AUTOMAATIOTESTAUS

Case: ServiceNow Automated Test Framework (ATF)

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
kevät 2023

Jenny Holmberg

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus — Tiivistelmä

Tekijä Jenny Holmberg

Vuosi 2023

Työn nimi Automaatiotestaus – Case: ServiceNow Automated Test Framework (ATF)

Ohjaajat Lasse Seppänen, Paula Heikkinen

Tämän opinnäytetyön tarkoituksena oli selvittää millaisia testejä kannattaa automatisoida, miten testituloksia voidaan hyödyntää jatkuvassa kehittämisessä, sekä mitkä ovat ServiceNow'n Automated Test Framework -sovelluksen hyödyt ja haitat. Opinnäytetyö käsittelee laajasti ohjelmistotestausta, sekä automaatiotestausta ServiceNow'n automaatiotestaustyökalulla. Opinnäytetyö on tehty toimeksiantona julkishallinnon organisaatiolle.

Opinnäytetyön teoriaosuudessa käydään läpi ohjelmistotestauksen eri osa-alueita automaation näkökulmasta, sekä esitellään ServiceNow ja ATF-sovelluksen perusteet. Tietopohjana työssä käytetään lisäksi toimeksiantajan kehitystiimille järjestetyn tutkimuskyselyn tuloksia, sekä puolistrukturoidun haastattelun vastauksia. Opinnäytetyö on tyypiltään toiminnallinen. Toiminnallisen osuuden myötä lukija pystyy itsenäisesti ottamaan ATF-sovelluksen käyttöön ja luomaan omia automatisoituja testejä.

Selvityksen perusteella ei pysty aukottomasti sanomaan, voiko ympäristöjen toimivuuden varmistaa ATF-sovelluksella ja vähemmällä manuaalisella työllä, sillä sovelluksen käyttö itsessään vaatii manuaalista työtä. Testien luontiin käytetty aika korreloi suoritusaikaan ja siitä saatuun ajalliseen hyötyyn. Sovelluksen avulla voidaan todistetusti vähentää manuaaliseen testaukseen kuluvaa aikaa. Opinnäytetyön tuloksina työssä valmistui toimeksiantajalle ATF-sovelluksella luotuja automatisoituja testejä, priorisointimatriisi, vinkkejä parhaisiin käytäntöihin sekä käyttöönottosuunnitelma.

Avainsanat ATF, ServiceNow, automaatiotestaus, regressiotestaus, ohjelmistotestaus

Sivut 95 sivua ja liitteitä 13 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology **Abstract**

Author Jenny Holmberg Year 2023

Subject Test automation – Case: ServiceNow Automated Test Framework (ATF)

Supervisors Lasse Seppänen, Paula Heikkinen

The purpose of this thesis was to find out what kind of tests should be automated, how the test results can be used in continuous development, and what the benefits and drawbacks of ServiceNow's Automated Test Framework application are. The thesis deals extensively with software testing, as well as test automation with ServiceNow's automation testing tool. The thesis was done as an assignment for a public administration organization.

The theoretical background deals with different aspects of software testing and is reviewed namely from an automation point of view. ServiceNow and the basics of the ATF application are introduced as well. Research data obtained through a survey organized for the client's development team, as well as answers from a semi-structured interview were used to benefit the thesis. The thesis is functional in type and meant as a guide for creating one's own automated tests independently with the ATF application.

Based on the results obtained in the thesis, it is not possible to definitively say whether the functionalities of the environments can be ensured with the ATF application and less manual work because the application itself requires some manual work. The time spent on creating the tests correlates with the execution time and the time benefit obtained from it. Usage of the application proved to reduce the time spent on manual testing. The thesis work resulted in automated tests created with the ATF application, a prioritization matrix, tips for best practices, and an implementation plan.

Keywords ATF, ServiceNow, test automation, regression testing, software testing

Pages 95 pages and appendices 13 pages

Sanasto

Automaatiotestaus	Kone suorittaa testauksen annettujen parametrien mukaan.
ATF-sovellus	ServiceNow Automated Test Framework (ATF) on sovellus, jonka avulla luodaan ja suoritetaan automatisoituja testejä.
Automatisoitu testi	Testitapaus, jonka suorittaja on kone. Automatisoitu testi voidaan ajastaa tai suorittaa manuaalisesti.
Black Box -testaus	Testaustapa, jossa testaaja tietää mitä ohjelmisto tekee.
Gray Box -testaus	Testaustapa, jossa testaaja tietää mitkä ohjelmiston vaatimukset ovat.
Järjestelmätestaus	Järjestelmätestauksella varmistetaan ohjelmiston toiminta kokonaisuudessaan. Järjestelmätestauksessa testataan koko järjestelmää tai sen osaa.
Ohjelmistotestaus	Ohjelmistotestauksen tarkoituksena on antaa sidosryhmille tietoa testattavan ohjelmistotuotteen tai -palvelun laadusta.
Regressiotestaus	Regressiotestauksella varmistetaan, ettei järjestelmään tehty muutos tai päivitys vaikuta olemassa olevaan koodiin, eli ei aiheuta virhetilannetta.
ServiceNow	Pilvipohjainen toiminnanohjausjärjestelmä.
Testitapaus	Kirjallinen ohje, miten jokin toiminto tai ominaisuus testataan, sisältää myös tiedon testauksen lopputuloksesta.
Toiminnanohjausjärjestelmä	Laaja kokonaisvaltainen järjestelmä, jolla tyypillisesti yksi tietokanta, jota eri toiminnot käyttävät.
Update set	Suomennetaan usein päivityspaketiksi, sisältää kehittäjän tekemät muutokset esim. kehitysympäristössä. Update set siirretään XML-tiedostona ympäristöjen välillä.
White Box -testaus	Testaustapa, jossa testaaja tietää miten ohjelmisto toimii.

Sisällys

Kuvat ja taulukot

Liitteet

Liite 1 Aineistonhallintasuunnitelma

Liite 2 Tutkimuskysely

Liite 3 ATF-testitulosten säilytyskäytännön muokkaus

Liite 4 Taulut, joista ATF ei voi palauttaa tehtyjä muutoksia

Liite 5 ATF-käyttöliittymätestien ajo Headless Browser:illa

1 Johdanto

Automaatiotestaus tarkoittaa testausta, jonka kone suorittaa. Automatisoidun testin testitapauksessa kerrotaan vaiheittain mitä ja miten kone suorittaa testauksen. Automaatiotestausta hyödynnetään useimmiten toistuvissa ja toistettavissa prosesseissa ja / tai toiminnoissa. Automatisoinnilla pyritään vaikuttamaan kustannusten alentamiseen, ajankäyttöön, kehittämisen tehostamiseen ja ohjelmistovirheiden vähentämiseen.

Opinnäytetyössä tutustutaan ServiceNow'n automaatiotestauksen työkaluun. ServiceNow on pilvipohjainen toiminnanohjausjärjestelmä, joka koostuu monista eri sovelluksista, tässä työssä syvennytään yhteen niistä. ServiceNow Automated Test Framework (ATF) on sovellus, jonka avulla luodaan ja suoritetaan automatisoituja testejä. Työssä esitellään ServiceNow'n käyttömahdollisuudet tarvittavin osin.

Opinnäytetyön aihe valikoitui omasta kiinnostuksesta automaatiotestaukseen sekä toimeksiantajan tarpeesta selvittää sen tuoma mahdollinen hyöty kehitystiimille. Automaatiotestaus on aiheena entuudestaan tuttu, mutta ServiceNow'n automaatiotestaustyökalu ei ole. Tavoitteena on selvittää millaisia testejä kannattaa automatisoida, miten testituloksia voidaan hyödyntää ja mitkä sovelluksen hyödyt ja haitat ovat. Työssä pyrin selvittämään myös sovelluksen hyödyt ja mahdolliset haitat verrattuna manuaaliseen järjestelmätestaukseen.

Opinnäytetyön tuloksena on selvitys, voidaanko ATF:n avulla ja vähemmällä työllä varmistaa, että eri ympäristöt (kehitys ja testi) toimivat tehdyn muutoksen jälkeenkin, verrattuna manuaaliseen työhön. Lopputuloksena on myös sovelluksella luotuja testejä, jotka jäävät toimeksiantajan käyttöön.

Opinnäytetyön tutkimuskysymykset ovat:

- Minkälaisia testejä kannattaa automatisoida?
- Miten automatisoitujen testien tuloksia voidaan hyödyntää kehitystyössä?
- Mitkä ovat ATF-sovelluksen hyödyt ja haitat?

2 Ohjelmistotuotanto

Seuraavissa luvuissa käydään läpi, miten ohjelmistotuotanto on vaikuttanut testaukseen. Lyhyt katsaus historiaan muistuttaa mistä aikanaan on lähdetty, kun vertaillaan tilannetta nykypäivän ohjelmistojen uudelleenkäyttöön ja ketterään kehittämiseen. Vaikkakin ohjelmistotuotteet voivat olla suuria ja monimutkaisia, ihmiset voivat kuvitella mitä ohjelmisto voi tehdä heidän hyväkseen ennen kuin näkevät valmiin tuotteen (Mili & Tchier, 2015).

2.1 Ohjelmistotuotannon historia

Ohjelmistotuotanto on nuorin tekniikan ala, ohjelmistokehitys aloitettiin 50-luvun lopulla korkean tason ohjelmointikielten tulon myötä. 90-luvulla tuli tutuksi uudelleenkäyttöön perustuva ohjelmistotuotanto. Mili ja Tchierin (2015) mukaan uudelleenkäyttö on erottamaton osa suunnitteluprosessia kaikilla tekniikan aloilla. He käyttävät usein autoa esimerkkinä vertaillessaan eri tekniikan aloja. Kun markkinoille tuodaan uusi auto, sitä ei ole keksitty uudestaan alusta. Tällä he vertailivat sitä miksi uusien ohjelmistotuotteiden kanssa ei voisi käyttää uudestaan olemassa olevia koodirivejä. Ohjelmistotuotannon uudelleenkäytettäviä osia ovat esim. ohjelmiston määrittely, lähdekoodi sekä testidata. 2000-luvun ensimmäisellä vuosikymmenellä ketterä ohjelmointi painottaa joustavaa ja nopeaa reagointia muutoksiin. Ohjelmistosuunnittelusta tuli komponenttipohjaista. Kiinnostusta tekoälypohjaiseen ohjelmistosuunnitteluun oli jo 80-luvulla, vasta 2000-luvun toisella vuosikymmenellä sillä on suuria vaikutuksia ohjelmistokehitykseen, kuten esim. koneoppiminen ja neuraaliverkot. Sosiaalinen media ja esineiden internet ovat luoneet valtavan kysynnän ohjelmistosuunnittelun osaamiselle. (Mili & Tchier, 2015)

2.2 Ohjelmistotuotannon uudelleenkäyttö

Laadukkaitten ohjelmistotuotteiden rakentaminen ei ole vain kriittistä, vaan myös vaikeaa. Laatustandardia voidaan valvoa prosessinhallinnan ja tuotevalvonnan kautta. Prosessinhallinnalla tarkoitetaan varmistusta siitä, että ohjelmistot kehitetään alan standardien tai hallituksen määräämien standardien mukaisesti. Tuotevalvonnalla

tarkoitetaan toimenpiteitä tuotteiden laatuominaisuuksien varmistamiseksi, kuten dynaamista testausta, luotettavuuden arviointia ja vikasietoisuutta. (Mili & Tchier, 2015)

Vaikka ohjelmistotuotteet ovat suuria, monimutkaisia ja kalliita, voidaan uudelleenkäytöllä vaikuttaa kustannuksiin ja niiden monimutkaisuuteen. Suomentaen vapaasti alkuperäistä lähdettä, Mili ja Tchier (2015) kiteyttivät ajatuksen lauseeseen ”Suunnittele kerran, käytä uudelleen usein”. Ohjelmiston uudelleenkäyttö on kuitenkin erittäin vaikeaa ja epäkäytännöllistä. Mahdollisuudet uudelleenkäytölle ovat hyvin rajalliset monista syistä, esim. ohjelmistovaatimukset, jotka vaihtelevat hyvin laajasti, toiminnalliset yhteensopivuudet, jossa arkkitehtuuri ei kohtaa, tai toimialuekohtaiset mallit, esim. tietojen- ja tapahtumakäsittelyssä, joissa materiaalia on hyvin rajoitetusti uudelleen käytettäväksi. (Mili & Tchier, 2015)

2.3 Ohjelmistotuotannon kustannustenhallinta

Useimmilla toimialoilla kustannuksia hallitaan tuotantomäärillä. Mitä enemmän tuottaa, sitä pienemmät yksikkökustannukset, ohjelmistoissa tämä on toisinpäin. Muilla toimialoilla yhtenäiset tuotteet ovat toisistaan riippumattomia, kun taas ohjelmistoissa ne ovat toisistaan riippuvaisia. Mitä enemmän ohjelmistoja, sitä enemmän riippuvuuksia on kyettävä hallitsemaan. (Mili & Tchier, 2015)

Syntyy paradoksaalinen talous. Ohjelmistokehitys on erittäin työvoimavaltainen ala, suurin osa tuotteen kustannuksista johtuu suunnittelusta, jota ei voida automatisoida. Verrattaessa ohjelmistotuotantoa autoon, suurin osa auton kustannuksista muodostuu valmistuksesta, ei sen suunnittelusta. Ohjelmistoilla tämä on päinvastaista. Perinteisillä tekniikan aloilla tuotteita valmistetaan suuria määriä ja niiden odotetaan toimivan määritelmien mukaisesti, niin prosessihallinnan kuin tilastollisten havaintojen näkökulmasta katsottuna. Tekniikan aloilla kustannukset muodostuvat yli 90 % valmistuksesta ja alle 1 % testaamisesta. Ohjelmistoissa lähestymistapa on eri, sillä jokainen tuote on testattava. Ohjelmistojen kustannukset jakautuvat tasan, valmistukseen käytetään 50 % ja testaukseen 50 %. Mili ja Tchier (2015) haastavat ajattelemaan, miten paljon autoon luotettaisiin, jos puolet sen hinnasta on käytetty testaamiseen? (Mili & Tchier, 2015)

Ylläpitokustannukset voidaan jakaa kahteen kategoriaan, korjaavaan ja mukautuvaan. Korjaava ylläpito tarkoittaa suunnittelun muuttamista tai toteutuksen virheiden korjausta. Ohjelmistoissa kaikki korjaavat ylläpitotoimenpiteet liittyvät virheelliseen suunnitteluun ja toteutukseen. Mukautuva ylläpito tarkoittaa mukauttamista muuttuviin vaatimuksiin. Mukautuvaa ylläpitoa ei ole muualla kuin ohjelmistotuotannon toimialalla. Havainnollistava esimerkki on, että ostat liikkeestä kaksiovisen auton ja palaat takaisin pyytämään kahta ovea lisää. (Mili & Tchier, 2015)

3 Ohjelmistotestaus

Tässä luvussa käydään läpi ohjelmistotestausta automaation näkökulmasta. Pääluku kuvaa lyhyesti mitä ohjelmistotestaus on. Alaluvuissa kerrotaan ohjelmiston laadusta ja sen ominaisuuksista, sekä millä eri tavoin ohjelmistoja voidaan testata. Käydään myös läpi automaatiotestauksen näkökulmasta regressiotestausta, automaatiotestausta yleisesti, verrataan manuaalista testausta automaatiotestaukseen, sekä kerrotaan niiden hyödyt ja haitat.

