Excercise 2

Markku Velinen & Niko Hämäläinen

m0701890 & m0701849

To07

1. Tavallisesti teollisuudessa käytetty ohjelmiston laadun mittari on virheiden määrä tuhatta ohjelmariviä kohden. Vertaa, kuinka hyödyllinen mittaustapa on ohjelman kehittäjille tai käyttäjille. Millaisia ongelmia voi aiheutua, mikäli tätä pidetään ainoana ohjelmiston laadun mittarina?

2. Selvitä, millaisia päätelmiä voisit tehdä (tai olla tekemättä) seuraavasta väittämästä:
   "The quality of program X was twice as great as program Y if integration testing revealed program X to have twice as many faults per KLoC as program Y."
   Vapaa suomennos: ohjelman X laatu oli tuplasti parempi kuin ohjelman Y, mikäli integraatiotestauksessa ohjelmasta X löydettäisi kaksinkertainen määrä virheitä ohjelmaan Y verrattuna

3. Selvitä, miksi ohjelman koodirivien määrä (LoC) voi olla hyvä/huono ohjelmiston mittari.

1.

Virheiden määrä tuhatta ohjelmariviä kohden voi antaa kohtuullisen yleiskuvan ohjelmistosta. Menetelmässä hyviä puolia on selkeys, yksinkertaisuus ja helppo verrattavuus. Käyttäjän kannalta tämä menetelmä on loistava. Yhdellä numerolla ohjelmistojen laadun vertailu on käyttäjälle erittäin helppoa.

Menetelmä ei silti ole läheskään aukoton ja ohjelmistoa ei voi mitenkään luotettavasti yksinkertaistaa yhdeksi taianomaiseksi kaiken kertovaksi numeroksi. Esimerkiksi ohjelma A, jossa ei sinällään ole yhtään koodivirhettä, mutta se ei vain ole halutun protokollan mukainen. Sitten ohjelma B, jossa on yksi virhe per 1000 riviä ja täysin protokollan mukainen. Yksinkertaisen numerovertauksen mukaan käytännössä käyttökelvoton ohjelma A olisi parempi tässä tapauksessa.

Lisäongelmia menetelmä tuo jos kehittäjät ovat tietoisia siitä, että heidän koodinsa virhetiheyttä rivejä kohden tutkitaan. Siinä tapauksessa voidaan esimerkiksi kirjoittaa koodia, jossa jokaisella rivillä on yksi kirjain (mikäli kieli sallii tämän). Näin saadaan helposti kertaluokkaa pienempi virhetiheys ohjelmaan. Tosin koodin ajaminen koodinormalisoijan läpi korjaa osittain tämän ongelman.

Lisäksi tämä voi aiheuttaa ylimääräistä koodin kahdennusta. Jos jossain on esimerkiksi 50 rivinen funktio, jonka toiminta on taattu, niin sehän on erittäin kannattavaa kopioida ja liittää niin moneen paikkaan kuin mahdollista. Näin toimivan koodin osuus nousee huomattavasti ja virheiden suhteellinen määrä vähenee.

Oikeastaan varsinainen neronleimaus tämän systeemin kanssa olisi tehdä funktio, joka ei tee mitään. Sitten tehdään toinen funktio, jossa kutsutaan tätä ensimmäistä funktiota esimerkiksi 100000 kertaa (kutsu per rivi, käyttämättä mitään toistorakenteita tai vastaavia) tai niin paljon kuin rivejä nyt halutaankaan. Sitten kumpaakaan näistä funktioista ei kutsuttaisi missään ulkopuolella, joten kääntäjä optimoi ne pois, eikä luodun binäärin koko kasva tai suorituskyky kärsi turhaan.

Toteaisin siis, että menetelmä voi toimia jossain määrin kunhan arvioitavan koodin kirjoittajat eivät ole arvioinnista tietoisia.

2.

Tehtävän 2 lausahdus oli itsessään hyvin epämääräinen, eikä kuulostanut kovinkaan järkevältä. Tutkinnan seurauksena kuitenkin löytyi, että nämä mittausmenetelmät ovat yliobjektiivisia menetelmiä. Lausahdus perustellaan siten, että mitä enemmän virheitä löydetään integraatiotestauksen aikana, niin sitä enemmän niitä keretään korjata ennen tuotteen valmistumista. Tämä johtaa siihen, että pienemmillä virhemäärillä odotetaan suurempaa virhemäärää lopputuotteessa. Kyseinen lähestymistapa on siis hyvin objektiivinen, eikä ota kantaa muuhun kuin numeroihin."

