Tmc-cli-projektin testidokumentti

1 Testauksen rooli ohjelman kehityksessä

Ohjelma kehitettiin ketterällä menetelmällä, käytännössä Scrumilla. Testaus oli siten osa jokaista sprinttiä. Testit automatisoitiin mahdollisimman pitkälti.

Testauksen osuus kunkin sprintin työmäärästä kuitenkin vaihteli. Yksi syy tähän oli asiakkaan toive saada ohjelmasta nopeasti aikaan versio (minimal viable product), jota voi tarjota käyttöön ohjelmointikurssien opiskelijoille. Tällainen versio valmistui neljännen sprintin aikana. Testauksessa painotettiin projektin alussa erityisesti keskeisen ohjelmalogiikan testausta. Projektin edetessä testikattavuus kasvoi.

Testauksen rivikattavuutta seurattiin Coberturalla. Tulokset saa näkyviin esimerkiksi repositorion README-tiedoston linkistä.

Rivikattavuus ei kerro suoraan testien hyödyllisyydestä. Testauksen ensisijaisena tavoitteena ei ollut 100 prosentin rivikattavuus vaan hyödyllisten testien laatiminen.

2 Testaustavat

Ohjelman rakenne vaikutti luonnollisesti testien toteutukseen ja testimenetelmiin. Ohjelman keskeinen toiminnallisuus koostuu yksittäisistä komennoista, joita käyttäjä voi suorittaa. Komennot toteutettiin komentotehdasmallilla siten, että lähes jokainen komento kirjoitettiin omaksi luokaksi. Komentojen lisäksi ohjelmassa on komentojen käyttöön apuluokkia, jotka pääosin liittyvät syötteiden ja tulostusten sekä tiedostojen käsittelyyn.

2.1 Yksikkötestit

Etenkin apuluokille kirjoitettiin testejä, jotka ovat luonteeltaan selvästi yksikkötestejä. Nämä testit testaavat etupäässä yksittäisten luokkien metodien toimintaa.

2.2 Integraatio- ja hyväksymistestit

Komentoluokille kirjoitetut testit ovat suurelta osin luonteeltaan enemmän integraatio- tai hyväksymistestejä kuin yksikkötestejä. Testeissä luodaan ohjelmasta sovellusolio, jolle annetaan parametreina komento vastaavasti kuin ohjelman käyttäjä antaisi. Testit testaavat käytännössä usean luokan yhteistoimintaa. Toisin sanoen testeillä pyritään varmistamaan, että koko ohjelman sovelluslogiikka toimii suunnitellusti.

2.3 Regressiotestaus

Automatisoidut testit mahdollistivat vaivattoman regressiotestauksen projektin aikana. Testit ajettiin automaattisesti ohjelmaa käännettäessä.

2.4 Ohjelman demoaminen

Ohjelman ominaisuuksia esiteltiin asiakkaalle jokaisen sprintin päättyessä. Tällöin asiakkaalla oli mahdollisuus kommentoida ohjelman toimintaa.

2.5. Tutkiva testaus

Automatisoitujen testien ohella käytettiin sprinttien aikana tutkivaa testausta. Ohjelmaa kokeiltiin ensin käyttää normaalin käyttäjän tavoin. Ohjelman antaman palautteen perusteella pyrittiin löytämään virheitä ohjelman toiminnasta. Tämän lisäksi ohjelmaa kokeiltiin käyttää epätavallisilla tavoilla, esimerkiksi selvästi virheellisillä syötteillä. Näin saatiin osaltaan varmistettua, että ohjelma toimii odotetulla ja asiakkaan kannalta luontevalla ja informatiivisella tavalla.

Manuaalisissa testeissä oli käytössä testipalvelin, jossa oli testikursseja. Ohjelman toimintaa kokeiltiin myös aidolla ohjelmointikurssilla.

3 Testityökalut

Kaikki automatisoidut testit toteutettiin teknisesti Javalla JUnitilla. Tämä koskee myös integraatiotestejä. JUnit sopi tässä projektissa hyvin myös niiden toteutukseen, eikä tältä osin nähty tarpeelliseksi käyttää esimerkiksi EasyB:tä.

Testauksessa hankalimmaksi osoittautui se, että ohjelma kommunikoi verkon yli TestMyCode-palvelimen TmcCore-luokan kanssa. Testeissä haluttiin eroon verkkoriippuvuudesta.

Verkkoriippuvuuden eliminoimiseksi testeissä käytettiin Mockitoa. Näin voitiin luoda TmcCore-luokasta mock-olioita testejä varten. Mockitoa hyödynnettiin ohjelman omienkin luokkien testauksessa.

Testauksessa hyödynnettiin jonkin verran myös PowerMockia. Sillä oli mahdollista luoda mock-olioita muun muassa staattisia metodeja sisältäville luokille. Ohjelmassa on staattisia metodeja muutamassa apuluokassa.