Henri Haverinen H6

Jokaisella testauskattavuuden kriteerillä on omat vahvuudet ja heikkoudet. Samoin jokaisella on myös omat käyttökohteensa, joihin se soveltuu muita paremmin. Peruslohko- ja haarakattavuus eivät toisistaan poikkea paljoa, joskin näkisin haarakattuvuuden parempana vaihtoehtona näistä kahdesta. Peruslohkokattavuudella ei voi esimerkiksi if-lauseita testata ehdolla, joka evaluoituu epätodeksi. Esimerkki 1 kuvastaa tilannetta, jossa pelkällä tosi-arvolla testaaminen tuottaa jo 100% kattavuuden kun käytetään peruslohkokattavuuden kriteeriä, piilottaen potentiaalisesti bugin mikäli ehto onkin false. Haarakattavuus antaa siis todenmukaisemman tuloksen, ja esimerkissä 1 haarakattavuuden perusteella 100% kattavuus saavutetaankin vain tosi- ja epätosi-arvoilla testatessa. Haarakattavuus pyrkiikin ratkaisemaan ongelmia joita peruslohkokattavuudessa on. Haarakattavuudessakin on kuitenkin puutteita, esimerkiksi ehtolauseiden oikosulkukäyttäytymisen johdosta kaikkia ehtoja ei välttämättä oikeasti testata, vaikka haarakattavuus antaisikin 100% kattavuuden.

Sekä peruslohko- että haarakattavuuden etuja ovat niiden helppo ja edullinen toteuttaminen. Myös erilaisia työkaluja useille ohjelmointikielille löytyy. [1] [2] [3] Lisäksi testitapauksien kehittäminen on mielestäni varsin helppoa ja yksinkertaista: tee testitapauksia, aja testit, katso mitä osioita tesit eivät kata, kehitä testitapauksia jotta kattamatta jääneet osiot katetaan, aja testit uudestaan, tutki tulokset jne.

Molemmat sopivat esimerkiksi perusluontoiseen testaamisen jolla pyritään varmistamaan, että kaikki kirjoitettu koodi suoritetaan, ts. ohjelmassa ei ole kuollutta koodia. Haarakattavuus testaa lisäksi, että ehtolauseet tulee testattua niin tosi- kuin epätosi-arvoilla. Edellämainituista syistä haarakattavuus on parempi vaihtoehto.

Esimerkki 1 (mukailtu sivulta: `http://www.bullseye.com/statementCoverage.html#b1`).

```
Integer x = null;
if (ehto) {
   x = 1;
}
y = x * 2; // jos ehto on epätosi niin tässä nousee NullPointerException
```

Aktiivisten ehtojen kattavuus on hyvä tapa testata monimutkaisempia ehtolauseita. Lisäksi aktiivisten ehtojan kattavuus vähentää testitapauksia verrattuna kaikkien mahdollisten yhdistelmien testaamiseen. Aktiivisten ehtojen kattavuuden heikkoutena on, että se on mahdollista toteuttaa ilman että haarakattavuus täyttyy. [4] Lisäksi itse koen aktiivisten ehtojen kattavuuden toteuttamisen välillä hieman työlääksi, lisäksi valmiita työkaluja ei taida juuri löytyä.

Syötealueen osituksen kattamiselle on useita eri tapoja, joskin yksinkertaisin lienee yhden osituksen kattaminen jossa jokainen ekvivalenssiluokka tulee kattaa vähintään yhdellä testitapauksella. Ekvivalenssiluokkiin jakaminen ja yhden osituksen kattaminen vähentää tarvetta useille testitapauksille. Yhdistelmäkattavuus sen sijaan vaatii, että kaikki mahdolliset ekvivalenssiluokkien yhdistelmät tulee testata. Tämä takaa laajan ja kattavan testauksen, mutta tilanteesta riippuen testitapauksien määrä voi kasvaa suuriksi ja työläiksi tehdä. Tilanteesta riippumatta näen itse ekvivalenssiluokkien käyttämisen joka tapauksessa hyödylliseksi, sillä se selkeyttää ja vähentää

1. https://coverage.readthedocs.io/en/coverage-4.4.1/ (python)
2. http://www.eclemma.org/jacoco/ (java)
3. https://stackify.com/code-coverage-tools/ (list of different coverage tools)
4. http://swtv.kaist.ac.kr/courses/cs453-fall09/Ch3-2-Logic.pdf

testitapausten laatimista. Erityisesti yksikkötestauksessa ekvivalenssiluokkien ja erilaisten osituksien tekeminen on suotavaa ja työmäärää vähentävää.

Mutaatiotestaus on hyvä tapa testitapauksien kehittämiseen, sillä työkalu paljastaa hyvin tarpeita uusille testitapauksille. Mutaatiotestauksen työkalut ovat kuitenkin ilmeisesti harvassa ja ne eivät ole välttämättä kovin hyvälaatuisia. Omat kokemukset mutaatiotestaustyökalusta PIT:stä olivat positiiviset, sillä sen käyttö oli helppoa. PIT (niinkuin ilmeisesti muutkin työkalut) ovat kuitenkin raskaita ja hitaita käyttää.

Lähteinä käytetty pääasiassa tämän kurssin luentokalvoja.