Modernit koheesiometriikat käytännössä

TURUN YLIOPISTO
Informaatioteknologian laitos
Tietotekniikan kandidaatin tutkielma
Toukokuu 2015
Konsta Sinisalo

TURUN YLIOPISTO
Informaatioteknologian laitos

KONSTA SINISALO: Modernit koheesiometriikat käytännössä

TkK-tutkielma, 20 s., 3 liites.
Tietotekniikka
Toukokuu 2015

Koheesio on koodin yhtenäisyyttä arvioiva suure. Sillä pyritään tulkitsemaan koodin uudelleenkäytettävyyttä ja ymmärrettävyyttä. Korkean koheesion omaava projekti on selkeä ja tarkoituksenmukainen kokonaisuus. Projekti, jolla on matala koheesio, tulisi jakaa osiin. Koheesiota arvioimaan on kehitetty monia metriikoita, jotka perustuvat erilaisiin tapoihin laskea projektin luokkien koheesiolle numeerisia arvoja. Niiden etujen ja puutteiden tunteminen ennen käytännön soveltamista on äärimmäisen hyödyllistä.

Tässä tutkielmassa esitellään yleisimpiä koheesiometriikoita, sovelletaan kolmea niistä esimerkkiprojekteihin ja arvioidaan niiden hyödyllisyyttä käytännöllisinä koheesion arviointimenetelminä. Sovellettaviksi metriikoiksi valitaan LCOM4 (LCOM = Lack of Cohesion in Methods), LCOM5 ja CAMC (=Cohesion Among Methods in Class). Niitä arvioidaan sekä niiden tuottaman informaation että soveltamisen vaaditun työn perusteella. Lopuksi vertaillaan metriikoiden keskinäistä korrelaatiota.

Tuloksia analysoitaessa tullaan huomaamaan, minkälaista informaatiota, ja millaista työmäärää vastaan, metriikat antavat käytännössä. LCOM5 ja LCOM4 todetaan osittain toisiaan vastaaviksi ja vakaiksi, kun taas CAMCin huomataan olevan nopeudestaan huolimatta epäluotettava.

Asiansanat: koheesio, LCOM, LCOM4, LCOM5, CAMC, kytkentä, TCC, LCC

Sisällysluettelo:

1 JOHDANTO

Nykyaikaisessa ohjelmistotuotannossa kasvu ja kilpailu ovat kovaa. Uusien ohjelmistojen tuottaminen vaatii verrattain vähän resursseja aloittelevalta yritykseltä ja uusia yrityksiä syntyy paljon. Erottuakseen on tärkeää saavuttaa ja säilyttää imago luotettavana ohjelmistonkehittäjänä. Varsinkin nuoren yrityksen on vaikea toipua imagon menetyksestä.

Ohjelmistonkehittäjän imago riippuu paljon tämän tuottamien ohjelmien laadusta. Ohjelmiston laatu ei riipu pelkästään sen toiminnallisuudesta, vaan myös sen kyvystä mukautua uusiin vaatimuksiin ja rajoituksiin. Jos ohjelma on toteutukseltaan raskas, monimutkainen ja vaikeasti muokattava, edellä mainittu ei ole mahdollista. Niin uusien kuin vanhojenkin yritysten on voitava luotettavasti arvioida koodinsa laatua.

Koodin laadun arviointi ei ole kovin suoraviivaista. Intuitiivisesti tärkein aspekti on nopeus, jolla ohjelmaa suoritetaan. Usein koneen ohjelman läpikäyntiin kuluttama aika on kuitenkin niin mitätön, ettei mahdollisten suoritukseen kuluvien aikojen vertaaminen ole mielekästä. Vähemmän intuitiivisesti ajatellen, huono koodi johtaa myös heikkoon muokattavuuteen ja uudelleenkäyttöön. Koodin näiden ominaisuuksien arvioimiseksi onkin kehitetty mittareita. Olio-ohjelmoinnin kannalta tärkeitä metriikoita ovat koheesio ja siihen läheisesti liittyvä kytkentä.

Koheesion (engl. cohesion), eli luokan yhtenäisyyden, arviointi auttaa ohjelmistotuottajaa arvioimaan olion tai luokan sisäistä rakennetta ja sen vahvuutta. Korkean koheesion omaava luokka on yhtenäinen ja selkeä kokonaisuus. Tämä johtaa usein myös hyvään uudelleenkäytettävyyteen. Koheesio liittyy läheisesti kytkentään (engl. coupling), joka mittaa luokan kytköksiä muihin luokkiin. Usein korkea koheesio pyritään liittämään matalaan kytkentään, jotka ovat molemmat ohjelmiston kannalta hyviä ominaisuuksia. Molempia mittaamaan on kehitetty erilaisia metriikoita, jotka toimivat hieman eri periaattein.

Esitetyn pohjalta olisi äärimmäisen hyödyllistä tietää, jos metriikoissa on suuria eroja tai puutteita toisiinsa nähden. Jos yksi metriikka on toista yleispätevämpi ja helpommin sovellettava, on sille enemmän käyttöä. Jos jokin puute metriikassa estää sen käytännöllisen soveltamisen ohjelman koodiin, ei kyseistä metriikkaa kannata arvioinnissa soveltaa. Erilaisten metriikoiden puutteiden tiedostaminen on tärkeää.

Tutkielmassa käsitellään kolmea eri koheesiometriikkaa ja arvioidaan niiden käytännöllisyyttä. Tämä toteutetaan soveltamalla niitä ennalta valittujen projektien lähdekoodeihin ja tulkitaan näin saatuja tuloksia. Mahdolliset erot (tai niiden puute)

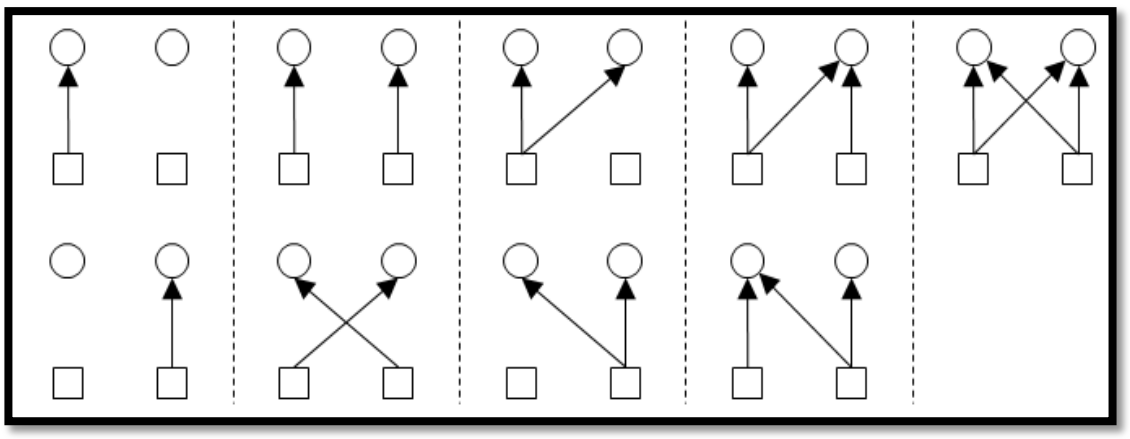
metriikoissa selviävät näiden testien pohjalta. Jos kaksi metriikkaa sovellettuna samaan koodiin luovat erilaisen kuvan toteutuksen koheesiosta, voidaan metriikoiden katsoa olevan ratkaisevasti erilaisia. Luvussa kaksi perehdytään eri metriikoihin ja niiden toimintaperiaatteisiin. Luvussa kolme suoritetaan valittujen metriikoiden soveltaminen ja luvussa neljä analysoidaan saatuja tuloksia. Työn aikana pyritään näin kartoittamaan eri metriikoiden soveltuvuutta koodin koheesion arvioimiseen ja niiden käytännönläheistä arvoa.

