintään 20 kpl.	Rajapinnat, laskenta, logiikka, alustukset ja tietorakenteet.
Infosys	Tulee valita projektiryhmästä. Kokoon ei oteta kantaa.	Iteratiivisessa projektissa joka iteraation lopussa.	57 kpl.	Loogiset virheet, standardit, redundantti koodi, käyttöliittymä.
IBM	Koko ohjelmistokehitysryhmä sekä toimenpideryhmä, 3-10 henkilöä.	Integroitu kehitysprosessiin. Jokaisen kehitysvaiheen jälkeen.	Noin kolmasosa virheistä, suosittelee noin 20 kpl.	Kommunikaatio, huolimattomuus / ylenkatsominen, koulutus, transkriptiot.
HP	12 henkilöä, suurin osa ohjelmistokehittäjiä.	Projektin jälkeen.	476 kpl.	Todennäköisen syyn perusteella.

nin ja vaikutusalueen perusteella. Pienillä virhemäärillä luokittelu ei ole yhtä oleellista kuin suurilla virhemäärillä. Kuten taulukosta 3.3 nähdään, kaikki RCA-toteutukset, joilla oli noin 20 virhettä per RCA, kävivät kaikki virheet läpi perimmäisiä syitä analysoitaessa. Jos virheitä oli enemmän kuin 20 kappaletta, apuna käytettiin luokittelua sekä Pareto-analyysia tai sektoridiagrammeja helpottamaan suurimpien luokkien löytämistä. Perimmäisiä syitä etsittiin RCA-toteutuksissa usein pelkästään keskustelemalla tapahtuneista virheistä. Toinen suosittu tapa oli käyttää kalanruotokaaviota.

Taulukosta 3.4 voidaan havaita, että RCA:n kustannukset ja vaikutukset ohjelmistokehityksen laatuun olivat kaikilla yrityksillä saman suuntaisia. Ohjelmistokehitykseen integroiduissa RCA-prosesseissa ohjelmistovirheiden keräämiseen käytettiin keskimäärin puoli tuntia ja loput Root Cause -analyysista suoritettiin noin kahdessa tunnissa. HP:n pienet kustannukset verrattuna CSC:n tai IBM:ään johtuvat siitä, että HP:n kustannuksissa ei ole otettu huomioon parannusehdoitusten käyttöönotosta johtuvia kuluja.

Taulukko 3.3: Yhteenveto perimmäisten syiden etsinnästä.

Yritys	**Minkälaiselle virhejoukolle etsittiin perimmäiset syyt?**	**Miten syiden etsintä suoritettiin?**	**Miten ratkaisuideoita synnytettiin?**
Lucent Tech.	2 eri joukkoa: julkaisun jälkeen löytyneet virheet ja Pareto-analyysin kautta löytyneet virheet.	Valmis 4-uloitteinen perimmäisten syiden avaruus, johon virhe yritetään sijoittaa, 2 päivän mittainen workshop.	kahden päivän aivoriihi.
CSC	Virhejoukosta löytyneille systemaattisille virheille.	Palaverissa joko keskustelemalla tai kalanruoto-kaavion avulla.	Samassa palaverissa syiden löytymisen jälkeen on oma osio ratkaisujen ideoimiselle.
Infosys	Valittiin Pareto-analyysistä virhejoukkoja siten, että valinnan kattavuudeksi tuli 80%.	Haettiin vastausta kalanruotokaaviolla ongelmaan "Liian monta virhettä ryhmässä X".	Ratkaisuja ideoitiin samassa palaverissa syiden etsinnän kanssa keskustelemalla ryhmässä.
IBM	Kaikille virheille jotka valittiin RCA-prosessiin mukaan.	Ohjelmistokehittäjät virheitä yksilöllisesti ja pohtivat mahdollisia syitä, sen jälkeen he yrittivät tunnistaa systemaattisia virheitä koko virhejoukosta.	Samassa palverissa syiden etsinnän kanssa haettiin ratkaisuja keskustelemalla virheistä.
HP	Valittiin kaksi suurinta virhejoukkoa.	Kalanruoto-kaavioilla.	Tarkkaa menetelmää ei mainita.

Taulukko 3.4: RCA:n kustannukset ja vaikutukset ohjelmiston laatuun sekä organisaatioon.