Ohjelmistotestauksen tarkoituksena on antaa sidosryhmille tietoa testattavan ohjelmistotuotteen tai -palvelun laadusta. Sitä mukaan, kun ohjelmisto kehittyy, on myös testausta kehitettävä sen mukana. Ohjelmistotestauksella varmistetaan, että ohjelmisto vastaa määrittelyjä ja että se toimii kuten on haluttu. Voidaan myös sanoa, että testauksen suunnittelussa ja suorittamisessa tavoitteena on selvittää millä tavoin ohjelmisto on rikki. (Hoffman, 2019)

Hamilton (2021e) määrittelee ohjelmistotestauksen tarkoituksena olevan virheiden, puutteiden ja puuttuvien vaatimusten tunnistaminen. Yksinkertaisesti sanottuna ohjelmistotestaus tarkoittaa testattavan sovelluksen varmentamista. (Hamilton, 2021e)

Ohjelmistotestauksen hyötyihin lasketaan kustannustehokkuus, ohjelmiston laatu, turvallisuus ja asiakastyytyväisyys. Minkä tahansa ohjelmiston testaaminen ajoissa säästää rahaa pitkällä aikavälillä. Jos viat havaitaan aikaisessa vaiheessa testausta, maksaa niiden korjaaminen vähemmän. Ohjelmiston laatu on olennainen vaatimus kaikille ohjelmistotuotteille. Testaamisella varmistetaan, että asiakas saa laadukkaan tuotteen. Ohjelmistoihin pitää voida luottaa ja testaamisen avulla varmistetaan, ettei haavoittuvuuksia löydy ja poistetaan mahdolliset ohjelmistoriskit. Ohjelmiston käyttöliittymällä on suuri vaikutus asiakastyytyväisyyteen. Kun halutaan taata asiakkaille paras käyttökokemus, testataan ohjelmiston käyttöliittymää. (Hamilton, 2021e)

3.1 Ohjelmiston laatu

Kun ohjelmistoa testataan, on tärkeää tietää myös sen laatuominaisuuksista. Ohjelmiston laatua voidaan mitata ja se vaikuttaa myös suunnitteluun. Laadukkaan ohjelmiston ominaisuuskategoriat ovat; toiminnalliset-, käyttö-, käytettävyys-, liiketoiminta- ja rakenteelliset ominaisuudet. Testauksella yleensä todistetaan, että jotkin prosessit tai tuotestandardit täyttyvät. Voidaan testata, että jokin laatuominaisuus saavutetaan, esim. oikeellisuus, luotettavuus, tai epäonnistumisaste. Testaamisella voidaan parantaa tai seurata laatuominaisuutta, esim. ohjelmiston luotettavuutta tai vikasietoisuutta. Testauksen avulla voidaan myös luoda tilastollisia todisteita laatuominaisuuden tueksi epäonnistumisasteella. (Mili & Tchier, 2015)

Ohjelmiston ominaisuudet kiinnostavat sidosryhmiä eri tavalla, riippuen heidän roolistaan. Ohjelmiston toiminnalliset-, käyttö- ja käytettävyysominaisuudet kiinnostavat järjestelmän käyttäjiä, kun taas liiketoiminnan- ja rakenteellisista ominaisuuksista ovat kiinnostuneet järjestelmän kehittäjät ja johtajat. Liiketoiminnan ominaisuuksiin kiinnittävät enemmän huomiota projektipäälliköt kuin tekninen ryhmä (suunnittelijat, insinöörit, analyytikot, jne.), joita kiinnostaa ohjelmiston rakenteelliset ominaisuudet. (Mili & Tchier, 2015)

3.1.1 Toiminnalliset ominaisuudet

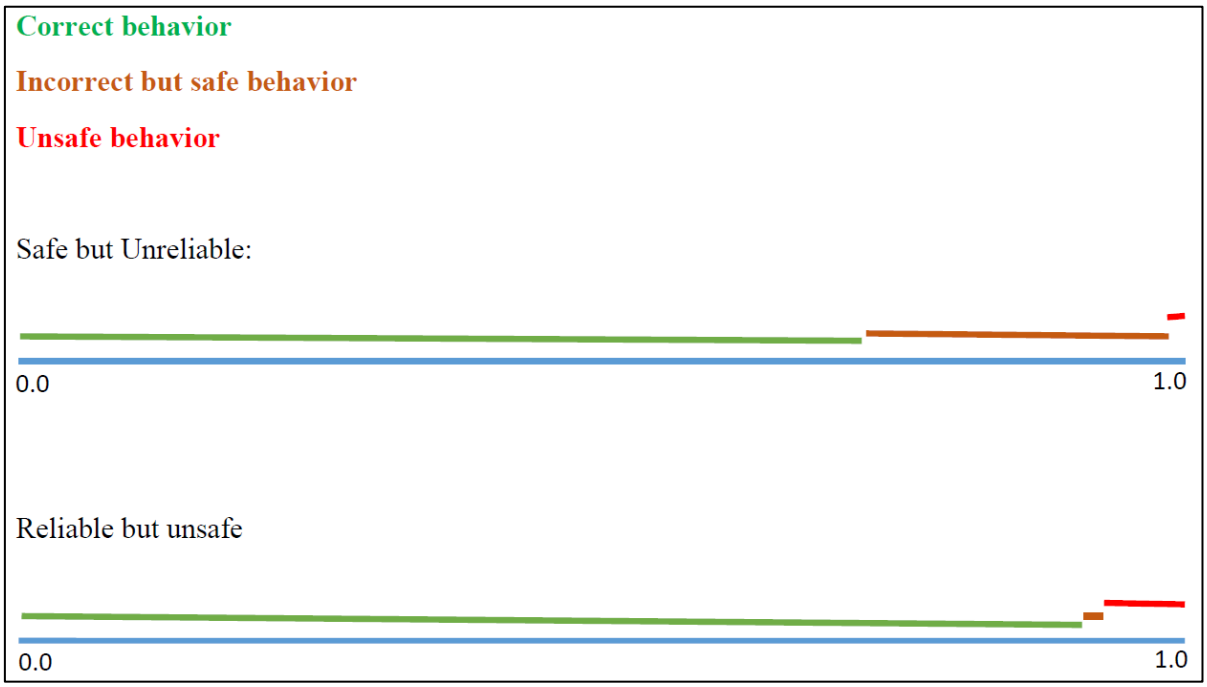
Ohjelmiston toiminnalliset ominaisuudet ovat määrittely, toimialue ja käyttäytymismalli. Nämä kaikki ovat attribuutteja, jotka liittyvät ohjelmiston tulo- ja lähtökäyttäytymiseen. Määrittelyssä kuvataan ohjelmiston tarvitsemat ja pakolliset tulo- ja lähtökäyttäytymiset. Toimialueessa on kuvattu syötteet ja tilanteet, jotka ohjelmiston on käsiteltävä. Käyttäytymismallissa käydään läpi todennäköisyysjakaumaa, eli kuinka usein jokainen syöte tai tilanne tapahtuu ohjelmiston normaalissa käytössä. (Mili & Tchier, 2015)

Toiminnallisten ominaisuuksien tilastolliset attribuutit ovat luotettavuus ja turvallisuus. Luotettava ohjelmisto tarkoittaa, että käyttäjät voivat luottaa siihen, että ohjelmisto ei kaadu ja toimii kuten sen pitäisikin. Luotettavuutta voidaan mitata MTTF:llä (Mean Time To Failure) tai MTBF:llä (Mean Time Between Failures). Ero näiden kahden välillä on, ettei MTTF:llä ole muistia, milloin järjestelmä viimeksi kaatui. Turvallinen ohjelmisto kuvastaa sen

kykyä toimia normaalisti tai epänormaalisti ilman että se vaarantaisi ihmishenkiä tai vahingoittaisi järjestelmän ympäristöä. Voidaan sanoa, että järjestelmä on vikasietoinen, jos sen luotettavuutta ei pystytä varmistamaan, mutta voidaan vakuuttaa, että se ei aiheuta katastrofia, vaikka kaatuisikin. Ohjelmiston turvallisuutta voidaan mitata MTTF:llä tai MFC:llä (Mean Failure Cost). (Mili & Tchier, 2015)

Kuva 1 osoittaa, että järjestelmä voidaan todeta turvalliseksi mutta epäluotettavaksi tai luotettavaksi mutta vaaralliseksi. Turvallinen mutta epäluotettava järjestelmä täyttää vaatimukset 0,80 todennäköisyydellä, mutta kaatuessaan se harvoin aiheuttaa katastrofia. Luotettava, mutta vaarallinen järjestelmä taas täyttää vaatimukset 0,99 todennäköisyydellä, mutta kaatuessaan aiheuttaa usein katastrofin. (Mili & Tchier, 2015)

Kuva 1 Vertailu turvallisen, mutta epäluotettavan ja luotettavan, mutta vaarallisen järjestelmän välillä (Mili & Tchier, 2015)



3.1.2 Käyttöominaisuudet

Siinä missä toiminnalliset ominaisuudet käsittelevät tuotteen toimintoja ja / tai palveluita, niin käyttöominaisuudet käsittelevät ehtoja, joissa näitä palveluja tarjotaan. Mitattavia käyttöominaisuuksia on viisi; latenssi, läpäisykyky, tehokkuus, kapasiteetti ja skaalautuvuus.

Kaikki nämä ominaisuudet voivat olla tavoitteena testauksessa. Latenssilla tarkoitetaan keskimääräistä vasteaikaa sekunneissa, eli käyttäjän syötteen ja järjestelmän vasteen välinen viive, esim. interaktiivisissa järjestelmissä. Läpäisykyvyllä tarkoitetaan käsiteltyä määrää aikayksikköä kohti, esim. eräajossa suoritettavia tapahtumia. Tehokkuudella tarkoitetaan resurssien säästöä, esim. keskusmuistia, levytilaa, verkon kaistanleveyttä ja prosessorin syklejä. Järjestelmän kapasiteetti viittaa käyttäjien määrään, jota se pystyy palvelemaan samanaikaisesti säilyttäen riittävän palvelun laadun. Skaalautuvuus tarkoittaa järjestelmän kykyä jatkaa palvelujen toimittamista, kun työmäärä ylittää nimelliskapasiteetin, eli hienoista huononemaa. (Mili & Tchier, 2015)

3.1.3 Käytettävyysominaisuudet

Käytettävyyteen liittyvät ominaisuudet ovat käyttäjän ja järjestelmän väliset suhteet. Ominaisuudet ovat helppokäyttöisyys, opittavuus, muokattavuus, kalibroitavuus ja yhteentoimivuus. Järjestelmän helppokäyttöisyys ja opittavuus tarkoittaa, että käyttäjät voivat helposti käyttää järjestelmää perustehtävien suorittamiseen. Kun järjestelmä on suunniteltu intuitiivisesti, käyttäjät eivät tee virheitä eikä toiminnot vie kuin muutaman klikkauksen. Muokattavuudella tarkoitetaan, että järjestelmää on mahdollista muokata käyttäjien erityistarpeisiin. Kalibroitavuudella tarkoitetaan kykyä kalibroida järjestelmä erityisiin toiminnallisiin tarpeisiin. Yhteentoimivuus tarkoittaa kykyä olla vuorovaikutuksessa muiden sovellusten kanssa, esim. kommunikointi, tietojen vaihto ja sen käyttäminen. (Mili & Tchier, 2015) Käytettävyyteen liitetään kaikki käyttäjäkokemukseen liittyvät elementit, miten käyttäjät oppivat, löytävät sisältöä ja voivat tehdä enemmän järjestelmällä (Interaction Design Foundation, n.d.).

3.1.4 Liiketoimintaominaisuudet

Ohjelmiston liiketoiminnallisia ominaisuuksia ovat kehityskustannukset, ylläpidettävyys ja siirrettävyys. Kehityskustannuksiin sisältyy aika henkilökuukausina ohjelmiston määrittelystä testaukseen. Ylläpitokustannukset lasketaan vuositasolla käyttäen kaavaa henkilökuukaudet vuodessa jaettuna koodirivien määrällä. Siirrettävyys tarkoittaa tässä yhteydessä kustannuksia, jotka aiheutuvat sovelluksen siirtämisestä ympäristöstä toiseen. Kustannuksia hallitaan eristämällä ja minimoimalla alustasta riippuvaisia toimintoja. (Mili & Tchier, 2015)

Liiketoimintaominaisuuksiin kuuluu myös uudelleenkäytettävyys. Mahdollisuus käyttää uudelleen joko kokonaan, osittain, sellaisenaan tai muutosten jälkeen muun kehittämisen yhteydessä. Uudelleenkäytettävyys riippuu kahdesta ominaisuudesta. Ensimmäinen ominaisuus on ohjelmiston käyttökelpoisuus, onko ohjelmistolle kysyntää ja missä määrin kysyntää on. Toinen ominaisuus on käytettävyys, miten helposti ohjelmistokomponentit voidaan sovittaa muihin sovelluksiin. (Mili & Tchier, 2015)

3.1.5 Rakenteelliset ominaisuudet

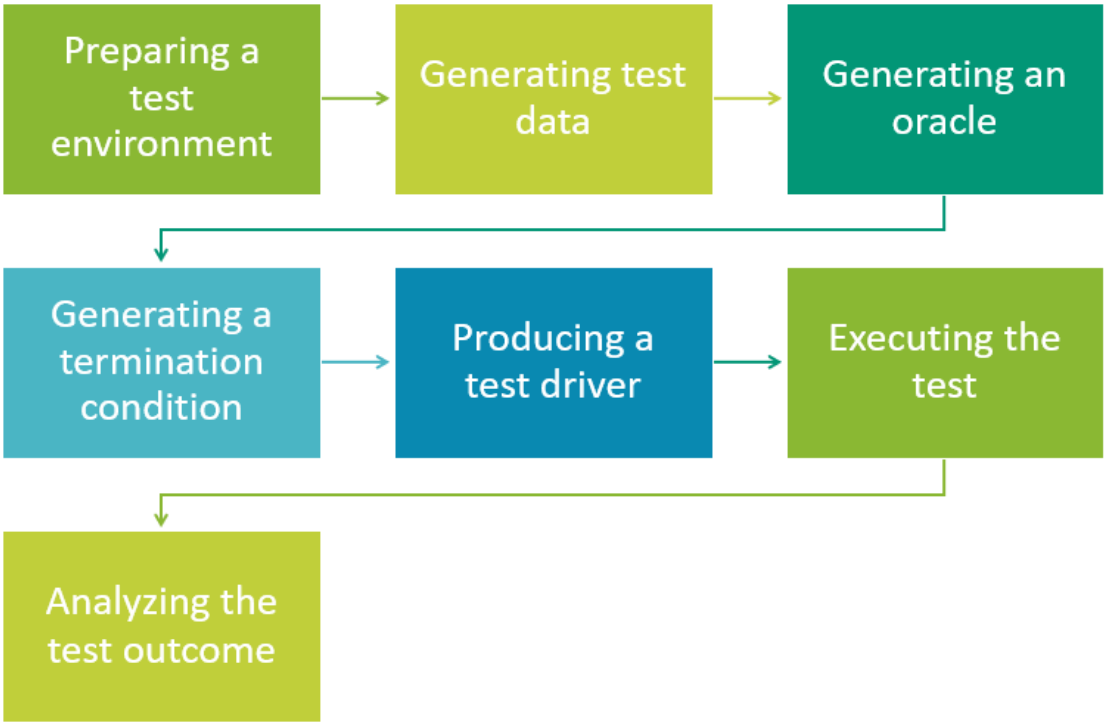
Ohjelmiston rakenteellisiin ominaisuuksiin kuuluvat suunnittelun eheys, modulaarisuus, testattavuus ja sopeutuvuus. Suunnittelun eheydellä tarkoitetaan johdonmukaisuutta suunnittelussa, huomioidaan että värit, fontit ja kuvat, eli ohjelmiston elementit ovat yhteensopivia alustasta riippumatta. Osa eheyttä on myös yksinkertainen suunnittelu, käytettävä ja käyttäjäystävällinen muotoilu. Modulaarisuudella tarkoitetaan tiedon piilottamista, joko koodin puolella tai käyttöliittymässä. Koska moduulien välillä siirretään tietoja ja yksittäiset moduulit sisältävät erilaista tietoa, hyvin suunniteltu modulaarisuus pitää tiedonsiirron matalana ja yhtenäisyyden korkeana. Testattavuuteen liittyy kaksi attribuuttia. Attribuutit ovat ohjattavuus ja havaittavuus, molempiin liittyy kaistanleveys, jolla voidaan ohjata järjestelmään tulevia syötteitä, sekä tarkkailla järjestelmästä lähtevää tietoa. Sopeutuvuudella tarkoitetaan ohjelmiston muuttamista muuttuvien vaatimusten mukaiseksi. Sopeutuvuus eroaa muokattavuudesta ja ylläpidettävyydestä, siten että kyseessä on muuttunut vaatimus, ei virheen korjausta, ei päivittämistä uuteen versioon, eikä täydellistä ylläpitoa, jolla parannetaan ohjelmistoa ja sen arvoa. (Mili & Tchier, 2015)

3.2 Ohjelmistotestauksen elinkaari

Ohjelmiston testaus suoritetaan esimerkkisyötteellä ja analysoidaan sen tulokset. Toisin kuin muut tuotekehitystuotteet, ohjelmistotuotteet ovat tyypillisesti viallisia. Niissä esiintyy määrittely-, suunnittelu- ja / tai toteutusvirheitä. Toisin kuin muut tekniset tuotteet, viat johtuvat epätäydellisestä kehityksestä. Kun ohjelmistoa ei ylläpidetä, seuraa jatkuva heikkeneminen ohjelmiston laadussa. (Mili & Tchier, 2015)

Kuva 2 on luotu materiaalin pohjalta ja kuvaa ohjelmistotestauksen elinkaaren seitsemän vaihetta. Vaiheet ovat testiympäristön valmistelu, testidatan luonti, oraakkelin luonti, lopetusehdon luonti, testisuorittajan luonti, testin suorittaminen ja testitulosten analysointi. (Mili & Tchier, 2015)

Kuva 2 Ohjelmistotestauksen elinkaaren 7 vaihetta (Mili & Tchier, 2015).



Seuraavissa alaluvuissa kuvataan tarkemmin ohjelmistotestauksen elinkaaren vaiheita.

