Luomala Joni-Petteri

Testinhallintaohjelman käytön analysointi

Insinööri (AMK)

Tietotekniikka

Kevät 2017

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Luomala Joni-Petteri

Työn nimi: Testauksenhallintaohjelman käytön analysointi ja korvaus

Tutkintonimike: Insinööri (AMK), tietotekniikka

Asiasanat: Testinhallinta, ohjelmistotestaus, testaus automaatio, Robot Framework, SpiraTest, Jenkins

Tämän opinnäytetyön kohde on tuottaa testinhallintaohjelmiston analyysi ja parantaa testinhallintaohjelmiston käyttöä ohjelmistonsuunnitteluprojekteissa. Testihallintaohjelmistona käsitellään SpiraTest-testinhallintaohjelmistoa. Opinnäytetyö käsittelee myös yleisesti ohjelmistotestausta ja sen järjestämistä ohjelmistoprojektissa. Työn tilaaja on Oululainen ohjelmistoalan yritys Bittium Oyj, jonka ohjelmistotestausympäristö otetaan opinnäytetyön lähtökohdaksi.

Työssä käydään läpi kuinka testinhallintaohjelmisto SpiraTest toimii ohjelmistoprojektissa. Löydettyjä piirteitä arvioitiin suhteessa ohjelmiston testauksessa tarvittaviin työkaluihin ja metodeihin. Työssä havaittuja ohjelmistotestausta hyödyttävät piirteet ovat yleisesti tarpeellisia testinhallinta ominaisuuksia ja ovat sellaisenaan siirrettävissä muihin ohjelmistoprojekteihin.

Työssä kehitettiin vaihtoehtoisia toteuttamismalleja testauksenhallinnan toteutukselle ilman testinhallintaohjelmistoa. Toteuttamismallit huomattiin toteuttamiskelpoisiksi. Kuitenkin toteutuessaan muutosehdotukset tulisivat vaatimaan aikaisempaa suurempaa kurinalaisuutta ohjelmistotestauksen suunnittelijoilta ja toteuttajilta.

Abstract

Author: Luomala Joni-Petteri

Title of the Publication: Test Management Software Usage Analysis

Degree Title: Bachelor of Engineering, Information Technology Engineering

Keywords: Test Management, Software Testing, Test Automation, Jenkins, Robot Framework, SpiraTest

The objective of this Bachelor's thesis was to produce an analysis of test management software and improve the usage of test management software in software programming projects. SpiraTest shall serve as the test management software for this thesis. This thesis also covers basics of software testing and managing of software testing. This thesis was made to Bittium Oyj and the software testing methods of Bittium serve as the basis of this thesis.

In this thesis, the working of test management software SpiraTest in software projects shall be analyzed. The findings of this analyze will be reviewed in accordance of needs and requirements of software projects. The findings of this thesis are applicable to other similar software testing solutions and are possible to be used in variety of software projects

The final outcome of this thesis is alternative testing management models for test management without test management software. The alternatives for test management were found plausible to be used in software management. However, the changes to software test management proposed by this thesis require more rigorous work ethics from test planners and test makers.

ALKUSANAT

Bittium Oyj on oululainen ohjelmistoalan yritys. Sain harjoittelupaikan Bittiumin Kajaanin toimistolta syksynä 2016. Opinnäytetyön aihe tuli harjoittelun aikana käydyistä keskusteluista ja ajatuksesta, että voitaisiinko testinhallintaohjelmistoa korvata jatkossa muilla tavoilla.

Kiitän kaikkia opinnäytetyötäni avustaneita tahoja. En olisi onnistunut tässä ilman teitä. Kiitos.

Kajaani, 25, huhtikuuta, 2017

Joni-Petteri Luomala

SISÄLLYS

KÄYTETYT LYHENTEET JA TERMIT

Automoitu testaus	Ohjelmistotestaus, joka on suoritettu automaattisesti, esimerkiksi tietokoneen ajamana. [Engl. Test Automation]
Avainsana	Engl. Keyword, Avainsana on avainsanapohjaisessa testauksessa käytetty käsky, jolla ilmaistaan yhtä toimintoa.
Avainsanapohjainen testaus	Engl. Keyword-Driven Testing, Avainsanapohjainen testaus on Testiautomaatiokäytänne, jossa testidata ja avainsanat luetaan ulkoisesta datalähteestä. [M. Fewster and D. Graham. Software Test Automation. Addison-Wesley, 1999.]
Integraatio	Yhdistäminen. Integraatiolla ohjelmistosuunnittelussa tarkoitetaan uusien ohjelmistolisäysten yhdistämistä jo valmiiseen ohjelmistoon.
Jatkuva integraatio	Engl. Continuous Integration eli jatkuva integraatio on periaate, jossa integraatiota suoritetaan jatkuvasti automatisoitua testausta vasten, jotta integraatio ongelmat havaittaisiin aikaisessa vaiheessa ja olisi mahdollista säilyttää ohjelmiston toimintakyky ja toimitettavuus aina.
Manuaalitestaus	Manuaalisesti ihmisen työllä suoritettu ohjelmistotestaus.
Oraakkeli	Engl. Oracle eli oraakkeli on ohjelmistotestauksen automatisoinnissa käytetty apuväline, jolla voivat testi-insinöörit tai automatisoidut testit määrittää, onko testi hyväksytysti suoritettu vai ei. [I. Burnstein. Practical Software Testing. Springer 2003.]
End to End -testaus	Eli päästä päähän -testauksella tarkoitetaan metodologiaa, jossa sovelluksen testaus suoritetaan alusta loppuun. Sen tavoitteena on löytää systeeminkeskinäiset riippuvaisuudet ja varmistaa oikean informaation kulkeminen eri komponenttien ja systeemien välillä.
Testiaskel	Engl. Test Step, Testitapauksen alakohtia, joissa määritellään tietyn testivaiheen syötteet.

Testitapaus	Engl. Test case, Testitapaukset ovat yksittäisiä testejä, joille on määritetty suoritustapa, annettavat syötteet ja halutut vastaukset testattavasta ohjelmistosta. Jos nämä suoritteet epäonnistuvat tai halutut vastaukset eivät ole oikeita, on testitapaus epäonnistunut, muutoin se on onnistunut.
Tietokäyttöinen testaus	Engl. Data-Driven Testing eli tietokäyttöinen testaus on testausskriptaustekniikka, jossa testisyötteitä ja ennakoituja vastauksia säilytetään tiedostossa, jonka avulla testausskriptillä ajetaan kaikki määrätyt testitapaukset. [M. Fewster and D. Graham.Software Test Automation. Addison-Wesley, 1999.]
Vaatimus	Engl. Requirement, Vaatimus on ohjelmistolta vaadittu ominaisuus tai toiminto. Tämä vaatimus pohjaa joko sopimukseen, standardiin, spesifikaatioon tai muuhun dokumentaatioon.

1 Johdanto

Nykyaikaiset ohjelmistot ovat monimutkaisempia kuin koskaan ja tulevat jatkamaan kehitystä vielä hankalammiksi kokonaisuuksiksi kehittää ja ylläpitää. Tämä kehitys asettaa haasteita ohjelmistojen laadulle, joka varmistaa ohjelmistojen toiminnan kaikissa tilanteissa ja estää vioista johtuvat kulut. Ohjelmistojen toiminnassa on kyseessä hyvin usein jopa miljoonien eurojen kokoiset summat [1]. Ohjelmiston kaikki toiminnot ja ominaisuuksien toimiminen tulee tarkistaa monissa eri tilanteissa, jolloin voidaan olla varmoja ohjelman toimimisesta. Moderneilta ohjelmistoilta vaadittu laatu on siis varmistettava kattavalla ja monipuolisella testauksella.

Moderni ohjelmistotestaus on monitahoista, ja testauksen painopisteet voivat vaihdella eri alueiden välillä ohjelmointiprojektien eri vaiheissa. Testauksen tulee olla kattavaa niin pienimpiä ohjelmisto komponentteja kuin koko ohjelman suoritusta testattaessa. Testausta joudutaan toistamaan ohjelmointiprojektin edetessä määräajoin, jotta voidaan olla varmoja kaikkien ominaisuuksien toiminnasta. Täten pienessäkin ohjelmistossa testattavien asioiden määrä käy nopeasti hyvin suureksi, tarvitaan testauksen nopeuttamiseksi automatisaatiota. Automatisaatiolla voidaan testata hyvin nopeasti valtavia määriä testejä, mikä ihmistyövoimalla suoritettuna olisi kestänyt jopa kymmeniä kertoja pidempään [2]. Automaatiolla saavutetaan myös hyötyjä toistettavuudessa, laadun hallinnassa ja pienentyneessä työvoiman tarpeessa [3].

Ohjelmistotestaus on laaja ja vaikeasti hallittava kokonaisuus, jolle tarvitaan oma testistrategia. Testistrategiassa kuvataan kuinka testaus ja laadun valvonta projektissa suoritetaan. Testistrategiassa kuvattujen laadunvarmistusmenetelmien varmistamiseksi tarvitaan erilaisia testaustyökaluja. Yksi näistä työkaluista on testinhallintaohjelmistot, jotka auttavat hallinnoimaan testausta ohjelmistoprojekteissa.

Tämä opinnäytetyö tulee käsittelemään testinhallintaa ja sen järjestämistä ohjelmistoprojekteissa. Kuitenkin ennen kuin opinnäytetyön pääaihetta eli testinhallintaohjelmistoa aletaan käsitellä, on syytä perehtyä ohjelmistotestaukseen. Seuraava luvussa käsitellään taustatietoa ohjelmistotestauksesta.

1.1 Opinnäytetyön tavoitteet

Opinnäytetyön tavoitteena on analysoida testinhallintaohjelmiston hyödyt ja kehityskohteet. Kerätä mitä testinhallintaohjelmistolla voi tehdä ja onko se mahdollista tehdä muin keinoin. Onnistuakseen tulee opinnäytetyössä suorittaa seuraavat kolme vaihetta:

1. Analysoida nykyisen testauksenhallintaohjelman käyttöä
2. Tutkia tämän analyysin perusteella mahdollisia erilaisia suoritustapoja testinhallinnalle
3. Tutkia voiko näitä suoritustapoja käyttää ohjelmistoprojektin testinhallinnassa

1.2 Opinnäytetyön rakenne

Tämä luku sisälsi johdannon, opinnäytetyön tavoitteet ja rakenteen.

