Tampereen ammattikorkeakoulu

Yksikkötestaus

Petri Irri

Backend-kehitys testausraportti

Lokakuu 2021

Tietojenkäsittely
Ohjelmistotuotanto

SISÄLLYS

1 JOHDANTO

Tässä Backend-kehityksen testausraportissa kerrotaan testaukseen ja tarkemmin yksikkötestaukseen liittyvistä asioista. Raportti on toteutettu osana Backend-kehitys opintojaksoa ohjelmistokehitys kurssia varten. Raportissa kerrotaan lyhyesti testauksesta yleisesti ja perehdytään nopeasti yksikkötestaukseen. Raportti ei käsittele aihealueita laajasti vaan toimii pintaraapaisuna aihealueisiin ja sen tarkoituksena onkin tarjota lukijalle nopea tapa tutustua yksikkötestaukseen, sekä ohjelmistotestaukseen yleisesti. Raportin päälähde on Marc Cliftonin kirja: Unit testing succintly.

Raportissa ensimmäiseksi käsitellään ohjelmistotestaamista yleisesti, jossa pyritään antamaan lukijalle tarvittavat yleistiedot, jotta tämä voi ymmärtää raportin myöhempiä aihealueita. Raportti kertoo ensiksi testauksesta yleisellä tasolla sivuten erilaisia testimenetelmiä, sekä niiden kategorisointia. Raportti antaa muutamia esimerkkejä muista testausmenetelmistä, sekä laadukkaan testauskokonaisuuden sisällöstä.

Raportin pääosiossa kerrotaan yksikkötestaamisesta ja niiden suunnittelusta, sekä vaatimuksista. Raportti Kertoo ensiksi yleisellä tasolla mitä yksikkötestaus on ja miksi sitä kannattaa tehdä. Tästä jatkaen raportti selittää tarkemmin mitä yksiköt ovat ja millaisia ovat hyvät yksiköt. Raportti myöskin kertoo samalla minkälaisia ovat huonot yksiköt ja miksi hyvien yksiköiden valitseminen on tärkeää.

Raportin kerrottua mitä yksiköt ovat siirtyy se kertomaan hyödyllisestä yksikkötestistä ja sen suunnittelemisesta. Tässä osiossa raportissa tarkastellaan yleisellä tasolla testauksen suunnittelemista ohjelmistoprojektin alussa ja kuinka yksikkötestausta voi suunnitella. Raportti kiinnittää erityistä huomiota testauksen suunnittelemiseen koko projektin ajaksi huomioiden projektin tulevan ylläpitovaiheen ja korostaen aikaisin aloitetun testauksen merkitystä.

Raportin viimeisessä osassa kerrotaan erilaisista testausprosesseista ja niiden hyödyistä. Testausprosesseista kerrotaan yleistasolla tuoden niiden hyödyt ja haitat esille.

2 Mitä on ohjelmistotestaus?

2.1 Testauksesta yleisesti

Ohjelmistotestauksella pyritään varmistamaan ohjelmiston täyttävän sille asetetut vaatimukset. Ohjelmistotestauksessa on paljo erilaisia testausmenetelmiä, jotka keskittyvät ohjelmiston tietyn osa-alueen testaamiseen. Testimenetelmät on yleisesti jaettu kahteen kategoriaan: **Toiminnallinen testaaminen**, jossa testa-

taan ohjelmiston toiminnallisuuden täyttävän sille asetetut vaatimukset, sekä toimivan oletetusti, **Ei-toiminnallinen testaus**, jossa testataan ohjelmiston käyttökuntoisuutta.

Laadukkaassa ohjelmistotestauksessa käytetään Ei-toiminnallisia testausmenetelmiä, sekä toiminnallisia testausmenetelmiä, joiden avulla luodaan toimiva testauskokonaisuus. Tämän raportin puitteissa käsitellään tarkemmin yksikkötestaamista, mutta muita toiminnallisen testauksen testimenetelmiä ovat: integraatiotestaus, järjestelmä testaus ja hyväksyttämistestaus. Ei-toiminnallisen testaamisen testimenetelmiä ovat: suorituskykytestaus, turvallisuustestaus, käytettävyystestaus ja yhteensopivuustestaus.

