Elyas Addawe

Testihallintatyökalu PLM-Zonelle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

8.5.2023

Tiivistelmä

Tekijä:	Elyas Addawe
Otsikko:	Testihallintatyökalu PLM-Zonelle
Sivumäärä:	34 sivua
Aika:	8.5.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmointituotanto
Ohjaajat:	Lehtori Jorma Räty

Lopputyön tavoitteena oli hankkia PLM-Zonelle testihallintatyökalu, mikä vastaa yrityksen tarpeita. Syy tälle on, että PLM-Zonen tulevat kehitysprojektit ovat entistä enemmän monimutkaisempia eikä yrityksellä ollut vakiintunutta menetelmää testaukselle. Testihallintatyökalun avulla organisaatiot voivat parantaa testausta ja varmistaa, että tuottavat laadullisia ohjelmistoja.

Tämä edellytti, että ensin kerään PLM-Zonen vaatimukset työkalulle. Tämän jälkeen vaatimuksien pohjalta kerään markkinoilta kolme testihallintatyökalua. Vertailen näitä kolmea testihallintatyökalua ja valitsen yhden organisaatiolle. Viimeiseksi koulutan PLM-Zonen työntekijöitä käyttämään työkalua ja luon testiprosessin, mikä soveltuu työkaluun.

Saavuttaakseni tavoitteeni loin kyselyn henkilökunnalle. Kyselyn tavoitteen oli kerätä vaatimuksia työkalulle. Kyselyn vastaamisen jälkeen järjestin henkilökohtaiset haastattelut, joissa kävin läpi osallistujan vastaukset ja keräsin lisävaatimuksia. Työkalujen vertailua varten kehitin pistejärjestelmän, missä kävin läpi, kuinka hyvin työkalu täyttää vaatimukset. Tämä mahdollisti, että tunnistin kyseisen työkalun heikkoudet ja vahvuudet. Viimeiseksi järjestin demotilaisuuden, missä esitin valitsemani testihallintyökalut. Lopullinen päätös tehtiin yhdessä sidosryhmien kanssa.

Vaatimuksien tuloksessa tuli ilmi, että tärkeimmät vaatimukset työkalulle pois lukien tyypilliset testihallintatyökalun ominaisuudet kuten testien hallinta olivat Jira-Integraatio ja helppokäyttöisyys. Valitsin vertailua varten Xray-, Zephyr- ja AIO Tests- testihallintatyökalut. Demotilaisuudessa päätimme, että valitsemme Xray-testihallintatyökalun. Xray täytti asetetut vaatimukset, ja meidän mielestämme se oli kaikista yksinkertaisin käyttöliittymä. Samalla työkalu oli hinnaltaan edullisempi, kuin Zephyr-testihallintatyökalu.

Koulutuksessa ja implementoinnissa loin käyttöohjeen työkalun käytölle ja konfiguroinnille. Samalla loin testiprosessin, mikä soveltuu Xray-työkaluun.

Avainsanat: Xray , AIO Tests, Zephyr

Abstract

Author:	Elyas Addawe
Title:	Test Management Tool For PLM-Zone
Number of Pages:	34 pages
Date:	8.5 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Jorma Räty, Senior Lecturer

The goal for the study was to select a suitable test management tool for PLM-Zone. The reason for this is that PLM-Zone's upcoming developments projects are more complex, and the company does not have a specific tool nor a standardized test process. With a test management tool, one can improve the testing efforts and deliver higher quality software.

First the requirements for the tools were gathered. Then based on the requirements three test management tools were selected. Next the selected test management tools were compared with each other and then the best suited one was selected. The next step would be to train the staff to use the selected tool.

To achieve this, a survey for stakeholders was created. The goal of the survey was to gather requirements for the tool. After the survey there were 1 on 1 interviews with participants to go through the survey answers and to gain additional insight. For the tool comparison a point scoring system was created. It was used to see how well the tool fulfilled the set requirements and to see the strengths and weaknesses of the tool. Finally, a demo event with the stakeholders was held to convey the strengths and weaknesses of each tool and finally to decide on the tool.

In the requirement gathering phase Jira integration and user-friendliness were highlighted above all else excluding the expected test management tool features such as test management. Based on the requirements Xray, Zephyr and AIO Tests were selected. In the demo event it was decided to select Xray for Jira. Xray fulfilled each requirement and was more affordable than Zephyr. Xray also had the best UI and was more user-friendly.

In the implementation and training stage a manual for configuration was created as well as a second manual that outlines how to use the tool effectively. Furthermore, a test process that each tester can follow was created based on the tool selected.

Keywords: Xray, Zephyr, AIO Tests

Sisällys

Lyhenteet

JIRA: *Attlassian yrityksen kehittämä työkalu projektinhallintaan ja tehtävien seurantaan.*

BITBUCKET: *Attlassian yrityksen kehittämä työkalu versionhallintaan.*

NIST: *National Institute of Standards and Technology on yhdysvaltalainen valtionosasto.*

.

1 Johdanto

Tämän lopputyön tavoitteena on hankkia testihallintatyökalu yritykselle, jolla ei ollut aikaisemmin vakiintunutta testiprosessia tai työkalua. Lopputyössä tullaan ensin kartoittamaan vaatimukset työkalulle. Seuraavaksi verrataan testihallintatyökaluja keskenään kerättyjen vaatimuksien pohjalta. Samalla suunnitellaan lyhyt koulutusohjelma, missä käydään työkalun toiminnot, jolla varmistetaan, että työntekijöillä on tarvittavat taidot työkalun tehokkaaseen käyttöön. Lopputyö tulee myös lisäämään tietämystä ohjelmistotestauksesta ja testihallintatyökaluista.

2 Lähtökohdat

2.1 Tausta

Lopputyö tehtiin PLM-Zonelle. PLM-Zone on ohjelmistoyritys, joka luo ohjelmistoja rakennus- ja valmistusyrityksien puolelle. Yritys myös tarjoaa yrityskonsultointia ja sovelluskehitystä asiakkaille.

Yrityksessä ei ollut testihallintatyökalua, mikä johti sisäiseen projektiin testihallintatyökalun hankkimiselle. Tunnistin, että kyseinen projekti olisi kiinnostava lopputyön aihe. Näin ollen otin projektin omalle vastuulleni.

PLM-Zonella on omaa tuotekehitystä sekä myös muille asiakkaille tehtävää kehitystyötä. Kehitysprojektien sisältöä hallitaan Jira-ohjelmistossa ja koodi säilytetään Bitbucket-ohjelmistossa. Tällä hetkellä ei ole olemassa erityistä testinhallintatyökalua tai vakiintunutta testiprosessia.

PLM-Zonen tulevat sovellus- ja tuotekehitysprojektit ovat yhä monimutkaisempia, ja sovellukset sisältävät monimutkaisia ominaisuuksia. Tämä edellyttää systemaattista ja hyvin hallittua testiympäristöä.

2.2 Ongelma

Tämän lopputyön käsiteltävä ongelma on standardoidun testausprosessin ja työkalun puute yrityksessä. Ilman standardoitua testausprosessia testaajat voivat käyttää erilaisia menetelmiä ja työkaluja, mikä johtaa epäjohdonmukaisuuksiin testauksessa ja kasvattaa riskiä ohjelmistojen erilaisille virheille. Esimerkiksi testaus ei kata kaikkia ohjelmistosovelluksen ominaisuuksia ja toimintoja. Tämä voi johtaa ohjelmistokehityksen elinkaaren viivästymiseen ja ohjelmistotuotteiden laadun heikkenemiseen. Lisäksi standardoidun testausprosessin ja työkalun puuttuminen voi vaikuttaa negatiivisesti asiakkaisiin johtamalla ohjelmistovirheisiin, tietojen katoamiseen tai tietoturvaloukkauksiin. Asiakas voi myös joissain tilanteissa vaatia kattavan dokumentaation tuotteesta ja suorittaa testit itse. Tämä voi johtaa negatiivisen palautteen saamisen asiakkailta ja alan asiantuntijoilta, mikä lopulta heikentää yrityksen uskottavuutta. Siksi ongelmanratkaisuna on löytää sopiva testinhallintatyökalu ja laatia koulutusohjelma standardoidun testausprosessin luomiseksi. Tämän avulla voidaan varmistaa laadukkaat ohjelmistotuotteet ja tyydyttää asiakkaiden tarpeet.

2.3 Tavoitteet

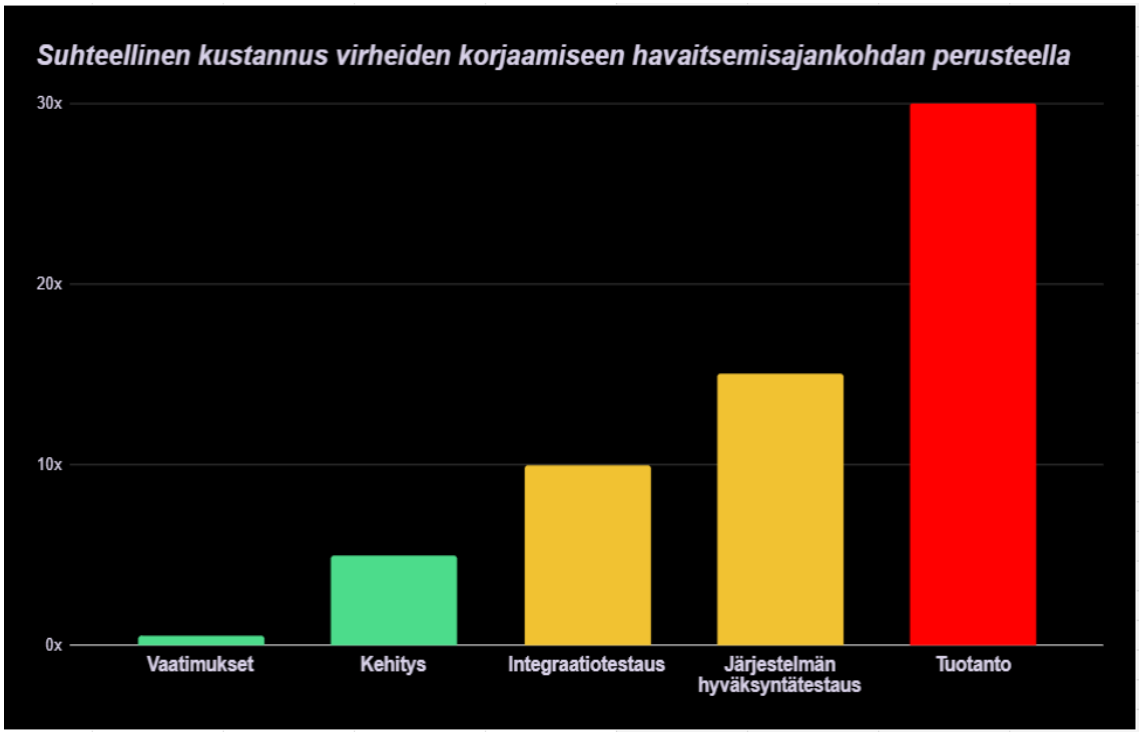
Tavoitteena on valita sopiva testihallintatyökalu yritykselle, jolla ei ollut aikaisemmin standardoitua testiprosessia tai testihallintatyökalua. Tämä edellyttää, että ensin tunnistetaan yrityksen vaatimukset testihallintatyökalulle, minkä jälkeen verrataan markkinoilla olevia testihallintatyökaluja keskenään kerättyjen vaatimuksien pohjalta. Viimeiseksi on kouluttaa työntekijät käyttämään valittua testihallintakalua ja luoda standardoitu testausprosessi.