2 KOHEESIOMETRIIKAT

Koheesio mittaa luokan yhtenäisyyttä. Se on perusajatukseltaan intuitiivinen mittari koodin laadulle. Mitä yhtenäisempi luokka on, sitä selkeämpi sitä on käsitellä ja helpompi käyttää uudelleen eri yhteyksissä. Olio on selkeä kokonaisuus, jota on helppo muokata. Koheesion kasvaessa myös ulkopuolisen on helpompi tulkita koodia.

Koheesiometriikat eivät yleensä keskity moduulin tai luokan niin sanottuun semanttiseen koheesioon, eli siihen onko olio rakennettu järkevästi edustamaan tarkoitustaan. Tämä riippuu liikaa suunnittelijan näkökannasta. Metriikoissa keskitytään nimenomaan syntaktisen koheesion arvioon, eli siihen miten moduulin (luokan) sisäiset attribuutit ja metodit ovat kytköksissä. Jos olion attribuuttia ei käytetä tai metodi ei vaikuta muihin sen komponentteihin, laskevat ne luokan koheesioarvoa (kuva 1). Jollei näiden niin sanottujen turhien komponenttien uudelleensijoittamista vastaan ole pätevää syytä, tulisi ne toteuttaa joko eri luokassa tai luoda niitä varten kokonaan oma olionsa. [2]

Koheesiometriikoita on useita ja moni niistä rakentuu aikaisemmin kehitetyn metriikan päälle lisäten siihen jotain. Suurimmat erot ovat komponenttien kytkeytyneisyyden määrittelyissä ja koheesioarvojen laskukaavoissa.



Kuva 1 - Luokan metodien kytköksiä attribuutteihin visualisoituna. Oikealle siirryttäessä koheesio on korkeampi [1]

2.1 LCOM1-4

LCOM, eli Lack of Cohesion in Methods (koheesion puute metodeissa), kuuluu käytetyimpiin koheesiometriikoihin. LCOMilla saadut arviot ovat suoraviivaisia ja arvojen vaihteluväli on tyypillisesti nollasta äärettömään tai nollasta yhteen. Kun LCOM-arvo on yksi tai nolla, on luokan koheesioarvo korkea. Arvon ollessa suurempi kuin yksi on arvo heikentynyt. Koska LCOM mittaa nimenomaan koheesion puutetta, tarkoittaa suuri arvo heikentynyttä koheesiota. LCOMista on monia versioita, joihin viitataan eri nimin joko numeroiden tai alkuperäiseen metriikan kehittäjään perustuen.

LCOMia on kehitetty eteenpäin alkuperäisestä muodostaan, joka tunnetaan nykyään LCOM1:nä. LCOM1:n esittivät Shyan Chidamber ja Chris Kemerer vuonna 1991 [3]. Se antoi koheesiolle arvon yksinkertaisesti sen mukaan, kuinka monta paria voidaan muodostaa luokan metodeista siten, ettei niillä ole yhtään yhteistä attribuuttia. Yhteiseksi attribuutiksi lasketaan olion muuttuja, jota metodissa hyödynnetään. Kuvan 2 luokkiin pätisivät siis seuraavat lauseet: LCOM1 = 9 (vasen luokka) ja LCOM1 = 8(oikea luokka). [1]

Chidamber ja Kemerer esittivät myöhemmin LCOMille kaavan, joka tunnetaan nykyään myös nimellä LCOM2 [4]. LCOM2 on LCOM1:n tavoin yksinkertainen koheesion mitta. Se perustuu seuraavaan kaavaan:

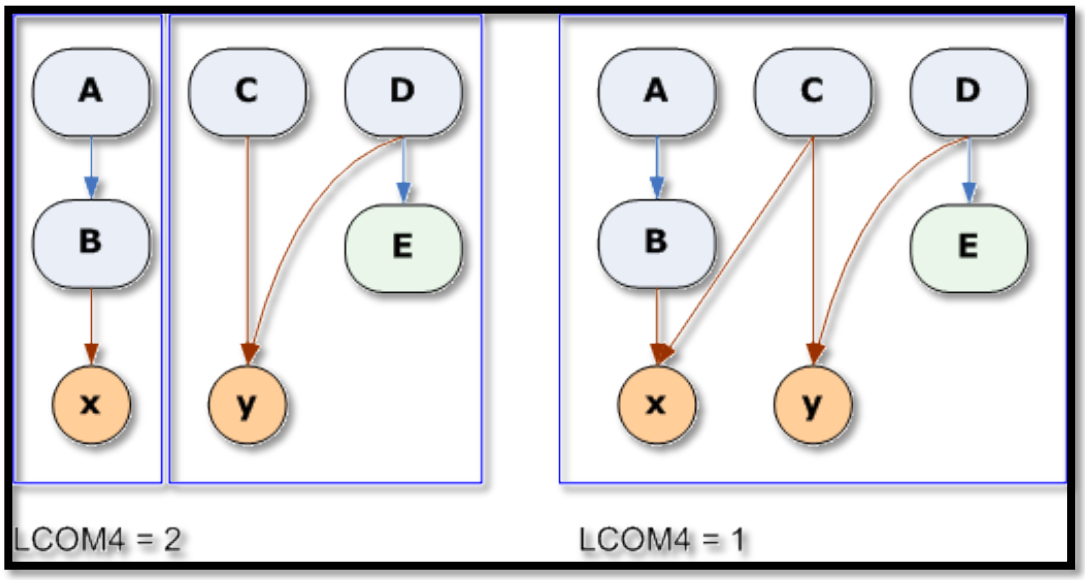
$$LCOM2 = \begin{cases} P - Q, kun\ P \geq Q \\ 0, kun\ P < Q \end{cases}, \qquad (k1)$$

missä P on niiden metodiparien määrä, joilla ei ole yhtään yhteistä attribuuttia ja Q niiden metodiparien määrä, joilla on. [1]

Wei Li ja Sallie Henry kehittivät metriikan, joka tunnetaan LCOM3:na [5]. LCOM3:n perusajatuksena on muodostaa metodeista yhtenäisiä kokonaisuuksia yhteisten rajapintojen kautta. Jos kahdella metodilla on yhteinen attribuutti, on niillä yhteinen rajapinta. Jos metodeilla A ja B on yhteinen rajapinta ja metodeilla B ja C on yhteinen rajapinta, muodostavat A, B ja C yhdessä kokonaisuuden. LCOM3 saa arvonsa suoraan näiden kokonaisuuksien määrästä. Jos metodien A, B ja C lisäksi luokassa on metodi D, jolla ei ole yhtään yhteistä rajapintaa muiden metodien kanssa, on luokassa kaksi kokonaisuutta ja näin ollen LCOM3 = 2. [1]