2.2 Testauksen hyödyt

Ohjelmistotestauksella säästetään rahaa, työtunteja ja lisätään asiakastyytyväisyyttä. Ohjelmistotestaus saavuttaa nämä hyödyt näyttämällä ohjelmiston kehityksen aikana erilaisia ohjelmointi- ja suunnitteluvirheitä. Näitä virheitä on usein työlästä ja kallista korjata myöhemmin ohjelmiston valmistuttua. Varsinkin toiminnallinen testaaminen on tärkeää, jotta voidaan varmistua kirjoitetun ohjelmakoodin toimivuudesta ja laadusta. Toiminnallinen testaaminen myöskin itsessään yleensä dokumentoi kirjoitettua ohjelmakoodia ja kertoo mitä koodin oletetaan tekevän.

3 Yksikkötestaus

3.1 Mitä on yksikkötestaaminen?

Yksikkötestaamisessa testataan ohjelmakoodin yksiköitä, jotka ovat pieniä pätkiä ohjelmakoodia. Yleensä mitä pienempi yksikkö sen parempi. Yksikkötestaamisen perusajatuksena on todistaa ohjelmakoodin toimivan oikein. Hyvin kirjoitettu yksikkötesti todistaa koodin toimivan oikein ja lisää sen laatua, sekä käytettävyyttä. Yksikkötestit eivät kuitenkaan itsessään takaa laadukasta koodia, eikä kehitys-

tai testaamisajan vähentymistä. Yksikkötestit vain todistavat ohjelmakoodin toimivan oikein ja täten pakottavat kehittäjän kirjoittamaan laadukasta koodia, joka nopeuttaa kehitystä ja testaamista.

3.1.1 Miksi tehdä yksikkötestausta?

Kuten aiemmassa kappaleessa totesin yksikkötestauksen perusajatuksena, on näyttää toteen ohjelmakoodin toimivuus. Hyvin kirjoitetut yksikkötestit antavat mitattavissa olevan varmuuden ohjelmakoodin toimivuudesta. Yksikkötestien kattavuus on yksi mitattavissa oleva asia. Yksikkötestien kattavuudella tarkoitetaan, kuinka suurelle osalle ohjelmakoodista löytyy yksikkötesti. Tämän avulla ei voida suoraan sanoa yksikkötestauksen vähentäneen ohjelmointivirheiden määrää tietyn prosenttiyksikön verran. Yksikkötestauksen kattavuutta voidaan kuitenkin hyödyntää miettiessä ohjelmistossa vielä olevien ohjelmointivirheiden määrää, katsomalla yksikkötestauksen kattamatonta osaa ohjelmistosta. Kirjoitettujen yksikkötestien määrää voi myöskin vertailla raportoitujen ohjelmointivirheiden määrään. Tämä voi paljastaa yksikkötestien laadullisia ongelmia, mikäli suuri osa ohjelmointivirheistä tulee ohjelmakoodin kohdista, joihin on jo kirjoitettu yksikkötestit. Vastaavasti se voi myöskin todistaa yksikkötestien olleen hyvin kirjoitettuja, kun ohjelmakoodin pätkistä, joilla on yksikkötesti, ei tule ohjelmointivirhe raportteja.

Toinen hyöty yksikkötestien kirjoittamisessa on niiden toistettavuus. Valmiiksi kirjoitettua yksikkötestiä voi toistaa niin monta kertaa kuin on tarve, eikä hyvin kirjoitettua yksikkö testiä tarvitse kirjoittaa uudestaan. Tämän avulla ohjelmiston kehityksen edetessä on mahdollista suorittaa vanhoja yksikkötestejä ja tarkistaa vanhan ohjelmakoodin toimivuus. Näin yksikkötestit lisäävät vanhan ohjelmakoodin luotettavuutta, vaikka uutta ohjelmakoodia tulisi lisää. Tämä myöskin vähentää virheenetsintään käytettävää aikaa uuden ohjelmakoodin rikkoessa vanhaa ohjelmakoodia.

3.1.2 Mitä ovat yksiköt?

Yksikkötestauksen keskiössä ovat yksiköt. Yksiköt ovat ohjelmakoodin loogisia osia, joita vasten voi kirjoittaa testin. Yksikkö on ihanteellisesti metodi, joka ei kutsu muita metodeja ja saa parametrejaan niiltä. Kuitenkaan aina ei ole mahdollista olla käyttämättä muita metodeja, mutta on hyödyllistä miettiä voisiko metodista kehittää puhtaamman yksikön. Tämä vähentää yhden yksikön monimutkaisuutta ja parantaa yksikkötestin laatua. Metodin tarvitessa muita metodeja kannattaa ne laittaa ylempään metodiin, joka kutsuu niitä ja antaa metodin kutsun parametrina ne metodille.