Luvussa 2 käydään läpi ohjelmistotestausta ja siihen liittyvän testauksenhallintaohjelmiston hyötyjä

Luku 3 tulee käsittelemään opinnäytetyössä mainittuja ja käytettyjä työkaluja ja esittelee ne ja niiden toiminnan.

Luku 4 antaa yleiskuvan opinnäytetyön lähtökohdista.

Luku 5 keskittyy SpiraTest -testauksenhallintaohjelmiston analysointiin ja erittelyyn.

Luvussa 6 annetaan vaihtoehtoisia toteuttamismuotoja luvun 5 erittelemille testauksenhallintaohjelmiston ominaisuuksille.

Opinnäytetyön tuloksia käsitellään luvussa 7, jossa pohditaan luvun 6 käsittelemiä vaihtoehtoja ja niiden toteuttamiskelpoisuutta arvioidaan.

Lopuksi Luku 8 tulee sisältämään yhteenvedon opinnäytetyöstä.

2 Ohjelmistotestaus

Ohjelmistotestaus on prosessi, jossa ohjelmisto ajetaan koesyötteellä ja analysoidaan lopputulos [4]. Tiivistettynä ohjelmistotestaus kuulostaa yksinkertaiselta. Kuitenkin suurin ongelma testaamisessa on testit itsessään. Testauksen on tarkoitus tarjota asiakkaalle varmuus siitä, että ohjelmisto täyttää kaikki sen vaatimukset [5]. Miten siis muotoilla testejä, jotka varmistavat ohjelmiston täyttävän vaatimukset ja huomaavat vaatimusten vastaisen suorituksen?

Vaatimus (Requirement) on lähtökohta testaamiselle. Vaatimus määritellään joko asiakkaan toimesta tai johonkin standardiin perustuen, ja se voi käsittää esimerkiksi käyttötapauksia tai toimintoja, joita ohjelmiston tulee kyetä suorittamaan. Vaatimukset ovat tärkeitä testaamiselle ja vaativat selkeyttä, jos vaatimukset eivät ole tarpeeksi hyvin määriteltyjä on ohjelmistoa ja sen testausta vaikea ryhtyä tekemään.

Testitapaus (Test case) on vaatimuksen pohjalta suunniteltu ohjelmistopolun suoritus, jonka toiminta hyväksytetään vaatimusta vasten. Testitapaukset koostuvat syötteistä, määritellyistä suorituksen lähtökohdista ja ennalta odotetuista lopputuloksista [6]. Testitapauksen laadinta ja suunnittelu ovat haastavia prosesseja ja niiden vaiheiden selostaminen ei ole tämän opinnäytetyön laajuudessa mukana.

Testauksen suorituksessa testitapaus ajetaan läpi. Testitapauksessa määritellyt syötteet syötetään testitapauksen määrittelemässä järjestyksessä ohjelmistoon, jonka lähtötilaksi on laitettu testitapauksen kuvaama lähtökohta testaamiselle. Suorituksen edetessä syötteiden laittamista ja ohjelmiston toimintaa seurataan ja verrataan odotettuja lopputuloksia vasten. Jos toiminta on testitapauksessa määritellyn kaltaista, tuottaa oikeita tuloksia ja tapahtuu määritellyn ajan kuluessa, on testi suoritettu onnistuneesti ja onnistuminen raportoidaan.

Huomattaessa testitapausta suoritettaessa ohjelmiston toiminnassa ja odotetuissa lopputuloksissa eroavaisuuksia, testaus keskeytetään. Tällaisessa tilanteessa on havaittu mahdollinen virhe ohjelman toiminnassa ja se tulee raportoida virheellisenä suorituksena.

Testauksen raportointi tapahtuu testitapauksen suorituksen päätyttyä, jolloin testistä syntynyt aineisto kerätään ja tulokset arkistoidaan ja liitetään testitapaukseen. Raportointitulos näyttää testin lopputuloksen. Testiraportti yksinkertaisimmillaan sisältää vain onnistumisen tai epäonnistumisen, mutta voi myös sisältää enemmän tietoja testitapauksesta. Näissä tiedoissa voidaan käydä kaikki testin vaiheet niistä saadut tuotokset ja testauksessa kulunut aika. Testiraporttista voi käydä myös ilmi testatun laitteen tila tai testauksessa käytetyt resurssit. Jos testitapaus on epäonnistunut, epäonnistumisen kohta testauksessa sisältyy testiraporttiin.

Testauksen edetessä ja testitapauksia uudelleen testatessa raporteista kertyy testitapaushistoria, josta voi todeta kuinka kauan testitapaus on ollut onnistuneesti testattu, milloin testin havaitsema virhe on korjattu ja uuden virheen sattuessa, missä välissä virhe on alkanut esiintyä testauksessa.

2.1 Ohjelmistotestauksen eri muotoja

Ohjelmistotestausta on mahdollista järjestää monin eri tavoin, jotka testaavat ohjelmistoa eri näkökannoista. Pelkillä yhden kaltaisilla testeillä ei ole mahdollista kattaa koko ohjelmiston toimintaa, eikä verrattain raskailla testeillä kannata tarkistaa koko ohjelmiston toimintaa. Jos ohjelmiston testaamisen osa-alueelle löytyy testi, joka pienemmällä vaivalla kattaa testitapauksen, sen käyttäminen kannattaa.[7]

Yksikkötestaus

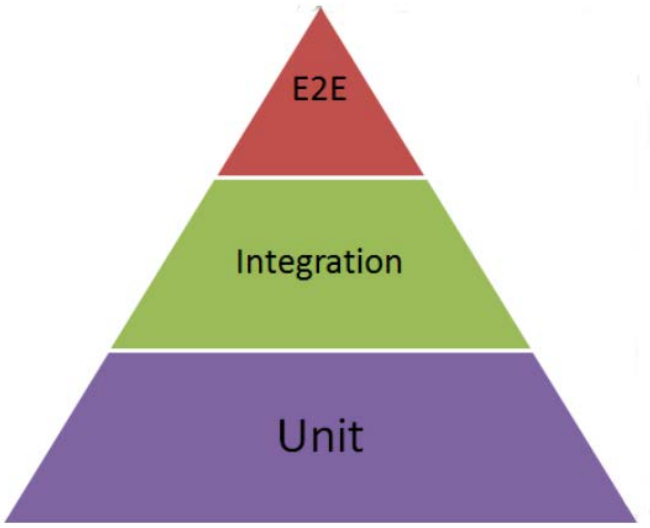
Yksikkötestaus (Unit Testing) on testausta, jossa testataan ohjelmiston pienimpiä testattavia osia. Tämä voi olla esimerkiksi luokka tai metodi, joille on kirjoitettu omat testinsä, jotka varmistavat toimivuuden kaikissa niin oikeissa kuin virheellisissä tilanteissa [8]. Yksikkötestit ovat nopeita suorittaa ja antavat näin nopeasti ja tarkasti tiedon siitä missä ohjelmistossa virhe on tapahtunut [9]. Kuitenkin yksikkötesti eivät testaa yksikkönsä ulkopuolista toimintaa, joten niillä ei voi kattaa koko ohjelmiston testausta. Eri ohjelmiston osasten keskinäistä toimintaa tulee testata muilla tavoilla.

Integraatiotestaus

Integraatiotestauksessa (Integration Testing) varmistetaan, että uudet lisätyt asiat ohjelmistoon ovat yhteen sopivia muiden niitä ennen tehtyjen ominaisuuksien kanssa, eivätkä aiheuta ongelmia. Integraatiotestaus perustuu usein ohjelmiston systeemiarkkitehtuuriin. Usein tällaista testausta käytetään CI-prosessissa (Continuous Integration / Jatkuva Integraatio), jossa haetaan jatkuvaa ohjelmisto-ominaisuuksien lisäämistä ohjelmistoon ja integraatio-ongelmien estämistä ohjelmistokehityksessä. [10]

End-to-End -testaus

End-to-End -testauksessa ohjelmiston ominaisuudet testataan ohjelmistolta esitettyjä vaatimuksia vasten. Testauksessa ohjelmiston suoritus tarkistetaan alusta loppuun seuraten ohjelmiston komponenttien ja systeemien toimintaa. Komponenttien ja systeemien välittämien tietojen virheettömyys testataan. [11]



Kuva 1 Esimerkki testien suhteellisesta määrästä projektissa

2.2 Ohjelmistotestauksen automaatio

Testaus vaatii hyvin suuren työpanoksen. Noin 50 prosenttia nykyisestä ohjelmistokehitykseen käytetystä työstä on ohjelmiston testaamista [12]. Kuitenkin suurin osa tästä työstä on arvojen tarkistusta ennalta määriteltyjä oikeita arvoja vastaan. Tämänkaltaisten yksinkertaisten tehtävien antaminen tietokoneen hoidettavaksi on helppoa ja kustannustehokasta. Lisäksi tietokone pystyy suoriutumaan paljon ihmistä nopeammin testauksesta, jolloin testausaika pienenee.

Testausautomaatiossa testitapauksesta kirjoitetaan testiskripti, joka annetaan tietokoneelle suoritettavaksi. Testiskriptissä on määritelty, miten tietokone ohjelmistoa testaa. Kun tietokone on ajanut testin läpi, antaa tietokone raportin testin tuloksista, joista näkyy, onnistuiko testi, ja jos testi epäonnistui, niin missä testi on epäonnistunut. Näistä tuloksista ohjelmistosuunnittelija voi päätellä, onko ohjelmistossa virheitä ja mitä pitää korjata.

Huolimatta kaikista testausautomaation hyödyistä kaikkea ei kannata testata tietokoneella. Jotkut testitapaukset voivat vaatia ihmismäistä intuitiota testauksessa syntyvien arvojen tulkinnassa, jota ei voi tietokoneella mallintaa [13]. Myös testitapaukset, jotka ovat luonteeltaan tutkivia ja joiden tuloksien avulla suunnitellaan varsinaisia automatisoitavia testitapauksia, on parasta suorittaa manuaalisesti. Tällaiset testitapaukset suoritetaan usein vain kerran ja kertasuoritusten automatisoinnille ei ole mitään syytä.