3.2.1 Testiympäristö

Testiympäristö tarvitaan kehitystyön tueksi, sillä suurin osa testauksesta tapahtuu jo kehitysvaiheessa. Testiympäristön on tarkoitus jäljitellä tuotantoympäristöä. Testaushetkellä tehdyt havainnot heijastavat käyttäytymistä, joka kuvitellaan käytön aikana. Testiympäristö on valmisteltu, kun onnistutaan simuloimaan tuotantoympäristöä ja sen työkuormitusolosuhteita. (Mili & Tchier, 2015)

3.2.2 Testidata

Testidata on ohjelmistolle annettu syöte testin suorittamisen aikana. Testidata edustaa tietoja, jotka vaikuttavat tai joihin ohjelmiston suorittaminen vaikuttaa testauksen aikana. Testidataa käytetään sekä positiiviseen testaukseen että negatiiviseen testaukseen. Positiivisella testauksella varmistetaan, että toiminnot tuottavat odotettuja tuloksia. Negatiivinen testaus varmistaa ohjelmiston kykyä käsitellä epätavallisia, poikkeuksellisia tai odottamattomia syötteitä, esim. kielletyt merkit, tai numeron syöttäminen kenttään, johon odotetaan tekstisyötettä. Huonosti suunniteltu testidata ei välttämättä testaa kaikkia mahdollisia testiskenaarioita, mikä heikentää ohjelmiston laatua. (Hamilton, 2021c)

Testidataa voidaan luoda käsin, massakopioimalla tuotantodataa tai automatisoiduilla testidatan työkaluilla (Hamilton, 2021c). Testidatan luontiin on kaksi näkökulmaa, syötetty testidata ja todennäköisyysjakauman ylittäminen. Testidatan tarvitsee olla tarpeeksi pieni ollakseen käytännöllinen, mutta tarpeeksi suuri ollakseen eduksi. Testidatan luonnissa on oleellista, että tietojen valintakriteerit ovat mietittynä, jotta varmistetaan ja voidaan optimoida niiden edustavuus. Valintakriteereitä on kolmen tyyppisiä. Toiminnalliset kriteerit löytyvät vaatimusmäärittelystä ja edustavat tietoa, joka kattaa tuotteen kaikki toiminnot ja / tai palvelut. Rakenteelliset kriteerit edustavat dataa, jota voidaan käyttää kaikkien ohjelman komponenttien ja / tai ominaisuuksien testaamiseen. Satunnaisvalintakriteerit edustavat valintatietoa, joka jäljittelee tuotteen käyttötapaa, eli todennäköisyysjakaumaa. (Mili & Tchier, 2015)

3.2.3 Testioraakkeli

Elinkaaren kolmas vaihe on oraakkelin luonti. Testioraakkeli on lähde, jota käytetään määrittämään odotettuja tuloksia, joita verrataan testattavan ohjelmiston todellisiin tuloksiin. Testioraakkeli voi olla olemassa oleva järjestelmä (vertailua varten), muu ohjelmisto, käyttöopas tai henkilön asiantuntemus, mutta se ei saa olla koodia. (ISTQB, n.d.)

Testioraakkeli on mekanismi, jonka avulla määritetään, toimiiko testattava ohjelmisto järkevästi vai ei. Kun oraakkeli antaa epäilyttävän tai järjettömän lopputuloksen, asian tutkimiseen vaaditaan aina ihminen. Kun oraakkeli toteaa, että jokin käyttäytyminen on

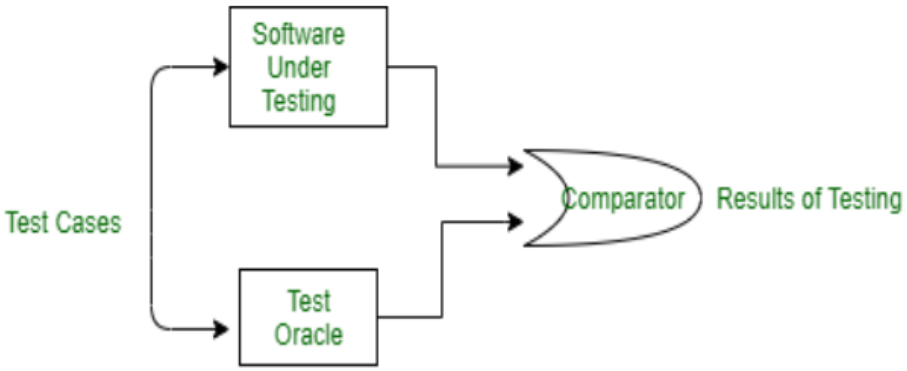
merkityksetöntä, siirrytään testissä eteenpäin. Onnistuneita testejä ei lähtökohtaisesti epäillä, vaan tutkitaan niitä, jotka epäonnistuvat. (Hoffman, 2019)

Jokaisessa testissä on jonkinlainen oraakkeli ja yhdessä testissä voidaan suorittaa useita vertailuja. Erilaisia käyttäytymismalleja voidaan tarkistaa oraakkelilla, esim. siirtyikö käyttäjä oikealle näytölle tai päivittyikö tietokantaan oikea tieto. Kun testeihin lisää testioraakkelin, testit vahvistuvat ja on todennäköisempää löytää virhe. Mitä enemmän oraakkeleita on käytössä, sitä enemmän voidaan löytää virheitä. Testatessa useimmiten tarkistetaan mitä ohjelmistossa tapahtuu. Hoffman (2019) toteaa, että testejä automatisoivat henkilöt, huomaavat usein testioraakkelin tarpeellisuuden. Varsinkin suuren volyymin automaatiotestauksessa tarvitaan hyviä oraakkeleita. (Hoffman, 2019)

Vertailuja voidaan tehdä kahdella tavalla. Ensimmäinen tapa on deterministinen lähestymistapa, se on binäärinen ja yksinkertainen, testi joko onnistui tai epäonnistui. Epäonnistunut tulos tarkoittaa, että käyttäytyminen on epänormaalia ja tulos ei ole odotetun lainen. Suurin osa testaajista testaa tällä tavalla. Toinen tapa on todennäköisyyspohjainen lähestymistapa, joka osaa kertoa milloin käyttäytyminen vaatii lisätutkimuksia, mutta ei tarkoita välttämättä, että virhe on löytynyt. Todennäköisyyspohjaisia testituloksia analysoimalla selvitetään ohjelmiston käyttäytymisen syitä henkilön toimesta. (Hoffman, 2019)

Vertailua voidaan siis suorittaa testin sisällä tai siitä voidaan luoda lokitiedosto, josta voidaan jälkikäteen tarkastaa, tekikö testi sen mitä haluttiin, vai sen mitä siltä odotettiin. Kuva 3 havainnollistaa testioraakkelin toimintaa. Testin sisällä tehty vertailu on useimmiten sisällytetty automaatiotesteihin. (Hoffman, 2019)

Kuva 3 Testioraakkeli (Upadhyay, 2021)



Seuraavaksi esitellään opinnäytetyön kannalta oleellisimmat testioraakkelin tyypit, sillä niitä on useita erilaisia.

Yksi tyypeistä on ei-oraakkeli (the No Oracle), jossa testit vain suoritetaan eikä tuloksia tarkasteta. Ei-oraakkelilla huomataan ohjelmiston kaatumiset tai muut vastaavat isot virheet. Toinen tyyppi on itsenäinen toteutus, jossa oraakkelin osia luodaan itse. Itsenäisellä toteutuksella useimmiten kattaa tietyn alueen syötteet, joitain muuttujia ja jopa tuloksia, mutta sillä myös luodaan virheelliset onnistuneet tulokset. Itsenäinen toteutus on myös ohjelmisto itsessään, mitä monimutkaisempi oraakkeli, sitä todennäköisemmin siinä on virheitä. Joissain tapauksissa oraakkeli on monimutkaisempi kuin testattava ohjelmisto, eikä silloin voida luotettavasti tehdä vertailua. (Hoffman, 2019)

Kolmas on kahden ensimmäisten tyyppien yhdistelmä. Tämä testioraakkeli tarkistaa muutokset ja erot testiajojen välillä. Tarkistus voi olla synkroninen tai asynkroninen. Synkroninen tarkistus tehdään samanaikaisesti testin kanssa ja on yleensä osa itse testiä. Asynkroninen tarkistus on riippumaton siitä mitä testissä tapahtuu, esim. muistovuotojen tarkistukset ovat asynkroniset. Automaatiotestauksessa useimmiten käytetään tätä yhdistelmää testioraakkeleista, jossa on sekä validoituja että validoimattomia tietoja. Validoitu tieto on tarkistettu tieto ja sen uskotaan olevan oikein. Validoitua tietoa voidaan pitää eräänlaisena referenssitietona. Suurimmaksi osaa testit sisältävät validoimatonta tietoa, niitä ei ole erikseen tarkistettu. Yleisin käytetty metodi on luoda aktiviteettiloki ja verrata sitä aiemmin luotuun lokiin. Validoitua lokitiedostoa voidaan myös kutsua referenssitiedoksi, sen sisältämät tiedot tiedetään oikeiksi ja vertaillaan suoritettuja aktiviteettilokeja siihen. (Hoffman, 2019)

Neljäs testioraakkelin tyyppi on ihmisoraakkeli, joka on kaikista yleisin oraakkeli. Henkilö tarkkailee ohjelmistoa ja seuraa mitä tapahtuu. Ihmisoraakkelia käytetään aina kun uskotaan, että ohjelmistossa on vika. (Hoffman, 2019) Ihmisoraakkelit voivat tehdä virheitä, tuloksia lasketaan käsin ja voi olla vaikeaa määrittää, vastaako tulos odotettua tulosta. Ihmisoraakkelit käyttävät määrittelyjä hyödyksi päättääkseen, toimiiko järjestelmä oikein. Oikean toiminnan määrittämisessä on tärkeää, että järjestelmän tai komponentin käyttäytyminen on tarkasti määritelty. (Upadhyay, 2021) Automatiikkaa vian tutkimiseen ei ole vielä keksitty (Hoffman, 2019).

Kaikki testit sisältävät oraakkeleita ja erityyppisiä oraakkeleita on useita. Kaikkea ohjelmistoon syötettyä tietoa on mahdoton tarkistaa, eikä niihin vaikuttavia vaikutuksia voida hallita kokonaan. Hyvän testioraakkelin ei tarvitse olla deterministinen ollakseen hyödyllinen, eikä sen tarvitse tietää tarkkaa tulosta. Testitulosten pitäisi todentaa esim. jokin laskukaava ja kertoa että käytetyt aputiedostot on suljettu, eikä tämä vaikuttanut muihin käynnissä oleviin toimintoihin. (Hoffman, 2019)

3.2.4 Lopetusehto

Testaus on tavoitteellista toimintaa, pitää tietää mikä on testin tavoite ja onko tavoite saavutettu. Testauksen tavoitteena voi olla esim. löytää vikoja, todistaa ettei vikoja ole, arvioida täyttääkö järjestelmä käyttäjän odotukset, arvioida luotettavuutta, saavuttaa liiketoimintatavoitteet jne. Lopetusehtoa varten tarvitaan tieto tavoitteen saavuttamisesta, tavoitteita voi olla erilaisia ja silloin tarvitaan myös erilaisia ehtoja. (Mili & Tchier, 2015) Lopetusehdoissa on yksi ennalta määritetty ehto. Jos lopetusehto vastaa testitulosta, luodaan testin yhteenvetoraportti. Yhteenvetoraportti sisältää tiedon tehdyistä testaustoimista ja lopullisista testituloksista. (Khimani, 2020)

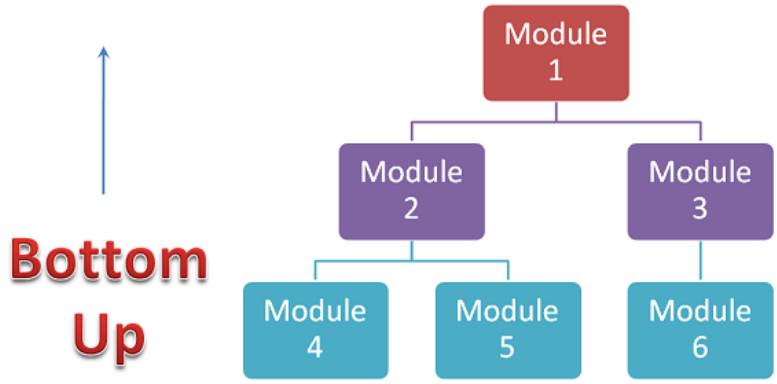
3.2.5 Testiajuri

Testiajuri simuloi tiedonsiirtoa puuttuvien tai puutteellisten komponenttien moduulien välillä. Ohjelmistotestaajat käyttävät testiajureita korvaamaan testauksen aikana tilapäisesti moduulin, joko puuttuvan tai kehityksen alaisen. Testiajurit täyttävät käytettävissä olevien

moduulien välttämättömät vaatimukset ja ovat hyödyllisiä odotettujen tulosten saavuttamisessa. (ProfessionalQA, 2020)

Kuva 4 esittää alhaalta ylöspäin suuntautuvaa testausstrategiaa, jossa alemman tason moduulit testataan ensin. Tällä tavalla ei hukata aikaa odotellessa, että kaikki moduulit olisivat valmiina ennen testauksen aloittamista. Testattuja moduuleita käytetään edelleen helpottamaan seuraavan tason moduulien testausta. (Hamilton, 2021b)

Kuva 4 Alhaalta ylöspäin suuntautuva -testaus (Hamilton, 2021b)



3.2.6 Testin suoritus

Testin suoritus voi olla manuaalinen tai automatisoitu. Jos testitapauksen suorituksen aikana havaitaan vika tai vikoja, niistä ilmoitetaan yleensä jonkin vianseurantajärjestelmän kautta. Kun kehittäjä on korjannut vian, suoritetaan testitapaus uudelleen. Jos testitapauksen tulos on epäonnistunut, se merkataan epäonnistuneeksi. Jos testitapauksen tulos vastaa sille asetettua tavoitetta tai tulosta, merkataan testitapaus onnistuneeksi. (Khimani, 2020)

3.2.7 Testitulosten analysointi

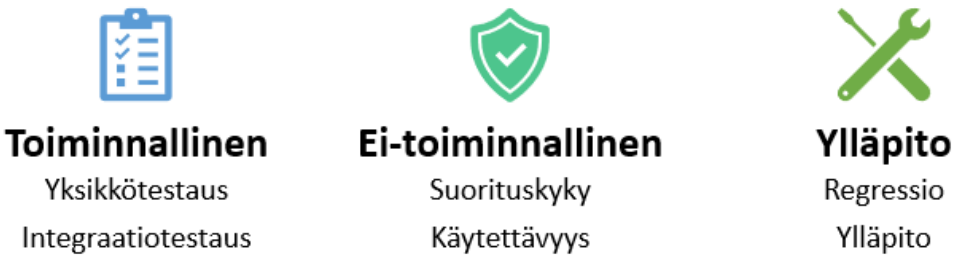
Ei tarvitse testata, ellei ole valmis analysoimaan testituloksia ja tekemään johtopäätöksiä. Testitulosten analysointi koostuu kaikkien suoritettujen testien raporttien tarkistamisesta ja johtopäätösten tekemisestä testien tavoitteiden mukaisesti. Jos testi epäonnistui, haetaan ratkaisu siihen missä vika on ja miten se korjataan. Jos testi onnistui, voiko siitä päätellä, että ohjelma on luotettava ja että se toimii oikein. (Mili & Tchier, 2015)

3.3 Erilaiset ohjelmistotestaustavat

Ohjelmistotestauksessa vertaillaan uuden toteutuksen ja ohjelmiston määrittelyjä toisiinsa, jotta varmistetaan ettei mitään jää puuttumaan. Ohjelmistotestaus myös varmistaa, että ohjelmisto toimii oikealla tavalla eri laitteilla tai olemassa olevan integraation kanssa, siksi se on yksi ohjelmistokehityksen laadunvarmistuksen muodoista. (Thelin, 2020)

Ohjelmistoja testataan yleensä kolmen eri kategorian testaustyypeillä. Testauskategoriat ovat toiminnallinen testaus, ei-toiminnallinen testaus sekä ylläpitotestaus. Kuva 5 on luotu materiaalin pohjalta ja esittelee vain muutamia testaustyyppejä, sillä niitä on tunnistettu yli 150 kpl. (Hamilton, 2021e)

Kuva 5 Testauskategoriat ja -tyypit (Hamilton, 2021e)



Ohjelmiston testaamiseen on monia eri tapoja. Kaikki testaustavat voidaan karkeasti jakaa Black Box- tai White Box -testauksen alle. Ero näiden välillä on siinä mitä testaaja tietää. Black Box -testauksessa testaaja tietää mitä ohjelmisto tekee, mutta ei miten se toimii. Useimmiten testaaja on kehitysprosessin ulkopuolinen henkilö, joka antaa mielipiteensä ohjelman käyttäytymiseen ja arvioi käyttäjäkokemustaan. White Box -testaus on Black Box -testauksen vastakohta. Testaaja tietää miten ohjelmisto on rakennettu ja miten ohjelmiston tulisi toimia. Testaaja on usein itse osallisena kehitysprosessissa ja arvioi ohjelmiston logiikkaa tarkoin määritellyn testitapauksen avulla. (Thelin, 2020)

Testaustapa, joka yhdistää Black Box- ja White Box -testauksen on Gray Box -testaus. Gray Box -testauksessa testaajalla ei ole tarkkaa tietoa mitä ohjelmisto tekee, mutta tietää osittain, miten ohjelmisto toimii ja tietää mitkä sen vaatimukset ovat. Gray Box -testausta käytetään web-sovelluksissa ja on hyödyllinen mm. integraatiotestauksessa. Testaustavan

päätavoitteena on löytää ongelmia, jotka johtuvat sovelluksen virheellisestä rakenteesta tai sen vääränlaisesta käytöstä. (Software Testing Help, 2021a)

Ohjelmistotestaus voi olla myös suunnittelematonta, jolloin puhutaan tutkivasta testauksesta. Testaajalla ei ole käytössään testitapausta, eikä välttämättä noudata mitään ennalta määrättyä kaavaa. Tutkiva testaus voi olla hyvinkin tehokasta, sen avulla voi paljastua käyttötapauksia, joita ei osattu ottaa huomioon ohjelmiston suunnitteluvaiheessa. (Pyykkö, 2010)

Pyykkö (2010) toteaa työssään, että testauksen automatisointia pidetään tärkeänä ketterissä menetelmissä. Ohjelmiston kehitystyössä testaajat osallistuvat aikaisessa vaiheessa ja kehittäjät saavat nopeasti palautetta työstään, jota ketterät menetelmät korostavat ja painottavat. Ohjelmistotestauksen automatisointi voi kuulostaa houkuttelevalta, kun ohjelmiston koko ja sen monimutkaisuus kasvaa. Manuaalinen ohjelmistotestaus ei takaa tällöin enää nopeita ja toistettavia tuloksia, jotka ovat keskeisiä piirteitä ketterissä menetelmissä ja automatisoinnissa. Ohjelmistotestauksen automatisoinnilla voidaan säästää aikaa, optimoida toteutusta ja lisätä laajuutta testaukseen. Tilanteita, joissa automatisoiduilla testeillä ei löydetä paremmin virheitä, voidaan välttää käyttämällä testien suunnitteluun aikaa. (Pyykkö, 2010)

Edellä mainitut Black Box-, White Box- ja Gray Box -testaustavat voidaan kaikki automatisoida. Tutkivaa testausta ei voida automatisoida, sillä kone suorittaa testin täsmällisesti sille annettujen parametrien mukaisesti.