Yksikön tulisi tehdä vain yksi asia. Tämä on yleensä seurausta puhtaan yksikön luomisesta, joka ei kutsu muita metodeja. Tämä helpottaa yksikkötestien tekemistä ja varmistaa yksikkötestien todistavan niiden tekevän yhden asian oikein. Lopputuloksena on myöskin usein helpommin luettavaa ohjelmakoodia ja luodut yksikkötestit kertovat paremmin ohjelmakoodin oletetun käyttäytymisen. Metodeja, jotka käyttävät pelkästään muita yksikkötestillisiä metodeja ei ole järkeä testata, koska sen käyttämät metodit jo toimivat todistetusti.

Yksiköillä ei tulisi olla useita mahdollisia koodin suoritus polkuja. Yksikkö, jossa on useita suorituspolkuja if tai switch lauseiden johdosta on huomattavasti vaikeampi testata kokonaisuudessaan. Mieluiten yksikössä ei ole useita suorituspolkuja aiheuttavia lauseita, vaan niistä tehdään omat yksikkönsä. Näiltä lauseilta ei kuitenkaan voi kokonaisuudessaan välttyä, jolloin on suositeltavaa laittaa näiden lauseiden sisältö omaan metodiinsa ja tehdä niille yksikkötesti. Näin luodun ylemmän metodin yksikkötesti varmistaa sen toimivan odotetulla tavalla ja näyttävän toteen ohjelmakoodin toimivuuden.

3.2 Hyödyllinen yksikkötesti

Yksikkötesti itsessään ei takaa sen hyödyllisyyttä. Huonosti kirjoitettu yksikkötesti tai huono testissä käytettävä yksikkö voivat olla turhia, jos ne eivät todista ohjelmakoodin oikeellisuutta. Tämän takia on tärkeää käyttää järkeviä yksikköjä, joille voi kirjoittaa järkeviä yksikkötestejä. Huonoihin yksikkötesteihin menee enemmän aikaa, joka vähentää muihin yksikkötesteihin käytettävää aikaa.

3.2.1 Yksikkötestin vaatimukset

Yksikkötestin päävaatimus on sen todistavan testattavan ohjelmakoodin oikeellisuuden eli ohjelmakoodin toimivan odotetulla tavalla. Tätä varten yksikkötestin on todistettava ohjelmakoodin käsittelevän: metodille annetut parametrit oikein, metodin suorittavan laskennallisen tehtävän oikein ja sen käsittelevän metodin sisäiset, sekä ulkoiset virheet oikein. Nämä ovat tärkeimpiä asioita, joita yksikkötestin pitäisi testata.

Ohjelmakoodin parametrien oikealla käsittelyllä tarkoitetaan ohjelmakoodin sietävän tilanteita, joissa metodille annetaan sopimattomia arvoja. Yksinkertainen esimerkki asiasta on nollalla jako, jolloin ohjelmakoodin pitäisi tulostaa virheilmoitus. Yksikkötesti, joka testaa vain tämän ei itsessään ole erittäin hyödyllinen sillä se on laadultaan heikko yksikkötesti.

Yksikkötestin tulisi myös varmistaa metodin laskennallisen tehtävän oikeellisuus. Tämän toteuttaminen riippuu testattavasta metodista, mutta yleisimmät metodit ottavat kaksi parametria ja palauttavat yhden vastauksen. Yleensä tämä on jokin laskutoimitus. Tässä tapauksessa yksikkötestin tulisi varmistaa metodin palauttavan datan oikein. Muissa tapauksissa yksikkötestien tulisi myös todistaa matemaattisten laskentojen toimivan oikein. Tähän kuuluu datan muuntaminen muodosta toiseen, sekä erilaisten listojen muuntaminen.