3 Teknologiat

3.1 Ohjelmistotestaus

Ohjelmistotestaus on prosessi, jossa arvioidaan ja varmistetaan, että ohjelmistosovellus täyttää asetetut vaatimukset ja toimii odotetulla tavalla. Testauksella pyritään tunnistamaan mahdollisia vikoja, virheitä ja heikkouksia ohjelmistosovelluksessa. Testauksen merkittävyys on noussut viime vuosina, sillä ohjelmistot ovat entistä enemmän monimutkaisempia. Ohjelmistojen pitää muun muassa toimia virheettömästi useassa eri ympäristössä samalla, kun ne täyttävät käyttäjien tarpeet. Testauksen tavoitteena on havaita ja korjata kohdatut viat, virheet ja heikkoudet ohjelmistoelinkaaren varhaisessa vaiheessa, ennen kuin sovellus päättyy asiakkaiden käsiin. [1.]

Ohjelmistotestauksen laiminlyönti voi johtaa bugeihin, kaatumisiin ja virheisiin. Tämä tulee olemaan haitallinen ohjelmiston käyttäjille ja tämä tulee lopulta vahingoittamaan yrityksen mainetta. Kattavan testauksen laiminlyönti voi myös aiheuttaa taloudellisia tappioita yritykselle. Esimerkiksi Reutersin vuoden 2012 raportin mukaan finanssiyhtiö Knight Capital Group menetti noin 440 miljoonaa dollaria vain 45 minuutissa puutteellisen testauksen aiheuttaman ohjelmistohäiriön vuoksi. [2.]



Kuva 1. Suhteellinen kustannus virheiden korjaamiseen havaitsemisajankohdan perusteella. [3.]

Kuva 1 havainnollistaa, kuinka vikojen korjaamisen hinta nousee, mitä lähemmäksi mennään tuotantoa. Paras aika testaamiselle on NISTin tekemän tutkimuksen mukaan jo heti vaatimuksen määrittelyn vaiheessa. Löytämällä ja korjaamalla ongelmia varhaisessa vaiheessa ohjelmistokehittäjät voivat säästää aikaa ja rahaa ja estää potentiaalista vahinkoa maineelleen ja asiakassuhteilleen. [3.]

3.2 Erilaiset testaustyypit

Ohjelmistotestaukseen kuuluu monia eri osa-alueita ja tasoja. Testauksessa ei ole olemassa yhtä ja ainoaa standardoitua mallia. Kaikilla on omat mielipiteensä ja omat lajittelut. Voidaan silti sanoa, että ohjelmistotestaus jakautuu kahteen päähaaraan. Nämä ovat toiminnallinen ja ei-toiminnallinen testaus. Nämä kaksi ovat ohjelmistotestauksen perustyyppejä, joiden avulla varmistetaan, että ohjelmisto täyttää käyttäjän vaatimukset ja toimii odotetulla tavalla.

Toiminnallinen testaus on testausmuoto, missä varmistetaan, toimivatko ohjelmistosovelluksen toiminnot vaatimusten mukaisesti. Se tarkistaa, suorittaako ohjelmistosovellus sille asetetut toiminnot, kuten esimerkiksi tietojen käsittelyn, kirjautumisen tai laskelmien suorittamisen käyttäjän odotusten mukaisesti. Toiminnallinen testaus on todella tärkeää, sillä se varmistaa, että ohjelmistosovellus vastaa käyttäjän tarpeita ja toimii odotetulla tavalla.

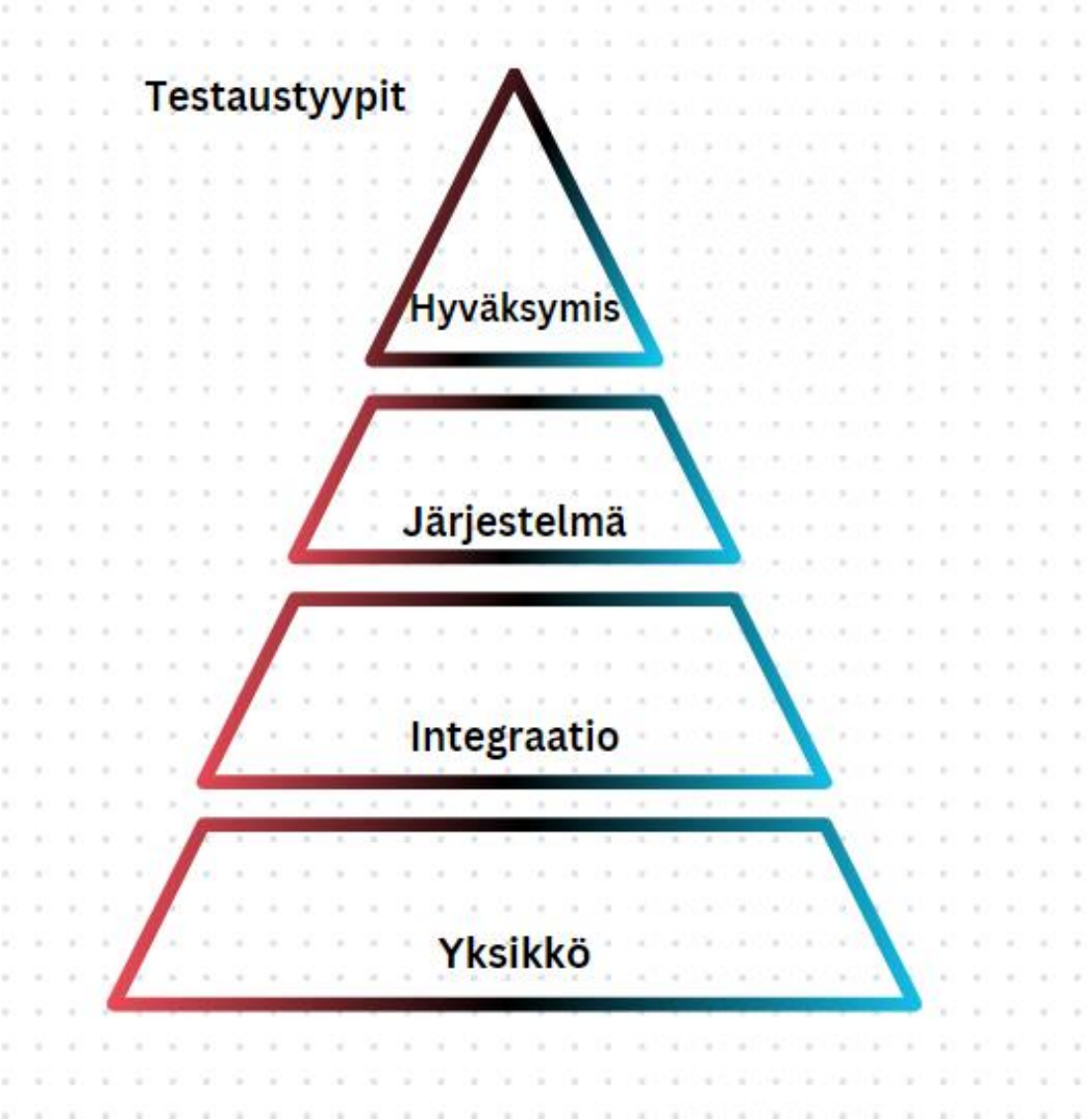
Ei-toiminnallinen testaus taas auttaa meitä varmistamaan, että ohjelmistosovellus on tehokas, luotettava ja käyttäjäystävällinen. Se auttaa tunnistamaan ja ratkaisemaan kaikki ei-toiminnalliset ongelmat, jotka voivat vaikuttaa käyttäjän kokemukseen. Esimerkki ei-toiminallisesta testauksesta on seurata, miten kauan kestää yhden sivun latautuminen käyttäjän näkökulmasta.

Ei-toiminnallinen testaus sisältää erilaisia testejä, joita ovat suorituskykytestaus, tietoturvatestaus ja käytettävyystestaus. Jokainen ei-toiminnallinen testaustyyppi arvioi ohjelmistosovelluksen tietyn puolen ja auttaa varmistamaan, että se täyttää käyttäjän ei-toiminnalliset vaatimukset. [4.]

Nyt kun tiedämme toiminnallisen ja ei-toiminnallisen testauksen merkitykset, perehdytään enemmän toiminnalliseen testaukseen. Toiminnalliseen testaukseen kuuluu useita eri testityyppejä, joita ovat esim.

- yksikkötestaus
- integraatiotestaus
- järjestelmätestaus
- hyväksymistestaus.

Nämä testit suoritetaan kehitysprosessin eri vaiheissa. Alla olevassa kuvassa 2 havainnollistetaan testit suoritusjärjestyksessä.



Kuva 2. Toiminnallisen testauksen pyramidi

Kuva 2 havainnollistaa testauksen eri tasoja. Alhaalla on yksikkötestaus, mikä esiintyy heti kehitysvaiheen alussa, ja ylimpänä hyväkysymistestaus, joka tapahtuu kehitysvaiheen lopussa.

Yksikkötestauksessa testataan ohjelmiston yksikköä eristyksessä muista koodista. Yksikkö taas kuvaa ohjelmiston pienintä palasta, mikä voi olla esimerkiksi luokan metodi. Yksikkötestauksen tavoitteena on tunnistaa ohjelmistovirheet mahdollisimman varhaisessa vaiheessa. Käyttämällä yksikkötestausta voidaan taata, että jokainen moduuli eristyksessä toimii tarkoituksenmukaisesti. Yksikkötestauksen tekee itse kehittäjä.

```
@Test
public void unit_test_esimerkki() {
//luodaan olio ja asetetaan hinta

Laiteosa muisti=new MuistiPiiri(10);

          assertEquals(10.0, muisti.getHinta(),"Ei toimi halutulla ta-
valla");

}
```

Esimerkkikoodi 1. Esimerkki yksikkötestauksesta.

Esimerkkikoodissa 1 on esimerkki yksikkötestauksesta. Testissä on MuistiPiiri-niminen luokka, missä asetetaan hinta olion luonnin yhteydessä. Tässä testataan, palauttaako getHinta()-metodi oikean luvun. Ensin asetetaan oikea vastaus, mikä tässä tilanteessa on 10, jonka jälkeen verrataan lukua, getHinta()-metodin palauttamaan vastaukseen. Kun testiä ajetaan, niin saadaan saman tien palaute testin onnistumisesta tai epäonnistumisesta.

Integraatio-testaus on testaustyyppi, jossa testataan useamman moduulin toimintaa yhdessä ja varmistetaan, että ne toimivat odotetulla tavalla. Integraatiotestaus ikään kuin testaa useita yksikköjä kerralla. Integraatiotestauksen tekee itse kehittäjä.

```
@Test
     public void integration_test_esimerkki() {
          //luodaan olio ja asetetaan hinta
          Laiteosa muisti=new MuistiPiiri(10);
          Laiteosa prosessori=new Prosessori(20);

          //lisätään osat emolevyyn
          Emolevy emolevy = new Emolevy(100);
          emolevy.addOsa(prosessori);
          emolevy.addOsa(muisti);

        assertEquals(130, emolevy.getHinta(),"emolevy ei palauta oi-
keaa hintaa");
     }
```

Esimerkkikoodi 2. Esimerkki integraatiotestauksesta.

Esimerkkikoodissa 2 on esimerkki integraatiotestauksesta. Koodissa ensin luodaan oliot kuten prosessori ja muistipiiri ja alustetaan hinnat. Sen jälkeen luodaan emolevyolio ja asetetaan emolevylle oma hinta, minkä jälkeen lisätään osat. Viimeiseksi varmistetaan, palauttaako emolevy, joka käsittelee useita osia, oikean hinnan.