3.4 Regressiotestaus

Seuraavissa alaluvuissa käsitellään regressiotestauksen tarkoitusta sekä regressiotestausta automaation näkökulmasta. Tarkastellaan myös regressiotestauksen tarvetta ohjelmistojen ylläpidossa sekä ketterään kehitykseen liittyen.

3.4.1 Regressiotestauksen tarkoitus ja määritelmä

Pienillä muutoksilla voi olla suuria seurauksia, toteaa Kinsbruner (2019b) artikkelinsa alussa. Regressiotestauksella varmistetaan, että koodiin tehdyt muutokset tai päivitykset eivät aiheuta virhetilannetta, eivätkä riko mitään. Ilman regressiotestausta virheiden havaitseminen on aikaa vievää ja kallista. Ohjelmistoon tehdyt päivitykset muuttuvat hyödyttömiksi, jos ne rikkovat muita osia ohjelmistossa. (Kinsbruner, 2019b)

Regressiotestausta tehdään myös varmistamaan, ettei tehdyillä päivityksillä ole sivuvaikutuksia olemassa oleviin toimintoihin. Regressiotestauksen tarve syntyy aina kun koodia muutetaan, lisätään uusi ominaisuus, korjataan vikoja tai suorituskykyongelmia. Regressiotestausprosessin suoritus alkaa virheen tunnistamisella. Kun virhe on tunnistettu, tehdään tarvittavat muutokset ja suoritetaan testaus uudestaan. Regressiotestaukseen valitaan testitapaukset, jotka kattavat sekä muokatun koodin että vaikutuksenalaiset osat. Ohjelmiston ylläpito, joka on yksi testauskategorioista, kattaa regressiotestauksen osalta kaiken toiminnan, joka liittyy parannuksiin, virheenkorjauksiin, optimointiin ja olemassa olevien ominaisuuksien poistamiseen. Nämä ylläpidon ominaisuudet voivat saada järjestelmän toimimaan väärin. Tehokkaalla regressiostrategialla säästetään aikaa ja rahaa. Erään tapaustutkimuksen mukaan regressiotestaus säästää virheenkorjauksissa jopa 60 % aikaa ja 40 % rahaa. (Hamilton, 2021d)

Regressiotestauksen haasteet liittyvät useimmiten testisarjaan, eli testattavat testitapaukset, joka helposti kasvaa erittäin isoksi. Aika- ja mahdollisten budjettirajoitusten vuoksi koko regressiotestisarjaa ei voida suorittaa. Mahdollisimman suuren testikattavuuden saavuttaminen samalla kun testisarjaa pienennetään, on haastavaa. Regressiotestitiheyden määrittäminen voi myös olla haastavaa, eli kuinka usein regressiotestausta suoritetaan, se voi olla jokaisen muokkauksen jälkeen, koontipäivityksen tai julkaisun yhteydessä tai vasta usean virheenkorjauksen jälkeen. (Hamilton, 2021d)

3.4.2 Regressiotestaus automaation näkökulmasta

Ketterässä kehityksessä regressiotestausta tehdään useimmiten testiautomaation ja testaustyökalujen avulla. Automatisoinnilla pyritään kattavampaan testaukseen ja lisäämään

siihen tarkkuutta, kun testitapausten määrä kasvaa. Kun muutoksia ja / tai päivityksiä tehdään paljon, uusia toimintoja iteroidaan ja käyttäjiltä saadaan palautetta tuotannosta, voi olla vaikeaa pysyä muutosten mukana. (Kinsbruner, 2019b)

Regressiotestauksessa käytettyjä testitapauksia tarkastellaan aina kun muutoksia ja / tai päivityksiä on tehty ohjelmistoon. Osa testitapauksista voivat olla käyttökelvottomia muutetussa ohjelmistossa. Varsinkin ne, joissa testataan muutoksen kohteena ollutta toimintoa. Testitapausten tarkastelusta käytetään nimitystä uudelleen validointiprosessi. Testitapauksen uudelleen validoinnissa on tarkoitus löytää käyttökelvottomat testitapaukset ja uudelleen validoida ne käytettäväksi regressiotestauksessa tai mahdollisesti jopa poistamaan, mikäli uudelleen validointi ei ole mahdollista. Prosessissa tarkastellaan testitapausten syötettä ja odotettua tulosta. Mikäli syöte on kelvoton, testitapaus voidaan poistaa tai käyttää negatiivisena testitapauksena. Mikäli syöte on kelvollinen, mutta testitapauksen odotettu tulos on kelvoton, pitää se päivittää vastaamaan muuttunutta toimintoa. (Holopainen, 2005)

Garousin ja Mäntylän (2016) artikkelissa, jossa he tutkivat milloin ja mitä kannattaa automatisoida ohjelmistotestauksessa, lopputulos oli että, yleisimmät yksittäiset vaikuttavat tekijät olivat regressiotestauksen tarve, taloudelliset tekijät sekä testattavan ohjelmiston kypsyys.

3.5 Automaatiotestaus

Automaatiotestaus on testaustekniikka, jolla testataan ja verrataan tulosta odotettuun tulokseen. Automaatiotestaukseen käytetään testiskriptejä tai automaatiotestaustyökaluja. Automaatiotestaus on suunniteltu nopeuttamaan testausprosessia ja käytetään useimmiten automatisoimaan toistuvia tehtäviä ja tehtäviin, joita on vaikea suorittaa manuaalisesti. Testiautomaatiotyökalut voivat kattaa useampia käyttöjärjestelmiä, laitteita ja testitapauksia kuin manuaalinen testaus. (Software Testing Help, 2021b)

Automaatiotestauksen käyttöönotto voi epäonnistua, jos ajoitus ei ole oikea, lähestymistapa on väärä tai asiayhteys vääristyy. Päätös, mitä ja milloin automatisoidaan testausta, on erittäin tärkeä. Väärällä päätöksellä on suuri vaikutus ja johtaa useimmiten pettymyksiin ja

vaikuttaa menoihin. Menoina tarkoitetaan resursseja ja tehtyä työtä. Päätöksenteon tukena voi käyttää erilaisia ohjeita siitä, mitä ja milloin automatisoidaan testaus. (Garousi & Mäntylä, 2016)

3.6 Manuaalinen testaus vs. automaatiotestaus

Manuaalinen testaus on perinteisin testaustyyppi. Manuaalisessa testauksessa testaajat luovat testitapaukset ja suorittavat ne käsin. Testit suoritetaan yksitellen ja yksilöllisesti. Testaaja suorittaa testit askel askeleelta, ilman testiskriptiä. Manuaalisessa testauksessa testaaja vahvistaa tärkeimmät ominaisuudet ohjelmistossa. Yleensä manuaalinen testaus on aikaa vievää, mutta hyödyllistä. Tämän tyyppisessä testauksessa testaajat ja kehitystiimi ovat vahvasti mukana kaikessa, aina testitapauksen kirjoittamisesta testin suorittamiseen. (Kinsbruner, 2019a)

Automaatiotestauksessa testit suoritetaan automaattisesti testaustyökalujen avulla. Testaajat käyttävät työkaluja ja komentosarjoja automatisoidakseen testaustyöt. Automaatiotestauksella suoritetaan määrällisesti enemmän testitapauksia ja parannetaan testien kattavuutta. Automaatiotestauksessa tarvitaan osaavia henkilöitä, he kirjoittavat testitapauksen perusteella testiskriptin, joka automatisoi testin suorittamisen. Automaatiotestaus vaatii koodausta ja testiskriptien ylläpitoa. (Kinsbruner, 2019a)

Suurin ero manuaalisella testauksella ja automaatiotestauksella on testin suorittaja. Manuaalisen testauksen suorittaa henkilö, kun automaatiotestauksessa testin suorittaa työkalu. Testaustapoja yhdistää tarkoitus havaita virheet, ennen kuin ne päätyvät ohjelmiston tuotantoympäristöön. Automaatiotestaus on tehokkaampaa ajankäytöllisesti verrattuna manuaaliseen testaukseen. Henkilön tekemä testaus on virhealttiimpaa. Molemmilla testaustavoilla on vahvuutensa ja heikkoutensa. Manuaalisen testauksen vahvuutena on, että se käsittelee paremmin monimutkaisia testitapauksia. Automaatiotestauksen vahvuutena taas on testien nopeampi suorittaminen ja sen kattavuus muutoksiin. (Kinsbruner, 2019a)

Software Testing Help (2021b) kertoo tarinallisen esimerkin manuaalisen testauksen ja automaatiotestauksen välillä. Testaaja saa testattavakseen toiminnanohjausjärjestelmän,

joka sisältää 100 lomaketta. Testaaja aloittaa työn tutkivalla testauksella lomakkeesta 1. Lomake 1 sisältää noin 50 erilaista kenttää. Syöttämällä satunnaisia arvoja kenttiin ja lomakkeen lähettämiseen menee 20 minuuttia. Testaaja onnistuu saamaan virheilmoituksen aikaiseksi ja kirjaa tulokset vianhallintajärjestelmään. Seuraavana päivänä kehittäjä on korjannut ongelman. Testaaja testaa lomaketta 1 uudestaan samoilla vaiheilla mitä kirjasi muistiin, kun raportoi viasta. Testaaja ei onnistu toistamaan virhettä, joten ongelma korjaantui. Kehittäjä ilmoittaa seuraavana päivänä, että on tehnyt päivityksen lomakkeeseen 1. Testaaja suorittaa regressiotestausta lomakkeeseen 1, tarkistaen ettei päivitys aiheuttanut uutta ongelmaa. Testaamiseen meni jälleen 20 minuuttia ja testaaja kyllästyy saman lomakkeen testaamiseen. Kehittäjä ylläpitää lomaketta kuukausittain, tehden siihen teknisiä parannuksia sekä muutoksia käyttäjiltä saadun palautteen perusteella. Testaajan tulee testata sama lomake 1 kuukausittain. Testaaja pitkästyy testaamaan samaa lomaketta uudestaan eikä syötä jokaiseen kenttään enää arvoja. Testaajan eikä testin tarkkuus ole sama, jos vain puolet kentistä saa syötteen. Testaajalla on muitakin lomakkeita testattavana. Eräänä päivänä käyttäjä ilmoittaa virheestä lomakkeessa 1. Testaajaa harmittaa, että hän ei löytänyt virhettä ennen käyttäjää.

Tarina pätee 90 % manuaaliseen testaukseen, regressiotestaus on raskasta. Ihmiset eivät jaksa tehdä samaa asiaa samalla nopeudella ja tarkkuudella päivittäin. Automaatiota tarvitaan toistamaan vaiheet samalla tarkkuudella kuin ensimmäiselläkin kerralla. (Software Testing Help, 2021b)

Tauriainen (2011) pohtii automaatiotestauksen hyötyä artikkelissaan. Hän vertaili ohjelmistotestauksen näkökulmasta robottia ihmiseen. Robotti pärjää hyvin, kunnes törmää ennalta arvaamattomaan tapahtumaan. Robotti tarvitsee ihmisen selvittämään mistä virhe johtui ja korjaamaan sen. Robotti ei voi korvata ihmistä. Tauriaisen (2011) esimerkissä viisi ihmistä tekevät samaa työtä kuin robotti, ajallisesti ihmiset suoriutuvat tehtävästä nopeammin kuin yksin puurtava robotti. Automaation hyöty tulee esille siinä, kun robotille annetaan jokin tehtävä ja ihmiset voivat tehdä jotakin muuta. Testaukseen liittyen, ihmisiä voisi mieluummin käyttää parantamaan testauskattavuutta ja kitkemään virheet ohjelmistosta. Automaatiotestaus on erittäin hyvä työväline ihmiselle. Lopputulos vertailussa oli, että annetaan robotin huolehtia toimivaksi todetusta tehtävästä, jotta se ei enää mene huomaamatta rikki. (Tauriainen, 2011)

Taulukko 1 on luotu mukaillen materiaalia, jossa vertaillaan kohteita manuaalisen testauksen ja automaatiotestauksen välillä. Vertailukohteet valikoituivat automaatiotestauksen käyttöönottoa ajatellen ja mitkä mahdollisesti vaikuttavat päätöksentekoon.

Taulukko 1 Manuaalinen testaus vs. automaatiotestaus (Hamilton, 2021a)

Vertailukohde	Manuaalinen	Automaatio
Toteutus	Testitapauksen suorittaa henkilö	Testitapaus suoritetaan automaatiotyökalujen avulla
Kustannus	Pienempi alkuinvestointi, tuotto pitkällä aikavälillä alhainen, vaatii henkilöresurssin	Suurempi alkuinvestointi, tuotto pidemmällä aikavälillä suuri, vaatii testaustyökalun sekä henkilöresurssin
Kustannustehokkuus	Ei kustannustehokasta suuren volyymin regressiolle	Ei kustannustehokasta alhaisen volyymin regressiolle
Luotettavuus	Vähemmän tarkka ja virhealttiimpi inhimillisten virheiden mahdollisuuden vuoksi	Tarkempi, työkalujen ja komentosarjojen avulla testi suoritetaan aina samalla tavalla
Soveltuvuus	Soveltuu parhaiten tutkivaan-, käytettävyys- ja ad hoc-testaukseen, sekä usein päivittyvän sovelluksen testaukseen (AUT-testaus)	Soveltuu parhaiten regressio-, suorituskyky-, ja kuormitustestaukseen, sekä toistettaviin toiminnallisiin testitapauksiin

3.6.1 Manuaalisen testauksen hyödyt ja haitat

Muutokset ja päivitykset ohjelmistoon on testattava ensin manuaalisesti. Manuaalisen testauksen tarkoitus on varmistaa ohjelmiston virheettömyys. Kaikkia mahdollisia virheitä on kuitenkin mahdotonta tunnistaa. Manuaaliseen testaukseen tarvitaan aina henkilö, joka suorittaa testin. Monenlaisia sovelluksia voidaan testata manuaalisesti, ja tietyn tyyppiset testit sopivat paremmin manuaalisesti suoritettavaan testaukseen, kuten ad hoc- ja tutkiva testaus. Testaaja voi kokea manuaalisen testauksen tylsänä, sillä samoja tai samankaltaisia testejä suoritetaan toistuvasti. Testaajan on kyettävä olla kärsivällinen ja tarkka, mutta myös

luova kun suorittaa testausta. Monesti on samaistuttava asiakkaan rooliin testauksessa, jotta loppukäyttäjän tarpeet osataan huomioida ohjelmistossa. (Kinsbruner, 2019a)

3.6.2 Automaatiotestauksen hyödyt ja haitat

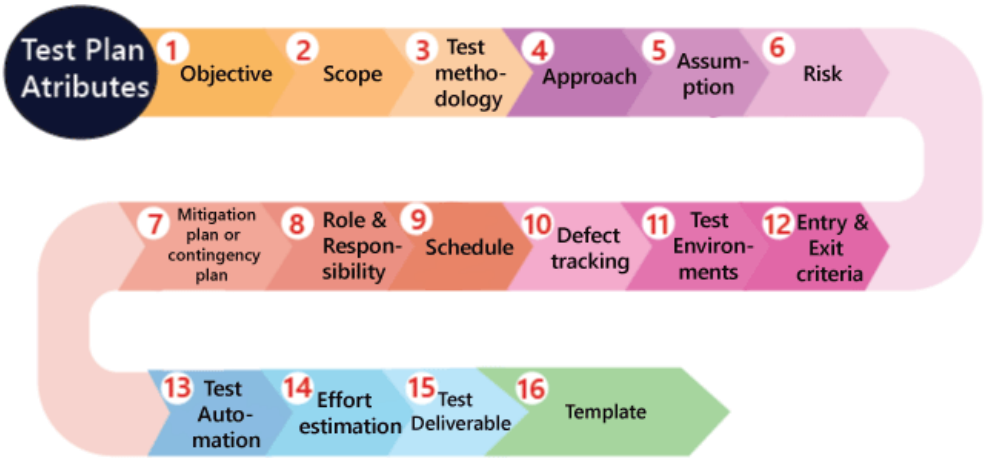
Automaatiotestaus ei tuo parempia tuloksia kuin manuaalinen testaus. Kaikkia testejä ei voida automatisoida, joten manuaalinen testaus on välttämätöntä. Kun manuaalinen testaus on suoritettu, voidaan sen perusteella kirjoittaa testiskripti automaatiota varten. Kun testiskripti on kerran toteutettu, se voidaan suorittaa yhä uudelleen, vaikka useita kertoja päivässä viikon jokaisena päivänä. Automaatiotestaus säästää aikaa, kustannuksia ja välillä jopa testaajan hermoja. Automaatiotestauksessa testaus suoritetaan työkalujen avulla, eikä työkalu väsy saman testiskriptin toistamiseen. Automaatiotestausta suositellaan vakaille ohjelmistoille ja käytetään usein regressiotestauksen apuna. (Kinsbruner, 2019a)