Martin Hitzin ja Behzad Montazerin LCOM3sta vielä kehittämä LCOM4 [6] huomioi rajapinnaksi myös metodien väliset kutsut. Jos metodit siis ovat keskenään näin

yhteydessä, kuuluvat ne samaan kokonaisuuteen (kuva 2). Jos LCOM3- tai LCOM4-arvo on nolla, ei luokassa ole yhtään metodia.[1]



Kuva 2 - Kahden luokan esitys ja LCOM4-arvot [2]

2.2 LCOM5

Nykyään LCOMista puhuttaessa tarkoitetaan yleisimmin Brian Henderson-Sellersin, Larry Constantinen ja Ian Grahamin kehittämää metriikkaa [7], jota yleisesti kutsutaan myös nimillä LCOM5 tai LCOM*. Tämä metriikka arvottaa LCOMin yhden ja nollan välille desimaalilukuna ja käyttää seuraavaa kaavaa:

$$LCOM5 = \frac{\left(\frac{1}{a}\sum_{j=1}^{a} m(A_j)\right) - m}{1 - m} \qquad (k2)$$

missä a = attribuuttien lukumäärä, j juoksee yhdestä ylöspäin, m = metodien lukumäärä ja $m(A_j)$ = yhteen attribuuttiin viittaavien metodien lukumäärä.

Huomattavaa on tämän metriikan pohjautuvan lähtökohtaan, jossa jokainen metodi viittaa vähintään yhteen attribuuttiin. Tällöin luokka, jossa neljä metodia vaikuttavat jokainen eri attribuuttiin kuvastaa huonointa mahdollista tapausta. Mitä suurempi arvo, sitä heikompi koheesio (yksi ollen huonoin mahdollinen arvo).

2.3 TCC ja LCC

TCC, eli Tight Class Cohesion (tiukka luokkakoheesio) [8], ja LCC, eli Loose Class Cohesion (löysä luokkakoheesio) [8] ovat Linda Ottin, James Biemanin, Byun-Kyoo Kangin ja Bindu Mehran vuonna 1995 kehittämät koheesiometriikat. Ne kuvaavat luokan metodien kytkeytyneisyyttä sen mukaan, mihin attribuutteihin ne vaikuttavat joko suoraan tai toista metodia kutsumalla. Jos metodeilla A ja B on yhteinen attribuutti, on niiden välillä suora yhteys. Jos A:n suoritukseen kuuluu B:n kutsu ja B kytkeytyy attribuuttiin, on A:n ja B:n välillä myös suora yhteys. Jos taas A kutsuu B:n suoritusta ja A kytkeytyy attribuuttiin, ei A:n ja B:n välillä ole suoraa yhteyttä.

TCC:n arvon laskenta perustuu seuraavaan kaavaan:

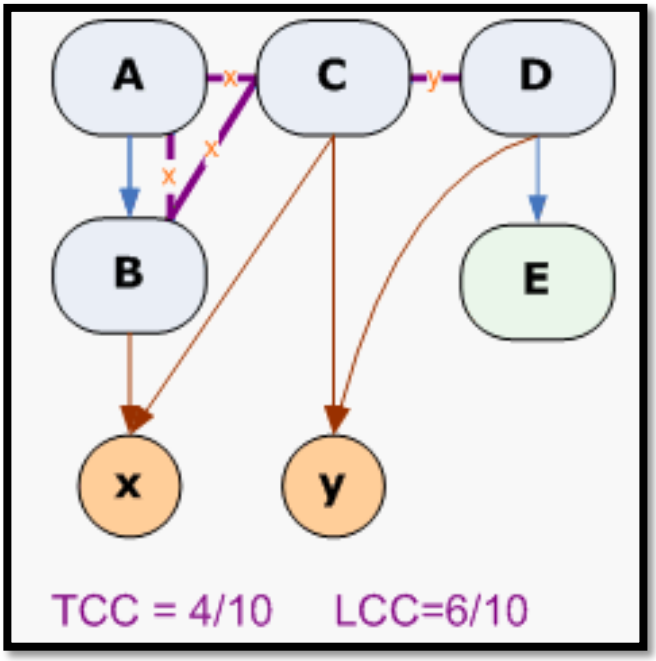
$$TCC = \frac{NDC}{NP},\ NP = \frac{N*(N-1)}{2}, \qquad (k3)$$

missä *NDC* = suorien yhteyksien määrä (Number of Direct Connections), *NP* = mahdollisten yhteyksien määrä ja *N* = metodien määrä. LCC on nimensä mukaisesti vain löysempi versio TCC:stä. LCC:ssä otetaan huomioon myös epäsuorat yhteydet. Metodilla C on epäsuora yhteys metodiin A, jos sillä on suora yhteys metodiin B, jolla on suora yhteys A:han. LCC:n kaava on:

$$LCC = \frac{NDC+NIC}{NP}, \qquad (k4)$$

missä *NIC* = epäsuorien yhteyksien määrä (Number of Indirect Connections).

Molempien TCC:n ja LCC:n mahdollisten arvojen vaihteluväli on nollasta yhteen, jossa nolla tarkoittaa heikointa mahdollista koheesiota (ei yhtään yhteyttä) ja yksi parasta mahdollista koheesiota (kaikki metodit ovat yhteydessä toisiinsa). Määritelmien pohjalta huomataan, että kun LCC = 0 pätee myös TCC = 0. Jos TCC:n tai LCC:n arvo on alle 0,5, katsotaan luokan koheesion olevan heikko. [2]



Kuva 3 - TCC- ja LCC-arvot luokalle [2]

2.4 CAMC

Vuonna 1998 Jagdish Bansiya, Letha Etzkorn, Carl Davis ja Wei Li kehittivät uuden metriikan koheesion mittausta varten. CAMC, eli Cohesion Among Methods in Class (koheesio luokan metodeiden välillä) [9], ei perustunut luokan attribuuttien ja metodien välisten yhteyksien määrään, vaan attribuuttien laadun ja metodien parametrien suhteeseen.

CAMCin lähtökohta on seuraava: koska kaikki metodit pääsevät käsiksi luokan omiin attribuutteihin, kertovat metodin mahdolliset parametrityypit sen saamasta ulkoisesta informaatiosta ja tätä kautta myös sen funktionaalisesta linkittyvyydestä. Toisin sanoen, jos moni luokan metodi saa vaikkapa kokonaislukutyyppisiä (`Integer`) arvoja parametreissaan, olisi oletettavaa että nämä luokat käsittelevät toisiinsa läheisesti liittyvää informaatiota ja ovat koherentteja informaatiotasolla.