4 Yksikkötestauksen suunnitteleminen

4.1 Suunnittelun aloittaminen

Yksikkötestausta suunnitellessa on hyvä ajatella ohjelmisto projektin tulevaisuutta. Tämä on helpointa tehdä heti projektin alussa, kun tehtävästä ohjelmistosta on käsillä vain vaatimuslista. Tässä kohtaa on helpointa miettiä valmiiksi asioita, sillä myöhemmin on huomattavasti vaikeampaa sopeuttaa vanhaa koodia uusiin vaatimuksiin. Ensimmäisenä on hyvä suunnitella, mille ohjelmakoodin osille yksikkötestejä halutaan tehdä. Mitättömälle koodille ei usein ole erityisen

hyödyllistä tehdä yksikkötestejä, vaan suunnitteluvaiheessa tulee asettaa etusijalle ohjelmiston laskennalliset vaatimukset ja erityisesti ohjelmiston erityisvaatimusten tuottama ohjelmakoodi.

Ohjelmiston ylläpidon suhteen on myös hyödyllistä miettiä yksikkötestaamista. Valmiina olevat yksikkötestit itsessään dokumentoivat ohjelmistoa, joka helpottaa sen ylläpitoa. Myöskin uuden vikakorjauksissa tulevan ohjelmakoodin mahdollisesti aiheuttamat ohjelmistovirheet vanhassa koodissa tulevat helpommin esille. Kannattaa myös miettiä kirjoittaako uusille ominaisuuksille tai vikakorjauksille omia yksikkötestejään. Tämä kaikki on vielä hankalampaa kehittäjän tullessa valmiiseen projektiin, joka on jo ylläpitovaiheessa ja jossa ei ole valmiiksi yksikkötestejä. Tässä kohtaa kehittäjän pitää tarkastella, onko valmiiseen koodikantaan järkevää luoda yksikkötestejä vai pitäisikö sitä muuttaa jotenkin ja mille metodeille yksikkötestejä tehdä ensin. Yksikkötestien tekeminen on usein hintansa arvoista, sillä se helpottaa tulevaa ylläpitoa, dokumentoi olemassa olevaa ohjelmakoodia ja auttaa ohjelmakoodin syvemmässä ymmärtämisessä.

4.2 Testausprosesseja

4.2.1 Testivetoinen kehitys

Testivetoinen kehitys on ohjelmointitekniikka, jossa ohjelmoitavaa toimintaa lähdetään rakentamaan testitapauksen kautta. Testivetoisessa kehityksessä luodaan ensiksi testitapaus ja testitapaukselle tehdään ohjelmakoodi, joka läpäisee testin. Vaikka tämä vaikuttaa aluksi erikoiselta tavalta kehittää ohjelmaa, tarjoaa se kuitenkin monia hyötyjä. Testivetoinen kehitys pienentää ohjelmointivirheiden määrää ja lisää testien kattavuutta. Tämän avulla ohjelmiston testaaminen helpottuu, vaikka ohjelmistoon lisättäisiin uusia ominaisuuksia, sillä vanhoille ominaisuuksille on jo olemassa testitapaukset. Testivetoinen kehitys myöskin usein parantaa kirjoitetun ohjelmakoodin tasoa ja tekee siitä kompaktimpaa.

4.2.2 Koodaa ensin, testaa sitten

Koodaa ensin testaa sitten menetelmä on erittäin luonnollinen tapa tehdä ja testata ohjelmakoodia. Nimensä mukaisesti ensiksi tehdään ohjelmakoodia ja sitten kirjoitetaan sille testitapaukset. Tässä on kuitenkin huonot puolensa, sillä testaaminen vaatii kurinalaisuutta, että testit tulevat oikeasti tehdyiksi. Tämä tulee usein kokemuksen ja tottumuksen kautta. Ohjelmakoodista tulee myös helposti sekavaa ja yhteen kietoutunutta, joka vaikeuttaa hyvien yksikkötestien tekemistä. Tämä vaatii enemmän kurinalaisuutta ja tarkempaa ohjelmakoodin tarkastelua. Mitä kokeneempi tiimi ohjelmakoodia on tekemässä, sitä sujuvammin tämä prosessi toimii. Kuitenkin kiireisinä ja korkea stressisinä aikoina tiimi vaatii vieläkin tarkempaa tarkastelua, koska silloin testaaminen unohtuu helpommin. Tämä johtuu yleensä lähestyvistä aikarajoista ja testivetoisessa kehityksessäkin näin voi käydä, mutta koodataan ensin, testataan sitten menetelmällä se jää helpommin tavaksi.