Automaatiotestauksen haasteina nähdään usein sen vaatima ohjelmointiosaaminen. Automaation vaatiman ohjelmointikielen oppiminen on varmasti hyödyllistä, mutta logiikan rakentamiseen ja kehittämiseen tulisi kiinnittää enemmän huomiota. Automatisoinnin ei pitäisi olla vain muutamien resurssien vastuulla, vaan koko tiimin pitäisi osallistua. Testitulosten raportointi vaatii koordinointia ja ylläpitoa, mikä itsessään lisää kustannuksia. Automatisoituihin testeihin tulisi voida luottaa, sillä työtunteja investoidaan automaation rakentamiseen. Tämän vuoksi tulee pyrkiä ylläpitämään skriptejä, jotta tulokset olisivat luotettavia. (Roy, 2021)

4 Testaussuunnitelma

Testaussuunnitelma on asiakirja, joka kuvaa ohjelmiston tarkoituksen, laajuuden ja testausmenetelmän. Testaussuunnitelmaa voidaan pitää testaustyön ohjekirjana. Kuva 6 esittelee testaussuunnitelman eri osia. Testaussuunnitelmassa on kuvattu testauksen tavoitteet, testauksen laajuutta sekä yleinen aikataulu toiminnoille. Testauksen tavoitteilla tarkoitetaan mitä ohjelmistosta tarkistetaan ja / tai validoidaan. Testauksen laajuus määrittää mitä testataan ja mitä rajataan testauksen ulkopuolelle. Aikataulu sisältää tiedon, miten ja milloin testataan. Testaussuunnitelmassa tulee olla tieto mahdollisista riskeistä ja niiden tasoista, jotta testausta voidaan priorisoida riskien mukaan. Yksi tärkeimmistä tiedoista testaussuunnitelmassa on tarvittavien resurssien määrittely. Resurssit voivat olla henkilöitä ja teknisiä resursseja, kuten testiympäristöt, testaustyökalut ja testidata. (Rice, n.d.) Testaustyökaluista kirjataan tieto mitä työkaluja käytetään testaukseen ja virheraportointiin. Vianhallinnasta kirjataan yksityiskohtaiset tiedot siitä, miten vioista ilmoitetaan, kenelle ne ilmoitetaan ja mitä kuhunkin virheilmoitukseen liitetään, esim. kuvakaappaus tai testiloki. Testaussuunnitelma sisältää myös tiedon poistumisparametreistä, eli milloin testaus lopetetaan. Poistumisparametreissä on kuvattu tulokset, joita testaukselta odotetaan ja ne antavat samalla testaajille vertailukohteen todellisten tulosten vertailuun. (Bose, 2020) Ennen kuin oikea testaustapa ja muut suunnittelun yksityiskohdat voidaan määritellä, tarvitaan laajempi näkymä organisaation tavoitteista. Tarvitaan tieto mitä testauksella halutaan saavuttaa, eli testausstrategia, joka on osa testisuunnitelmaa. (Rice, n.d.)

Kuva 6 Testaussuunnitelman osat (Jaiswal, n.d.)



Testaussuunnitelman laatii tyypillisesti testausvastaava tai testauspäällikkö. Hyvänä lähtökohtana testaussuunnitelman luonnissa on testausstrategian määrittely, joka auttaa ymmärtämään testauksen tavoitteita. Testaussuunnitelmassa tarvittavia tietoja ei aina ole heti saatavilla, vaan ne voivat tulla esille myöhemmin. Testaussuunnitelman luontia verrataan tutkimustyöhön, siinä ohjelmistoa tutkitaan ja yritetään selvittää sen yksityiskohtia. Hyvä käytäntö on antaa osa testaussuunnitelmasta muun tiimin dokumentoitavaksi. Testaussuunnitelman laatija kokoaa ja muokkaa tiedot lopulliseen muotoonsa. Tärkeä osa testaussuunnitelmaa on myös sen katselmointi. Asiasisällön tunteva tiimi katselmoi suunnitelman ensin. Muutosten jälkeen katselmointiin voivat osallistua asiantuntijasidosryhmät ja muut, jotka voivat tarjota hyödyllisiä näkökulmia testaussuunnitelmaan. (Rice, n.d.)

4.1 Testausstrategia

Testausstrategia on yksi osa testaussuunnitelmaa. Testausstrategia on asiakirja, joka selventää testaussuunnitelmaa ja määrittelee testausprosessin vaiheet. Testausstrategia kuvaa mitä ja miksi ohjelmistoa testataan. Käytännössä testausstrategia on helpompi määritellä ennen testaussuunnitelmaa, jotta sen tavoitteet ymmärretään. Testausstrategian myötä on käytössä perustiedot yksityiskohtaisen testaussuunnitelman laatimista varten. Sidosryhmät kannattaa ottaa mukaan pohtimaan testausstrategiaa, jolloin ymmärrys testauksen tärkeydestä kasvaa. Testausstrategiassa ei tarvita yksityiskohtia, sen voi määritellä jopa ennen ohjelmiston vaatimusten määrittelyä. (Rice, n.d.)

4.2 Testitapaus

Testitapaus viittaa toimiin, joita vaaditaan tietyn ominaisuuden tai toiminnallisuuden tarkistamiseksi ohjelmistotestauksessa. Testitapaus voi perustua prosessikuvauksiin, ohjelmiston määrittelyihin ja koulutusmateriaaleihin. Manuaalisen testitapauksen komponentit ovat testattavan kohteen nimi, attribuutit ja kuvaus. Automaatiotestitapauksilla on muitakin ominaisuuksia. Testitapauksen kuvaus on tarkoitettu pääasiassa testaajille, olivatpa he manuaalisia testaajia tai testiautomaation asiantuntijoita. Testitapauksen kuvaus sisältää karkean kuvauksen suoritettavista testitoiminnoista. Siinä on tiedot kaikista vaadituista valmisteluista ja edellytykset testin suorittamiselle.

Lähtötilanteella tai valmistelulla tarkoitetaan esim. minkä tyyppinen asiakas on ja missä roolissa asiakas toimii järjestelmässä. Testitapaus on ohje testaajalle, miten ja missä järjestyksessä tehdään asioita järjestelmässä. Testitapauksen kuvauksessa tulee myös olla tieto odotetusta tuloksesta tai tarkistuksista, jotka tulee suorittaa testin aikana. Verifiointivaiheilla testaaja varmistaa toimiiko järjestelmä kuten sen pitääkin. Testitapaukset suositellaan luotavaksi niin, että testitapauksen kirjoittaja ja testaaja eivät ole sama henkilö. (PAT Research, n.d.)

4.2.1 Testitapausten valinta regressiotestaukseen

Regressiotestausta tarvitaan aina kun ohjelmistoon lisätään uusi ominaisuus, korjataan vikoja tai suorituskykyongelmia. Regressiotestauksen kustannukset kasvavat, jos ohjelmistoa muutetaan usein. Regressiotestausta varten valitaan testitapauksia, jotka kattavat sekä muokatun koodin että sen vaikutuksenalaiset osat tai toiminnot. Tehokkaiksi regressiotesteiksi luokitellaan testitapaukset, joissa on usein vikoja, näkyvät käyttäjille, ovat raja-arvotapauksia sekä toiminnallisuudet, joihin on tehty paljon muutoksia viime aikoina. Lisäksi regressiotestaukseen tulisi valita testitapauksia, jotka varmistavat järjestelmän ydinominaisuudet sekä kaikki integraatio- ja monimutkaiset testitapaukset. (Hamilton, 2021d)

4.2.2 Testitapausten valinta automaatiotestaukseen

Manuaalisia testitapauksia automatisoidaan parantamaan tuottavuutta ja kattavuutta. Yksi tärkeimmistä vaiheista on sopivien testitapausten valitseminen ja sijoitetun pääoman tuottoasteen (return on investment) määrittäminen. Automaatiotestausta varten tarvitaan strategia, seurantaa ja ohjausta. Oikein toteutettu automaatio on hyödyksi tiimille, projektille ja viime kädessä myös organisaatiolle. Testitapauksia valittaessa, kannattaa selvittää tavat, jolla saadaan korkeampi ja positiivinen tuottoaste sijoitetulle pääomalle. Royn (2021) mukaan automaatiotestaukseen kannattaa valita testitapauksia, joihin liittyy monimutkaista liiketoimintalogiikkaa, ovat vaikeita suorittaa manuaalisesti tai on tarve monialustaiselle testaukselle, esim. erilaiset käyttöjärjestelmät ja selaimet. Myös testitapaukset, joissa tarvitaan erilaisia käyttäjiä, sisältävät suuren määrän dataa, joilla on

jokin riippuvuus ja jotka vaativat erikoistietoja, ovat hyviä automatisoinnin kohteita. (Roy, 2021)

Sijoitetun pääoman tuottoasteen määrittäminen voidaan tehdä pilkkomalla järjestelmän osat moduuleiksi. Moduulit analysoidaan ja tunnistetaan testitapaukset, jotka tulisi automatisoida parametrien perusteella (kuva 7). Tämän jälkeen moduulit ryhmitellään ja yhdistetään testitapausten määrään, jolloin saadaan määrällinen arvio testauksen suorittamisesta manuaalisesti, esimerkkinä kuva 8. Ryhmittelyn jälkeen tiedot voidaan vielä yhdistää (kuva 9), ja tieto on helpommin hyödynnettävissä sijoitetun pääoman tuottoasteen laskennassa. (Roy, 2021)

Kuva 7 Testitapausten jaottelu parametreittäin (Roy, 2021)

Module ID
Module Name

TC Id	Description	With different set of data?	Different browser	Different environment	Complex business logic	Different set of users	Involves large amount of data	Any dependancy	Requires Special data	Candidate for automation	Comments
TC01	xyz	Y	Y	Y	N	N	N	N	N	Y	Same test cases for n number of time with different data
TC02	xyz	Y	Y	Y	N	Y	N	Y	Y	Y	
TC03	xyz	N	N	N	Y	Y	Y	Y	Y	Y	Has complex logic which may go wrong when executed manually
TC04	xyz	N	N	N	N	N	N	N	Y	N	
TC05	xyz	Y	N	N	N	Y	N	N	Y	N	Requires to be executed with only 2 set of data
TC06	xyz	Y	Y	N	Y	N	Y	N	Y	Y	

Kuva 8 Testitapausten ryhmittely määrällisesti (Roy, 2021)

Module	No. of TCs with different set of data?	No. of TCs with different browser	No. of TCs with different environment	No. of TCs with Complex business logic	No. of TCs with Different set of users	No. of TCs which Involves large amount of data	No. of TCs having dependancies	No. of TCs which requires Special data	Time (In Mins) to execute manually
Mod 1	3	3	2	2	2	2	2	3	180
Mod 2	x	x	x	x	x	x	x	x	x
Mod 3	x	x	x	x	x	x	x	x	x
Mod 4	x	x	x	x	x	x	x	x	x
Mod 5	x	x	x	x	x	x	x	x	x

Kuva 9 Testitapausten laskennalliset tiedot moduuleittain (Roy, 2021)

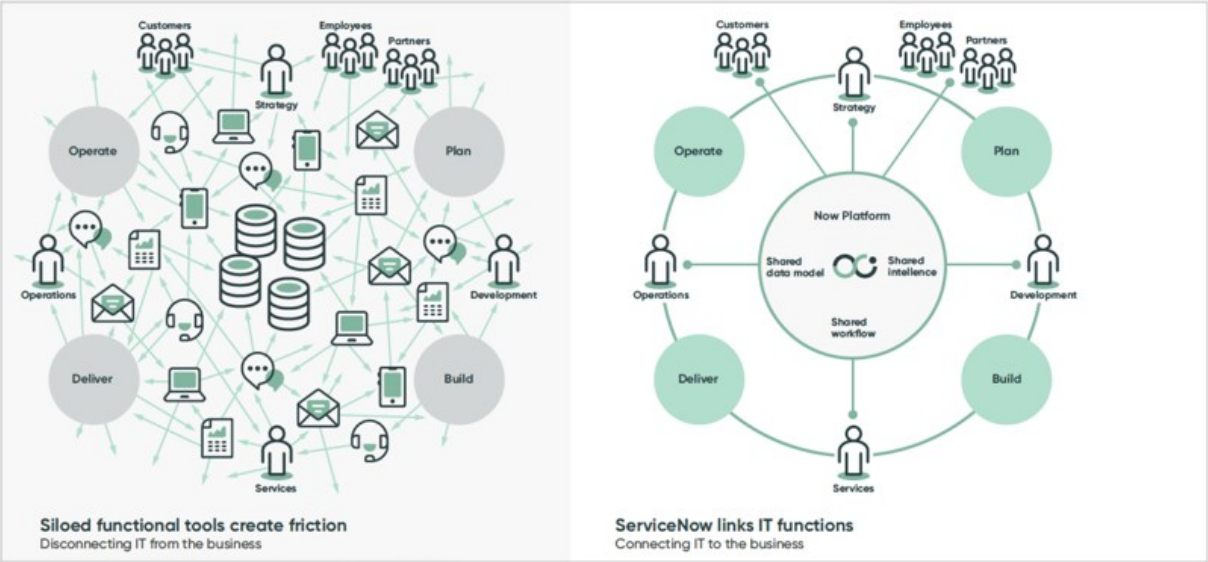
Name of module	Total number of test cases to Automate	Time required to execute manually	Time to run using automation scripts
Mod 1	19	5 hours	1.5 hours
Mod 2	x	x	x
Mod 3	x	x	x

Tässä luvussa käsiteltiin tärkeimmät kohdat testaussuunnitelmaan ja sen sisältöön. Testaussuunnitelma kertoo, miten testataan, milloin testataan ja kuka tarkistaa testitulokset. Testausstrategia kertoo mitä tekniikkaa tulisi noudattaa testauksessa. Testitapaus sisältää tietoa testaajalle millä ehdoilla, syötteillä ja missä järjestyksessä etenemällä pitäisi saavuttaa odotettu lopputulos. Regressiotestaukseen soveltuu parhaiten testitapaukset, jotka varmistavat järjestelmän ydinominaisuudet ja ne, jotka kattavat sekä muokatun koodin että sen vaikutuksenalaiset osat tai toiminnot. Automatisoitaviksi kohteiksi soveltuvat testitapaukset, joihin liittyy monimutkaista liiketoimintalogiikkaa ja ovat vaikeita suorittaa manuaalisesti.

5 ServiceNow

ServiceNow on perustettu 2004 Yhdysvalloissa ja sitä pidetään yhtenä lupaavimmista teknologiayrityksistä tällä hetkellä. ServiceNow on käytössä yli 6 000 yrityksessä maailmanlaajuisesti. (AW Academy, n.d.) ServiceNow on pilvipohjainen IT-palvelunhallinnan (IT Service management, ITSM) ohjelmistoalusta, jolla voidaan automatisoida IT-liiketoiminnan hallinta (kuva 10). ServiceNow pohjautuu ITIL prosessiviitekehykseen ja tarjoaa palvelulähtöisen näkökulman tehtäville, toiminnoille ja prosesseille. (Taylor, 2021)

Kuva 10 ServiceNow IT-palvelunhallinta (ServiceNow, 2021e)



5.1 Liiketoimintaratkaisut

ServiceNow’n tarjoamat pilvipalvelut sisältävät viisi liiketoimintakategoriaa, jotka ovat IT, henkilöstöhallinto, asiakaspalvelu, tietoturva ja liiketoiminta. ServiceNow tarjoaa työkaluja IT-palvelun-, IT-toimintojen-, IT-liiketoiminnan- ja IT-omaisuuden hallintaan. Näiden avulla voidaan hallita projekteja, ryhmiä ja asiakasvuorovaikutusta erilaisten sovellusten ja lisäosien kautta. ServiceNow voidaan integroida muihin työkaluihin ja siihen on saatavilla myös kolmansien osapuolien kehittämiä sovelluksia. Kaiken edellä mainitun lisäksi erilaiset työnkulut ovat osa järjestelmän ydintoimintoja. (Fitzgibbons, 2020)

ServiceNow-alusta käyttää koneoppimista hyödyntämällä dataa ja työnkulkuja, jotta sitä käyttävät yritykset pystyvät toimimaan nopeammin ja skaalautuvasti (Taylor, 2021). ServiceNow käyttää tekoälyä ja analytiikkaa tiedon tuottamiseen, ennustamiseen ja toistuvien tehtävien automatisoimiseksi. ServiceNow sisältää lukuisia ominaisuuksia, joiden avulla voidaan digitalisoida työnkulkuja ja lisätä niitä tarpeen mukaan. ServiceNow-sovellukset kuten Suorituskykyanalyysi (Performance Analytics), Ennakoiva älykkyys (Predictive Intelligence) ja Agentin työtila (Agent Workspace), ovat esimerkkejä näistä ominaisuuksista. (ServiceNow, n.d.)