CAMC arvo mittaa yksittäisten metodien parametrityyppien leikkauksen kaikkien luokan metodien parametrityyppilistan kanssa. Luokan vastaava CAMC arvo voidaan laskea käyttämällä seuraavaa kaavaa:

$$CAMC = \frac{\sum_{i=1}^{n} |P_i|}{|T| * n} \qquad (k5)$$

jossa n = metodien määrä, $|T|$ = kaikkien metodien parametrityyppien unionin parametrityyppien lukumäärä ja $|P_i|$ = yksittäisen metodin parametrityyppien lukumäärä. Tässä jokaisen metodin parametreihin lasketaan myös luokka itsessään. Siihen viitataan usein niin sanottuna `this`- tai `self`-parametrina. `This`-parametrista johtuen jokaisella metodilla on kaavassa *k5* vähintään yksi parametri. [9]

```
class Employee {

  public:
        Employee () { name = address = 0; }
        double Pay(int no_hours);
        double Tax(double tax_rate, int no_deductions);
        double SetPayRate (double new_pay_rate);
        void   ChangeAddress(char *new_address);
  private:
        char *name;
        char *address;
        double pay_rate;
        int hours_worked;
};

class EmployeeNode {
   public:
         EmployeeNode (Employee * obj,
                       EmployeeNode * _next = 0);
   private:
         Employee *pEmp;
         EmployeeNode *next;
};

class EmployeeList {
   public:
         EmployeeList (void);
         void Add(Employee obj);
         void Delete(Employee obj);
         int Length(void);

   private:
         EmployeeNode *start, *end;
         int length;
};
```

CAMC Calculations For Class Employee :

T = { this, char, int, double } and | T | = 4

$P_{Constructor}$ = | { this } ∩ T | = 1
P_{Pay} = | { this, int } ∩ T | = 2
P_{Tax} = | { this, double, int } ∩ T | = 3
$P_{ChangePayRate}$ = | { this, double } ∩ T | = 2
$P_{ChangeAddress}$ = | { this, char } ∩ T | = 2

$CAMC_{Employee}$ = ∑ {1, 2, 3, 2, 2}/ 4 × 5 = 10 / 20 = 0.5

CAMC Calculations For Class EmployeeNode :

T = { this, Employee } and | T | = 2

$P_{Constructor}$ = | { this, Employee } ∩ T | = 2

$CAMC_{EmpNode}$ = ∑ {2}/ 1 × 2 = 2 / 2 = 1.0

CAMC Calculations For Class EmployeeList :

T = { this, Employee } and | T | = 2

$P_{Constructor}$ = | { this } ∩ T | = 1
P_{Add} = | { this, Employee } ∩ T | = 2
P_{Delete} = | { this, Employee } ∩ T | = 2
P_{Length} = | { this } ∩ T | = 1

$CAMC_{LinkedList}$ = ∑ {1, 2, 2, 1}/ 2 × 4 = 6 / 8 = 0.75

Kuva 4 - Esimerkkejä luokista ja niiden vastaavista CAMC-arvoista [8]

Steve Counsellin, Stephen Swiftin ja Jason Cramptonin vuonna 2006 julkaisema tutkielma [10] päivitti CAMCin kaavaa seuraavasti:

$$CAMC = \frac{1}{kl} \sum_{i=1}^{k} \sum_{J=1}^{l} o_{ij} = \frac{\sigma}{kl} \qquad (k6)$$

jossa k = metodien määrä, l = luokassa parametrityyppeinä esiintyvien muuttujatyyppien määrä. o_{ij} = i:nnen metodin j:ttä parametrityyppiä edustavien parametrien määrä siten, että

$$o_{ij} = \begin{cases} 1, jos\ j{:}s\ paparametrityyppi\ löytyy\ i{:}nestä\ metodista \\ 0\ muuten \end{cases} \quad (k7)$$

Luonnollisesti $\sigma = \sum_{i=0}^{k} \sum_{j=0}^{l} o_{ij}$. Huomattavaa on, ettei kaava *k6* ota huomioon `this`-parametria.

Koska *k6* ei huomioi `this`-parametria, on mahdollista että o_{ij} = 0. Määritelmän pohjalta ei voi kuitenkaan käydä niin, että σ = 0, ellei jokainen luokan metodi ole parametriton. Tällöin toisaalta myös pätee, että kl = 0. CAMCin raja-arvoille pätee seuraava lause:

$$0 < CAMC \leq 1 \quad (k8)$$

Mitä suuremman arvon CAMC saa, sitä koherentimpi luokka on. CAMCin arvolla 1 jokainen luokan metodi saa parimetriensä joukossa jokaista luokassa esiintyvää parametrityyppiä.

2.5 Kytkentä

Kytkentä (engl. coupling) mittaa luokkien ja kokonaisuuksien välisiä yhteyksiä ja riippuvuutta. Käytännössä mitä enemmän moduulien välillä on riippuvuutta aiheuttavia tekijöitä, sitä suurempi niiden välinen kytkentäarvo on.

Koheesion siis mitatessa luokan sisäisiä kytköksiä, mittaa kytkentä luokan ulkoisia kytköksiä. Vaikka koheesio näin muistuttaakin kytkentää, kertoo suuri moduulienvälinen kytkentä huonosta ohjelmistosuunnittelusta. Näin koheesio ja kytkentä ovat toistensa vastakohtia ja yleensä korreloivat keskenään; mitä pienempi kytkentäarvo, sitä suurempi koheesioarvo ja päinvastoin.

Luokka, joka nojaa paljolti toisten luokkien toteutuksen varaan vaikeuttaa ohjelman yleistä muuttamista ja uudelleensoveltamista [3]. Jos jotain tiettyä osaa luokassa halutaan muokata, saattaa se vaikuttaa muiden kokonaisuuksien toimintaan. Tällöin joudutaan muitakin muokkaamaan, mikä ei yleensä ole toivottavaa tai tarkoituksenmukaista.

Kytkentä jaetaan mittauksen suhteen kahteen lajiin: afferenttiin (engl. afferent coupling, C_a) [11], eli sisäänpäin kantautuvaan, ja efferenttiin (engl. efferent coupling, C_e) [11], eli ulospäin kantautuvaan. Afferentti kytkentä mittaa sitä, kuinka moni luokka riippuu tämän luokan ominaisuuksista ja funktionaalisuudesta, kun taas efferentti kytkentä keskittyy luokan omaan riippuvuuteen muista [11]. Efferenttiä kytkentää on huomattavasti helpompi mitata, kuin afferenttia (yhden moduulin toiminnasta saattaa riippua vaikka kuinka monen muun moduulin

toiminta). Yleensä kytkennän mittausta ei tehdä luokkatasolla, vaan joko package- tai projektitasolla.

3 METRIIKOIDEN SOVELTAMINEN

Sovellettaviksi metriikoiksi on valittu luvussa kaksi esitettyjen metriikoiden joukosta LCOM4, LCOM5 ja CAMC. LCOM5:n laskemiseen on käytetty Eclipsen Metrics-liitännäistä [12], muut on laskettu käsin. Koska osa arvoista joudutaan laskemaan käsin, pyritään metriikoita soveltamaan luokkatasolla.

Osa kappaleessa 2 esitetyistä metriikoista on perusideoiltaan samanlaisia (LCOM4 on samankaltainen TCC:n ja LCC:n kanssa) ja sovelletut tekniikat on valittu niiden joukosta laskutapojen ja taustalla olevien ajatuksien erilaisuutta silmälläpitäen. LCOM4 ja LCOM5 laskevat eri tavoin metodien ja attribuuttien välisiä viittauksia, ja CAMC perustuu ajatukseen käsiteltävän informaation yhtenäisyydestä.

Työssä tarkasteltavat projektit rajoittuvat yhden package-tason sisään, eivätkä viittaa sen ulkopuolelle. Koska luvussa 2.5 todettiin kytkentää useimmiten tarkasteltavan package-tasolla, voidaan sen suoraan todeta kaikilla projekteilla olevan nolla. Niiden kytkennän tarkastelussa ei näin ole yhtään mielekästä aspektia.