Automatisaatiota suunniteltaessa kannattaa ottaa huomioon, onko testitapausten automatisointiin käytettävällä työmäärällä saavutettavissa tarpeeksi suuri hyöty. Testitapauksia, jotka testaavat vähämerkityksistä harvoin tapahtuvaa tapausta, jonka automatisoiminen olisi kallista, ei kannata automatisoida. Suunnittelussa kannattaa ottaa myös huomioon se, että automatisoituja testejä tulee ylläpitää ja jos ylläpidon kustannukset ylittävät automatisaatiosta saadut hyödyt, ei niitä kannata automatisoida.

Automatisoidussa testauksessa on myös mahdollista tehdä oraakkeleita (Oracle), jotka pystyvät katsomaan testitapauksen lopputuloksesta, oliko se onnistunut vai ei. Kuitenkin tällaisten oraakkeleiden työstäminen ja ylläpito voi tulla hyvin kalliiksi ohjelmistoprojektin edetessä. Siksi suurin osa testauksesta kannattaa jättää tietokäyttöiseksi testaukseksi (Data-Driven testing). Tietokäyttöisen testauksen hyöty on siinä, ettei oraakkeleita tarvita, vaan testaus suoritetaan ennakolta koottuja arvoja vastaan.

2.3 Testauksenhallintaohjelmiston hyödyt

Testauksenhallintaohjelmisto helpottaa ohjelmistoprojektin testauksen suunnittelua, testien ajamista, testien automatisointia, testien ja virheiden raportointia ja työmäärien ja aikataulujen seurantaa. Näitä ohjelmiston tarjoamat hyödyt eivät ainoastaan auta testauksen suunnitellussa, mutta myös testauksen hallinnoinnissa ja työmäärien mittaamisessa. Testauksenhallintaohjelmisto antaa käyttäjälleen työkalut, joilla seurata ohjelmistoprojektin työn edistystä.

Testauksenhallintaohjelmisto antaa mahdollisuuden kaikkien testauksen suunnittelijoiden käyttää samaa ohjelmistoa hyväkseen suunnittelussa ja tallettaa suunnitelmat kootusti. Testinhallintaohjelmisto voi tarjota myös testien ajamiseen työkaluja, joilla aloittaa, ajastaa tai poistaa automatisoidusta testauksesta testejä. Testien raportoinnin keruu ja säilytys ovat myös testinhallintaohjelmistoille tavallisia ominaisuuksia.

Nämä kaikki ominaisuudet yhdessä varmistavat, että kaikki ohjelmistotestauksen alueet ovat yhden työkalun hallinnassa ja helposti löydettävissä. Varastoitu tieto voidaan myös koota helppolukuisiksi kaavioiksi ja tietopaketeiksi, joista testauksen tilaa on helppo seurata ja näin arvioida tehdyn työn määrää ja vielä jäljellä olevan työn suuruutta.

Testauksenhallintaohjelmisto myös tallettaa tietojen ja tiedostojen muokkaushistorian ja liittää muokkaushistoriaan syyn muokkaukselle. Jos testitapauksen vaatimus on muuttunut tai testianalyysissä on paljastunut jonkun muun testin jo kattavan vaatimuksen, on testitapauksia muutettava tai poistettava turhana toistona. Muokkaushistoriasta voidaan näin seurata muuttuvien vaatimusten ja virhekorjausten vaikutusta testaukseen. Tällä mahdollistetaan testimateriaalin muuttuminen ja näiden muutosten seuraaminen ja liittäminen vaatimuksiin.

3 Käytetyt työkalut

Seuraavat alakohdat tulevat käsittelemään yleisesti testauksessa käytössä olevia työkaluja, joita tulen opinnäytetyössäni käsittelemään. Työkalut esitellään ja niiden käyttötapaa ohjelmistotestauksessa avataan. Opinnäytetyössä ei tulla käsittelemään näiden työkalujen käyttöönottoa.

3.1 SpiraTest

SpiraTest on kaupallinen Inflectra Corporationin kehittämä testinhallintaohjelmisto. SpiraTestiä käytetään ohjelmistotestauksessa tarvittavien tietojen säilömiseen, testauksen suunniteluun, testauksen automatisoimiseen ja testitulosten raportointiin. SpiraTest-ohjelmaa tullaan käsittelemään luvussa 4 testinhallintaohjelmisto.

3.2 Robot Framework

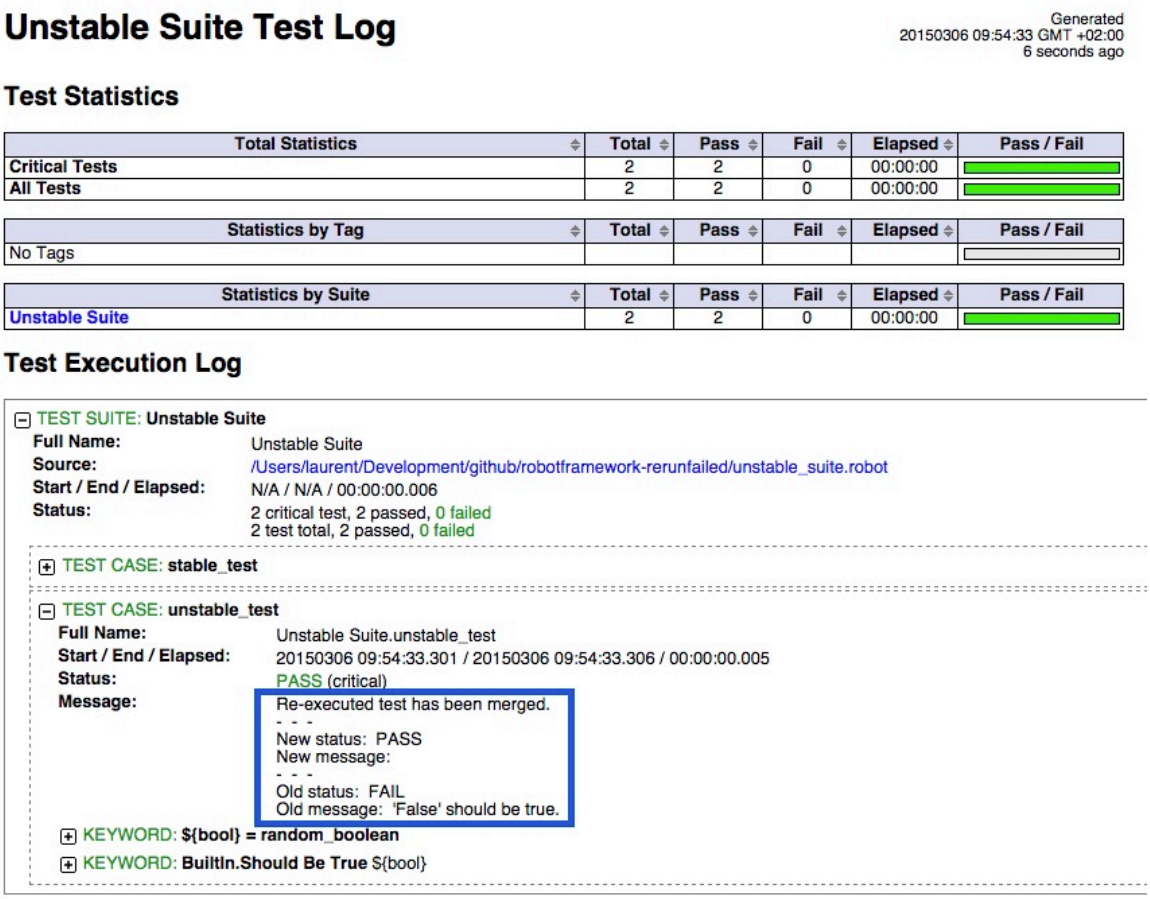
Robot Framework on avoimen lähdekoodin testausautomaation viitekehys, jonka avulla on mahdollista suorittaa hyväksyntätestausta [14]. Robot Framework käyttää testien laatimiseen Avainsanapohjaista (Keyword Based) skriptausjärjestelmää. Avainsanapohjaisessa testauksessa käyttäjä hyödyntää Frameworkin tarjoamia valmiita tai itse tekemiään avainsanoja testaustapauksen kirjoittamiseen. Avainsanojen käyttö testauksen kirjoittamisessa helpottaa testitapausten laadintaa ja avainsanoja on mahdollista käyttää muiden testitapausten laadintaan.

```
*** Settings ***
Documentation     A test suite containing tests related to invalid login.
...
...               These tests are data-driven by their nature. They use a single
...               keyword, specified with Test Template setting, that is called
...               with different arguments to cover different scenarios.
...
...               This suite also demonstrates using setups and teardowns in
...               different levels.
Suite Setup       Open Browser To Login Page
Suite Teardown    Close Browser
Test Setup        Go To Login Page
Test Template     Login With Invalid Credentials Should Fail
Resource          resource.robot

*** Test Cases ***                USER NAME        PASSWORD
Invalid Username                  invalid          ${VALID PASSWORD}
Invalid Password                  ${VALID USER}    invalid
Invalid Username And Password     invalid          whatever
Empty Username                    ${EMPTY}         ${VALID PASSWORD}
Empty Password                    ${VALID USER}    ${EMPTY}
Empty Username And Password       ${EMPTY}         ${EMPTY}

*** Keywords ***
Login With Invalid Credentials Should Fail
    [Arguments]    ${username}    ${password}
    Input Username    ${username}
    Input Password    ${password}
    Submit Credentials
    Login Should Have Failed

Login Should Have Failed
    Location Should Be    ${ERROR URL}
    Title Should Be    Error Page
```

Kuva 2 Robot Framework skriptiesimerkki [17]

Robot Framework pohjautuu tietokäyttöiseen testaukseen (Data-Driven testing) ja tarvitsee siis tiedoston, johon syötteet ja ennakoidut arvot on liitetty Robot Framework avainsanasyntaksin mukaan. Avainsanat Robot Frameworkissa ohjelmoidaan Pythonilla, mutta voidaan myös liitännäisen avulla ohjelmoida Javalla.

Robot Frameworkin testaamat testitapaukset on mahdollista raportoida, jolloin ne näkyvät helposti tutkittavassa muodossa, josta näkee onnistuneet ja epäonnistuneet avainsanat ja kaikki testissä käytetyt syötteet ja testatut arvot. Raportista näkyy selvästi, missä kohtaa testi on epäonnistunut.