5.2 Versiopäivitys

Versiopäivitysten tarkoitus on parantaa ServiceNow-alustan yleistä suorituskykyä. Versiopäivitykset voivat sisältää myös uusia moduuleja, sovelluksia, sekä parannuksia ja korjauksia olemassa oleviin tuotteisiin. Järjestelmän versiopäivitys voi olla työläs, mutta julkaisutietojen avustamana päivityksen vaiheet ja tehtävät suorittamalla takaa onnistuneen päivityksen. Julkaisutiedot ovat saatavilla ServiceNow-sivustolla. Kuva 11 esittää versiopäivityksen eri vaiheita. Ympäristöjen päivittäminen uuteen versioon vaatii suunnittelua, testausta ja validointia. Jos ympäristöihin ei halua tehdä versiopäivitystä kahdesti vuodessa, on mahdollista pitäytyä vuoden syklissä päivitysten suhteen. ServiceNow tukee nykyistä ja edellistä versiota, sitä vanhempaan versioon ei ole saatavilla tuotetukea. ServiceNow ilmoittaa suunnitellut julkaisupäivät kolme vuotta etukäteen. Julkaisut ovat osa ServiceNow tilaus- tai lisenssisopimusta, eivätkä täten aiheuta lisäkustannuksia. (Plat4mation, n.d.)

Kuva 11 Versiopäivityksen ohjeistus (Plat4mation, n.d.)

Viimeisimmässä versiopäivityksessä oli muutoksia ja lisäyksiä myös ATF-sovellukseen, lisäten testauksen kattavuutta ja parempia toimintoja. Suurin muutos sovelluksessa on päätön

testaus (headless testing). Käyttöliittymätestausta on parannettu päättömällä selaimella (Headless browser), jossa automatisoidaan selainten luonti ATF-testien käsittelyä varten. Muutoksen myötä voidaan testata käyttöliittymän toimivuutta ilman manuaalista selaimen avaamista testin suorittajaa (Client Test Runner) varten, jota käytetään käyttöliittymätestien käsittelyyn paikallisessa selaimessa. Muut uudet ominaisuudet ovat Select2-toiminnot, jolla voidaan hakea tietoa pudotusvalikoista, sekä tekstin asettaminen (AssertText on page) työtiloissa. (Tilton, 2021)

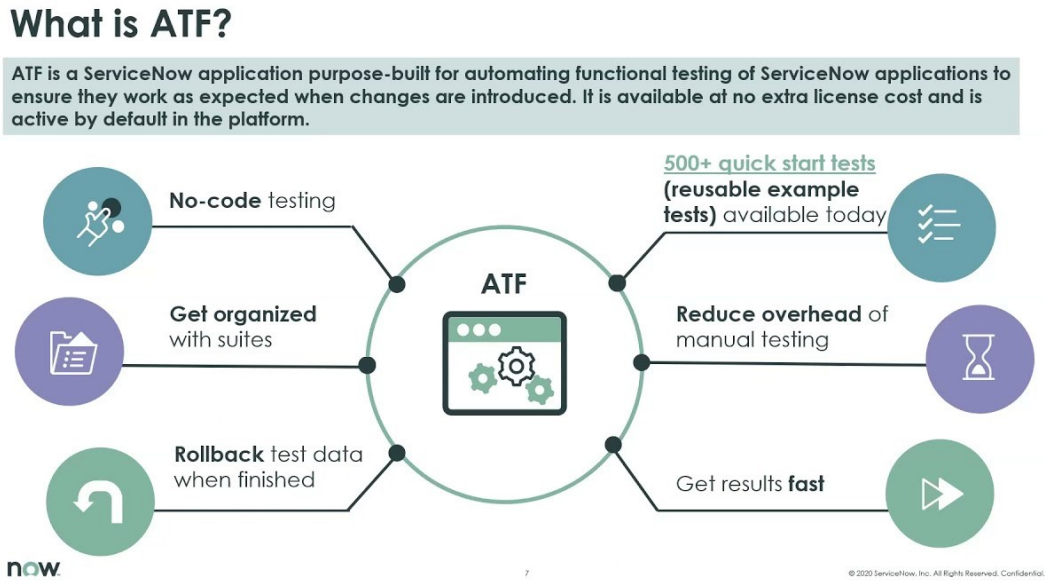
Seuraavan luvun jälkeen lukija, jolla on oma ServiceNow-ympäristö käytössään tai tekee töitä ServiceNow-kehityksen parissa, pystyy itsenäisesti ottamaan Automated Test Framework -sovelluksen käyttöön, sekä luomaan ja suorittamaan omia automatisoituja testejä.

6 Automated Test Framework -sovellus (ATF)

Pääluku esittelee Automated Test Framework -sovelluksen yleisellä tasolla ja kertoo sen hyödyistä. Alaluvuissa esitellään sovelluksen käyttötarkoitus ja parhaat käytännöt sen käyttöön kuin myös testien suunnitteluun. Tutustutaan myös sovellukseen liittyviin rooleihin, sen moduuleihin ja testauksen automatisointiin. Opinnäytetyön toiminnallinen osuus käsittelee testien luontia ja suorittamista sovelluksessa.

ATF on sovellustyökalu ja kuuluu ServiceNow'n Now Platform:in tarjoamiin sovelluksiin ja ominaisuuksiin. ATF-sovellus on itsessään ilmainen ja käyttää samoja kehitystyökaluja, jotka ovat tuttuja ServiceNow-kehittäjille. Sovelluksen avulla luodaan ja suoritetaan automatisoituja testejä. Sovelluksen testituloksia tarkastelemalla voidaan havaita mikä vaihe testistä on epäonnistunut ja millä tavalla. Epäonnistunutta vaihetta analysoimalla voidaan tunnistaa virheeseen johtanut muutos. ATF-sovelluksen mahdollisuuksista ja hyödyistä ovat kooditon testaus, pika-aloitustestit, testipaketit, manuaalisen testauksen kustannusten vähentäminen, testitietojen palautus, sekä nopeat tulokset. (ServiceNow, 2021a)

Kuva 12 Kuvaus ATF testaustyökalusta (ServiceNow, 2021a)



ServiceNow kuvailee ATF-sovelluksen hyötyjen olevan monipuoliset (kuva 12). Sovelluksen avulla voidaan kasvattaa ylläpito- ja kehitysaikaa korvaamalla manuaalinen testaus

automaattisella testauksella. Testiympäristöt palautetaan alkutilaansa poistamalla tehdyt testitiedot ja muutokset jokaisen testiajon jälkeen. Testipaketteja luomalla voidaan järjestää ja suorittaa testejä erissä, nämä voidaan myös ajastaa ja suorittaa testipaketit haluttuun aikaan. Sovellus vähentää testien suunnitteluun kuluvaa aikaa, sillä testipaketteja ja pika-aloitustestejä voidaan kopioida, myös testien uudelleenkäyttö on mahdollistettu. Sovelluksella voidaan myös luoda mukautettuja testivaiheita testauksen kattavuuden laajentamiseksi. Sovellus mahdollistaa ei-tekniselle henkilölle mahdollisuuden luoda testejä standardi Now Platform -toiminnallisuuksista (OOB-toiminnallisuudet). (ServiceNow, 2021a)

6.1 ATF-sovelluksen käyttötarkoitus ja parhaat käytännöt

Automated Test Framework (ATF) edistää luottamusta alustan vakauteen. ATF-sovellus tarjoaa toiminnallista testausta, joka mahdollistaa toiminnot kuten tietueiden luomisen, kenttäarvojen asettamisen ja niiden tulosten tarkistamisen. Jos testi epäonnistuu, sovellus kertoo, että tulos ei vastaa odotettua tulosta ja ilmoittaa yksityiskohdat taustalla olevasta ongelmasta tai virheestä. (ServiceNow, 2021d)

Ympäristön toimivuus varmistetaan suorittamalla ATF-testit aina kun siihen tehdään muutos, kehitystyötä tai versiopäivitys. Tämä tarkoittaa kaikkien laajuuksien (scope) sovelluksia, mukautettujen sovellusten tai valmisratkaisusovellusten (OOB-sovellus) ATF-testien suorittamista. ATF:llä testipakettien suorittaminen on nopeaa ja testituloksia analysoimalla selvitetään, vaatiiko muutokset korjaamista. Samoja testejä ja testipaketteja voidaan uudelleenkäyttää, jos niitä ylläpidetään tehtyjen muutosten osalta. ATF:llä on tarkoitus testata muutoksia, joita ServiceNow-ympäristöön tehdään, ei järjestelmän perustoimintoja. Käyttöliittymän toiminnallisuuksien testaaminen on sovelluksessa mahdollista ilman käyttöliittymää, jolloin voidaan välttää siihen tehtyjen muutosten aiheuttamat testien epäonnistumiset. Käyttöliittymäriippumattomuus on etu verrattuna kolmannen osapuolen automaattisiin testaustyökaluihin, joissa testien epäonnistumiset käyttöliittymämuutosten takia ovat yleinen ongelma. (ServiceNow, 2021d)

Lähdeartikkelia mukaillen on luotu taulukko 2 ATF:n soveltuvuus testaustyypeittäin.

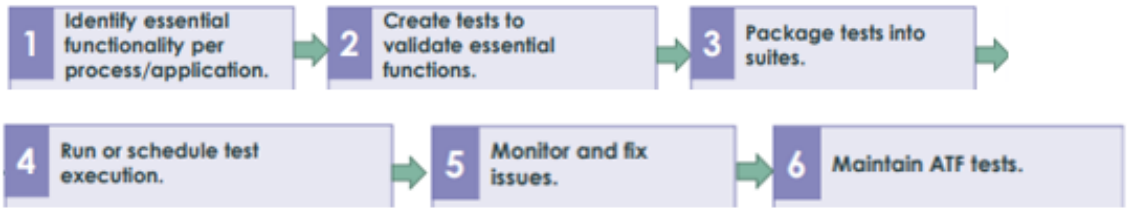
Taulukko 2 ATF:n soveltuvuus testaustyypeittäin (ServiceNow, 2021d)

Suositellaan	**Ei suositella**
ATF on tarkoitettu liiketoimintalogiikan toiminnalliseen testaukseen, joten sillä voidaan testata tiettyjä liiketoimintaprosesseja. Esim. jokaisella organisaatiolla on oma muutoksenhallintaprosessinsa. ATF-testissä luodaan muutospyyntö ja viedään se sen elinkaaren läpi, jotta voidaan varmistaa prosessin toimivuus.	ATF-testien suorittaminen tai muiden toiminnallisten testaustyökalujen käyttöä tuotantoympäristössä. Esim. ATF-testissä voidaan tekeytyä käyttäjäksi, jolla laajat oikeudet. Käyttäjä päivittää tietueen, jonka päivitys käynnistää tapahtuman, jolla lähetetään sähköposti-ilmoitus organisaation ylimmälle johdolle.
ATF testaa muutokset selain- ja käyttöjärjestelmäyhteensopivasti. ATF-testi luodaan kerran ja varmistetaan, että sama toiminto toimii odotetusti kaikissa tuetuissa selaimissa ja ympäristöissä.	ATF ei ole tarkoitettu ServiceNow-ympäristön yksittäisten käyttöliittymä-komponenttien testaamiseen. Hyvä nyrkkisääntö on testata asiaa, toimintoa tai prosessia, joka on omassa ympäristössä uniikki.
ATF on tehokas regressiotestauksessa. Luodaan kattavat testaussuunnitelmat, joilla varmistetaan, että nykyiset prosessit toimivat odotetulla tavalla päivityksen tai kehityksen jälkeen. Sama testaussuunnitelma suoritetaan aina ennen suuria muutoksia sekä niiden jälkeen lähtötilanteen saamiseksi.	ATF:ää voidaan käyttää, mutta ei suositella yksikkötestaukseen. ATF-testejä ei kannata luoda ominaisuuksille, jotka muuttuvat usein. On parempi luoda testit, kun muutokset alkavat olla vakaita, jotta ylimääräiseltä työltä vältytään.
ATF voidaan integroida esim. avoimen lähdekoodin JavaScript-testauskehykseen.	ATF ei ole tarkoitettu kuormitus- tai suorituskyvyn testaamiseen.

ServiceNow’n parhaat käytännöt ATF-sovellukseen liittyen jakautuvat kuuteen kategoriaan. Kuva 13 esittelee kategoriat visuaalisesti. Ensimmäinen käytännöistä on tunnistaa olennaiset toiminnot prosessi- tai sovelluskohtaisesti. Tämä mainittiin myös taulukossa, jossa tarkasteltiin testaustyypeittäin soveltuvuutta ATF:ssä (taulukko 2). Kun tiedetään mitkä toiminnot ovat olennaisia, mietitään niihin liittyviä riskejä. Riskejä taas verrataan käynnissä oleviin tai tuleviin muutoksiin ja miten paljon ylläpitoa tämä vaatii. Toinen käytännöistä on luoda testejä, jotka vahvistavat olennaiset ja välttämättömät toiminnot. Testien luonnin voi aloittaa, kun uuden ominaisuuden tai toiminnon jatkokehitys on vähäistä. Jos uutta kehitystä ja testiä tehdään rinnakkain, on huomioitava mahdolliset muutokset toiminnallisuuteen, joka johtaa ylimääräiseen työhön testin luonnissa. Kolmas käytäntö liittyy testipaketteihin.

Testipaketit tulee suunnitella siten että ne voidaan suorittaa yhtenä sarjana. Yksittäinen testi voidaan jakaa useampaan testipakettiin. Testipakettien ja sen sisältämät testit kannattaa suunnitella huolellisesti testauksen optimoimiseksi. Neljäs käytäntö liittyy myös testipaketteihin, jossa suoritetaan tai ajastetaan testit tarpeen ja tilanteen mukaan. Viides käytäntö sisältää testitulosten seurannan ja ongelmien korjaamisen. Testituloksia suositellaan tarkistamaan, kun testit on suoritettu. Yksityiskohtaiset testitulokset näkyvät omalla välilehdellään testipaketin-, testi-, sekä testivaiheen sivulla. Testin suoritusaikaiset kuvakaappaukset ovat nähtävillä testituloksissa ja toimivat apuna virheenselvittelyssä ongelmatilanteissa. Kuudes käytäntö ja yhtä tärkeä kuin muutkin käytänteet, on testien ylläpito. Aina kun jokin muuttuu, lisätään tämä muutos myös testeihin. Muutos arvioidaan ja päivitetään tarvittavat testit vastaamaan viimeisintä muutosta. (ServiceNow, 2021d)

Kuva 13 ServiceNow ATF:n parhaat käytännöt (ServiceNow, 2021d)



ServiceNow’n (2021b) parhaat käytännöt testien suunnitteluun ja luontiin liittyvät pääosin käytäntöihin, miten testejä kannattaa toteuttaa, osin ATF-sovelluksen käyttötarkoitukseen, kuin myös tiettyjen toimintojen hyödyntäminen testeissä.

Keskeiset testausskenaariot ovat käytäntö mitä ATF:ssä painotetaan erityisesti, sillä se on tarkoitettu liiketoimintaprosessien toiminnalliseen testaukseen. Testauksen tulisi keskittyä tärkeimpiin käyttäjäpolkuihin (user flow) tai prosessikulkuihin (process flow). **Tekeytyminen** on tarvittava toiminto, kun suoritetaan testivaihe käyttäjän oikeuksilla tai ryhmätiedolla. Testeissä luodaan useimmiten ensin käyttäjä, joksi tekeydytään. **Paketointi** tai testipaketit mahdollistavat testien ryhmittelyn tiettyyn järjestykseen sovelluksen tai liittyvien ominaisuuksien testausta varten. **Rinnakkaistestit** vähentävät testausaikaa, sillä testejä on mahdollista suorittaa rinnakkain. Testit, jotka käyttävät samoja tietoja suorituksen aikana, voidaan merkitä toisensa poissulkeviksi, jolloin varmistetaan, ettei niitä suoriteta samanaikaisesti. **Timeout** on älykäs sisäänrakennettu odotusmekanismi, jota voidaan

hyödyntää käyttöliittymän testivaiheissa, jotka edellyttävät käyttöliittymämuutoksen valmistumista ennen seuraavaa vaihetta. **Lyhyet testit** helpottavat testien rakentamista ja virheenselvitystä. Testeissä on tarkoitus testata erillisiä kohteita, esim. luo käyttäjä, avaa lomake. **Vältä toistoa**, samaa käyttöliittymän toiminnallisuutta ei kannata toistaa useassa testissä. Jos testit vaativat samaa toimintoa voidaan käyttää palvelinpuolen testivaihetta, jotka toimivat nopeammin kuin käyttöliittymätestivaiheet. **Testaa lopputulos** keskittymällä testin tulokseen toimivuuden varmistamisessa, etenkin liiketoimintasääntöjen testeissä. **Käyttöoikeudet** kannattaa tarkistaa epäonnistuneen testin virheenselvittelyssä. Testivaihe saattaa epäonnistua luodun testikäyttäjän puutteellisten oikeuksien takia. **Tarkista tietuepäivitykset** sisällyttämällä tietueen validointi (record validation) jokaiseen tietuepäivitys (record update) testivaiheeseen, varmistamaan että tietue päivitettiin. **Toiminto tiedon sijaan**, testataan toiminnallisuus ja arvojen muuttuminen tarkoitetulla tavalla, sen sijaan että testataan kaikkia mahdollisia yhdistelmiä datasta. **Ehdollinen testaus**, jossa suoritetaan testivaiheita annettujen kriteerien perusteella tai haaroitetaan testi eri polkuja pitkin, vaatii että luodaan erillinen testi jokaiselle tilalle. (ServiceNow, 2021b)

6.2 ATF-roolit

ServiceNow’ssa on roolipohjaiset käyttöoikeudet, joilla hallitaan sovellusten, moduulien ja ominaisuuksien käyttöä. Järjestelmänvalvojan roolilla on pääsy kaikkeen. Rooleja voidaan lisätä olemassa olevaan rooliin ja tällöin käyttäjä perii myös uuden roolin sisältämät käyttöoikeudet. Tyypillisesti luodaan rooli peruskäyttäjälle, jolla on pääsy perusjärjestelmän alustan ominaisuuksiin ja sovelluksiin. Käyttäjälle voidaan lisätä useita rooleja. (ServiceNow, 2021h) ATF-sovelluksen käyttö vaatii jonkin seuraavista rooleista, atf_test_admin, atf_test_designer tai atf_ws_designer (ServiceNow, 2021a).