Yritys	**RCA:n kustannukset**	**RCA:n vaikutukset**
Lucent Tech.	Ei mainita.	Kustannukset vähenivät 53%, tehokkuus kasvoi 24%.
CSC	1.5% kehitysbudjetista.	Virheet vähenivät keskimäärin 50%.
Infosys	Ei mainita.	DIR-arvo (Defect Injection Rate) pieneni noin 70% kahdessa iteraatiossa.
IBM	0.85 henkilötyövuotta/vuosi.	Virheet vähenivät keskimäärin 54%, säästöä 2.2 henkilötyövuotta/vuosi.
HP	1.5-7 tuntia ryhmätyötä.	Ennen käyttöliittymävirheitä 20% virheistä, RCA:n jälkeen 5%. Seuraava projekti 34% suurempi ja siihen kului 27% vähemmän aikaa.

Luku 4

RCA:n tulokset ja pohdintaa

4.1 Mistä ohjelmistovirheet johtuvat?

Edellisen luvun RCA-sovellusten lopputulokset olivat samankaltaisia. Yleisimpiä perimmäisiä syitä olivat ohjelmistokehittäjien huolimattomuus ja liian tiukat aikataulut sekä henkilöstön tietotaidot. Yu (1998):n tuloksissa riittämätön huomio yksityiskohtiin oli osasyynä 75 prosentissa ohjelmistovirheistä. Aikatauluongelmat olivat vastaavasti syynä 76 prosentissa virheistä. Leszak et al. (2000) suorittamassa RCA:ssa aikatauluongelmat olivat osasyynä keskimäärin 40 prosentissa ohjelmistovirheitä. Muita perimmäisiä syitä olivat mm. riittämättömät katselmukset (Leszak et al., 2000) ja riittämätön ymmärrys vanhasta koodista, käyttökuvauksista sekä tietokannoista sekä oliomalleista (Jalote ja Agrawal, 2005). Grady (1996) ei suoraan kerro mitkä syyt olivat perimmäisiä syitä, mutta esitetyistä kalanruotokaavioista löytyi monia edellä mainituista syistä. Mays et al. (1990) eivät myöskään mainitse löytämiään perimmäisiä syitä, mutta kertovat että erään tuotteen kohdalla yli puolet, 54%, parannusehdoituksista koski prosessien parantamista ja vain 12% henkilöstön koulutusta. Työkalujen osuus parannusehdoituksista oli 18%. Myös Card (1998) sai vastaavia tuloksia omassa analyysissaan. Card (1998) mukaan 65% systemaattisista virheistä johtuu huonoista toimintatavoista, virheistä eri prosesseissa sekä huonosta kommunikaatiosta.

4.2 Parannusehdoitukset

IBM:n RCA:ia lukuunottamatta kaikilla muilla parannusehdoituksia syntyi RCA:n tuloksena 5-10 kappaletta. Mays et al. (1990) ei mainitse, paljonko yksi RCA tuottaa parannusehdoituksia, mutta kertoi kahden eri tuotteen parannusehdoi-

tusten kokonaismääräksi 317 ja 335 kappaletta. Yleisimpiä parannusehdoituksia oli erilaisten tarkastuslistojen sekä ohjeistuksien käyttö ohjelmoitaessa sekä ohjelmakoodin katselmuksien lisääminen, jotka löytyivät jokaisesta luvussa kolme esitetyissä esimerkeissä. Muita parannusehdoituksia olivat mm. koulutuksien lisääminen, paremmat ohjelmakoodin analyysi- sekä testaustyökalut ja kattavammat yksikkötestit (Leszak et al., 2000; Mays et al., 1990).

4.3 RCA:n tulosten arviointi

RCA:n käytöstä saadut tulokset olivat poikkeuksetta hyviä. Parhaimmillaan raportoitiin jopa virhemäärien puolittumista ja jopa miljoonien dollareiden säästöjä virheiden korjaamiseen kuluvan ajan vähentyessä. RCA:n toteuttamista pidettiin suhteellisen edullisena ja tehokkaana tapana parantaa ohjelmistojen laatua. Tuloksiin tulee kuitenkin suhtautua kriittisesti, koska yksikään mainituista RCA-sovelluksista ei ollut puolueettoman tahon tekemä. Kalinowski et al. (2008) mukaan RCA:sta on tehty puolueeton tutkimus, mutta kyseinen tutkimus on kirjoitettu portugaliksi.