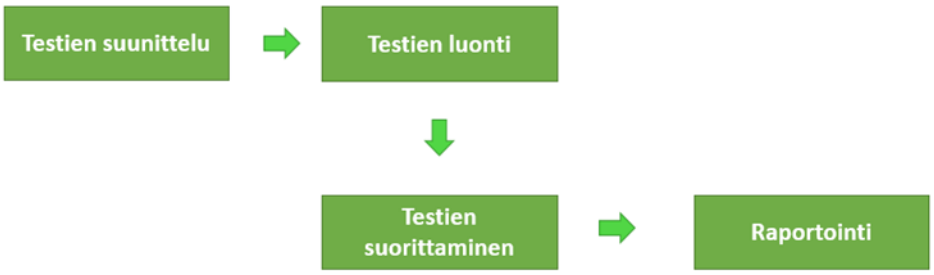
Järjestelmätestaus on testaustyyppi, joka arvioi koko järjestelmän tai sovelluksen kokonaisuutena. Järjestelmätestauksen tavoitteena on varmistaa, että järjestelmä täyttää asetetut vaatimukset ja toimii oikeassa tilanteessa. Järjestelmätestaus tehdään kehitysvaiheen lopussa, missä kehitetty sovellus on valmis tai lähes valmis. Järjestelmätestausta voivat suorittaa kehittäjät itse tai testausryhmä. Esimerkkinä voivat olla kehittäjät, jotka ovat rakentaneet ravintolasovelluksen. Testaajat sen jälkeen testaavat, täyttääkö sovellus asetetut vaatimukset kuten esimerkiksi tilaus, maksun suorittaminen ja kirjautuminen odotetulla tavalla.

Hyväksymistestaus on testaustyyppi, joka arvioi, täyttääkö sovellus sidosryhmien asettamat vaatimukset. Tämä sisällyttää, toimiiko järjestelmä odotetulla tavalla ja vastaako sovellus asiakkaan tarpeita. Testaaminen suoritetaan kehitysvaiheen lopussa. Testaamisen suorittaa itse loppuasiakas. Testin voi myös suorittaa loppuasiakkaan edustaja, jolla on ymmärrystä asiakkaiden vaatimuksista. [5.]

Edellä mainittujen testien avulla voidaan varmistaa, että ohjelmistosovellus on laadukas. Saavuttaakseen laadukkaan ohjelmiston on tärkeää myös määritellä testiprosessi, joka kuvaa testauksen vaiheet ja toiminnot selkeästi. Testiprosessiin kuuluvat testien suunnittelu, testien luonti, testien suorittaminen ja raportointi.

3.3 Testiprosessi

Ohjelmistotestaus on monimutkainen prosessi, mikä vaatii selkeästi määritellyt askeleet. Ohjelmistotestaukseen tyypillisesti kuuluu useita vaiheita kuten testien suunnittelu, testien luonti, testien suorittaminen ja raportointi.



Kuva 3 Esimerkki testiprosessista

Kuva 3 havainnollistaa testiprosessin ja järjestyksen. Ensimmäinen askel testiprosessissa on itse testien suunnittelu. Tässä vaiheessa asetetaan testaukselle tavoitteet. Määritellään testausmenetelmä, esimerkiksi, onko kyseessä järjestelmätestaus. Samalla määritellään mahdolliset testitapaukset. Tähän vaiheeseen voi osallistua useita toimijoita kuten kehittäjät, testaajat ja muut sidosryhmät.

Suunnittelun jälkeen luodaan itse testitapaukset. Testien luontivaiheessa otetaan suunnitteluvaiheessa tunnistetut testitapaukset ja luodaan selkeät askeleet testitapauksille, jotta ne olisivat toistettavassa. Luodut testitapaukset kattavat ohjelmiston eri osapuolet. Testitapauksessa määritellään selkeästi, mitä testataan. Testitapauksilla varmistetaan, että ohjelmistosovellus toimii odotetutulla tavalla.

Testien suorittamisvaiheessa juoksutetaan luodut testitapaukset. Tähän vaiheeseen kuuluu myös, että dokumentoidaan testitapauksien testitulokset ja ilmoitetaan syntyneistä virheistä. On hyvä tiedostaa, että testejä ei ole aina pakko suorittaa manuaalisesti, jos kyseessä on toistuva testitapaus kuten esimerkiksi kirjautuminen, tällöin voidaan testien suorittamista huomattavasti nopeuttaa luomalla automatisoitu skripti, mikä suorittaa kyseisen testin testaajan puolesta.

Viimeinen askel testiprosessia on raportointi. Tässä vaiheessa luodaan raportteja testauksesta ja nämä raportit välitetään sidosryhmille. Raportoinnin avulla sidosryhmät ja itse testaajat saavat yleiskuvan testauksesta ja siitä, mitä asioita voidaan kehittää jatkossa.

Olen esittänyt yksinkertaisen testiprosessin, mikä kattaa testauksen tärkeimmät vaiheet. On hyvä tiedostaa, että ei ole yhtä standardimallia eikä esittämäni testiprosessi sovellu kaikkiin tilanteisiin. Testiprosessi voi vaihdella riippuen eri tekijöistä kuten organisaation rakenteesta, ohjelmiston monimutkaisuudesta, ohjelmistokehitysmenetelmästä ja vaatimusten mukaan.

Testiprosessiin vahvasti vaikuttaa kehityksessä käytettävä menetelmä. Ohjelmistokehitysmenetelmällä tarkoitetaan prosesseja ja käytäntöjä, joita ohjelmistokehittäjät käyttävät projektinhallintaan. Kehitysmenetelmä tarjoaa selkeän kehyksen ohjelmistokehitykselle, jakaa rooleihin ja kuvaa selkeät vaiheet. Yleisiä ohjelmistokehitysmenetelmiä ovat ketterät menetelmät ja vesiputousmalli.

Viime vuosina hurjaan suosioon noussut ohjelmistokehitys menetelmä on ketterät menetelmät. Ketterässä menetelmässä ohjelmistokehitystä lähestytään toistuvasti ja inkrementaalisesti. Tämä tarkoittaa, että kehitysprosessi jaetaan pienempiin ja hallittavampiin sykleihin tai sprintteihin, jotka keskittyvät toimivan ohjelmiston tuottamiseen. Tämä korostaa yhteistyötä, joustavuutta ja pystytään helpommin vastaamaan asiakkaan muuttuviin vaatimuksiin. Esimerkiksi olemme luomassa verkkokauppaa, niin voidaan jakaa kyseinen projekti useisiin sprintteihin. Ensimmäinen sprintti voi olla sivun rakenne ja navigaatio. Toinen voi olla ostoskorin toiminnallisuus. Jokaisen sprintin jälkeen saamme palautetta

asiakkailta ja muilta sidosryhmiltä ja pystytään helposti vastaamaan asiakkaan tarpeisiin.

Tämä tarkoittaa, että testaus tulee olemaan integroitu kehitysprosessiin. Testaamista tehdään jatkuvasti, ja se tulee ilmenemään jokaisessa sprintissä. Tämä myös edellyttää joustavuutta meidän esittämästämme testiprosessista, sillä testisuunnitelma tulee muuttumaan kehityksen aikana ja testitapauksia luodaan jatkuvasti. [6;7;8.]

3.4 Testihallintatyökalut

Testihallintatyökalut ovat ohjelmistoja, jotka on luotu helpottamaan testiprosessia. Ajatellaan esimerkki tilannetta, missä ei käytetä testihallintatyökalua. Ensin luodaan testisuunnitelma käyttäen jonkinlaista asiakirjadokumenttia. Sen jälkeen luodaan testitapauksia manuaalisesti. Sitten kopiomme luodut testitapaukset uudestaan Exceliin. Testien suorittaminen tapahtuu taas Excel-tyyppisessä tiedostossa. Viimeiseksi käytämme Excelin raportointiominaisuuksia.

Taulukko 1 Esimerkki testiprosessista

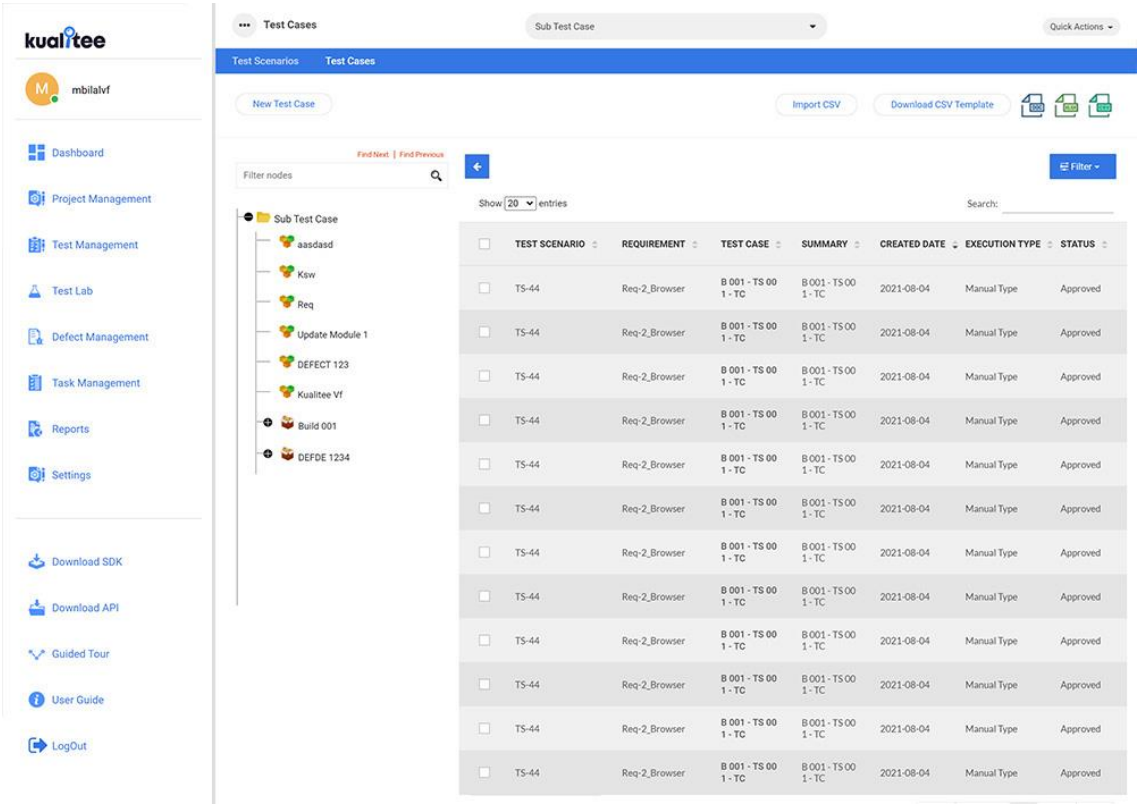
ID	**TESTITA-PAUS**	**KUVAUS**	**ODOTETTU VASTAUS**	**SAATU VAS-TAUS**	**SAATU VASTAUS (MOBIILI)**	**Lisätieto**
1	Kirjautumi-nen	Kirjaudu Systee-miin	Etusivu nä-kyy	14.2.2023 Toimii	15.2.2023 Toimii	
2	Kirjaudu ulos	Kirjaudu ulos ap-plikaatiosta	Login sivu näkyy	14.2.2023 Toimii	15.2.2023 Toimii	
3	Luo Käyt-täjä	Rekisteröidy	Vie sinut lo-gin sivulle	eii toimi	ei toimi	
4	Luo Sopi-mus	Etusivulta klikkaa "luo sopimus pai-niketta"	Sopimus il-mestyy etu-sivulle	14.2.2023 ei toimi	15.2.2023 ei toimi	

Taulukko 1 kuvaa tilannetta, missä olemme käsin luoneet Exceliin testitapauksia. Sanotaan, että haluamme nähdä, mitkä testitapaukset ovat epäonnistuneet, niin joudumme käyttämään Excelin filtteritoimintoa ja hakea ”ei toimi” -tekstillä epäonnistuneet testitapaukset. Mutta kuten näemme ID 4:n testitapauksesta, niin pelkkä kirjoitusvirhe voi tehdä epäonnistuneiden testien keräämisestä mahdottoman. Ajatellaan toista tilannetta, missä esimerkiksi haluamme suorittaa samat testitapaukset eri ympäristössä tai eri ajankohtana. Tämä edellyttää, että kopioimme testitapaukset uudestaan manuaalisesti. Kuten huomaamme, olemme ikään kuin testaajan armoilla. Pieni virhe voi tehdä raportoinnista mahdottoman.