Unstable Suite Test Log

Generated
20150306 09:54:33 GMT +02:00
6 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	2	2	0	00:00:00	
All Tests	2	2	0	00:00:00	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Unstable Suite	2	2	0	00:00:00	

Test Execution Log

TEST SUITE: **Unstable Suite**
Full Name: Unstable Suite
Source: /Users/laurent/Development/github/robotframework-rerunfailed/unstable_suite.robot
Start / End / Elapsed: N/A / N/A / 00:00:00.006
Status: 2 critical test, 2 passed, 0 failed
2 test total, 2 passed, 0 failed

TEST CASE: **stable_test**

TEST CASE: **unstable_test**
Full Name: Unstable Suite.unstable_test
Start / End / Elapsed: 20150306 09:54:33.301 / 20150306 09:54:33.306 / 00:00:00.005
Status: PASS (critical)
Message:
Re-executed test has been merged.
- - -
New status: PASS
New message:
- - -
Old status: FAIL
Old message: 'False' should be true.

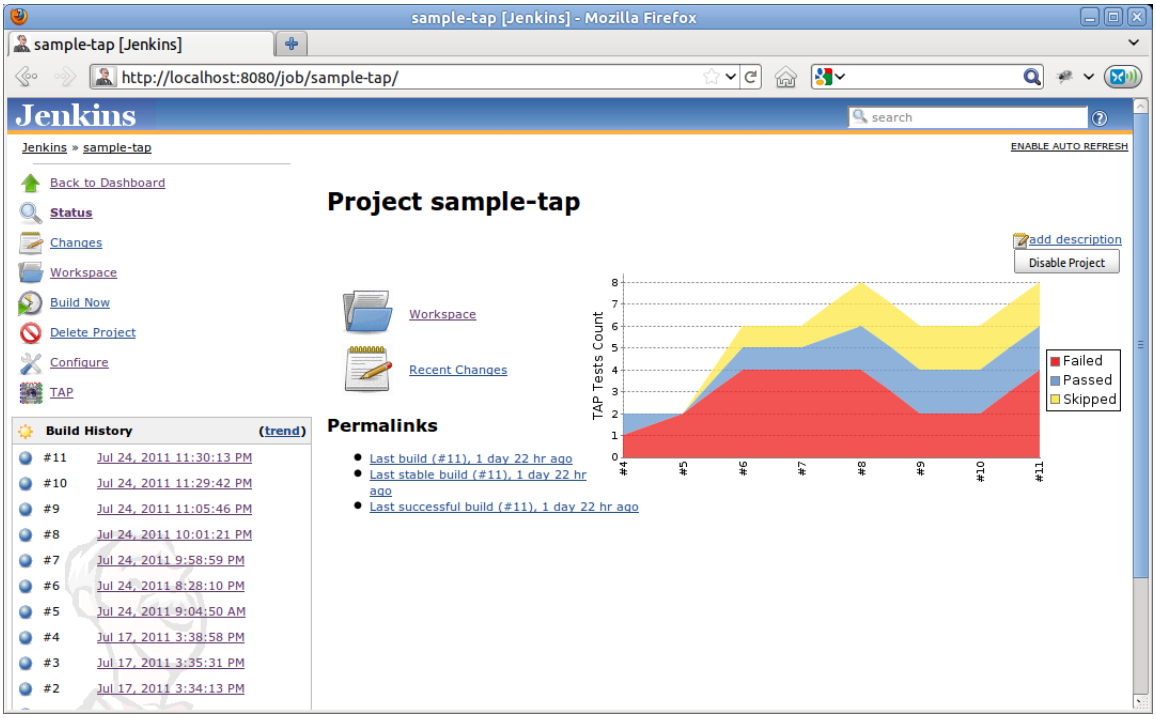
KEYWORD: **${bool} = random_boolean**
KEYWORD: **BuiltIn.Should Be True** ${bool}

Kuva 3 Robot Framework -testiraportti

3.3 Jenkins

Jenkins on avoimen lähdekoodin automaatioserveri. Automaatioserverillä on mahdollista suorittaa testitapauksia automaattisesti, joko käyttäjän käynnistämänä tai testaus voi myös käynnistyä määritellystä käynnistyskriteeristä. Kriteeri voi olla kellon aika, toinen testitapaus tai jokin muu heräte.

Jenkins toimii osana CI-järjestelmää. Jenkinsiin luodaan tehtäviä (job). Jenkins tehtävät ovat testaustietoja säilyttäviä testaussuorittajia, joille annetaan resurssit ja ohjelmoidaan suoritusskripti. Suoritusskripti voi olla esimerkiksi ohjelmistotesti, joka suoritetaan tehtävälle annetulla tietokoneella. Tämä tehtävä voidaan käynnistää halutuilla kriteereillä, jolloin suoritusskripti ajetaan ja sen lopputulos raportoidaan tehtävän suoritushistoriaan.

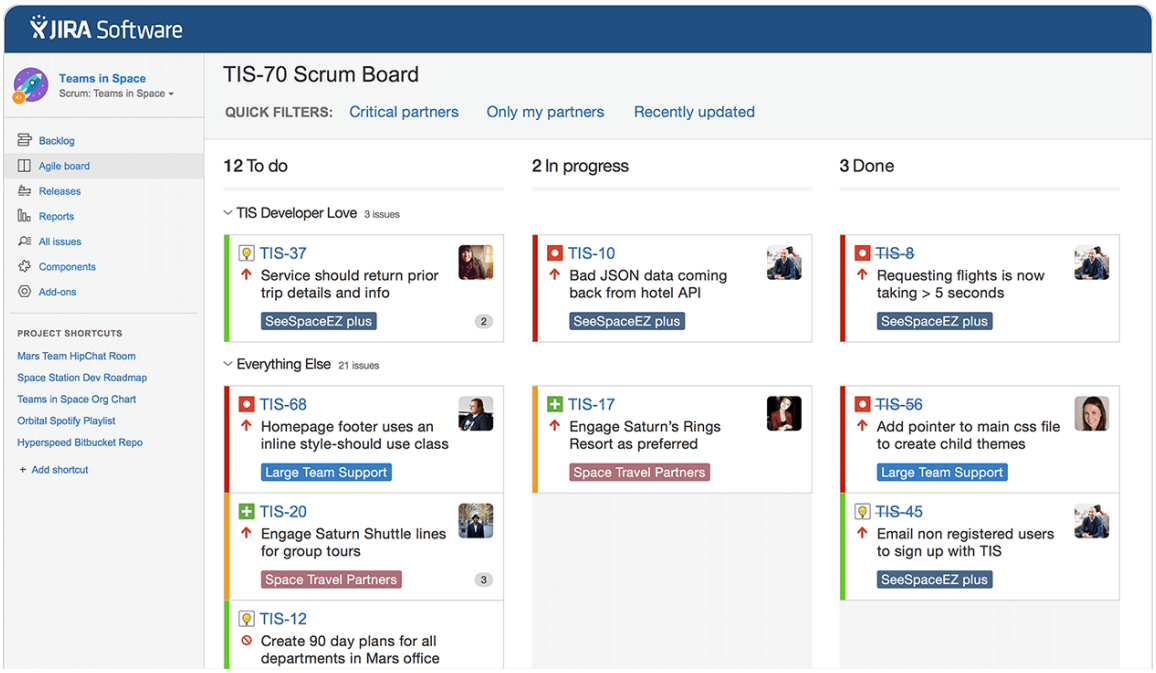


Kuva 4 Jenkins-tehtävänäkymä

Jenkinsiä käytettään yleisesti Robot Framework -testiskriptien ajamiseen ja näiden testiskriptien tulosten raportointiin. Jenkinsin tehtäviä käynnistetään sekä ajoitettuina, joka päivä tehtävinä testeinä että ohjelmoijien niin halutessa. [15]

3.4 JIRA

JIRA on Atlasianin kehittämä, laajasti käytössä oleva maksullinen ketterän ohjelmistosuunnittelun työkalu, jolla voidaan hallinnoida työtehtäviä ohjelmistoprojekteissa [16]. Jira hallinnoi työntekijöitä ja työtehtäviä ja mahdollistaa ketterän ohjelmistokehityksen yhdistämällä työntekijät työtehtäviin. Työntekijöiden on mahdollista ottaa itsenäisesti Jiraan listattuja työtehtäviä itselleen työn alle ja Jira pitää kirjaa näistä työtehtävistä. Työtehtäviä on monia erilaisia: tehtäviä, tarinoita (story), pää- ja sivuvaatimuksia (main requirement / requirement) ja monia muita.



Kuva 5 Jiran Scrumtaulu

Jiran työtehtäviä on mahdollista kommentoida ja niiden tila päivittyy työn edetessä. Työtehtäviin on mahdollista liittää tiedostoja. Työtehtävien toisiinsa yhdistäminen on helppoa käyttämällä työtehtäville annettavaa yksilökohtaista ID:tä. Työtehtävissä näkyy myös, mitkä muut työtehtävät viittaavat niihin.

3.5 Git

Git on ilmainen ja avoimen lähdekoodin versionhallintaohjelma [17]. Versionhallintajärjestelmä säilyttää tiedot edellisistä versioista historiassaan ja mahdollistaa vanhempaan versioon siirtymisen. Git käyttää tiedostojensa kontrollointiin hajautettua mallia. Git mallissa ohjelmoijalla on kokonainen klooni (clone) kaikista tiedostoista repositoriosta tietokoneellaan, mikä mahdollistaa työskentelyn ilman yhteyttä jaettuun repositorioon. Git mahdollistaa nopean haaroittumisen (branching) ja muut versionhallintakomennot, sillä ne suoritetaan paikallisella tietokoneella eikä yhteisessä repositoriossa.

Git kehitettiin alun perin Linux alustalle, mutta se on saatavilla myös muille alustoille. Gittiä käytetään yleensä komentoriviltä käsin, mutta on myös mahdollista asentaa graafinen käyttöliittymä GUI Gitin käyttöön.