3.1 Valitut projektit

Työssä tarkasteltavat projektit on valittu kahden pääperiaatteen mukaan. Ensin, projektien tulee olla tarpeeksi yksinkertaisia, jotta niiden tarkasteleminen ja metriikoiden tulosten yksityiskohtainen analysointi olisi mahdollista. Toiseksi, projekteissa täytyy olla jokin aspekti, joka tekee niistä työn kannalta mielenkiintoisia. Jokaista projektia tullaan esittelemään yksityiskohtaisemmin. Huomattavaa on, ettei projekteissa oteta kantaa koodin sellaisiin aspekteihin, jotka eivät suoraan liity työhön ja täten ole mielekkäitä sen kannalta. Erityisenä huomautuksena: lähdekoodin ei tarvitse olla toimivaa tai palvella spesifistä tarkoitusta jotta se voisi olla analysoinnin kohteena.

Ensimmäiseksi projektiksi on valittu freesourcecode.net-sivustolta Pong-projekti. Tässä koodissa tarkasteltavassa luokassa on kaksi aliluokkaa, 27 luokka-attribuuttia ja 19 metodia (attribuutit ja metodit listattu liitteessä A). Voidaan myös huomata, että kolme metodeista (`mouseDragged`, `keyReleased`, `keyTyped`) on vain nimetty, eivätkä omaa varsinaista toiminnallisuutta. Tämä on tehty, jotta metodi toteuttaisi kaikki tarvittavat rajapinnat. Projektia aiotaan käsitellä kolmen erillistapauksen kautta:

1.a Projekti sellaisenaan

1.b Projekti, jossa aliluokat on muutettu erillisiksi omiksi luokikseen

1.c Projekti, josta on poistettu kolme toteuttamatonta metodia

Huomattavaa on, että 1.c-kohtaa varten joudutaan luokkaa muokkaamaan ja sen rakenteeseen vaikuttamaan, jotta sen funktionaalisuus säilyisi ja pysyisi validina arvioinnin kohteena. Alustavasti tarkasteltuna projekti vaikuttaa melko keskinkertaiselta: luokan koheesio vaikuttaa normaalilta, osa metodeista viittaa moneen muuttujaan, osa ei viittaa kuin muutamaan. On selvää, että osa metodeista on lähinnä toteutettu näennäisesti ja rajapintojen vaatimusten vuoksi. Kohdassa 1.a ei myöskään oteta kantaa aliluokkien koheesioon. Ennen metriikoiden soveltamista voidaan luokan koheesio arvioida keskivertoa heikommaksi.

Toiseksi projektiksi on valittu freesourcecode.net-sivustolta InsertionSort-projekti. Projektin lähdekoodissa ei ole yhtään attribuuttia, mikä on mielenkiintoinen aspekti. Huomataan myös projektin lähdekoodin LCOM4 ja -5 arvojen olevan parhaat mahdolliset (1 ja 0), koska kaikki kolme metodia viittaavat toisiinsa ja muodostavat täten yhden kokonaisuuden. Esimerkkiluokka on pieni ja alustavasti tarkasteltuna selkeästi korkean koheesion omaava tiivis paketti (luokan metodit ja attribuutit listattu liitteessä B).

Kolmanneksi projektiksi on valittu LapMaster 2013, joka on Javalla toteutettu autopeli. Projektissa on kymmenen luokkaa, jotka kuuluvat `game`-nimiseen package-tasoon. Luokat vaihtelevat pituudeltaan kymmenistä riveistä tuhansiin. Näin laajasta projektista on nopealla katsauksella vaikea todeta koodin laadusta juuri mitään. Projektin `Game`-luokalla on kaksi alaluokkaa, `Score` ja `lapTimeComparator`. Projektia esitellään tarkemmin liitteessä C. Alaluokkien ominaisuudet on merkitty erikseen yläluokan vastaavaan yhteyteen. Projektin kohdalla aiotaan soveltaa koheesiometriikoita sekä luokkakohtaisesti, että suuremmassa mittakaavassa package-tasolla.

3.2 Projekteihin soveltaminen

Koska valituille projekteille saadaan LCOM5 arvo suoraan Eclipsen Metrics-liitännäisen kautta, ei sen kohdalla soveltamiseen paneuduta sen yksityiskohtaisemmin. Liitännäinen toteuttaa LCOM5 laskennan luvun 2.2 mukaisesti.

LCOM4 laskiessa pyritään ensin linkittämään luokkien kattavimmat metodit yhdeksi kokonaisuudeksi. Tämän jälkeen pyritään jokaisen attribuutin ja kokonaisuuden välillä löytämään linkki. Tähän suurempaan kokonaisuuteen pyritään vielä liittämään loput metodit. Jos kaikkia ei saada linkitettyä yhdeksi isoksi kokonaisuudeksi, tulisi pienempien

kokonaisuuksien olla helppo määrittää. Tämän jälkeen LCOM4:n arvo saadaan suoraan kokonaisuuksien määrästä kappaleen 2.1 mukaisesti.

CAMCille arvon laskeminen on vielä edellistä kappaletta suoraviivaisempaa. Eclipsellä saa suoran näkymän metodien nimiin ja niiden parametrityyppeihin, josta on helppo lukea metodien parametrityypit ja laskea sen pohjalta CAMCin arvo.

Sovellettaessa metriikoita projektiin kolme, LCOM4 ja CAMC lasketaan package-tasolla ottamalla kaikkien luokkien arvoista keskiarvo. Tämän lisäksi LCOM4 lasketaan vielä soveltaen luokkatason laskutapaa package-tasolle siten, että määritellään luokkien keskinäiset viittaukset ja määritellään siitä laskettavat kokonaisuudet. Jos luokan toteutuksessa käytetään toista luokkaa, on niiden välillä yhteys. Näin saatu tulos on esitetty taulukossa 2 erikseen nimikkeellä LCOM4*.

3.3 Tulokset

Tässä kappaleessa on esitetty projekteille LCOM4- ja CAMC-metriikkaa käyttäen lasketut, sekä Eclipsen Metrics-liitännäisestä kerätyt LCOM5-koheesioarvot. Projektin 3 `Game`-luokan yhteydessä esiintyville `Score`- ja `lapTimeComparator`-alaluokille on myös laskettu arvot erikseen, ja ne on otettu huomioon package-tason keskiarvoja laskettaessa. Lisäksi kappaleessa 3.2 esitetty LCOM4* on esitetty erillisenä arvona.