Testiprosessi on aina erilainen ja tapauskohtainen riippuen projektista. Samoin on hankalaa saada yleiskuvaa testien etenemisestä emmekä voi siististi uudelleen käyttää testitapauksia.

Testihallintatyökalun avulla koko prosessi digitalisoituu. Nämä työkalut auttavat meitä testien suunnittelussa, luomisessa, suorittamisessa, virheiden käsittelyssä ja raportoinnissa. Testihallintatyökalun avulla voi myös linkittää

vaatimukset itse testitapauksiin. Testihallintatyökalut saattavat myös tarjota integraation muihin sovelluksiin kuten esimerkiksi Jira-ohjelmaan. Viimeiseksi testihallintatyökalut tarjoavat keskitetyn sovelluksen, missä tiimi voi hallita testiprosessia.



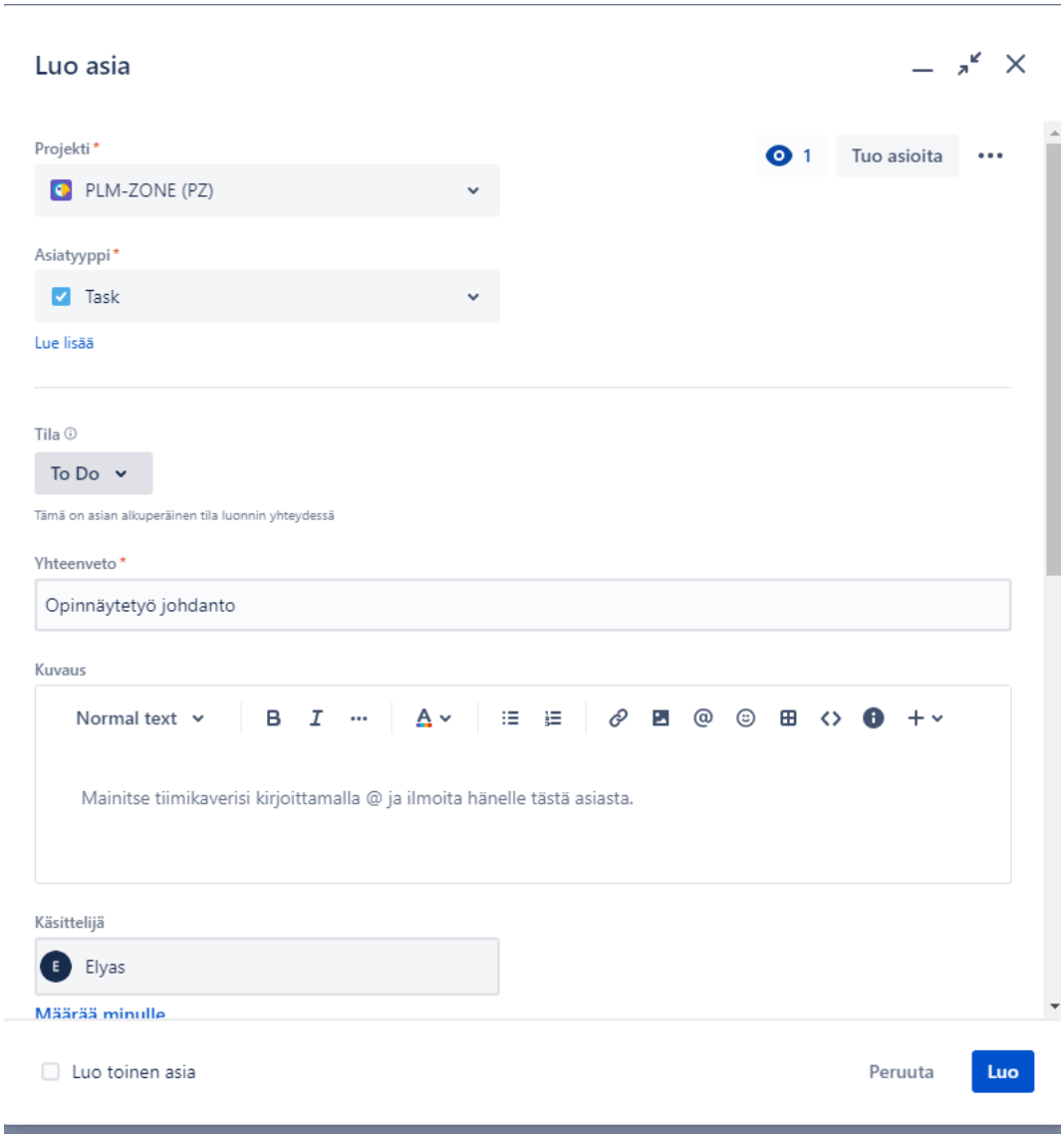
Kuva 4. Kualitee-testihallintatyökalun käyttöliittymä.

Kuva 4 näyttää Kualitee-testihallintatyökalun käyttöliittymän. Kuvassa näkyvät luodut testitapaukset ja niiden tila.

Käyttämällä testihallintatyökalua voidaan varmistaa, että testausta on dokumentoitu oikealla tavalla. Samoin sidosryhmät saavat reaaliajassa selkeän yleiskuvan testauksen etenemisestä ja tuloksista. Testihallintatyökalun avulla tiimit säästävät aikaa, vähentävät virheitä ja parantavat ohjelmiston laatua. [9.]

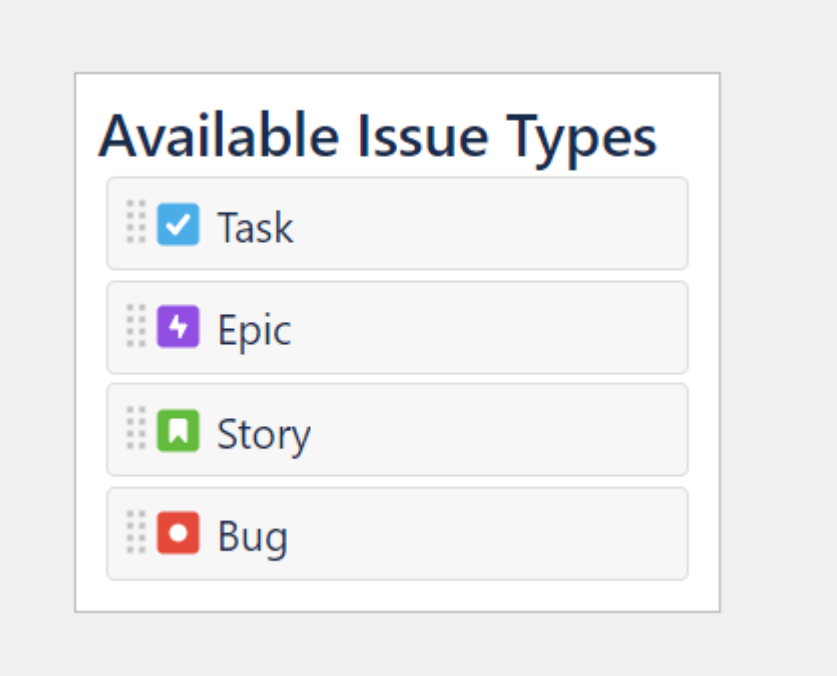
3.5 Jira

Työpaikalla käytämme Jira-ohjelmistoa. Jira on suosittu projektihallintatyökalu ja tehtävienhallintatyökalu. Sen on alun perin kehittänyt Atlassian, ja nyt sitä käytetään laajasti ohjelmistokehityksessä. Jira on kehitetty auttamaan tiimejä suunnittelemaan, seuraamaan ja hallitsemaan projekteja.



Kuva 5. Esimerkki tehtävän luonnista.

Jira-ohjelman tehtävänhallinnalla tiimit voivat asettaa tehtäviä ja seurata edistymistä projektin edetessä. Kuva 5 havainnollistaa, kuinka tehtävien luonti onnistuu Jira-ohjelmistossa. Kuvassa luodaan task-tehtävätyyppi PLM-ZONE-projektille ja asetetaan tehtävälle vastuuhenkilö, joka kuvassa on Elyas. Jira tarjoaa oletuksena task-, story bug-, epic-tehtävätyypit.



Kuva 6. Jiran oletustehtävätyypit.

Kuva 6 esittää oletustehtävätyypit Jira-ohjelmassa. Task kuvaa yksittäistä tehtävää projektissa. Story-tehtävätyyppi kuvaa tiettyä toiminnallisuutta asiakkaan näkökulmasta. Epic-tehtävätyyppi käyttötarkoitus on yleensä toimia säiliönä usealle Task-ja Story-tehtävätyypeille, joita yhdistää jokin asia.

Kuva 7. Projektin luonti ja pohjan valitseminen Jira-ohjelmistossa.

Jira tukee erilaisia kehitysmenetelmiä kuten ketterät menetelmät. Jira tarjoaa erilaisia projektipohjia, mitä voi vapaasti valita muun muassa Kanban. Kuva 7 havainnollistaa pohjan valitsemisen.

Mikä tekee Jira-ohjelmasta suositun, on sen joustavuus. Jira tukee useita projekteja, ja samalla ne ovat muokattavissa omien halujen mukaan. Voi muun muassa muokata tehtävätyypin oletettua työnkulkua, mikä on Jira-ohjelmistossa To Do, In Progress ja Done. Samoin voi luoda omia tehtävätyyppejä ja muokata olemassa olevia. Jira-ohjelma myös tarjoaa helpon integraation muihin sovelluksiin kuten esimerkiksi testihallintatyökaluihin. [10.]

4 Työn toteutus

4.1 Johdanto

Tutkimuksen tavoitteena on hankkia sopiva testihallintatyökalu yritykselle, jolla ei ole aikaisempaa testihallintatyökalua tai vakiintunutta prosessia kerättyjen vaatimuksien pohjalta. Tämä edellyttää, että ensin tunnistetaan vaatimukset, arvioidaan vaatimukset, kerätään testihallintatyökaluja vaatimuksen pohjalta, vertaillaan testihallintatyökaluja keskenään ja lopuksi implementoidaan valittu työkalu. Tutkimuksessa käyn läpi mainitsemani vaiheet, ja lukija voi käyttää kyseistä mallia samanlaisen ongelman ratkaisemiseen.

4.2 Vaatimusten kerääminen

Ensimmäinen vaihe on vaatimuksen kerääminen. Vaatimuksen keräämisellä varmistamme, että valittu testihallintatyökalu vastaa yrityksen tarpeita. Markkinoilla on satoja testihallintatyökaluja, ja asettamalla vaatimukset työkalulle voidaan rajoittaa lukua huomattavasti. Lähestyin vaatimusten keräämistä kahdella tavalla. Loin kyselyn ja laadin henkilökohtaisen haastattelun kaikille osallistujille.

Kyselyn tavoitteena oli kerätä vaatimuksia testihallintatyökalulle ja saada yleiskuva nykyisestä testauksesta. Huomasin tässä vaiheessa, että osallistujien tuntemus testihallintatyökaluista erosi, sillä monella osallistujilla ei ollut kokemusta testihallintatyökaluista ennen ja näin ollen ei voinut asettaa suoranaisesti vaatimuksia työkalulle. Näin ollen kyselyssä otettiin huomioon molemmat tapaukset, missä osallistuja tarkasti tietää, mitä testihallintatyökalun pitää täyttää, ja toinen tapaus, missä ei suoranaisesti tiedetä. Asetin kyselykielen englanniksi, sillä yrityksessä toimi englanninkielisiä henkilöitä. Kyselyn laatimisen jälkeen lähetin kyselyn yrityksen sidosryhmille, johon kuuluivat muun muassa kehittäjät, testaajat ja toimitusjohtajat. Näin ollen varmistin, että kaikkien näkökulma tulee esille. Esimerkiksi testaajalle ei ole ehkä tärkeitä raportointiominaisuudet, mutta se on kriittinen johdolle saada raportteja testauksesta. Tämän jälkeen analysoin kyselyn vastaukset ja listasin Exceliin alustavat vaatimukset.