Roolilla atf_test_admin on ATF-sovellukseen kattavimmat oikeudet. Tällä roolilla voidaan luoda ja muokata sovelluksen ominaisuuksia. Roolilla on oikeus nähdä testisivu, testivaiheet, testin suoritusikkuna, testitulokset sekä testipaketin tulos. Roolilla on oikeus luoda, muokata ja poistaa testejä, testivaiheita ja testipaketteja. Roolilla on lisäksi oikeus luoda ja muokata testivaiheiden tietueita. (ServiceNow, 2021a)

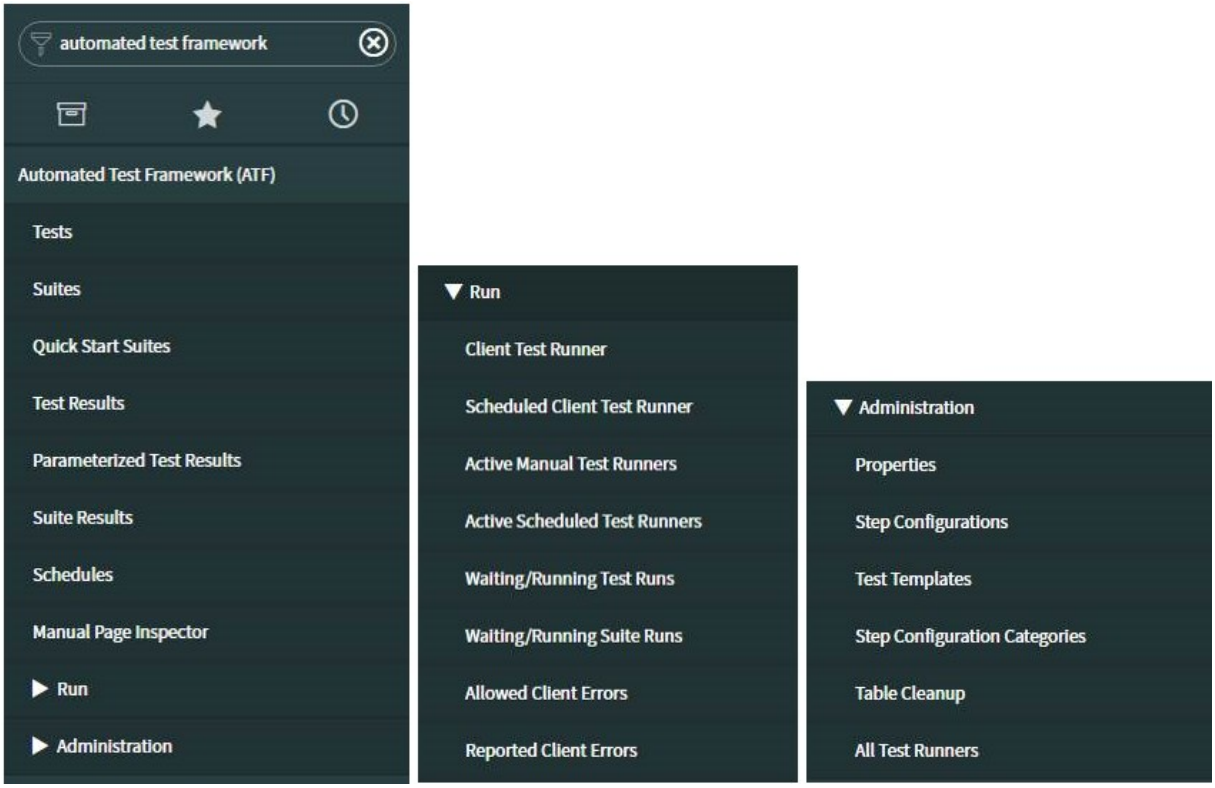
Roolilla atf_test_designer voidaan tarkastella sovelluksen ominaisuuksia, mutta ei luoda tai muokata niitä. Roolilla on oikeus nähdä testisivu, testivaiheet, testin suoritusikkuna, testitulokset sekä testipaketin tulos. Roolilla on oikeus luoda, muokata ja poistaa testejä, testivaiheita sekä testipaketteja. (ServiceNow, 2021a)

Roolilla atf_ws_designer voidaan tarkastella tai luoda tarvittavia perusprofiileja, joilla on oikeus verifioida toimintoja REST APIn kautta. (ServiceNow, 2021a)

6.3 ATF-moduulit

Tässä luvussa esitellään toimeksiantajan käytössä olevat ATF-sovelluksen moduulit. Kuva 14 esittää käytössä olevat moduulit, joista käydään läpi työn kannalta olennaisimmat. Koska ServiceNow'n suomenkieliset käännökset eivät ole vielä vakiintuneita, käytetään moduuleista niiden englanninkielisiä nimiä ja suluissa on toimeksiantajan järjestelmän suomenkielinen käännös.

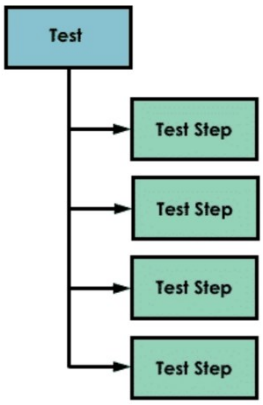
Kuva 14 Automated Test Framework -moduulit



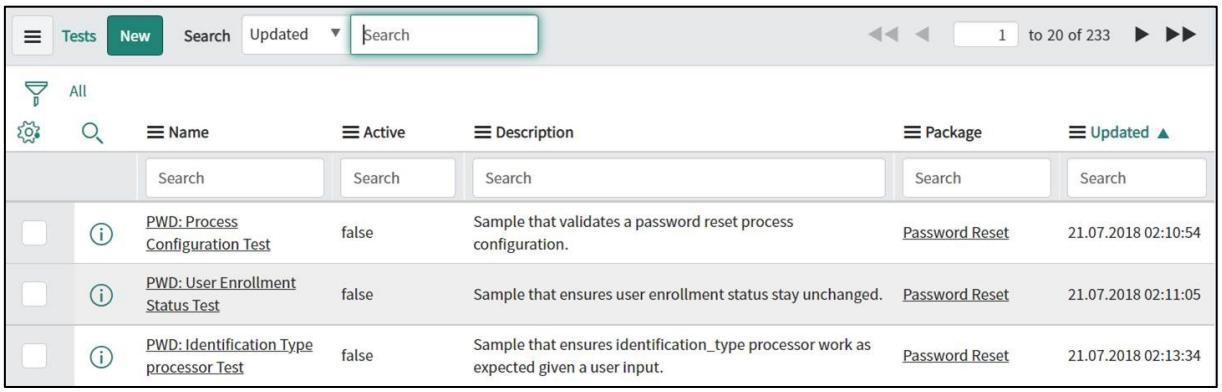
6.3.1 Tests

Tests (testit) on looginen ryhmittely toisiinsa liittyvistä automaattisista testivaiheista, jotka varmistavat jonkin toiminnallisuuden tai ominaisuuden järjestelmässä. Kuva 15 havainnollistaa testin rakennetta ATF-sovelluksessa. Kaikki testit ovat tietueita Test [sys_atf_test] -taulukossa (kuva 16). Testejä luodaan testaamaan yhtä ominaisuutta tai siihen liittyvien ominaisuuksien ryhmää. Jokainen testi sisältää luettelon testivaiheista ja -tuloksista. (ServiceNow, 2021a) Testin luonti käydään tarkemmin läpi luvussa 6.4.1.

Kuva 15 Testin rakenne (Streyda, 2021)



Kuva 16 Esimerkki [sys_atf_test] -taulun tietueista

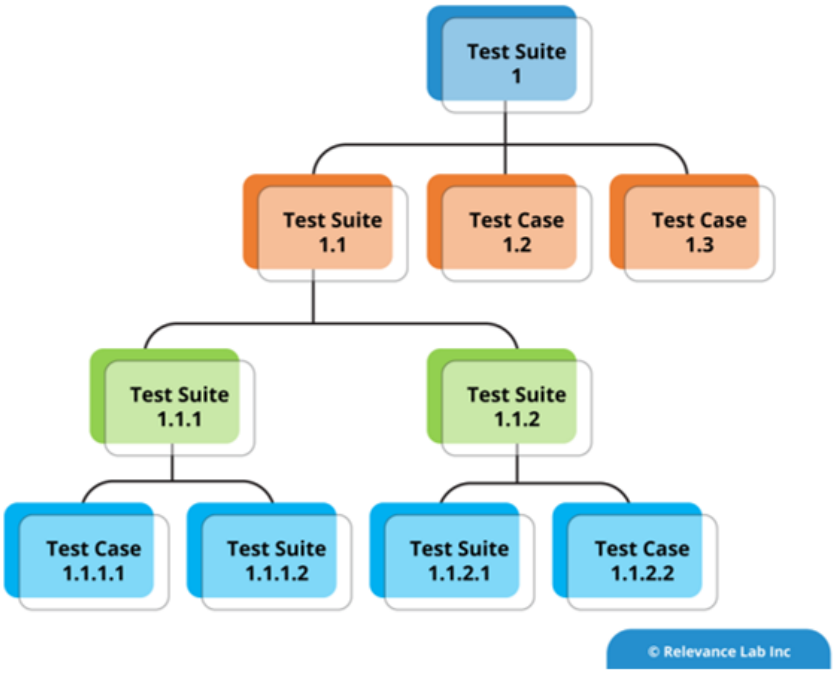


6.3.2 Suites

Suites (testipaketit) on kokoelma testejä, jotka suoritetaan tietyssä järjestyksessä. Testipaketit voidaan luoda testaamaan sovelluksen tai siihen liittyvien ominaisuuksien

ryhmää. Testipakettien suorittaminen voidaan ajastaa tai ajaa manuaalisesti. Kuva 17 havainnollistaa esimerkkiä testipaketin rakenteesta. Testipaketti voi sisältää testien lisäksi myös muita testipaketteja. (ServiceNow, 2021a)

Kuva 17 Esimerkki testipaketin rakenteesta (Relevance Lab, 2020)



6.3.3 Quick Start Suites

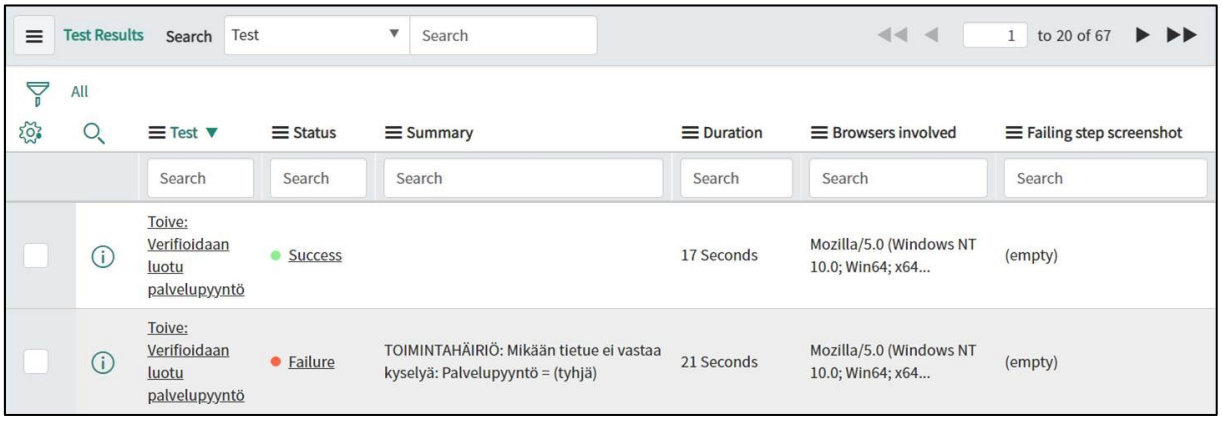
Quick Start Suites -moduuli (pika-aloituspaketit) sisältää ServiceNow'n tarjoamia testejä ja testipaketteja, joita voidaan käyttää malleina omien testien ja -pakettien rakentamiseen. (ServiceNow, 2021a) Pika-aloituspakettien testit suoritetaan onnistuneesti vain, jos oma ympäristö sisältää ServiceNow'n demodataa. Niitä ei pidä käyttää sellaisenaan, vaan tarkoituksena on kopioida testi tai testipaketti ja muokata ne omaan ympäristöön sopivaksi. Pika-aloituspaketit otetaan käyttöön sovellus- ja toimintokohtaisilla laajennuksilla. (ServiceNow, 2021g)

6.3.4 Test Results

Test Results (testitulokset) tallentaa suoritetun testin tai testisarjan tulosteen. Kaikki testitulokset löytyvät yksittäisinä tietueina Test Results [sys_atf_test_result] -taulusta (kuva

18). Taulun sarakeasetuksia muokkaamalla voidaan näyttää esim. testin tila (status), testin suorituksen kesto (duration) sekä kuvakaappaukset epäonnistuneista testivaiheista, jos niitä on saatavilla (failing step screenshot). (ServiceNow, 2021a) Järjestelmä ottaa kuvakaappauksen vain, jos testi sisältää käyttöliittymäkomponentin (ServiceNow, 2021j). Testituloksia käytetään tunnistamaan epäonnistuneet tai suorittamattomat testit. Testilokeja (test log) käytetään testitulosten lisätietoina. Testi- ja testipakettien tulokset poistetaan automaattisesti 30 päivän kuluttua niiden luonnista, ellei käyttöönoteta vaihtoehtoa, jossa tulokset säilytetään rajoittamattoman ajan. (ServiceNow, 2021a) Testitulosten säilytyskäytännön muokkaaminen on kuvattu liitteessä 3.

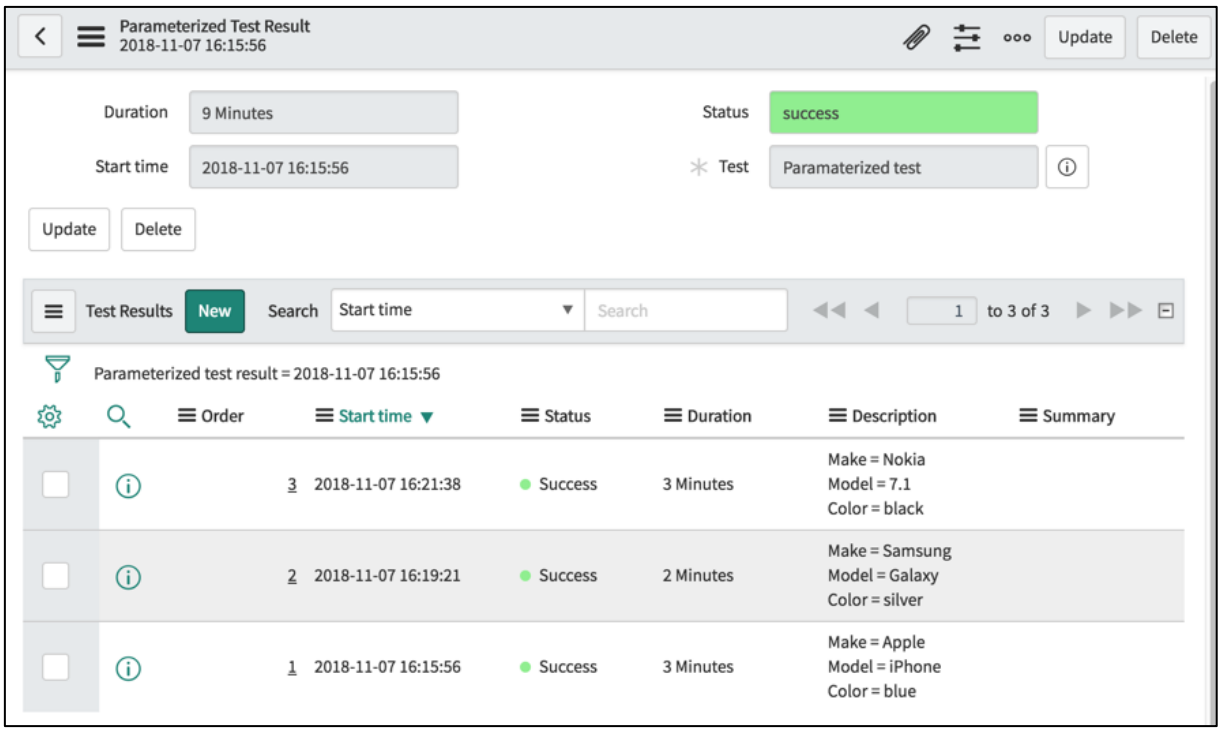
Kuva 18 Esimerkki [sys_atf_test_result] -taulun tietueista