Taulukko 1: Projektit 1 ja 2

	LCOM4	LCOM5	CAMC
1.a	5	0,872	0,175
1.b	5	0,872	0,175
1.c	2	0,872	0,146
2.	1	0	0,444

Taulukko 2: Projekti 3

	LCOM4	LCOM5	CAMC	LCOM4*
package-taso	15,667	0,673	0,354	1

Drawhandler	5	1,125	0,333
Menu	142	1,022	0,300
Physics	7	0,982	0,036
Car2	3	0,971	0,078
GameSession	20	0,921	0,278
Track	1	0,9	0,125
StopWatch	2	0,767	0,143
Game	2	0,722	0,167
- Score	1	0	1,000
- lapTimeComparator	1	0	1,000
Key	1	0,667	0,500
InputManager	3	0	0,292

3.4 Huomioita työstä

3.4.1 Ilmenneitä haasteita

Työtä tehdessä kävi ilmi, ettei helppokäyttöisiä tai monia koheesio- ja kytkentämetriikoita laskevia ohjelmia ole juuri olemassa. Tähän vaikuttaa ohjelmien käyttötarkoitus: monet projektikokonaisuudet koostuvat useista pienemmistä projekteista, joissa jokaisessa on itsessään usein monia package-kokonaisuuksia, jotka vastaavasti sisältävät useita luokkia. Tämän kokoisten ohjelmistokokonaisuuksien kokonaisvaltainen arviointi on todella työlästä ja nimenomaan tähän analyysiin on kehitetty paljon automatisoivia ohjelmia (kuten SonarQube[13] tai NDepend[14]). Saatavilla olevat laskentaohjelmat antavat erilaisia manuaalisen laskennan kannalta raskaita arvoja, kuten koodirivien määrän tai staattisten attribuuttien määrän. Ne myös tarjoavat monia koodin laatua arvioivia metriikoita.

Koodin laatua arvioivien ohjelmien ongelmaksi muodostuu kuitenkin toistuvuus ja rajallisuus; monet ohjelmat kierrättävät samoja metriikoita ensin projektitasolla, sitten package-tasolla, sitten luokkatasolla. Tämä käytännössä tarkoittaa, että kun ohjelmat ilmoittavat tarjoavansa useita koheesiometriikoita, ne oikeasti tarjoavat samaa metriikkaa moneen kertaan. Kaiken lisäksi useat ohjelmat keskittyvät laskemaan tiettyjä metriikoita, joita muutkin ohjelmat

laskevat. Useat koheesiota laskevat ohjelmat laskevat joko arvon nimikkeellä koheesio, tai sitten LCOM tarkoittaen LCOM5:tä. Lisäksi mitä syvemmälle luokkahierarkiassa edetään, sitä vähemmän metriikoita on tarjolla; oletettavasti johtuen siitä, että suurempien kokonaisuuksien analysoinnilla on korkeampi prioriteetti.

Huomattavaa on myös, ettei LCOM5 rajoitu taulukon 2 tuloksissa välille 0-1, vaikka Eclipsen Metrics-liitännäinen laskee LCOM-arvon kappaleen 2.2 kuvaamalla tavalla [12]. Mahdollinen syy tälle voi olla, että liitännäinen laskee mukaan myös tapaukset, joissa metodi ei viittaa yhteenkään attribuuttiin tai että esimerkkiprojektiin sovellettaessa jokin onnistui rikkomaan liitännäisen laskukaavan.

3.4.2 Työmäärän jakautuminen

Pohjatyöhön, erilaisten laskutyön automatisointivaihtoehtojen läpikäyntiin ja sopivien esimerkkiprojektien hankkimiseen kului luonnollisesti aikaa, mutta hyödyllisemmäksi nähdään varsinaiseen projektien käsittelyyn kuluneen ajan arviointi.

Koska LCOM5 saatiin työssä suoraan Eclipsen liitännäisestä, sen laskemiseen kulunutta aikaa on vaikea arvioida: kun projekti oli valmiina, ei tarvinnut kuin suorittaa build projektille Metrics-liitännäisen ollessa sallittuna. Tällöin LCOM5 saatiin niin sanotusti saman tien.

CAMCin kohdalla manuaaliseen laskentatyöhön kului aikaa. Huomattavaa oli kuitenkin, että CAMCin laskeminen ei vaatinut luokkien laajamittaista läpikäyntiä, vaan niiden pintapuolinen tarkastelu riitti. Tämän vuoksi laajojenkin luokkien tarkastelu CAMCin avulla oli nopeaa. Yhteensä CAMCin vaatima laskuaika per luokka oli maksimissaan viisi minuuttia ja kaikille projekteille yhteensä noin 30-45 minuuttia.

LCOM4:n soveltaminen vaati eniten työtä. Luokissa piti määrittää olennaiset kytkökset, mikä vaati koodien laajamittaista tarkastelua. Vaikka kaikkia mahdollisia kytköksiä ei tarvinnut määrittää, oli tarpeellisten kytkösten paikallistaminen silti aikaa vievä prosessi. Luokkien koko, tarkemmin attribuuttien ja metodien määrä, vaikutti ratkaisevasti laskuaikaan. Pieni luokka, jossa oli alle kymmenen luokkamuuttujaa ja metodia ei vaatinut kovin montaa minuuttia läpikäymiseen. Sen sijaan useita kymmeniä attribuutteja ja metodeja sisältävä luokka vaati riittävään soveltamiseen 30-75 minuuttia. Yhteensä LCOM4:n soveltaminen vaati työssä arviolta noin 3,5-4,5 tuntia.

4 ANALYYSI

Analyysissä pyritään löytämään sellaisia ominaisuuksia kappaleessa 3. sovelletuista metriikoista, jotka vaikuttavat niiden käytettävyyteen käytännön koodin arviointivälineenä. Arvioinnin kohteena on metriikoiden välittämän informaation laatu, sekä metriikoiden sovellettavuus.

4.1 LCOM5

4.1.1 Informatiivisuus

Ensimmäisessä projektissa LCOM5 tuotti positiivisia tuloksia. Projektin kosmeettinen muokkaaminen ei missään vaiheessa vaikuttanut siitä saatuihin arvoihin. LCOM5 osoittautui siis tältä osin vakaaksi koheesiometriikaksi. Projektissa kaksi LCOM5-arvoksi saatiin nolla, kuten oltiin osattu odottaakin.

Projektissa kolme LCOM5 antoi edelleen tasaisia arvoja, mikä yhä osoitti metriikan vakautta koheesion mittapuuna. Yllättävää kuitenkin oli, ettei arvoja alle 0,5:n ilmennyt nollan lisäksi yhtäkään. Tämä todennäköisesti johtuu siitä, että korkean koheesion omaavat luokat ovat usein hyvin koherentteja ($0,5 < LCOM5 < 0,8$) tai täysin koherentteja ($LCOM5 = 0$). Niin sanotun äärimmäisen korkean koheesion omaavat luokat ovat harvinaisia. Myös package-tasolla LCOM5 saa totuudenmukaisen arvon, mikä edelleen johtuu sen kyvystä tuottaa vakaita arvoja luokkatasolla.

4.1.2 Sovellettavuus

LCOM5 on testatuista metriikoista ehdottomasti helpoin soveltaa. Koska laskentatyö voidaan automatisoida, ei tulosten saantiin kulunut aikaa. Tämä toimii myös metriikan ainoana lievänä haittapuolena. Koska LCOM5:ttä laskettaessa ei luokan tarkempi tarkastelu ole tarpeen, on luokan koheesio täysin laskennan tuloksena saadun arvon varassa. Metriikan on kuitenkin osoitettu saavan arvonsa kohtuullisen luotettavasti, eikä sitä ole mitään syytä epäillä.

4.2 LCOM4

4.2.1 Informatiivisuus

Projekteissa yksi ja kaksi LCOM4 toimi juuri kuten odotettu. Projekteissa 1.a ja 1.b sen arvoa nostivat erityisesti rajapintojen vaatimat niin sanotut toteuttamattomat metodit (metodi nimetty, koska rajapinta vaatii sitä, ja jätetty kuitenkin toteuttamatta). Kun nämä tekijät poistettiin projektissa 1.c, saatiin huomattavasti parempi kuva luokan varsinaisesta koheesiosta. Projekti kaksi oli lähtökohtaisesti LCOM4-arvoltaan 1, mikä testeissä todennettiinkin.