Kyselyn luomisessa käytin Microsoft Forms -ohjelmaa. Sen avulla oli helppo välittää kysely osallistujille ja saada vastaukset reaaliajassa takaisin. Loin kyselyyn viisi kysymystä. Kyselyssä ensin kysytään nykytilanteesta ja potentiaalisista kipukohdista, minkä jälkeen siirrytään testihallintatyökalujen toimintoihin.

Kun kyselyyn oli vastattu, järjestin vastaajille 30-45 minuutin yksityishaastattelun. Henkilökohtaisen haastattelun tavoitteena oli saada lisätietoja osallistujan kysely vastauksista ja selventää tarvittaessa. Haastattelussa näytin myös Exceliin keräämäni vaatimukset ja lisäilin myös haastattelussa kerätyt vaatimukset.

Vaatimusten keräämisen jälkeen järjestin yhteisen tapaamisen sidosryhmien kanssa missä esitin keräämäni vaatimukset. Tavoitteena oli käydä läpi vaatimuksia yhdessä ja tarvittaessa karsia ja lisäillä vaatimuksia. Samalla tunnistetaan, mitkä vaatimukset ovat kaikista tärkeimpiä.

4.3 Testihallintatyökalujen vertailu

Seuraava vaihe oli kerätä testihallintatyökaluja ja vertailla niitä vaatimuksien pohjalta. Ensimmäiseksi keräsin testihallintatyökaluja kysymällä PLM-Zone asiantuntijoita, mitä työkaluja he ovat käyttäneet ennen ja, mitkä soveltuisivat nykyiseen tilanteeseen. Seuraavaksi hain itsenäisesti kaksi muuta testihallintatyökalua, joilla on korkeat arvostelut ja jotka täyttävät tärkeimmät vaatimukset.

Testihallintatyökalun arvioinnissa loin pistejärjestelmän Exceliin, jolla arvioin, kuinka hyvin kyseinen työkalu täyttää asetetut vaatimukset. Pistejärjestelmän asteikko oli 1-5, jossa 1 tarkoittaa, että ei täytä vaatimusta, ja 5 tarkoittaa, että täyttää vaatimukset täysin. Tämä mahdollisti sen, että kävin läpi kaikki vaatimukset työkalulle ja tunnistin kyseisen työkalun vahvuudet ja heikkoudet. Tulen esittämään pistejärjestelmätaulukon tulokset luvussa.

Tämän jälkeen järjestin demotilaisuuden sidosryhmille, missä kävin läpi testihallintatyökalun toiminnot ja esitin työkalun heikkoudet ja vahvuudet. Tilaisuuteen osallistuivat sidosryhmät. Demotilaisuuden päättyessä päätimme keskenämme, minkä testihallintatyökalun valitsemme.

4.4 Koulutus ja implementointi

Viimeinen vaihe on koulutus. Koulutusvaiheessa loin manuaalin käyttämällä PowerPoint -ohjelmistoa. Loin kaksi erillistä manuaalia, Yksi manuaali on työkalun konfiguraatio. Toinen manuaali taas sisältää ohjeet työkalun käytölle. Manuaali on suunniteltu, että se olisi helposti seurattavissa ja ymmärrettävä. Valitsin Power Point -ohjelman, sillä ideana on näyttää työkalun käyttöliittymä.

5 Tulokset

5.1 Vaatimuksien tulokset

Kuten mainitsisin työn toteutusosioissa, vaatimuksen keräämiseen loin kyselyn, järjestin haastattelut ja viimeiseksi pidin yhteisen tapaamisen, missä valittiin tärkeimmät vaatimukset. Vaatimukset ovat:

- Jira-Integraatio
- budjettia säästävä
- helppokäyttöinen
- testien hallinta
- raportointi
- automaatio testaus
- testi data ominaisuus
- vaatimuksien ja testien linkitys
- joustavuus.

On hyvä huomioida, että jätin pois listasta salassa pidettäviä vaatimuksia. Samalla listaamani vaatimukset ovat lopulliset vaatimukset työkalulle. Seuraavaksi käyn läpi kerätyt vaatimukset ja selitän, mitä kukin tarkoittaa meidän näkökulmastamme.

Jira-integraatiolla tarkoitan, että valitsemanne työkalu toimisi moitteettomasti Jira-ohjelmassa. Tavoitteena on, että työkalu ei olisi erillinen sovellus vaan olisi ikään kuin liitännäinen Jira-ohjelmalle. Eli ei haluta sellaista tilannetta, missä työkalu olisi erillinen sovellus. Syynä tälle vaatimukselle on se, että käytämme työpaikalla projektihallintatyökaluna Jira-ohjelmaa. Sidosryhmien tapaamisessa kyseinen vaatimus valittiin kaikista tärkeimmäksi eli valitun työkalun pitää soveltua Jira-ohjelmaan.

Seuraava vaatimus on, että valitsemanne testihallintatyökalu olisi mahdollisimman edullinen ja silti täyttäisi vaaditut toiminnot. Samalla työkalun pitää olla helppokäyttöinen. Tämä tarkoittaa, että olisi mahdollisimman yksinkertainen käyttöliittymä.

Testien hallinnalla tarkoitan, että työkalu sisältää kaikki tyypilliset toiminnallisuudet testihallintatyökaluissa, joita ovat testien luominen, organisointi, jäljittäminen, suorittaminen ja virheiden käsittely.

Työkalulla pitää olla raportointiominaisuudet. Tämä on tärkeä vaatimus sidosryhmille, sillä tämä antaa näkyvyyttä. Esimerkiksi on oleellista tuntea, kuinka paljon sovellusta on testattu. Eli valitsemamme työkalu pitää vähintään sisältää testien kattavuusraportin.

Työkalun pitää myös tukea automaatiota. Ideana on hyödyntää skriptejä, mitkä suorittavat luodut testitapaukset. Työkalun pitää integroida erilaisten automaatiotyökalujen kanssa.

Testidataominaisuudella tarkoitetaan kykyä liittää dataa itse testeihin. Tämä voi olla esimerkiksi yksinkertainen kuvakaappaus tai tiedosto. Esimerkiksi jotkut testit vaativat tiettyjä tiedostoja tai muuten haluamme vain liittää kuvia.

Työkalun pitää mahdollistaa linkitys testien ja vaatimusten välillä. Tämä käytännössä tarkoittaisi esimerkiksi Jira-ohjelman Story- tehtävätyypin linkitystä työkalun luomaan testiin. Tämä mahdollistaisi, että pystymme seuramaan, mitkä vaatimukset on testattu ja mitkä ei.

Viimeiseksi työkalun pitää olla joustava. Tämä voi olla esimerkiksi kyky konfiguroida työkalu halutulla tavalla.

Mainitsemani vaatimukset tulevat olemaan perusta testihallintatyökalujen keräämiselle ja vertailulle. Vertaamalla testihallintatyökaluja vaatimusten perusteella voidaan varmistaa, että valittu tuote tulee vastaamaan yrityksen tarpeita ja auttaa meitä testien hallinnassa. Tulen esittämään työkalujen vertailua vaatimusten pohjalta myöhemmin.

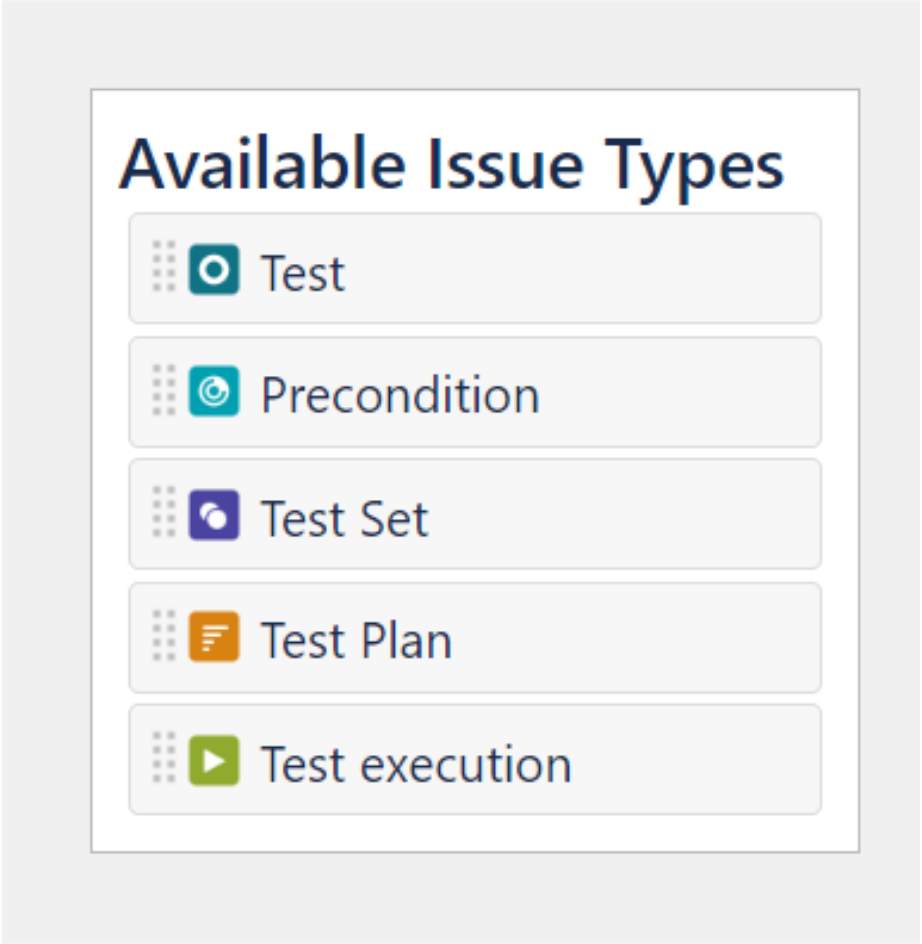
5.2 Valitut testihallintatyökalut

Keräsin 3 testihallintatyökaluja, jotka täyttävät asetetut vaatimukset, jotka mainitsin edellisessä luvussa. Tärkein vaatimus oli Jira-integraatio, joten työkalujen keräämisessä käytin Jira Marketplace -sivua. Jira Marketplace on verkkokauppa, jonka on kehittänyt Atlassian, sama yritys, mikä on luonut Jira-ohjelman. Kyseinen alusta tarjoaa mahdollisuuden löytää liitännäisiä Jira-ohjelmalle, johon sisältyy myös testihallintatyökaluja.