4 Lähtötilanne

Lähtötilanne opinnäytetyölle on tapa, jolla SpiraTest testinhallintaohjelmistoa käytetään. Jos ohjelmistotestauksen järjestäisi jollain muulla tavalla, voitaisiinko tällä uudelleen järjestelyllä saada parempia testausprosesseja. Parhaassa tapauksessa testausprosessit voisi saada nopeammiksi toteuttaa ja laadun takaamiselle jäisi enemmän aikaa. Näistä hyödyistä syntyisi laadukkaampia ohjelmistotuotteita pienemmillä kustannuksilla.

Testauksessa käytetyt toimintamallit tulee tutkia kriittisesti ja arvioida niiden vaikutusta testauksessa. Toimintamalleista saadut hyödyt tulee tunnistaa ja näille toimintamalleille vaihtoehtoismalleja suunniteltaessa nämä hyödyt tulee löytyä myös esitetyistä ratkaisuista.

Opinnäytetyöltä vaaditaan ratkaisuehdotuksia, jotka takaavat ohjelmiston laadun ja tuottavat selviä hyötyjä ohjelmistotestauksessa ja sen suunnittelussa. Ratkaisuehdotukset tulee kuvata niin, että ne on toteutettavissa ja niistä saadut hyödyt tulee kertoa.

5 Testinhallintaohjelmisto

Lähtiessämme tarkastelemaan testinhallintaa otamme lähtökohdaksi SpiraTest-testinhallintaohjelmiston. Testinhallintaohjelmiston tämän hetkistä tapaa käyttää selvitetään ja eritellään. Lopuksi SpiraTestin käyttöä analysoidaan työskentelyn parantamisen ja helpottamisen näkökulmasta.

5.1 Testinhallintaohjelmistoanalyysi

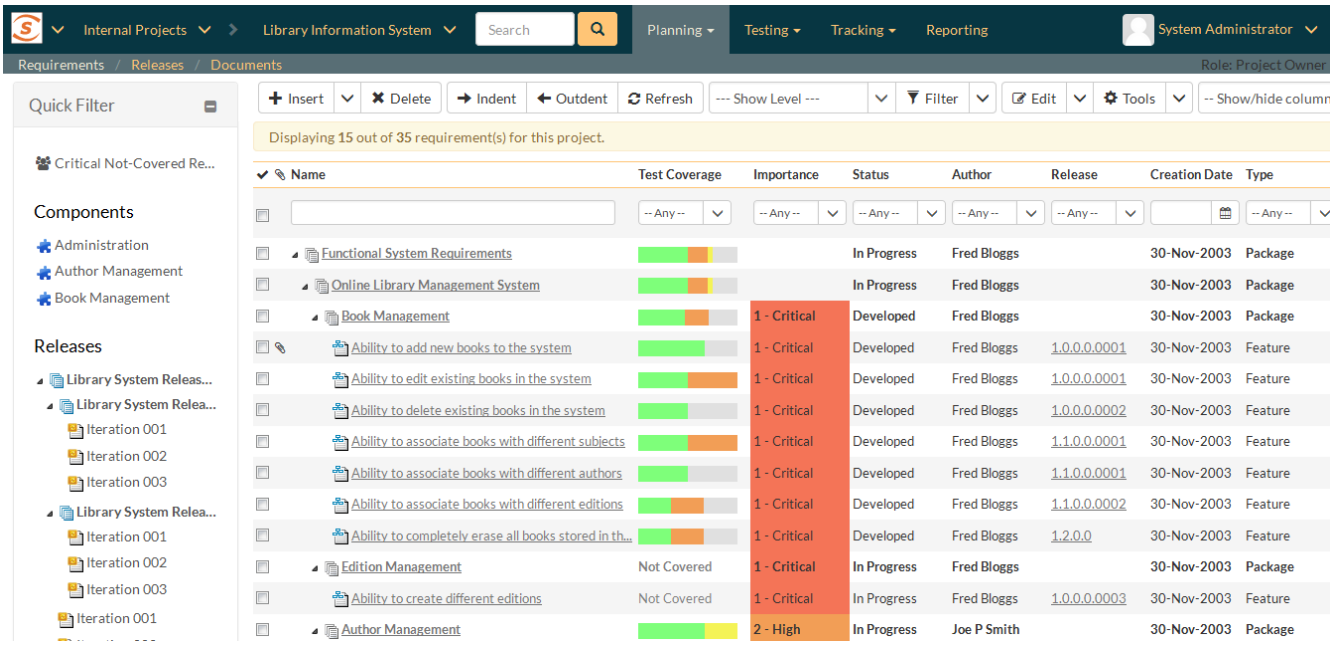
SpiraTest testinhallintaohjelmalla on monia ominaisuuksia, jotka helpottavat testauksen hallinnoimista alkaen testin vaatimuksista testin kirjoittamiseen, testin hyväksymisprosessiin, testien suorittamiseen ja testien tulosten raportoimiseen. Seuraavat kappaleet tulevat käymään läpi nämä ominaisuudet ja kuinka ne on toteutettu SpiraTestissä.

5.1.1 Vaatimukset

Testitapauksen luonti lähtee vaatimuksista (requirement). Vaatimuksella tarkoitetaan tässä opinnäytetyössä ohjelmistosysteemiltä vaadittujen ominaisuuksien ja funktioiden kuvausta. Vaatimuksessa tulee olla ohjelmistolta vaadittu ominaisuus tai toiminto tarpeellisen hyvin kuvattuna. SpiraTest mahdollistaa vaatimusten säilyttämisen ja tarkkailun vaatimusnäkymästä SpiraTest ohjelmistossa. Vaatimusnäkymästä näkyy selvästi, kuinka monta testiä on vaatimusta kohden ja kuinka moni niistä on onnistunut, epäonnistunut, estetty tilassa tai suorittamatta. Vaatimuksille on myös mahdollista määrittää tärkeystaso.

SpiraTest myös yhdistää vaatimukset testitapauksiin. Testitapausten yhdistäminen on kaksipuolista: vaatimuksesta on mahdollista nähdä testitapaukset, jotka testaavat vaatimusta ja testitapauksista on mahdollista nähdä mitä vaatimusta testitapaus varmistaa. Näin on nopeasti havaittavissa mitkä testit eivät onnistu vaatimuksesta ja kuinka lähellä täysimääräistä toimintakuntoa vaatimus on.

Jiran ja SpiraTestin vaatimusten yhdistäminen on tuettu SpiraTestin Jira liitännäisellä. Tämä tarkoittaa Jiraan luotujen vaatimusten automaattista kopioimista SpiraTestiin, jolloin vaatimukset ovat yhtenevät ja löydettävissä molemmista työkaluista. Jirasta on kopioinnin jälkeen mahdollista siirtyä Jira vaatimuksen kautta SpiraTest vaatimukseen ja nähdä miten vaatimus on katettu testeillä ja kuinka lähellä vaatimuksen täysi kattaminen on testeillä. Kun testeillä saavutetaan vaatimuksen täysi kattavuus, voidaan vaatimus sulkea ja suoritusta voidaan alkaa seurata automatisoiduilla testeillä.



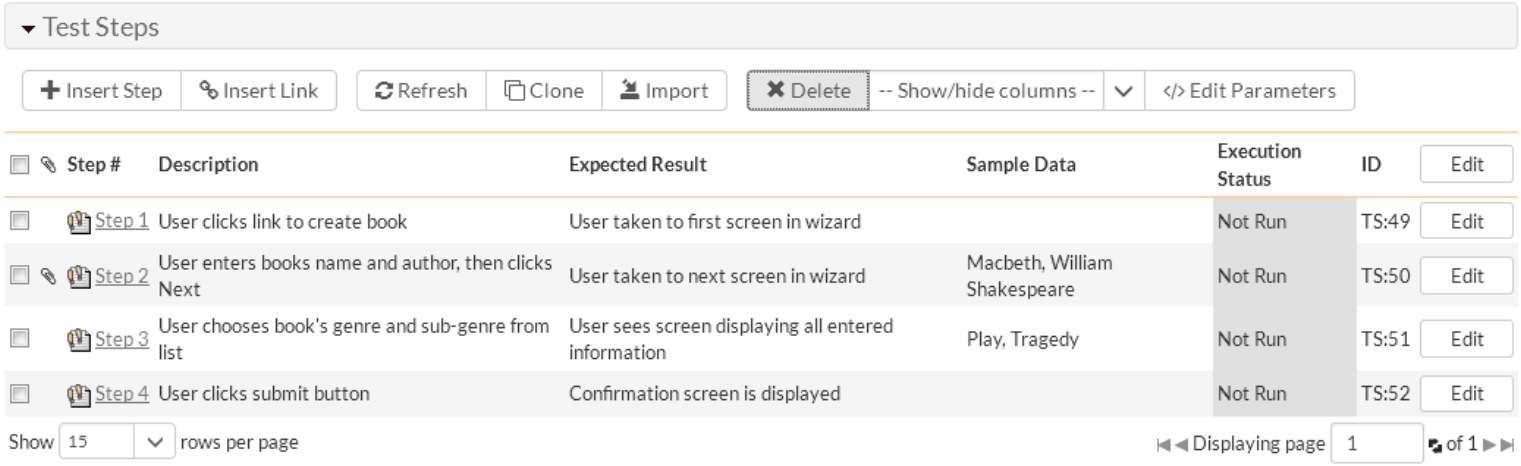
Kuva 6 Testivaatimusnäkymä SpiraTest

5.1.2 Testitapaus

SpiraTestin päätehtävä on hallinnoida testitapauksia (Test Case). Testitapaukset sisältävät testivaiheet ja testin ajamiseen liittyvän informaation. Testitapaukset suunnitellaan vaatimusten ja testaussuunnittelutekniikoiden avulla.

Testitapausta kirjoittaessa SpiraTest antaa testitapausaihion, johon testin kirjoittajan on mahdollista lisätä testi-informaatio. Testi-informaatiolle on monia esitehtyjä kenttiä joihin voi laittaa testissä käytettävän ohjelmistoversion, laitteiston, testin omistajan, testin kirjoittajan ja muita testin kannalta hyödyllisiä tietoja.

Testitapauksen tärkein informaatiokenttä on testin vaiheet, jonka SpiraTest jakaa testiaskeliin (Test Step). Testiaskelissa kuvataan testin suoritus vaihe vaiheelta: miten vaihe suoritetaan, mikä tulisi olla suorituksen lopputulos ja mahdollinen syötettävä arvo suoritukselle. Testitapauksesta näkyy myös, onko se suoritettu vai ei ja onko suoritus mahdollisesti estetty. Jos suoritus on estetty, este on mahdollista linkata esteen aiheuttamaan vika raporttiin SpiraTestissä.