Projektin kolme luokat vaihtelivat koheesioarvoiltaan huomattavasti. LCOM4 antoi luokille arvot, jotka vastasivat yleensä luokkien todellista koheesiota. Huomattavaa oli, että luokkien koon kasvaessa myös LCOM4-arvo usein kasvoi. Tämä on ymmärrettävää ja tulisi ottaa huomioon metriikkaa sovellettaessa. Luokissa `Gamesession` ja `Menu` ilmeni huomattava piikki LCOM4-arvossa, jota käsitellään myöhemmässä kappaleessa.

LCOM4:n ongelmaksi ilmenivät mahdolliset luokkamuuttujien väliset viittaukset. LCOM4 ei määrittele hierarkiassaan tilannetta, jossa luokkamuuttuja viittaa toiseen luokkamuuttujaan. Projektin kolme luokissa `Gamesession` ja `Menu` käytettiin osaa luokkamuuttujista ainoastaan toisten luokkamuuttujien konstruoimiseen, mikä nosti huomattavasti niiden LCOM4-arvoa. Mikäli kyseiset viittaukset olisi käsitelty attribuuttien välisenä ja hyväksyttynä linkkinä, olisi `Gamesessionin` LCOM4-arvo ollut 13 ja `Menun` kuusi.

LCOM4 laskettiin myös kahdella tavalla projektin kolme package-tasolla: luokkien arvojen keskiarvona ja etsien luokkien välisiä viittauksia ja muodostaen niistä package-tason kokonaisuuksia (LCOM4*). Keskiarvoinen laskutapa antoi tulokseksi yli 15, mikä on kyseisen projektin koheesioarvoksi huono arvio. Keskiarvoa nostivat huomattavasti luokkien `Gamesession` ja `Menu` suunnattomiksi kasvaneet arvot. Sen sijaan LCOM4* kuvasi projektin kokonaiskoheesiota paremmin. Projektin luokat muodostivat kokonaisuuden, jota ei voi järkevästi jakaa.

Vaikka LCOM4:n arvo yksinään ei kerro vielä tarpeeksi luokan toteutuksesta, indikoi se kuitenkin mahdollisia epähuomioita. Usein suuri LCOM4-arvo johtui hyödyntämättömistä luokkamuuttujista (luokkamuuttuja lisätty, todettu hyödyttömäksi ja kuitenkin jätetty poistamatta) tai rajapintatoteutusten vaatimista toteuttamattomista metodeista. Ongelmalliseksi muodostuu hyvän ja huonon arvon määrittäminen. Luokan koon kasvaessa myös LCOM4:n raja-arvo kasvaa.

4.2.2 Sovellettavuus

Vaikka LCOM4 on hyvin informatiivinen soveltajalleen, on se samalla raskas. Sen antama käsitys luokan koheesiosta ei varsinaisesti välity pelkän numeron kautta, vaan enemmänkin metriikan soveltamisen aikana. LCOM4:n soveltaminen vaatii luokan läpimittaista tarkastelua, joka antaa itsessään käyttäjälle hyvän kuvan luokan koheesiosta. Samalla metriikka myös vaatii huomattavan määrän työtä. Tottumattomalta soveltajalta menee LCOM4-metriikan selvittämiseen yhdelle laajalle luokalle helposti tunteja. Vaikka soveltamisessa kehittyvä rutiini tiputtaa vaaditun keskimääräisen ajan alle tuntiin, on tarkastelu silti raskasta.

4.3 CAMC

4.3.1 Informatiivisuus

Projektissa yksi CAMC-arvot tuottivat huonoja tuloksia. Projektien 1.a ja 1.b tulokset olivat identtiset, mutta 1.c:n arvo oli näitä huonompi. 1.c:stä oli poistettu kolme toteuttamatonta luokkaa, jolloin koheesion pitäisi parantua. CAMCin arvo sitä vastoin huononee. Samankaltainen käyttäytyminen on huomattavissa projektissa kaksi. Vaikka luokka on äärimmäisen koherentti, saa CAMC arvokseen 0,444. Yhden ollessa maksimi ei annettu arvo korreloi luokan koheesion kanssa ollenkaan.

Projektille kolme CAMC tuotti yhä epävakaita arvoja: kaksi luokkaa, joissa on vain yksi metodi kummassakin, sai täyden koheesioarvon (`Score` ja `lapTimeComparator`). Näiden lisäksi yksikään projektin luokka ei saanut yli 0,5:n CAMC-arvoa. Vaihteluväli arvoille oli 0,036-0,5. CAMCin arvot korreloivat vain ajoittain ja vain osittain luokan varsinaisen koheesion kanssa, eivätkä tulokset silloinkaan osoittaneet (`Score`- ja `lapTimeComparator`-luokkien lisäksi) että kyse olisi muusta kuin sattumasta. Package-tasolla CAMCin keskiarvo ei lähtöarvojen epäluotettavuudesta johtuen ole lainkaan luotettava kuvaus projektin koheesiosta

4.3.2 Sovellettavuus

CAMCin hyvänä puolena voidaan todeta sen olevan todella nopea ja helppo laskea. Luokassa tarvitsee käydä läpi vain metodien parametrit, mikä ei vaadi paljoa aikaa. Laajallekin luokalle CAMCin laskeminen vie aikaa vain muutaman minuutin.

4.4 Metriikoiden korrelaatio

Metriikoiden vastaavuutta arvioidessa on luontevinta analysoida projektia kolme. Projekti on esitetty kappaleen kolme taulukossa siten, että LCOM5 arvot ovat luokkakohtaisesti paremmuusjärjestyksessä huonoimmasta parhaaseen (ylinnä huonoin). On huomattavissa, että vaikkeivät LCOM4 ja LCOM5 täysin korreloikaan keskenään, on osittainen vastaavuus kuitenkin nähtävissä. Taulukon yläpuoliskolla löytyvät huonommat LCOM4-arvot ja alapuoliskolla paremmat. LCOM4 ja LCOM5 vastaavat toisiaan siis ainakin näiltä osin ja työn pohjalta voidaan suurella varmuudella sanoa, että molempien metriikoiden antaessa hyvän koheesioarvon, on luokka lähes varmasti koherentti.

Sen sijaan CAMCia tarkasteltaessa voidaan huomata, ettei minkäänlaista merkittävää järjestystä ole. CAMCin huonoimmat arvot ovat kolmannella ja neljännellä paikalla taulukossa, kun taas parhaat arvot (kun `Score` ja `lapTimeComparator` jätetään huomiotta) sijoittuvat 11:ksi ja ensimmäiseksi. CAMC ei myöskään korreloi lainkaan LCOMien kanssa.

5 YHTEENVETO

Luokan koheesio tarkoittaa luokan yhtenäisyyttä. Koska koheesio vaikuttaa koodin luettavuuteen ja sitä kautta uudelleenkäytettävyyteen ja muokattavuuteen, on sitä mittaamaan kehitetty paljon koheesiometriikoita. Tässä työssä näistä esiteltiin LCOM1-5, TCC, LCC ja CAMC. Lisäksi käsiteltiin hieman kytkennän roolia koheesion vastakappaleena.