Loppusuoralle pääsivät seuraavat työkalut:

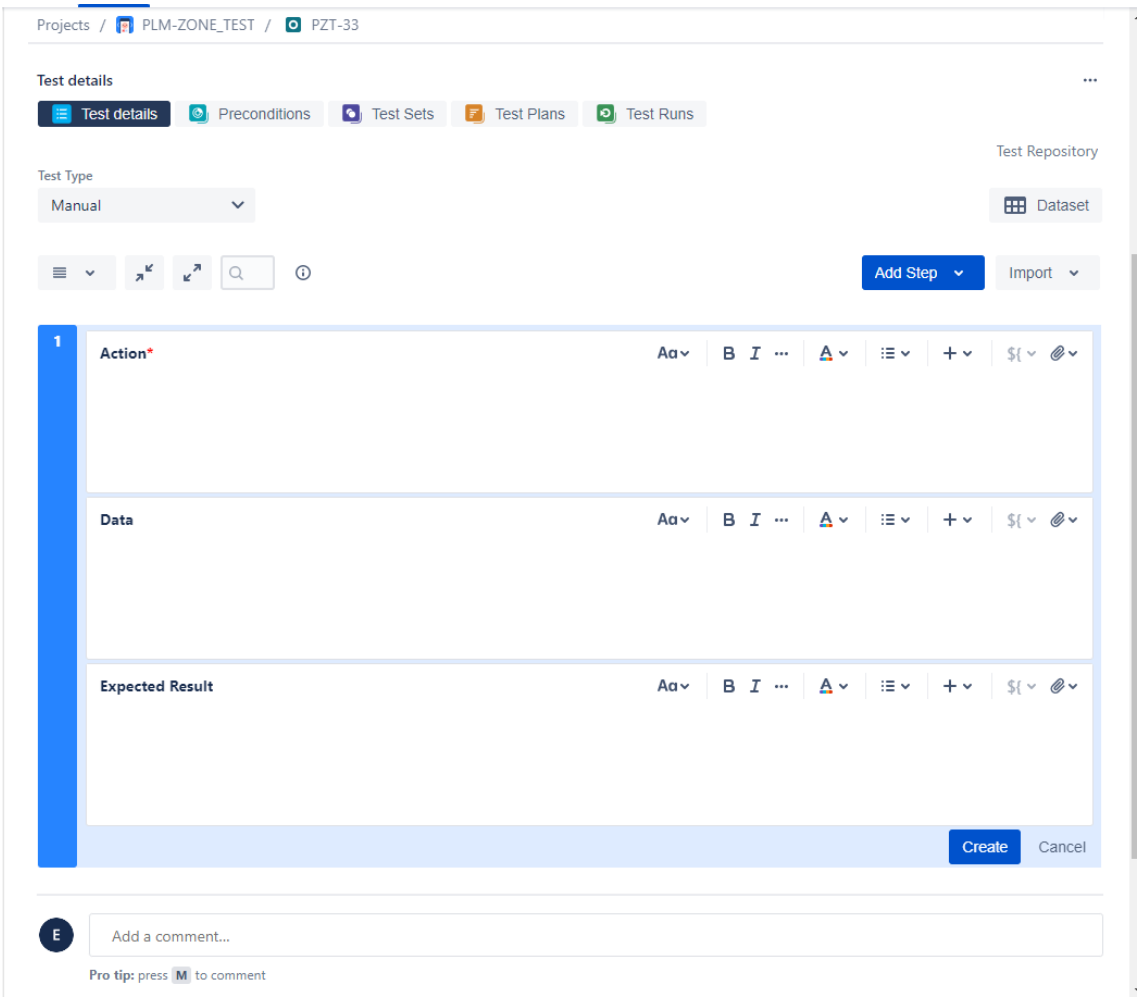
- Xray
- Zephyr
- AIO Tests.

Xray on testihallintatyökalu, mikä on kehitetty soveltumaan moitteettomasti Jira-ympäristöön. Xrayn avulla voidaan luoda testejä, organisoida, suorittaa ja luoda kattavia raportteja. Työkalu tukee myös integraatioita suosittuihin automaatio työkaluihin kuten muun muassa Junitiin, Seleniumiin, Cucumberiin ja TestNG:iin. Tämä työkalu on mahdollista saada pilvipohjaisena tilauspalveluna. Xray tarjoaa ilmaisen 30 päivän kokeiluversion. Hinnoittelu vaihtelee käyttäjien määrän mukaan. Hinnat alkavat kymmenen euroa kuukaudessa käyttäjää kohti. Kuva 9,10 ja 11 havainnollistaa Xray-testihallintatyökalun käyttöliittymän. [11.]



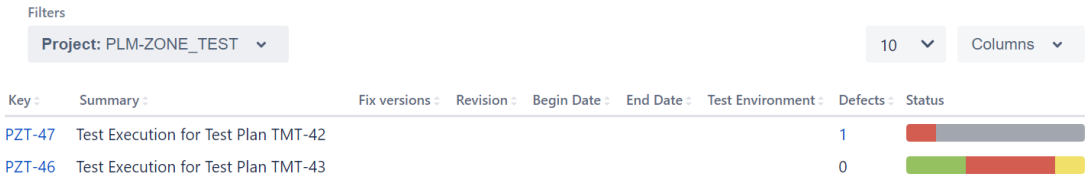
Kuva 8. Xray -tehtävätyypit.

Kuva 8 näyttää meille Xray-työkalun tehtävätyypit. Test-tehtävätyyppi on yksittäinen testitapaus. Precondition-tehtävätyypin käyttötarkoitus on antaa edellytykset ennen testien suorittamista. Test Set -tehtävätyypin käyttötarkoitus on toimia säilönä Test-tehtävätyypeille. Sen avulla organisoidaan testitapaukset. Test Execution -tehtävätyyppi käytetään suorittaakseen Test-tehtävätyypit. Test Plan -tehtävätyyppi on säiliö Test- ja Test execution -tehtävätyypeille. Tämä antaa mahdollisuudet luoda raportteja, ja ne voidaan myöhemmin suodattaa Test planin mukaan. Näiden tehtävätyyppien luonti tapahtuu samalla tavalla, kuin Jira-ohjelman oletustehtävätyypit. [11.]

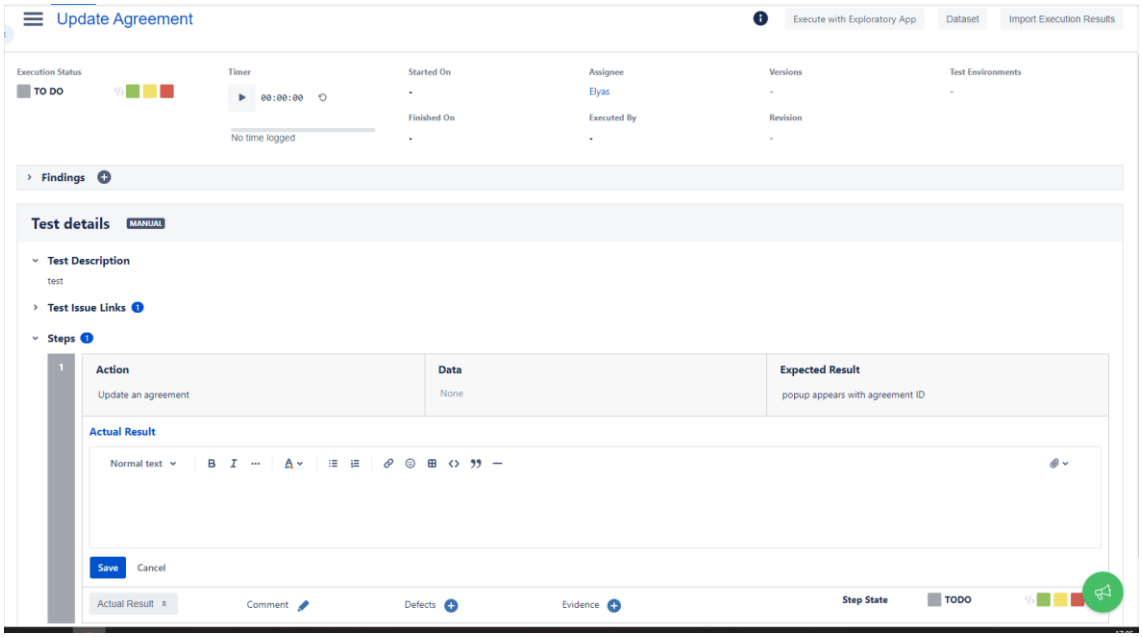


Kuva 9. Testiaskeleiden luonti Xray-ohjelmassa.

Kuva 9 havainnollistaa testien askeleiden luomista Xray-ohjelmassa. Testitapauksiin voi liittää tiedostoja ja kuvia. Oletuksena askeleen vaiheet ovat action, data, expected result. Xray tarjoaa mahdollisuuden muokata vaiheita.

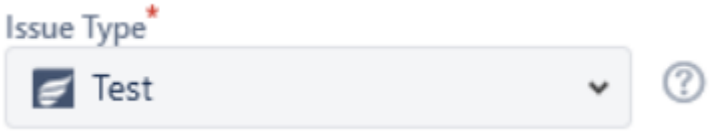


Kuva 10. testien suorittaminen Xray-ohjelmassa.



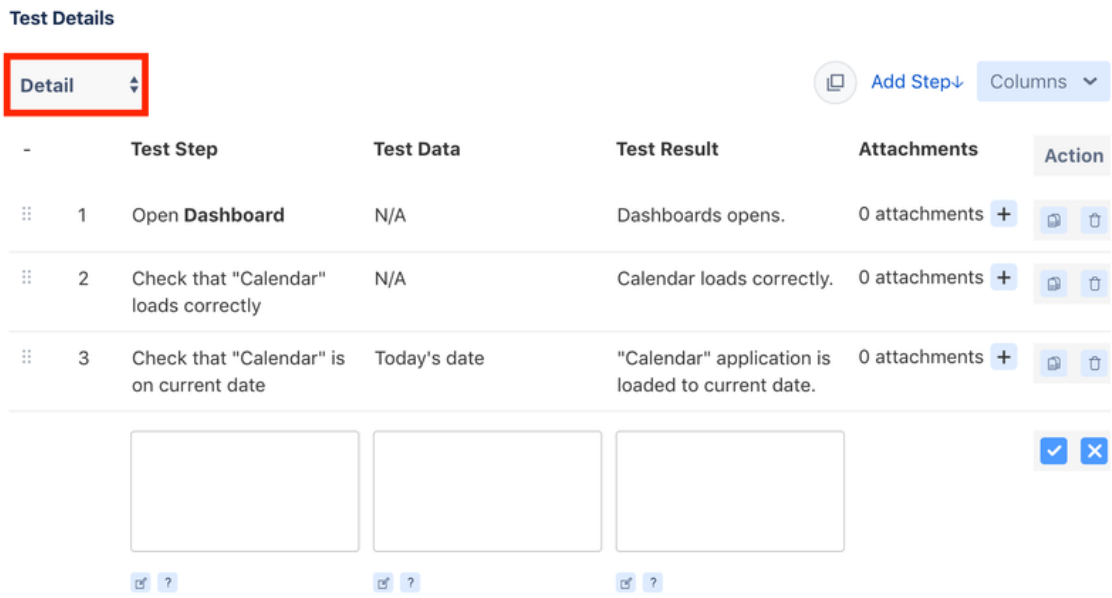
Kuva 11. Testien suoritus raportti Xray-ohjelmassa.

Zephyr Squad on toinen suosittu testihallintatyökalu, joka on kehitetty soveltumaan Jira-ympäristöön. Samoin kun Xray-testihallintatyökalussa, Zephyr-ohjelmalla voi luoda testejä, organisoida, suorittaa ja tuottaa kattavia raportteja. Zephyr tukee integraatiota suosittuihin automaatiotyökaluihin kuten Seleniumiin, Cucumberiin ja TestNg:iin. Tämä työkalu on mahdollista saada pilvipohjaisena tilauspalveluna. Zephyr tarjoaa ilmaisen 30 päivän kokeiluversion. Hinnoittelu vaihtelee käyttäjien määrän mukaan. Hinnat alkavat kymmenen euroa kuukaudessa käyttäjää kohti. Kuva 13,14 ja 15 havainnollistaa Zephyr-ohjelman käyttöliittymän. [12.]