3.

Koodirivien määrä antaa yleensä hyvän kuvan ohjelmistokomponentin laajuudesta, mikä usein myös helpottaa ylläpitoon liittyvien arviointien tekemistä. Tämä tarkoittaa sitä, että tunnettaessa koodirivien määrä, voidaan antaa suuntaa näyttävä arvio siitä, kuinka kauan esimerkiksi uuden ominaisuuden lisääminen kyseiseen komponenttiin tulisi kestää. Koodirivien määrän perusteella voidaan myös laskea arvio ohjelmiston hinnasta.

Aivan näin objektiivisesti asiaa ei kuitenkaan voida käsitellä, sillä koodirivien määrä ei kerro ollenkaan koodirivien laadusta mitään, saati sitten ohjelman toiminnallisesta toteutuksesta. Bill Gates aikoinaan sanoi, että ohjelmakehityksen mittaaminen koodirivien

määrässä on kuin mittaisi lentokoneiden painolla lentokoneiden kehitystä. Eikä hän ole väitteessään millään tavalla väärässä, sillä koodirivien määrää pystytään hyvin monella eri tavalla manipuloimaan. Yksi saattaisi kirjoittaa kaiken yhdelle riville, jolloin koodirivien mukaan kyseisen ohjelmoijan tuottavuus on lähes nollassa. Kun taas toinen ohjelmoija kirjoittaisi jokaisen sana omalle rivilleen, jolloin kyseinen ohjelmoija olisi painonsa arvoinen timanteissa. Kumpikin edellä mainituista tavoista on kuitenkin erittäin huonolaatuista ohjelmointia. Pelkkiin numeroihin ei siis kannata luottaa.

Otan työelämästä hyvän esimerkin tilanteesta, jossa koodirivien määrä antaisi väärän kuvan komponentin kompleksisuudesta ja laadusta. Vanha komponentti A on koodirivien määrältä noin viisi tuhatta riviä, päällisin päin toteuttaa kaikki toiminnalliset vaatimukset, mutta on hyvin epäluotettava, testaamaton sekä ylläpitokelvoton. Koodi on dokumentoimatonta, vaikealukuista sekä useista paikoista löytyy koodin dublikointia. Edellämainittujen syiden takia kyseinen komponentti päätettiin kirjoittaa kokonaan uudestaan, jolloin se korvattiin komponentilla B. B:n lopulliseksi kooksi tuli noin kymmenen tuhatta riviä koodia, eli koodirivejä oli tuplasti enemmän kuin komponentti A:ssa.

Analysoimalla koodirivien määrän perusteella voitaisiin sanoa, että komponentti A on parempi ja komponentin B kanssa ollaan vain tuhlattu aikaa. Näin ei kuitenkaan ole, sillä komponentti B on hyvin testattu, sisältäen useita satoja yksikkötestejä sekä hyvin kattavat integraatiotestit. Komponentti B on myös hyvin dokumentoitu, täyttää toiminnalliset vaatimukset paremmin sekä alustavien kokeilujen perusteella myös luotettavampi. Koodia on esitelty muille kehittäjille ja heidän mielestään koodi oli myös paljon selkeämpilukuista kuin komponentin A lähdekoodi. Eli komponentti B näyttää olevan myös paremmin ylläpidettävissä kuin A.

Tuloksena on, että koodirivien määrän käyttäminen ohjelmakomponentin arvioimiseen voisi toimia, mikäli kaikki ohjelmakoodi kirjoitettaisiin aina samalla tavalla. Näin ei kuitenkaan ole, sillä jokaisella ohjelmoijalla on väistämättä erilainen tausta, erilainen kokemus ja ammattitaito. Toiset ohjelmoijat tuntevat ja keksivät ytimekkäitä algoritmejä, kun taas toisilla saman tekemiseen kuluu enemmän koodirivejä. Molempia tapoja käyttäen, toteuttaminen voi kestää yhtä kauan. Viisi laadukasta koodiriviä saattaa helpostikin vastata viittäkymmentä huonoa koodiriviä.