Tutkielmassa pyrittiin selvittämään koheesiometriikoiden toimivuutta käytännössä. Tarkasteltavaksi valittiin LCOM5, LCOM4 ja CAMC. Kyseisistä metriikoista arvioitiin ja asetettiin rinnakkain niillä saatuja koheesioarvoja tuoden näin samalla esille metriikoiden eroja. Tähän pyrittiin ensin soveltamalla metriikoita esimerkkiprojekteihin ja analysoimalla saatujen tulosten laatua ja niihin käytetyn työn määrää.

Työssä kävi ilmi että kolmesta sovelletusta metriikasta LCOM5 oli hyvin vakaa ja käyttökelpoinen metriikka, LCOM4 oli informatiivinen, mutta työläs metriikka, ja CAMC todella nopea metriikka koheesion laskemiseen. Tutkielmassa ilmeni LCOM4:lle laskennallisia ja informatiivisia puutteita. Lisäksi CAMC todettiin yleisesti epäpäteväksi koheesiometriikaksi, jonka käyttö työn puitteissa oli hedelmätöntä. Puutteistaan huolimatta LCOM4 voitiin syväluotaavamman (vaikkakin myös työläämmän) luokka-analyysin vuoksi nostaa LCOM5:n rinnalle koheesion arviointimetriikkana. Puutteistaan huolimatta LCOM4 antaa käyttäjälleen hyvän kuvan luokan rakenteesta. Manuaalisen analyysin kautta jopa paremman, kuin LCOM5, sillä analyysin jälkeen arvioija tietää, mikä metriikan arvoa tarkalleen nostaa.

Tutkielman puitteissa selvisi, ettei käyttäjä voi pelkän koheesiometriikan tuottaman arvon pohjalta muodostaa yksiselitteistä käsitystä luokan tai projektin koheesiosta. Vaikka metriikat ovat yleensä yksinkertaisia soveltaa, niitten tarjoama koheesioarvo on yleensä suhteellinen, pikemmin kuin absoluuttinen (esimerkiksi suuremmilla projekteilla tai luokilla on luonnostaan suurempi koheesioarvo). Parhaimmillaan metriikan tuottama huonoon viittaava koheesioarvo kertoo tarpeesta tarkastella projektia tarkemmin ja etsiä mahdollisia puutteita koheesiossa, huonoimmillaan (kuten CAMCin tapauksessa) metriikan arvo ei korreloi projektin koheesion kanssa mitenkään.

LÄHTEET

[1] Ivan Marsic, Rutgersin yliopisto. Luentokalvo 16: Class Cohesion Metrics. < http://www.ece.rutgers.edu/~marsic/books/SE/instructor/slides/> Vierailtu: 3.5.2015

[2] Aivosto oy. Koheesiometriikoiden kuvaus. < http://www.aivosto.com/project/help/pm-oo-cohesion.html> Vierailtu: 3.5.2015

[3] Shyam R. Chidamber & Chris F. Kemerer. 1991. *Towards a Metrics Suite for Object-Oriented Design*. Sloan School of Management, Massachusetts Institute of Technology, USA.

[4] Shyam R. Chidamber & Chris F. Kemerer. 1994. *A Metrics Suite for Object Oriented Design*. IEEE Trans. on Software Engineering, vol 20.

[5] Wei Li & Sallie Henry. 1993. *Maintenance Metrics for the Object Oriented Paradigm*. Software Metrics Symposium, Proceedings.

[6] Martin Hitz & Behzad Montazeri. 1995. *Measuring Coupling and Cohesion in Object-Oriented Systems*. International Symposium on Applied Corporate Computing, Proceedings.

[7] Brian Henderson-Sellers, Larry Constantine & Ian Graham, 1996. *Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design)*. Object Oriented Systems 3.

[8] Linda Ott, James Bieman, Byung-Kyoo Kang & Bindu Mehra, 1995. *Developing Measures of Class Cohesion for Object-Oriented Software*. Proc. Annual Oregon Workshop on Software Metrics.

[9] Jagdish Bansiya , Letha Etzkorn, Carl Davis & Wei Li. 1999. *A Class Cohesion Metric for Object Oriented Design*. The University of Alabama in Huntsville.

[10] Steve Counsell, Stephen Swift & Jason Crampton. 2006. *The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design*. ACM Transactions on Software Engineering and Methodology, Vol. 15.

[11] Robert C. Martin. 2000. *Design Principles and Design Patterns.* <www.objectmentor.com>

[12] Eclipsen Metrics-liitännäisen ohjeistus ja esittely. < http://metrics.sourceforge.net/> Vierailtu: 3.5.2015

[13] SonarSource SA. SonarQuben kotisivut. <http://www.sonarqube.org/> Vierailtu: 3.5.2015

[14] NDepend. NDependin kotisivut. <http://www.ndepend.com/> Vierailtu 3.5.2015

LIITE A: Pong-projektin data

Attribuutit:

1. `gameflag : int`
2. `player1 : boolean`
3. `INIT_ACTIVE : int`
4. `INTRO_ACTIVE : int`
5. `GAME1_ACTIVE : int`
6. `GAME2_ACTIVE : int`
7. `padx : int`
8. `pad1x : int`
9. `score : int`
10. `gameover : boolean`
11. `t : Thread`
12. `dbuffer : Image`
13. `back : Image`
14. `back1 : Image`
15. `ballpic1 : Image`
16. `ballpic2 : Image`
17. `pad : Image`
18. `pad1 : Image`
19. `brickpic : Image`
20. `gameoverpic : Image`
21. `logo : Image`
22. `title : Image`
23. `dbuffer_gfx : Graphics`
24. `brick : Brick[]`
25. `ball : Ball[]`
26. `ball2 : Ball`
27. `logoxy : Ball`

Metodit:

A. `init()`
B. `initgame1()`
C. `initgame2()`
D. `start()`
E. `stop()`
F. `update(Graphics)`
G. `PaintIntro(Graphics)`
H. `PaintGame1(Graphics)`
I. `PaintGame2(Graphics)`
J. `paint(Graphics)`
K. `physicsgame1()`
L. `physicsgame2()`

M. `physicsintro()`
N. `run()`
O. `mouseDragged(MouseEvent)*`
P. `mouseMoved(MouseEvent)`
Q. `keyPressed(KeyEvent)`
R. `keyReleased(KeyEvent)*`
S. `keyTyped(KeyEvent)*`

* = Metodi toteuttamatta

Attribuutteja: 27

Metodeja: 19

Koodirivejä: 706

LIITE B: InsertionSort-projektin data

Metodit:

A. `insert(int[], int, int)`
B. `sort(int[])`
C. `main(String[])`

Attribuutteja: 0
Metodeja: 3
Koodirivejä: 32

LIITE C: LapMaster 2013-projektin data

Luokat:	Koodirivejä:	Metodeja:	Attribuutteja:
1. `Car2`	225	48	34
2. `Drawhandler`	54	3	8
3. `Game`	384	6+2	7+3
4. `GameSession`	520	6	122
5. `InputManager`	39	6	1
6. `Key`	16	2	3
7. `Menu`	1676	10	276
8. `Physics`	942	61	76
9. `StopWatch`	34	7	6
10. `Track`	145	8	10