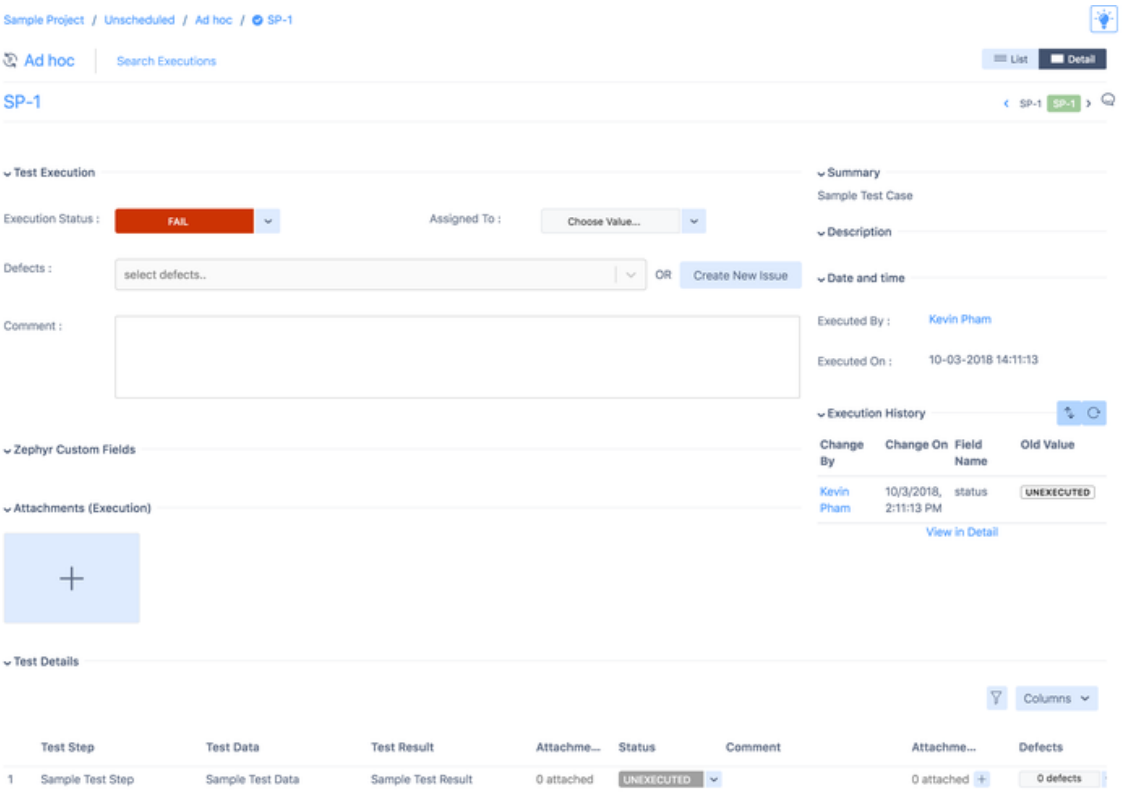


Kuva 12. Zephyr-ohjelman tarjoama tehtävätyypit.

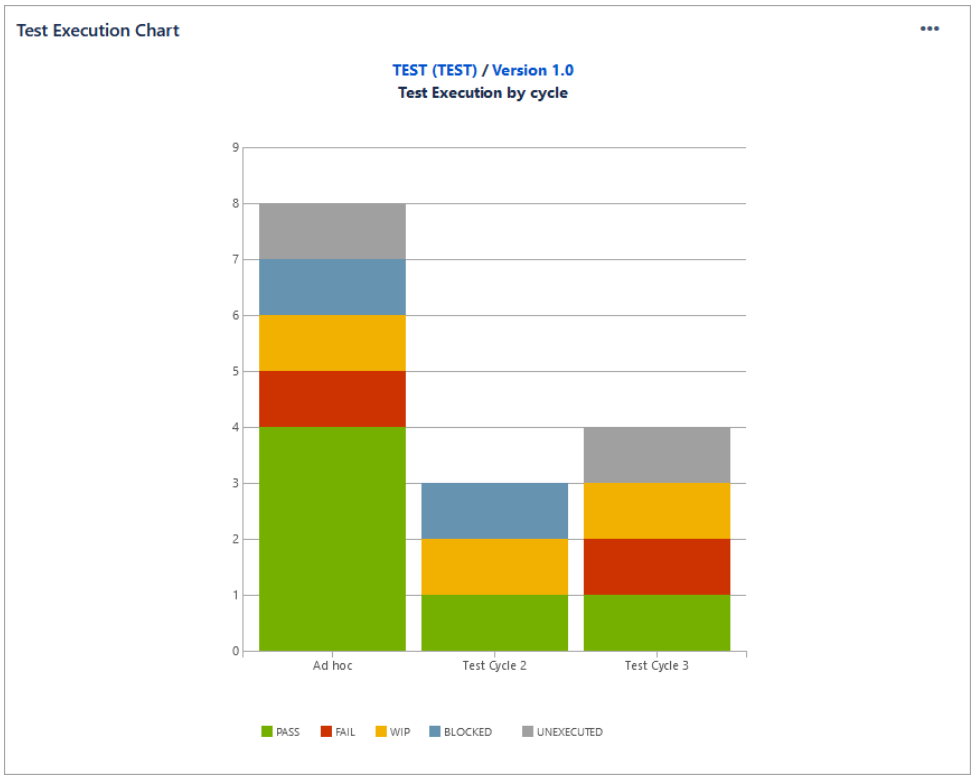
Kuva 12 havainnollistaa Zephyr-ohjelman tehtävätyypin. Tämä on ainoa Jira-ohjelmaan soveltuva tehtävätyyppi, minkä Zephyr-ohjelma tarjoaa. Kyseinen Test-tehtävätyyppi kuvaa yksittäistä testitapausta.



Kuva 13. Testiaskeleiden luonti Zephyr-ohjelmassa.

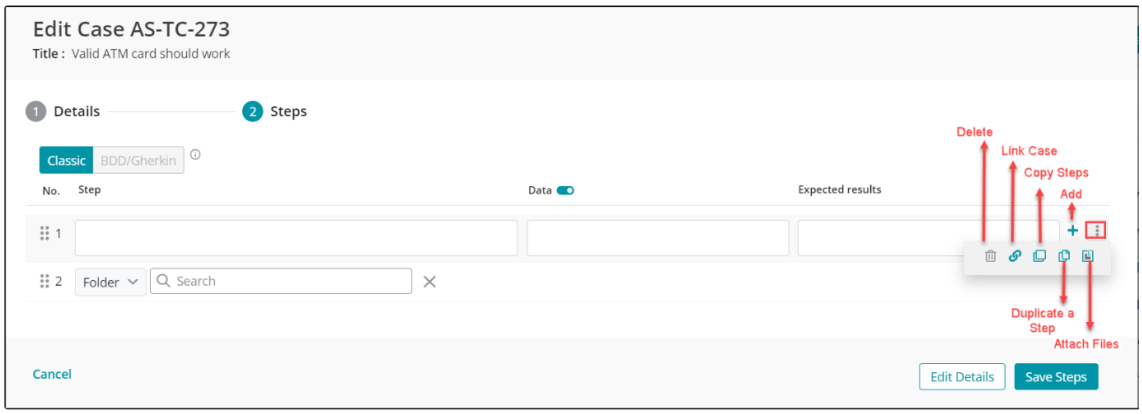


Kuva 14. Testien suorittaminen Zephyr-ohjelmassa.

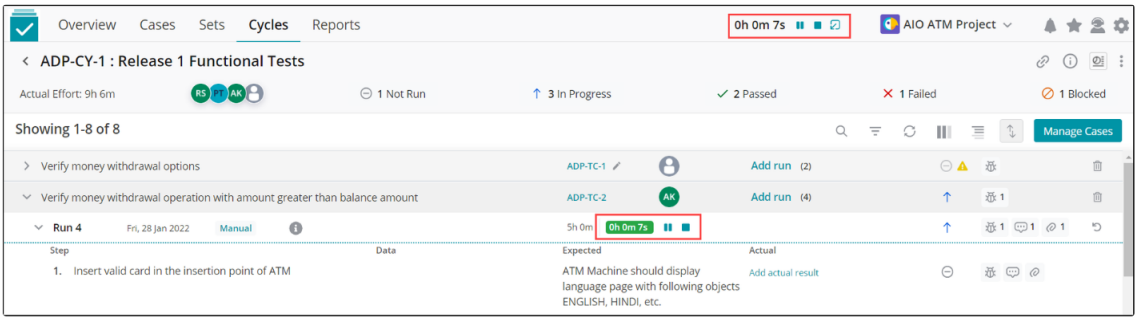


Kuva 15. Testien suoritus raportti Zephyr-ohjelmassa.

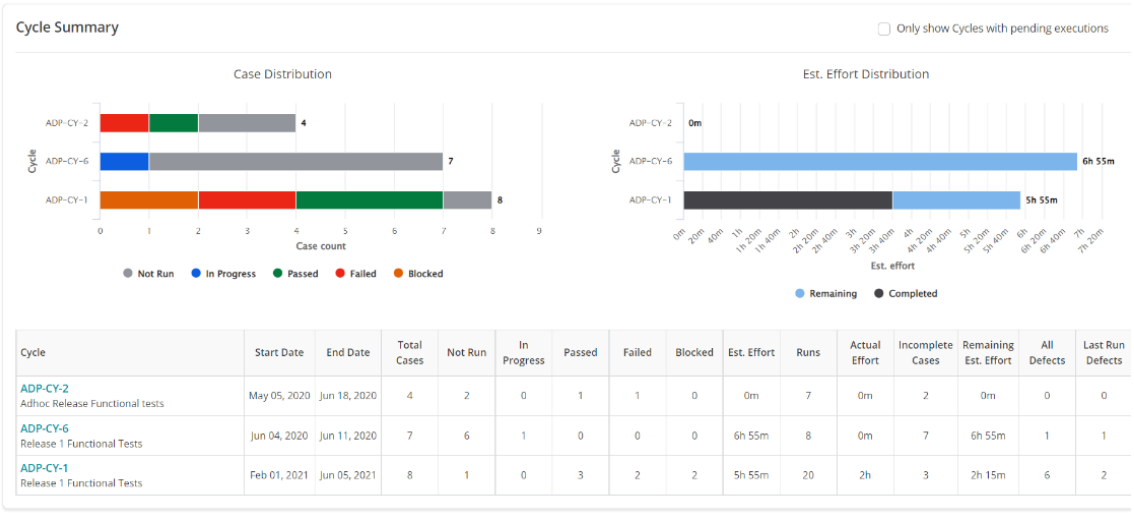
AIO Tests on viimeinen testihallintatyökalu, minkä valitsin. AIO Tests-ohjelman avulla voi luoda testejä, organisoida, suorittaa ja luoda kattavia raportteja. Työkalu tukee myös integraatioita suosittuihin automaatiotyökaluihin kuten muun muassa Junitiin ,Seleniumiin, Cucumberiin ja TestNG:iin. Tämä työkalu on mahdollista saada pilvipohjaisena tilauspalveluna. AIO Tests tarjoaa myös ilmaisen 30 päivän kokeiluversion. Hinnoittelu vaihtelee käyttäjien määrän mukaan. Työkalu on täysin ilmainen, jos organisaatiossa on alle 10 käyttäjää. Kyseinen työkalu ei ole yhtä suosittu, kuin Xray ja Zephyr, mutta työkalu silti täyttää asetetut vaatimukset ja on edullisempi. Kuva 16,17 ja 18 havainnollistaa AIO Tests-ohjelman käyttöliittymän. [13.]



Kuva 16. Testiaskeleiden luonti AIO Tests-ohjelmassa.



Kuva 17. Testien suorittaminen AIO Tests-ohjelmassa.



Cycle	Start Date	End Date	Total Cases	Not Run	In Progress	Passed	Failed	Blocked	Est. Effort	Runs	Actual Effort	Incomplete Cases	Remaining Est. Effort	All Defects	Last Run Defects
ADP-CY-2 Adhoc Release Functional tests	May 05, 2020	Jun 18, 2020	4	2	0	1	1	0	0m	7	0m	2	0m	0	0
ADP-CY-6 Release 1 Functional Tests	Jun 04, 2020	Jun 11, 2020	7	6	1	0	0	0	6h 55m	8	0m	7	6h 55m	1	1
ADP-CY-1 Release 1 Functional Tests	Feb 01, 2021	Jun 05, 2021	8	1	0	3	2	2	5h 55m	20	2h	3	2h 15m	6	2

Kuva 18. Testien suoritus raportti AIO Tests-ohjelmassa.