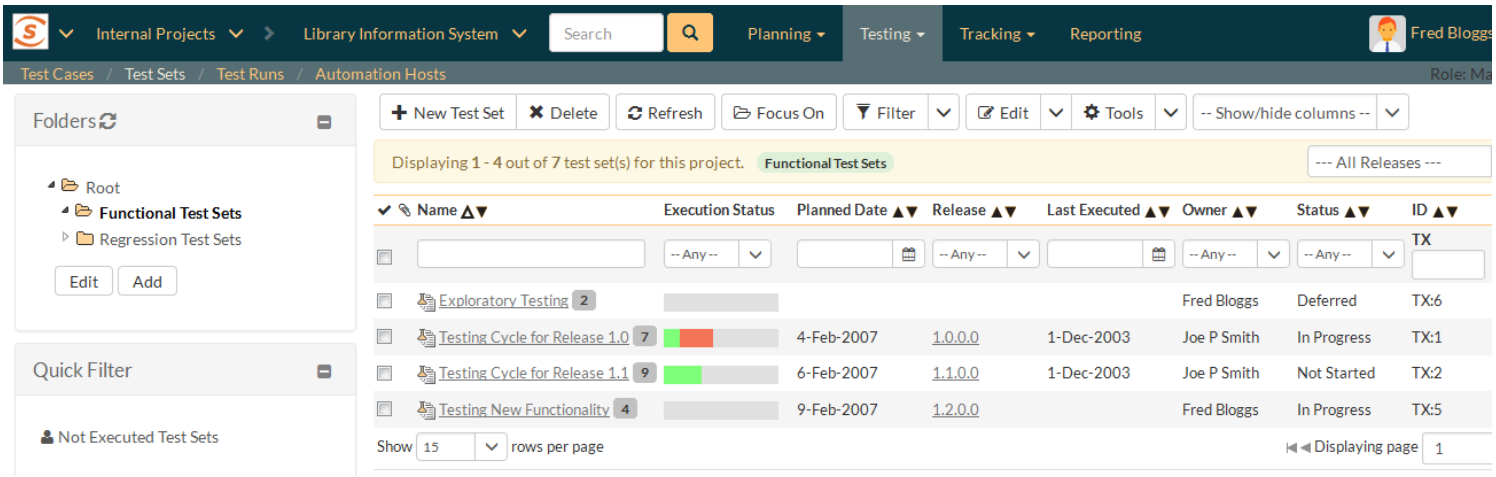


Kuva 7 Testivaiheet SpiraTest

Suorittaessa testitapauksen testausta on mahdollista käyttää SpiraTestiä niin manuaalisessa kuin automatisoidussa testauksessa. Manuaalisesti testatessa SpiraTest antaa käyttäjän käydä läpi testiaskeleet yksi kerrallaan ja merkitä testiaskeleen suorituksen tuloksen. Automatisoituun testaukseen SpiraTest antaa

mahdollisuuden ulkoisten testauskirjastojen käyttöön, jotka voivat raportoida SpiraTestiin suorituksensa. Näin testauksen tulokset jäävät näkymään testitapauksessa ja ne on helppo tarkistaa.

Testitapaukset myös ryhmitellään testiseteiksi (Test Set). Tämä helpottaa testien hallinnointia. Testisetistä on mahdollista katsoa kaikkien sen sisältämien testien tulokset. Tulokset ilmoitetaan värillisenä palkkina, jossa on vihreä onnistunut, punainen epäonnistunut, oranssi osittain epäonnistunut, keltainen estetty, ja harmaa ajamaton testi.



Kuva 8 Testisettinäkymä SpiraTest

5.1.3 Testien hyväksymisprosessi

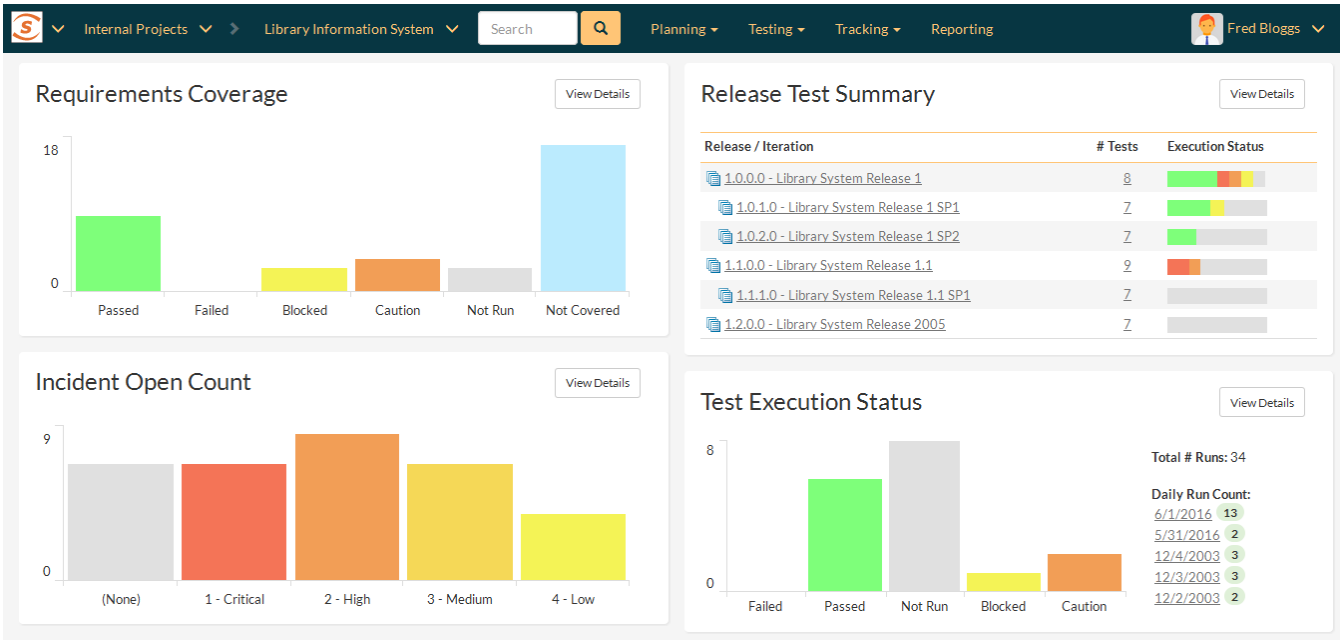
Kun testitapaus on kirjoitettu valmiiksi, tulee sen käydä läpi katselmointiprosessi. Prosessissa arvioidaan, onko testi vaatimuksen mukainen, onko testin testaama asia jo muun testin kattama ja onko se laadultaan tarpeeksi hyvä testaukseen lisäykseen.

SpiraTest antaa apuja testien katselmointiin. Testitapauksissa on katselmoinnin tila, josta käy ilmi, onko testi katselmoimaton, katselmoinnissa vai hyväksytty testaukseen. SpiraTestin testisettinäkymässä on mahdollista nähdä yhdellä vilkai-

sulla kaikkien testisetissä olevien testien katselmoinnin tilan. Tämä auttaa seuraamaan testien tekoprosessin edistymistä.

5.1.4 Testien raportointi

SpiraTest mahdollistaa testitulosten seurannan monin eri tavoin. SpiraTestin projektinäkymä antaa yleiskuvan ohjelmistoprojektin tilasta. Projektinäkymästä voi nopeasti havaita projektin testauksen etenemän ja jäljellä olevan työn määrän..



Kuva 9 Projectinäkymä SpiraTest

SpiraTest tarjoaa käyttäjälleen myös mahdollisuuden saada yksityiskohtaisen raportin testauksesta Excel-taulukkona, johon on listattu kaikki eri testausalueet ja niiden testitapausten määrä ja niiden suoritustilanne.

5.1.5 Muita SpiraTestin ominaisuuksia

SpiraTest tarjoaa myös monia muita ominaisuuksia, jotka helpottavat testauksen suunnittelua ja toteutusta, mutta jotka eivät jakaannu edellisiin isompiin kokonaisuuksiin. Osa ominaisuuksista on myös sellaisia, jotka ovat käytettävissä useassa SpiraTestin osassa.

Yksi ominaisuuksista on mahdollisuus liittää tiedostoja eri SpiraTest-osasiin kuten testitapauksiin, vaatimuksiin ja virheraportteihin. Tiedostot voivat sisältää lisätietoja helposti luettavassa muodossa, kuvia näyttämään visuaalisia tilanteita testauksesta tai syöte-esimerkkejä, joilla testiä on ajettu.

Testauksen tulokset ovat raportteja, joista näkyy, mitkä asiat toimivat ohjelmistossa ja missä kohtaa ohjelmistossa on virheitä. SpiraTest mahdollistaa näiden virheiden kirjaamisen. Näitä virheitä on mahdollista seurata ja niiden tilaa voi muuttaa, jolloin testauksessa tiedetään, mitkä virheet ovat testien löytämiä ja ovatko ne korjattu.

5.2 Huomioita SpiraTestin käytöstä

SpiraTestin suurimmat hyödyt ovat informaation säilyttäminen ja helppolukuisuus. SpiraTest tarjoaa keinon tiedon helppoon löydettävyyteen hakukenttien ja yksilöllisesti identifioitujen testitapausten, testisettien ja vaatimusten avulla. Lisäksi SpiraTestin sisäisen testitapausten ja vaatimusten yhdistäminen mahdollistaa oikein käytettynä yhtenäisen testaussuunnitelman ja testauksen järjestämisen.

SpiraTest mahdollistaa myös monien muiden työkalujen käyttämisen SpiraTestin kanssa. Näillä työkaluilla voi hoitaa esimerkiksi testien ajamisen, jolloin SpiraTestiin kirjatut testikoodit suoritetaan muilla työkaluilla ja tulokset kirjataan SpiraTestiin.