Iteratiiviseen kehitykseen integroidussa RCA:ssa ensimmäisten iteraatioiden analyyseilla oli merkittävä vaikutus ohjelmistovirheiden määrään. Jalote ja Agrawal (2005) pohtiikin, olisiko RCA:n suorittaminen ainoastaan alkuvaiheen iteraatioissa sekä virhemäärien aktiivinen seuranta riittävä toimenpide ohjelmiston laadun ylläpitämiseksi. Infosys:n RCA:ssa virhemäärät putosivat toisessa iteraatiossa kolmannekseen ensimmäisen iteraation virhemääristä, kun taas toisen ja kolmannen iteraation välinen ero oli vain 10%.

Mielenkiintoista oli huomata, että vaikka joidenkin RCA-sovellusten välillä oli noin 15 vuoden aikaero, joka teknisen kehityksen näkökulmasta on todella pitkä aika, olivat perimmäiset syyt pysyneet suurin piirtein samoina.

Luku 5

Yhteenveto

Kirjallisuudesta löytyi joitakin tapauksia, jossa RCA:ta on sovellettu eri ohjelmistoyrityksissä. RCA:n sovelluksista voidaan erottaa kaksi eri lähestymistapaa sen käyttöön: joko RCA suoritetaan ohjelmistoprojektin jälkeen omana projektinaan tai sitten se on integroitu ohjelmistokehitysprosessiin. Iteratiivisessa ohjelmistokehityksessä RCA suoritetaan yleensä jokaisen iteraation lopussa, jolloin kerralla analysoitava virhemäärä jää huomattavasti pienemmäksi kuin projektin jälkeisessä RCA:ssa Projektien jälkeiset RCA:t ovat kooltaan isompia ja niissä käsitellään vähintään edellisen projektin virheet, joita voi olla useita satoja kuten Yu (1998) tutkimat yli 600 virhettä tai Grady (1996) luokittelemat 476 virhettä. Grady (1996) mukaan projektin jälkeen suoritettava RCA on organisaation ensimmäinen askel kohti jatkuvan kehityksen prosessimallia. Kun historiallinen virhedata on analysoitu kerran, voidaan sen jälkeen keskittyä sen hetkisiin virheisiin ja käyttää RCA:ssa edellisen analyysin työkaluja ja tietoa hyväksi.

Luvussa kolme esitettyjen RCA-sovellusten tuloksien perusteella RCA on tehokas ja kustannuksiltaan pieni prosessi, jolla on merkittäviä vaikutuksia ohjelmistojen virhemääriin ja tätä kautta myös ohjelmistokehityksen tehokkuuteen. Useat yritykset, kuten IBM, Infosys ja Computer Science Corporation saavuttivat ohjelmistovirheiden puolittumisen RCA:ia käyttämällä.

Kirjallisuutta

Bjørn Andersen ja Tom Fagerhaug. *Root cause analysis : simplified tools and techniques*. ASQ Quality Press, Milwaukee, Wis., 2006. ISBN 0873896920 9780873896924.

D. N. Card. Learning from our mistakes with defect causal analysis. *Software, IEEE*, 15(1):56–63, Jan/Feb 1998. doi: 10.1109/52.646883.

Ram Chillarege, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray ja Man-Yuen Wong. Orthogonal defect classification–a concept for in-process measurements. *IEEE Transactions on Software Engineering*, 18(11):943–956, 1992.

R. B. Grady. Software failure analysis for high-return process improvement decisions. *Hewlett-Packard Journal*, 47(4):15–24, 1996.

Khurram Hashmi. Introduction and implementation of total quality management (tqm). Oct 2003. URL `http://www.isixsigma.com/library/content/c031008a.asp`. Käyty 18.04.2009.

P. Jalote ja N. Agrawal. Using defect analysis feedback for improving quality and productivity in iterative software development. osa 2005, sivut 701–714, 2005.

M. Kalinowski, G.H. Travassos ja D.N. Card. Towards a defect prevention based process improvement approach. sivut 199–206, Sept. 2008. doi: 10.1109/SEAA.2008.47.

Marek Leszak, Dewayne E. Perry ja Dieter Stoll. A case study in root cause defect analysis. *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, sivut 428–437, New York, NY, USA, 2000. ACM. ISBN 1-58113-206-9.

Robert G. Mays, Jones C. L., Holloway G. J. ja Studinski D. P. Experiences with defect prevention. *IBM Systems Journal*, 29(1):4–32, 1990.

Weider D. Yu. A software fault prevention approach in coding and root cause analysis. *Bell Labs Technical Journal*, 3(2):3–21, 1998.