5.3 Testihallintatyökalujen vertailu

Testihallintatyökalun vertailussa tein itseäni varten pisteytysjärjestelmän, missä arvioin, kuinka hyvin testihallintatyökalu täyttää asetetut vaatimukset. Pisteytysjärjestelmän asteikko on 1-5, missä 1 ei täytä vaatimusta ja 5 täyttää vaatimukset täysin. Toki on huomioitavaa, että itse päätös tehtiin demotilaisuudessa, minkä järjestin sidosryhmien kanssa. Kyseinen lähestymistapa oli tutustua työkaluun ja tunnistaa vahvuudet ja heikkoudet, mitä tuon esiin tapaamisessa.

Taulukko 2 Esimerkkimalli pisteytysjärjestelmästä

Työkalu	**Jira integraatio**	**Testien hallinta**	**raportointi**	**Helppokäyttöinen**	**Hinta**
Zephyr	4	5	5	5	3
Xray	5	4	5	5	4
AIO Tests	3	4	5	3	5

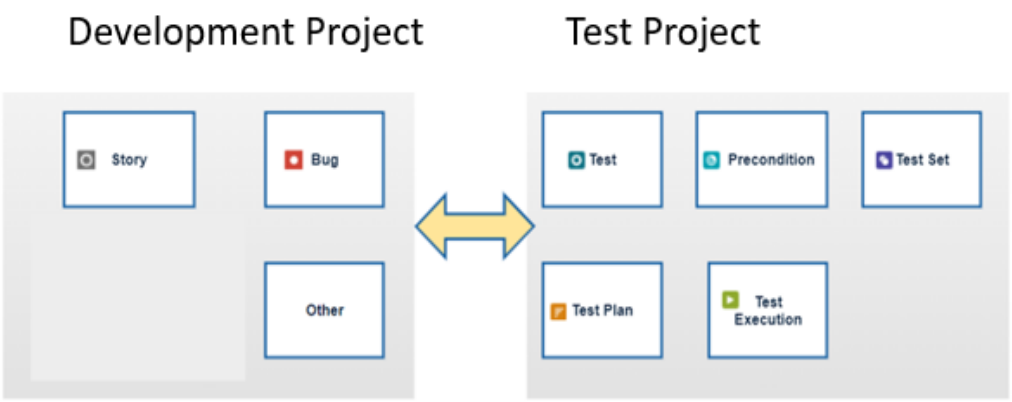
Taulukko 2 esittää, miten olin laatinut pisteytysjärjestelmän. On hyvä huomioida, että kyseessä on vain esimerkkimalli eikä todellinen versio.

Demotilaisuudessa toin esille työkalujen vahvuudet ja heikkoudet. Tilaisuudessa päätimme, että AIO Tests ei olisi meidän valintamme. AIO Tests täytti kaikki asetetut vaatimukset, mutta kyseinen työkalun käyttöliittymä oli heikompi, kuin kilpailijoilla. AIO Testsin vahvuus oli, että se on kaikista edullisin. Zephyr oli toisella sijalla. Kyseinen työkalu täytti asetetut vaatimukset, mutta kyseinen työkalu oli hinnaltaan kalliimpi, kuin Xray.

Valitsemamme työkalu on Xray-testihallintatyökalu. Tämä työkalu on edullisempi, kuin Zephyr, mutta tällä on enemmän ominaisuuksia ja selkeämpi dokumentaatio. Samoin Xray-työkalu on paljon joustavampi. Esimerkiksi sen avulla voi luoda erillisen testiprojektin, mikä mahdollistaa, että kehitysprojektiin ei sekoiteta testejä. Lisäksi firmassa löytyi henkilöitä, jotka ovat käyttäneet Xray-ohjelmaa ennen.

5.4 Koulutus ja implementaatio

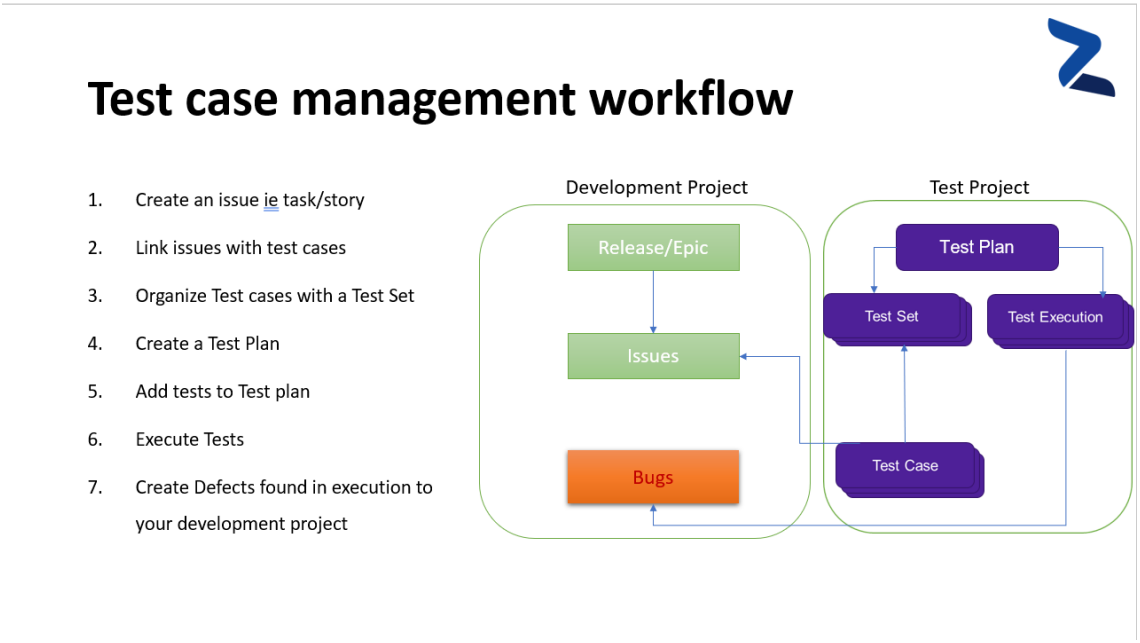
Testihallintatyökalun implementaatiota varten loin kaksi manuaalia. Yksi on työkalun konfiguraatio ja toinen on itse työkalun käyttöohje. Työkalun konfiguraatiota varten Xray tarjosi useita vaihtoehtoja. Yleinen valinta, mitä Xray suosittelee, on All in one -lähestymistapa, missä testaaminen tapahtuu kehitysprojektin sisällä. Toinen yleinen lähestymistapa on, että erotellaan testaaminen ja kehitystyö luomalla erillinen testiprojekti. Tämän lähestymistavan hyöty on, että testitapaukset eivät sotkeudu kehitysprojektiin ja nämä kaksi voidaan pitää erillään. Vaikka on erillinen testiprojekti Xray, mahdollistaa linkin muodostamisen testi- ja kehitysprojektin välille. Omalle organisaatiolle valitsin, että olisi erillinen testiprojekti jokaista kehitysprojektia varten. [14.]



Kuva 19. Yleiskuva projektin konfiguraatiosta.

Kuva 19 esittää, kuinka luomme erillisen testiprojektin kehitysprojektille. Samalla luomme linkin testien ja kehitystehtävien välille. Kyseinen kuva on myös laatimassani konfiguraatiomanuaalissa.

Työkalun käyttöohjetta varten kehitin testiprosessin, mikä soveltuu itse työkalun ominaisuuksiin asettamani konfiguraation mukaan.



Kuva 20. Yleiskuva testiprosessista.

Kuva 20 esittää, testiprosessin askeleet hyödyntäen Xray-työkalua. Kyseinen kuva on laatimastani käyttöohjeesta. Ensin luodaan Jira-ohjelman tehtävätyyppi Task tai Story. Näihin tehtävätyyppeihin linkitetään testitapaukset. Tavoitteena on, että jokainen kehitysprojektin tehtävä olisi testattu. Seuraavaksi organisoimme luodut testitapaukset, minkä jälkeen liitämme ne Test Plan -tehtävätyyppiin. Suoritamme asetetut tehtävät, ja jos ilmestyy virheitä, niin luomme Bug-tehtävätyypin kehitysprojektiin.

5.5 Yhteenveto

Lopputyön tavoitteena oli hankkia PLM-Zonelle sopiva testihallintatyökalu, mikä vastaa yrityksen tarpeita. Vaatimuksien keräämisessä nousi tärkeimmäksi Jira-ohjelman integraatio. Tämä vaatimus selkeytti, minkälaista testihallintatyökalua haemme. Päädyimme valitsemaan Xray-ohjelman, sillä se täytti asetetut vaatimukset ja oli parempi, kuin kilpailijat. Työ eteni hurjaa vauhtia, ja opin uusia asioita, sillä kyseinen projekti ei ollut tyypillinen ohjelmointiprojekti vaan oikea tutkimustyö ja ohjelmiston hankinta. Pääsin muun muassa järjestämään tapaamisia, haastatteluja ja ylipäätään sain suuren vastuun hankkia firmalle maksullista ohjelmistoa. Valitettavasti työkalua ei vielä ehditty implementoimaan virallisesti. Toki olin luonut manuaalin työkalulle ja sain palautetta, mitä asioita voi kehittää jatkossa. Loppujen lopuksi olen tyytyväinen saavutuksestani ja kiitän PLM-Zonea, että tarjosivat kivan projektin.

Lähteet

1. IBM. Software Testing. Verkkoaineisto: https://www.ibm.com/topics/software-testing. Luettu 1.4.2023.

2. Horowitz& Menn. 3.8.2012. Knight trading loss shows cracks in equity markets. Verkkoaineisto: https://www.reuters.com/article/knight-capital-trading-technology-idINDEE87200R20120803. Luettu 1.4.2023.

3. NIST. 1.4.2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. Verkkoaineisto: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [Luku 5,s. 4]. Luettu 1.4.2023.

4. Differences between Functional and Non-functional Testing. Verkkoaineisto: https://www.geeksforgeeks.org/differences-between-functional-and-non-functional-testing/. Luettu 15.4.2023.

5. Mitä on ohjelmistotestaus ja mitä hyötyä siitä on?. Verkkoaineisto: https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/. Luettu 15.4.2023.

6. Software Testing Life Cycle (STLC). Verkkoaineisto: https://www.javatpoint.com/software-testing-life-cycle. Luettu 25.4.2023.

7. Understanding Test Management Process. Verkkoaineisto: https://www.geeksforgeeks.org/understanding-test-management-process/. Luettu 25.4.2023.

8. What is Agile?. Verkkoaineisto: https://www.atlassian.com/agile. Luettu 25.4.2023.

9. Software Testing – Test Management Tools. Verkkoaineisto: https://www.geeksforgeeks.org/software-testing-test-management-tools/. Luettu 25.4.2023.

10. Jira Software features. Verkkoaineisto: https://www.atlassian.com/software/jira/features. Luettu 25.4.2023.

11. Xray. Verkkokauppa: https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira?hosting=cloud&tab=overview. Luettu 5.5.2023.

12. Zephyr. Verkkokauppa: https://marketplace.atlassian.com/apps/1014681/zephyr-squad-test-management-for-jira?hosting=cloud&tab=overview .Luettu 5.5.2023.

13. AIO Tests. Verkkokauppa: https://marketplace.atlassian.com/apps/1222843/aio-tests-all-in-one-test-management-for-jira?hosting=cloud&tab=overview. Luettu 5.5.2023.

14. Xray. Verkkoaineisto: https://docs.getxray.app/display/XRAY/Project+Organization+Use+Cases#ProjectOrganizationUseCases-Don'tmixmyRequirementsandDefectswithTests. Luettu 5.5.2023.