SpiraTestiä käytettäessä johtuvat sen eriytymisestä muista työkaluista. SpiraTest haluaa käyttäjänsä käyttävän kaikkia ominaisuuksiaan eikä helposti toimi yhdes-

sä muiden testaustyökalujen kanssa. Tämä johtaa siihen, että testauksessa muihin testaustyökaluihin joudutaan uudelleen kirjaamaan samat asiat, jotka ovat jo SpiraTestissä sillä erotuksella, että SpiraTestissä ei ole mitään toiminnollisuutta, vaan se toimii pelkästään tiedon säilyttäjänä. Kun vaatimukset on suljettu ja katettu testitapauksilla ja automatisoidut testitapaukset varmistavat vaatimusten täyttymisen, tällä tiedolla ei ole enää paljon merkitystä.

Myös se, että SpiraTest toimii yhteisessä repositoriossa, johon kaikki joutuvat ottamaan yhteyden työskennelläkseen. Jos tässä repositoriossa on ongelmia, työskentely on kaikilla mahdotonta. SpiraTest ei tarjoa myöskään testauksen hallintaa ilman yhteyttä repositorioon, mikä tuottaa vaikeuksia ohjelmiston käytössä.

6 Korvaavan vaihtoehdon implementaatio

SpiraTest on hyvin laaja ja käytännöllinen työkalu. Jos kuitenkin halutaan luopua sen käyttämisestä ohjelmistoprojektissa, on korvaavan ehdotuksen täytettävä hyvin suuret laatu- ja toiminnallisuusvaatimukset. Tässä korvaustilanteessa on kaikki SpiraTestin käytetyt ominaisuudet myös löydyttävä korvaavasta ehdotuksesta.

Tämä luku tulee käymään läpi edellisessä luvussa analysoidun SpiraTest-testinhallintaohjelmiston kaikki ominaisuudet ja antamaan niille mahdollisen korvausehdotuksen. Korvausehdotuksessa ei tulla käsittelemään muita testinhallintatyökaluja, joiden käyttö johtaisi vain samaan lähtötilanteeseen kuin SpiraTestin nykyinen käyttö.

Vaatimuksena korvausehdotuksille on säilyttää testinhallintatoiminnot samalla virtaviivaistaen testien hallintaa ja vähentäen päällekkäisyyttä ja turhaa toistoa.

6.1 Vaatimukset

Vaatimusten laatimiseen voidaan käyttää Jira vaatimusta (requirement), joka voidaan jakaa osavaatimuksiin, jotka taas ovat linkattu SpiraTestin kanssa yhteen. Vaatimusten siirtäminen SpiraTestiin luo turhaa kopiointityötä. Kopiointityöltä voidaan välttyä luopumalla kokonaan SpiraTestin vaatimusten tekemisestä ja antamalla vaatimusten olla kokonaan Jirassa, jolloin kaikki tieto löytyy yhdestä ja samasta paikasta.

Tällöin vaatimukset säilötään Jirassa ja linkkaaminen Jiran vaatimuksesta testitapauksiin tapahtuu Jira-tehtävässä, jossa määritellään tehtävät testitapaukset ja johon niiden tila raportoidaan ja tehtyjen testitapausten polut kirjataan. Tämä mahdollistaa vaatimuksen testitapausten suoran seurannan Jira-vaatimuksen kautta. Käyttäjän ei tarvitse enää tarkistaa montaa työkalua saadakseen informaatiota, vaan voi pelkän Jiran kautta nähdä vaatimuksen tilan.

Jira-vaatimuksessa on hyvin tärkeää säilyttää tietoa siitä, missä vaatimuksen testitapaukset ovat ja kuinka paljon niistä on saatu tehtyä. Jos kaikkia testitapauksia ei ole tehty, on työn tilan näyttävä selkeästi ja jäljellä olevan työmäärän olla helposti selvitettävissä. Tilanteessa, jossa vaatimus on katettu jonkin muun vaatimuksen testauksella, on nämä testitapaukset merkittävä vaatimukseen ja testianalyysistä saatu syy kattavuudelle on kirjattava vaatimukseen mukaan. Näin toimimalla varmistetaan työmäärän seurattavuus ja ennakoitavuus.

Vaatimusten muuttuessa on myös testitapausten muututtava tai syy muuttumattomuuteen on käytävä ilmi testianalyysistä. Muutokset ja niiden vaikutukset testitapauksiin on siis kirjattava vaatimuksiin. Jos muutoksien seurauksia ei raportoida tunnollisesti vaatimukseen, ei ole mahdollista olla varma testauksen kattavuudesta ja työmäärien arviointi vaikeutuu.

Jotta ohjelmistoprojektissa olisi mahdollista saada kuvaus, josta käy ilmi projektin tila, tarvitaan erillinen projektitestaustilaraportti. Tällainen yleiskuva testauksesta on mahdollista saada Jenkinsin tekemästä yhteisestä testiraporttien yhteenvedosta, jossa eri vaatimusten tai testausalueiden mukaan testien suoritus- ja onnistumisasteet on lueteltu. Jira-tehtävään voidaankin liittää linkki tähän raporttiin, jolloin vaatimuksen testauksen tila olisi aina ajantasainen.

6.2 Testitapaus

Luovuttaessa SpiraTestistä kaikki informaatio testitapauksesta täytyy säilyttää jossain muussa muodossa kuin SpiraTestin testitapauksessa. Mahdollisuus tälle säilytykselle on Robot Framework -testiskripteissä, joihin liitettynä testitapauksen dokumentointi on suoraan testissä itsessään. Testiaskeleet kirjataan niin laajasti kuin on tarpeellista askeleen suorittamiseksi, mutta testiaskeleen informaatiokenttään voi lisätä tarvittaessa lisätietoja.

```
#############################################################
#                        Step 1
#Preconditions:
#    - No calculator sofware started
#
#    Start calculator
#
#Sample Data
#    Run  Calculator
#
#Expected Results:
#    Calculator Started
#############################################################
```

Kuva 10 Testiaskeleen informaatiokenttä

Testitapauksen kaiken informaation ollessa suoraan testitapauksessa on mahdollista käyttää testiskriptiä kehikkona, johon testiautomaatio on helppo sijoittaa. Lisäksi jos testitapausta ei ajatella koskaan laitettavaksi manuaalitestattavaksi tai jos testauksessa järjestetään pelkkä automaatiotestaus, tämä käytäntö mahdollistaa testitapauksen automatisoinnin laatimisen välittömän aloittamisen. Välittömyydellä saadaan lisää tehokkuutta testaukseen, josta jää pois turha välivaihe, jossa testin vaiheet kirjattiin ilman minkäänlaista toiminnollisuutta.

Lisäksi testitapauksen tiedot voidaan sovittaa Robot Framework -skriptin tyyliseksi, jolloin raportoiminen onnistuu siinäkin tapauksessa, jos testitapaus on manuaalitesti. Näin manuaalitestienkin raportoituminen Jenkinsin kautta on mahdollista. Samalla mahdollistetaan testien helpompi seuranta, kun automatisoidusta raportista voi suoraan lukea mitä on testattu ja millä tavalla ja raportti päivittyy automaattisesti.

Testitapauksen muokkaushistoria päivittyy automaattisesti, kun testitapaukset säilytetään Git versionhallinnassa, joka kirjaa muutokset ja johon on mahdollista liittää kommentteja muutoksen mukaan.

```
*** Settings ***
Documentation   Test case to test calculator

*** Keywords ***
Manual Test Report
    [Documentation] Test SW: 10.4.2017
    ...             \n Tester: Joni-Petteri Luomala
    ...             \n HW: 1234abc567
    ...             \n Test date: 10.4.2017

*** Testcases ***
Calculator Addition
    [Documentation] Calculator addition test
    ...             Author: Joni-Petteri Luomala
    ...             Date: 10.2.2017
    [Tag]           ManualTest  Run
    [Timeout]       1 minutes
    [Teardown]      Manual Test Report

#############################################################
#                         Step 1
#Preconditions:
#   - No calculator sofware started
#
#   Start calculator
#
#Sample Data
#   Run  Calculator
#
#Expected Results:
#   Calculator Started
#############################################################

#############################################################
#                         Step 2
#   Try addition of 2 and 3
#Expected Results:
#   5
#############################################################

#############################################################
#                         Step 3
#   Press C to clear
#
#Expected Results:
#   Null
#############################################################
```

Kuva 11 Robot Framework -testitapaus, jossa on vain manuaalitesti ja raportointi

```
Calculator Addition
    [Documentation] Calculator addition test
    ...             Author: Joni-Petteri Luomala
    ...             Date: 10.2.2017
    [Tag]           AutomatedTest
    [Timeout]       1 minutes
    [Teardown]      Close Open Calculators

#############################################################
#                       Step 1
#Preconditions:
#   - No calculator sofware started
#
#   Start calculator
#
#Sample Data
#   Run  Calculator
#
#Expected Results:
#   Calculator Started
#############################################################

    Close Open Calculators

    Run  Calculator

#############################################################
#                       Step 2
#   Try addition of 2 and 3
#Expected Results:
#   5
#############################################################

    ${result}   Calculate  2 + 3

    Should Be Equal  ${result}  5

#############################################################
#                       Step 3
#   Press C to clear
#
#Expected Results:
#   Null
#############################################################

    ${result}   Clear Calculator

    Should Be Equal  ${result}  Null
```

Kuva 12 Robot Framework –testitapaus johon on liitetty automaatio

6.3 Testien hyväksymisprosessi

Kun testitapauksiin on yllä olevalla tavalla lisätty testausinformaatiota, on mahdollista ajaa kaikki testit päivittäin läpi Jenkinsin kautta, joka suoraan raportoi testit kootusti verkkosivulle. Jenkinsin raportista komponenteittain tai muulla halutulla tavalla testitapaukset voidaan listata ja jaotella.

Testitapauksen Robot skriptiin lisätään tunnisteita (Tag), joilla ilmoitetaan testien valmisteluprosessin vaiheet ja muuta tietoa testistä. Esim. Not Reviewed, Reviewed, Under Review tilat. Näiden tunnisteiden avulla voidaan suoraan raportoida, missä vaiheessa testien katselmointi on ja mitä komponenttia tai vaatimusta testi kattaa.

```
Calculator Addition
    [Documentation] Calculator addition test
    ...             Author: Joni-Petteri Luomala
    ...             Reviewer: Erkki Esimerkki
    ...             Date: 10.2.2017
    [Tag]           AutomatedTest  ComponentCalculator  Reviewed
    [Timeout]       1 minutes
    [Teardown]      Close Open Calculators
```

Hyväksymisprosessin tilan päivittymiseen Jiraan voi automatisoida Jenkins liitännäisellä joka automaattisesti päivittää testitapauksen tilan Jira tehtävään testin päätyttyä. Näin Jiran tehtävästä voidaan suoraan katsoa onko testitapaukset katselmoitu.