4.2.3 Ei yksikkötestaamista

On myöskin mahdollista jättää yksikkötestaaminen kokonaan pois. Tällöin testaaminen keskittyy usein muihin osa-alueisiin, kuten käyttöliittymätestaukseen tai integraatiotestaukseen. Vaikka yksikkötestaus on erittäin hyödyllinen menetelmä, ei se kuitenkaan korvaa muita menetelmiä tai ole joka projektissa paras.

5 POHDINTA

Ohjelmistotestaus on tärkeä osa minkä tahansa ohjelmiston kehittämistä ja ilman ohjelmistotestausta kehitetty ohjelmisto on erittäin altis ohjelmointivirheille. Aihealueena ohjelmistotestaus on erittäin laaja ja erilaisia testausmenetelmiä on paljon. Testausmenetelmät kategorisoidaan yleisesti toiminnallisiin- ja ei-toiminnallisiintestausmenetelmiin. Laadukkaassa ohjelmiston testaussuunnitelmassa käytetään kummankin kategorian menetelmiä ja niistä luodaan laadukas kokonaisuus, joka testaa ohjelmiston kaikki osa-alueet. Hyvin suunnitellusta testauksesta on kuitenkin erittäin paljon hyötyä, sillä se usein säästää rahaa ja aikaa.

Hyvin suunniteltu testisuunnitelma säästää rahaa varsinkin ohjelmistoprojektin ylläpitovaiheessa, jolloin ohjelmointivirheiden korjaaminen on usein hankalaa ja kallista.

Yksikkötestaus on yksi testitapa, jolla ohjelmistoa voi testata. Siinä ohjelmakoodi jaetaan pieniin osiin, joita kutsutaan yksiköiksi. Parhaimmillaan yksiköt ovat metodeja, jotka eivät kutsu muita metodeja ja tekevät vain yhden asian. Tämänkaltaisille yksiköille on helppo kirjoittaa yksikkötestejä ja ne ovat myöskin helpompia lukea, sekä ymmärtää. Yksikkötestit itsessään pakottavat ohjelmoijan kirjoittamaan laadukkaampaa ohjelmakoodia. Yksikkötestit myöskin toimivat dokumentaationa ohjelmakoodille, sillä ne kertovat metodin odotetusta toiminnasta. Uudet ohjelmoijat voivat käyttää tehtyjä yksikkötestejä tutustuakseen paremmin ohjelmakoodin toimintaan ja täten saada paremman ymmärryksen meneillään olevasta ohjelmistoprojektista.

Yksikkötestin tarkoitus on todentaa ohjelmakoodin oikeellisuus. Toisin sanoen olla todiste, että ohjelmakoodi toimii oletetulla tavalla ja käsittelee sille syötettyjä arvoja oikein, sekä palauttaa järkeviä arvoja. Yksikkötestiä itsessään voidaan pitää todisteena ohjelmakoodin toimivuudesta ja olemassaolosta. Yksikkötestit eivät kuitenkaan ole ohjelmistotestauksen pyhä Graalin malja. Yksikkötesti kertoo vain pienen osan ohjelmakoodista toimivan ja se ei kerro kuinka ohjelmakoodi toimii itsessään. Laadukasta testausta varten vaaditaan silti muita testimetodeja ohjelmiston muille osa-alueille. Kaiken kaikkiaan yksikkötestaus on loistava työkalu ohjelmistotestaukseen, mutta se toimii parhaiten muiden toiminnallisten testimetodien kanssa.

LÄHTEET

Marc Clifton: Unit Testing Succintly. Syncfusion Inc. 2013.

You Still Don’t Know How to Do Unit Testing (and Your Secret is Safe with Me). Stackify, https://stackify.com/unit-testing-basics-best-practices/. Luettu 28.10.2021.

Glossary TTD. Agile Alliance, https://www.agilealliance.org/glossary/tdd/. Luettu 29.10.2021.

What is Software Testing? Definition, Basics & Types in Software Engineering. Guru99, https://www.guru99.com/software-testing-introduction-importance.html. Luettu 29.10.2021.

Software Testing Methodologies. SmartBear, https://smartbear.com/learn/automated-testing/software-testing-methodologies/. Luettu 29.10.2021.

What Is Unit Testing? SmartBear, https://smartbear.com/learn/automated-testing/what-is-unit-testing/. Luettu 29.10.2021.

What is code testing and why is it important? Zeomag, https://www.zeolearn.com/magazine/what-is-code-testing-and-why-is-it-important. Luettu 29.10.2021.