Testitapauksien muutoshistoria säilyy versionhallinnassa, johon tässä tapauksessa käytetään Git versionhallintatyökalua. Kun käyttäjä muokkaa testi tapauksia ja tallentaa (commit) muutokset projektin Git repositorioon, muutoksesta säilyy kaikki tiedot. Näitä tietoja hyväksi käyttäen on myös mahdollista palata tarpeen vaatiessa vanhempaan tiedostoversioon. Gitin tallennuksien yhteydessä käyttäjältä vaaditaan viestiä, johon käyttäjä kirjoittaa muutoksen syyn. Näin Git pitää kirjaa testitapauksiin tehtävistä muutoksista ja mahdollistaa testitapauksen kehittymisen seuraamisen.

6.4 Testien raportointi

Muissa kohdissa mainittu Jenkins raportointi korvaa SpiraTestin raportoinnin. Testitapausten raportointi pystytään järjestämään paljon yksityiskohtaisemmin, kun käytetään kattavaa tunnistesysteemiä, joilla testitapaukset merkitään.

Jenkins-raportointi suoritetaan reaaliaikaisesti, jolloin virheiden määrää ja testikattavuutta on mahdollista seurata päivittäin yhdeltä ja samalta Jenkinsin raportti sivulta. Myös manuaalitestit on mahdollista liittää näihin testeihin. Jokainen Jenkins tehtävä, joka on raportoituna, on linkki tehtävään, josta on mahdollista katsoa testien suorituksen testitapaustasolla.

LASKIN PROGRAM AUTOMATED TESTING

Jenkins job	Review	Status	Pass rate	Comment
2.0_Calculator_Functions	Reviewed	Fail	3/5	#2 and #4 FAIL
2.0_Calculator_Robustness	Reviewed	PASS	3/3	
2.0_Calculator_Graphs	Reviewed	PASS	2/2	
1.0_Calculator_Regression	Reviewed	PASS	10/10	
1.0_Calculator_Upgrade	Reviewed	PASS	1/1	

LASKIN PROGRAM MANUAL TESTING AND TESTCASE REVIEW STATUS

Jenkins job	Review	STATUS	PASS_RATE	COMMENT
3.0_Calculator_Functions	Ready For Review	NOT RUN	0/1	
2.0_Calculator_New_Features	Under Review	NOT RUN	0/2	Issue P1-12345
2.0_Calculator_Upgrade	Reviewed	PASS	2/2	
2.0_Calculator_Functions	Not Ready for Review	NOT RUN	0/3	_
2.0_Calculator_Functions_manual	Reviewed	PASS	3/3	Last run on 23.1.2016

6.5 Muita SpiraTestin ominaisuuksia

SpiraTestin tarjoamat ominaisuudet liittää tiedostoja ja virheiden kirjaaminen löytyvät jo Jirasta, ja tästä syystä niistä luopuminen on helppoa. Tiedostojen liittäminen tullaan hoitamaan liittämällä ne Jiran tehtäviin. Virheiden korjaamisesta voi Jirassa tehdä oman tehtävänsä, jonka voi liittää Jiran sisäisesti sekä vaatimuksiin että testitapaustehtäviin.

7 Tulokset

SpiraTest testinhallintaohjelmisto on pitkään kansainvälisen yrityksen kehittämä ohjelmisto ja täten laadukas ja monipuolinen. Monet ohjelmiston ominaisuuksista eivät ole selviä nopeasti katsottuna, mutta kuitenkin nämä ylikatsotut ominaisuudet helpottavat ja tukevat testinhallintaa. Ilman niitä testauksen suunnittelu ja hallinta olisi paljon vaikeampaa toteuttaa.

On mahdollista siirtää kaikki SpiraTestin ominaisuudet muihin työkaluihin tai säilytysratkaisuihin. Kuitenkin tämä vaatii aikaisempaa enemmän huolellisuutta testauksen dokumentaatiossa ja laajempaa omatoimista testausmateriaalin tuottoa. Jos SpiraTest-ohjelmistosta luovutaan, monet itsestään selvät asiat, jotka ovat tällä hetkellä ohjelmiston tarjoamia, joudutaan tekemään manuaalisesti.

Siirrolla on myös hyviä puolia. Monet asiat, jotka SpiraTestin ollessa vaativat ylimääräistä kopiointia, olisi mahdollista järjestää paljon nopeammin. Esimerkiksi testaustapausten valmistaminen voitaisiin aloittaa suoraan ja siirtyä suoraan testiautomaatioon. Tämä voisi ehkäistä testaamisen ohjelmistokehityksestä jälkeen jäämisessä. Monet SpiraTestin ominaisuuksista jäävät tällä hetkellä käyttämättä ja ovat hyödyttömiä ja ominaisuuksilla on selvää päällekkäisyyttä muiden käytettyjen työkalujen kanssa. Myös suuren testimäärän ja testihistorian selaaminen on SpiraTestillä hankalaa. SpiraTestistä luopumiselle on monia hyviä syitä.

Luovutaan SpiraTestistä tai ei, joka tapauksessa testinhallinnassa on paljon kehitettävää. Päällekkäisyyden poisto ja turhan työn minimoiminen on hyvin tärkeää. Osa tässä opinnäytetyössä selvitetyistä asioista voisi ottaa harkintaan ohjelmistotestaukseen ja tiettyjä esille nostettuja ongelmia tulisi ratkaista.

8 Yhteenveto

Ohjelmisto suunnittelun kehittyessä ohjelmistojen rakenteet ovat monimutkaistuneet jatkuvasti. Näiden ohjelmistojen toimintavarmuuden selvittämiseksi tarvitaan jatkuvasti suurempia resursseja. Näiden resurssien hallintaan käytetyt testauksenhallintaohjelmistot ovat ohjelmistokehitysalalla yleisiä.

Näiden ohjelmistojen käytön optimointi säästää työtunneissa merkittäviä määriä ja samalla parantaa tuotteiden laatua. Ohjelmistojen käytössä ei kannata ainoastaan keskittyä ajan säästämiseen vaan ohjelmistotestauksen suunnittelua kannattaa kehittää myös selkeyden, informatiivisuuden ja toistettavuuden näkökohdista.

Opinnäytetyö kuvaa ohjelmistotestausta ja sen merkitystä ohjelmistoprojektissa. Ohjelmisto testauksenhallinnoinnin vaatimat työkalut ovat kuvattu ja niiden hyödyt eritelty. Työn pääpaino oli testauksenhallinnan vaihtoehtoisten suorittamistapojen esille tuonti ja testinhallinnan parantaminen. Työn lopputuloksena esitetyt vaihtoehtoisratkaisut ovat toteuttavissa ja niin ollen työ toteuttaa vaatimuksensa.

Lähteet

1. Homès, Bernard, and Bernard Homes. Fundamentals of Software Testing, edited by Bernard Homès, and Bernard Homes, John Wiley & Sons, Incorporated, 2013. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=1120766. s. 2 -3

2. Mili, Ali, and Fairouz Tchier. Software Testing, edited by Ali Mili, and Fairouz Tchier, John Wiley & Sons, Incorporated, 2015. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=4040909. s. 6

3. Elfriede Dustin, Jeff Rashka john Paul Automated Software Testing Introduction, Management and Performance Preface XVI - XVII

4. Mili, Ali, and Fairouz Tchier. Software Testing, edited by Ali Mili, and Fairouz Tchier, John Wiley & Sons, Incorporated, 2015. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=4040909. s. 23

5. Holcombe, Mike. Running an Agile Software Development Project, edited by Mike Holcombe, John Wiley & Sons, Incorporated, 2008. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=427745. s. 283

6. Homès, Bernard, and Bernard Homes. Fundamentals of Software Testing, edited by Bernard Homès, and Bernard Homes, John Wiley & Sons, Incorporated, 2013. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=1120766. s. XXV

7. Lisa Crispin, Janet Gregory, Agile Testing – A Practical Guide for Testers and Agile Teams s. 26-27

8. Microsoft developers Network – Unit testing Accessed on 11.4.2017 https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

9. *Mike Wacker Google Testing Blog Just Say No to More End-to-End Tests* https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html Accessed on 27.04.2017

10. Homès, Bernard, and Bernard Homes. Fundamentals of Software Testing, edited by Bernard Homès, and Bernard Homes, John Wiley & Sons, Incorporated, 2013. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=1120766. Glossary XXI

11. Lisa Crispin, Janet Gregory, Agile Testing – A Practical Guide for Testers and Agile Teams s. 486-487

12. Homès, Bernard, and Bernard Homes. Fundamentals of Software Testing, edited by Bernard Homès, and Bernard Homes, John Wiley & Sons, Incorporated, 2013. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=1120766. Glossary XXII

13. Mili, Ali, and Fairouz Tchier. Software Testing, edited by Ali Mili, and Fairouz Tchier, John Wiley & Sons, Incorporated, 2015. ProQuest EBook Central, https://kamezproxy01.kamit.fi:2252/lib/kajaani-ebooks/detail.action?docID=4040909. s. 12

14. Cem Kaner, James Bach, Bret Pettichord, Lessons Learned in Software Testing A Contex-Driven Approach s.99

15. Robot Framework website Accessed in 11.4.2017 http://robotframework.org/

16. Jenkins website Accessed in 11.4.2017 https://jenkins.io/

17. Atlassian website Accessed in 11.4.2017 https://www.atlassian.com/software/jira?_mid=e92f5d1dea5e48abbee395

0395c6288a&aceid=&adposition=1t1&adgroup=41572345111&campaign=742886319&creative=174460938173&device=c&keyword=jira&matchtype=e&network=g&placement=&gclid=CjwKEAjw_bHHBRD4qbKukMiVgU0SJADr08ZZuahdKfWHNnfeRuOcOgOmMO84E39xC0mOdZR3Nvg-NhoCr9Xw_wcB

18. Git website Accessed in 11.4.2017 https://git-scm.com/