6.3.5 Parameterized Test Results

Parameterized Test Results (parametroidut testitulokset) näyttää parametrisoitujen testien testitulokset syötedatan suoritusjärjestyksessä. Parametrisoitujen testien parametrit ja syötedata luetellaan kuvaus -kentässä. Kuva 19 on esimerkki parametrisoidun testin testituloksista. (ServiceNow, 2021f)

Kuva 19 Esimerkki parametrisoidusta testituloksesta (ServiceNow, 2021f)



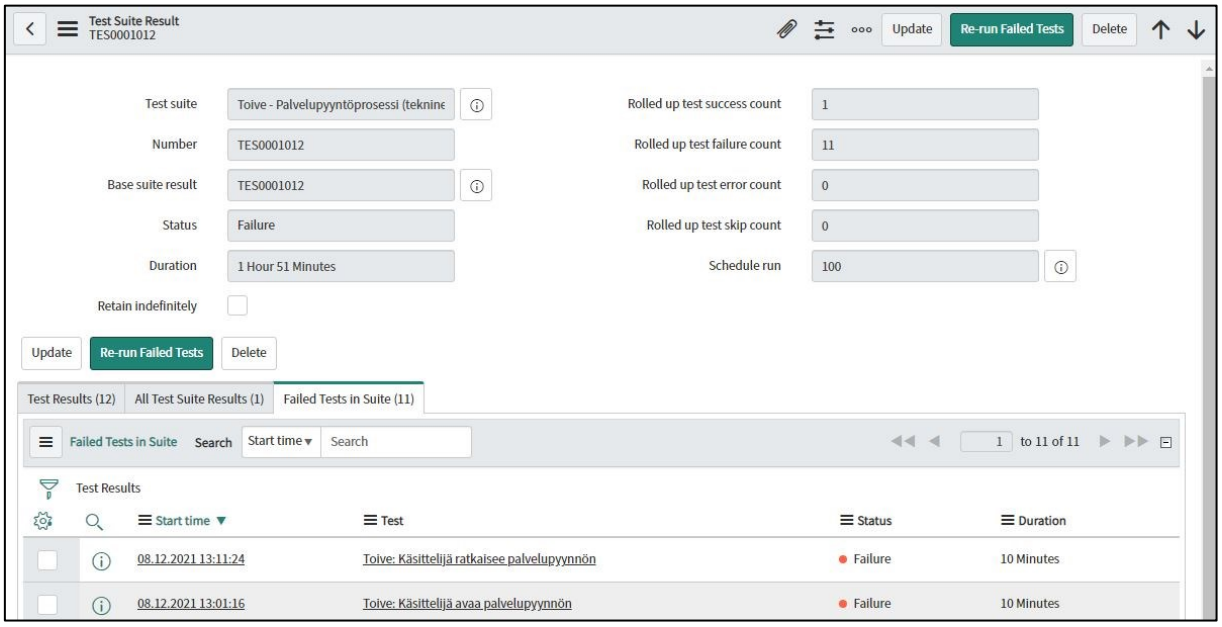
Parametrisoitu testi luodaan Tests-moduulin kautta. Uudessa testissä rastitetaan kohta Enable parameterized testing (ota käyttöön parametroitu testaus). Tämä tuo esille muiden välilehtien lisäksi välilehdet Parameter Definitions (parametrimääritelmät), Test Run Data Sets (testin suorittamisen tietopaketit), sekä Parameterized Test Results (parametroidut testitulokset). Parametrisoidun testin avulla testi suoritetaan useilla eri testitiedoilla. Parametrisoiduilla testeillä poistetaan tarve monistaa testivaiheita testitietojen muuttamista varten, esim. lomaketestissä valitaan eri vaihtoehtoja pudotusvalikoista yhden testin sisällä. Parametrisoiduilla testeillä lisätään testien uudelleenkäyttöä erottamalla testitoiminnot testitiedoista, sekä tuotetaan erilliset testitulokset jokaiselle syötedatalle. Parametrejä on kaksi eri tyyppiä, Exclusive Parameters (yksinomaiset parametrit) ja Shared Parameters (jaetut parametrit). Yksinomaisia parametrejä voidaan käyttää vain testissä, jossa ne on luotu. Jaettuja parametrejä voidaan käyttää missä tahansa parametrisoidussa testissä. (ServiceNow, 2021f)

Parametrisoidun testin luonnissa on huomioitava, että parametrin tietotyyppi tulee olla sama kuin kentän tietotyyppi. Jos lomakkeen kenttä on tietotyypiltään Choice (vaihtoehto), on luotavan parametrin oltava tietotyypiltään sama. Testivaihetta Run Server Side Script, eli palvelimella suoritettavia komentosarjoja, ei tueta parametrisoiduissa testeissä.

6.3.6 Suite Results

Suite Results -moduuli (testipakettitulokset) avaa [sys_atf_test_suite_result] -taulun listanäkymän. Testipaketin tuloksiin siirrytään valitsemalla halutun testipaketin aloitusaika (oletussarake). Tuloksia voidaan tarkastella testivaiheittain, testipaketeittain, mikäli paketissa useampi, sekä epäonnistuneiden testien kautta, kuten kuva 20 esittää. (ServiceNow, 2021c)

Kuva 20 Esimerkki testipakettituloksesta



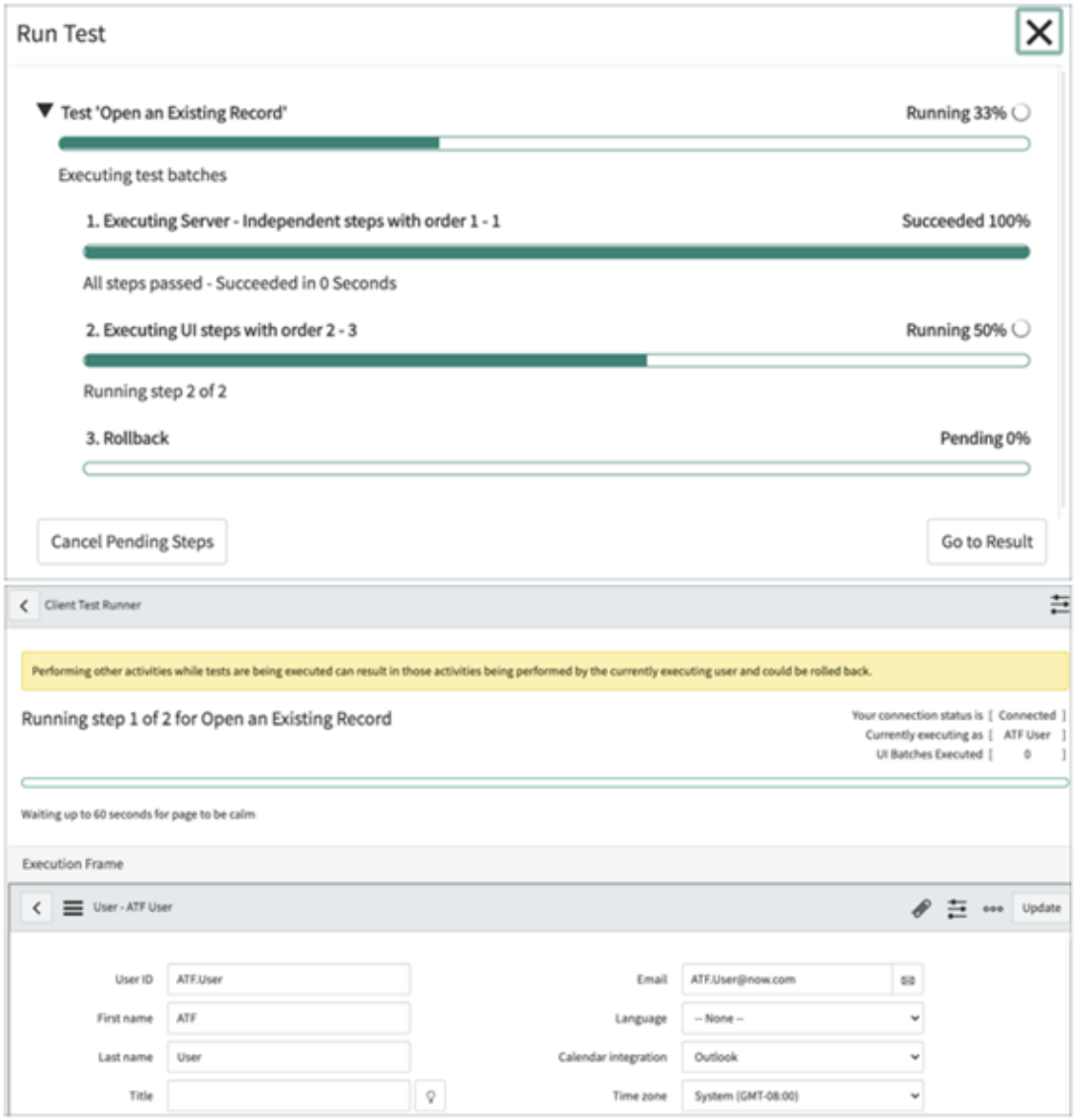
6.3.7 Schedules

Schedules-moduulin (aikataulut) kautta siirrytään Suite Schedules (ohjelmistopaketin aikataulut) listanäkymään, josta kaikki ajastetut testipaketit löytyvät. Ajastuksen voi tehdä ainoastaan testipaketeille, yksittäisten testien suoritusta ei voida ajastaa. Testipaketin ajastamiseen tarvitaan olemassa oleva testipaketti, automaattisen suorituksen ajankohta, sekä ajastetun testin suorittaja (scheduled client test runner). Ajastettu testipaketti suoritetaan vain, jos testipaketin selainehtoja vastaava testin suorittaja on aktiivinen. Ajastettuja testipaketteja ei suoriteta koneessa, joka on lukittu, sammutettu tai jossa ei ole selainta valmiiksi aktiivisena. (ServiceNow, 2021i) Testipaketin ajastus käytännön tasolla käsitellään luvussa 6.4.5.

6.3.8 Client Test Runner

Client Test Runner -moduuli (testin suorittaja) sisältää toiminnon, joka avaa selaimeen uuden välilehden, kun on tarkoitus suorittaa testi, testejä tai testipaketteja manuaalisesti. Client Test Runner -toiminto sisältyy Run Test -painikkeeseen, eikä sitä tarvitse erikseen avata ennen testien suorittamista. Run Test -painike avaa selaimeen välilehden, jos yksikin testin vaiheista sisältää käyttöliittymäelementin tai lomakkeen. Kun toiminto on aktiivinen, testin suoritusta voi seurata suorituskehyksessä (execution frame) tai Client Test Runner:issa. Kuva 21 havainnollistaa näiden väliset erot. Testituloksiin tallennetut kuvakaappaukset otetaan Client Test Runner:illa. Client Test Runner -välilehden ollessa aktiivinen, testit suoritetaan nopeammin verrattuna suorituskehykseen. Toiminnon asetukset löytyvät Administration-moduulin alta, kohdasta Properties. Asetuksissa voidaan lisäksi ottaa käyttöön virheenselvitys (debug), jolloin näkymässä on kaksi välilehteä. Ajastetun testipaketin suorittamisessa käytetään Scheduled Client Test Runner:ia. (ServiceNow, 2022)

Kuva 21 Execution Frame ja Client Test Runner



6.4 Testauksen automatisointi ATF-sovelluksella

Tämä luku esittelee parhaat käytännöt testien suunnitteluun ja luontiin. Alaluvuissa esitellään vaiheittain testin luonti, sen liittäminen testipakettiin sekä manuaalisen ja ajastetun testin suorittaminen.

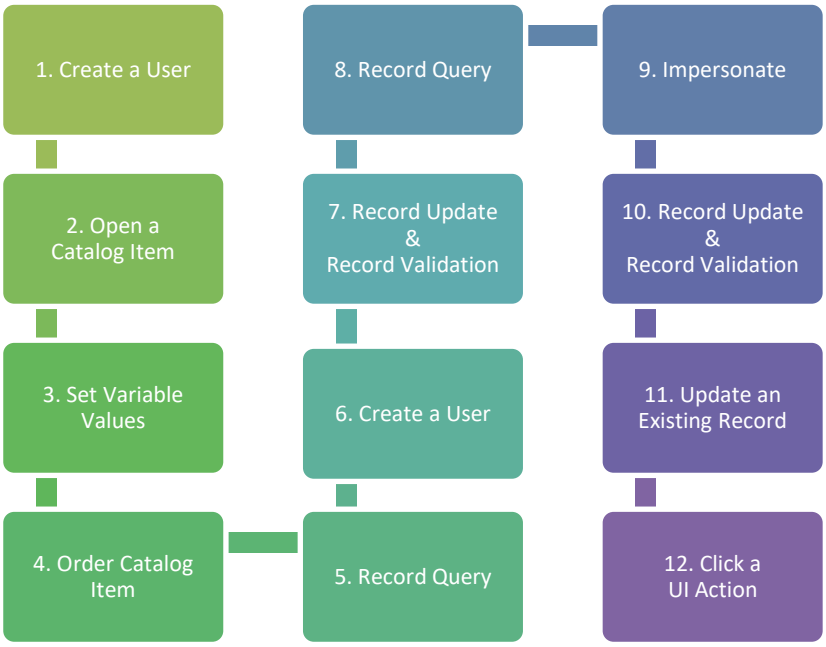
Yleisimmät testattavat ominaisuudet ja toiminnot, jotka voidaan ATF:llä automatisoida ovat lomakkeisiin liittyvät, palvelinpuolen toiminnot, käyttäjät, yleiset käyttöliittymätoiminnot, sähköpostit ja sovellusnavigointi coren puolella. Lomaketestaus voi olla kentän arvon asettaminen ja tarkistaminen. Voidaan tarkastaa kentän tyyli, sen näkyvyys, pakollisuus, tai

tyyppi, esim. vain-luku tai muokattavissa oleva. Se voi myös olla lomakkeella olevan käyttöliittymätoiminnon valinta ja lomakkeen lähettäminen. Palvelinpuolen toiminnolla tarkoitetaan tietueen etsintää, muokkausta ja lisäystä. Käyttäjien luonti on olennainen osa testausta ATF:llä. Luodaan testiä varten käyttäjä, jolla tietyt roolit ja / tai ryhmät. Yleistä käyttöliittymätoimintaa on esim. tarkistaa onko tietty teksti sivulla ja komponenttien valinta. Sähköpostiin liittyvät testivaiheet voivat olla viestin luonti ja lähetyksen varmistus. Sovellusnavigoinnilla tarkoitetaan testausta, jossa tarkistetaan moduulien näkyvyyttä ja niiden valintaa.

6.4.1 Testin luonti

Toimeksiantajan peruspalvelupyyntöprosessin testeistä esitellään tarkemmin vaiheet 1 Create a User (luo käyttäjä), 2 Open a Catalog Item (avaa luettelokohde), ja 7 Record Update & Record Validation (tietueen päivittäminen ja validointi). Kuva 22 havainnollistaa peruspalvelupyyntöprosessin kaikki tarvittavat testivaiheet.

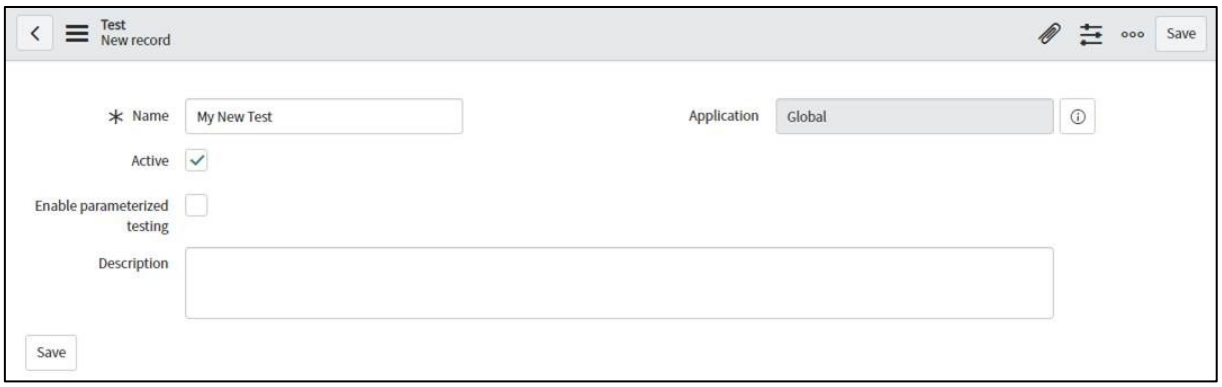
Kuva 22 Peruspalvelupyyntöprosessin testivaiheet



Uusi testi luodaan ATF:n Tests-moduulissa. Kuva 23 esittää uuden testin näkymää. Testille annetaan kuvaava nimi, joka kertoo mitä testissä tehdään, esim. testikäyttäjän luonti ja tekeytyminen. Vaikka testin nimi on ainoa pakollinen kenttä, on suositeltavaa lisätä testille

myös kuvaus. Kuvaus voi sisältää esim. tiedon testissä suoritettavista testivaiheista, kuten toimeksiantajan tapauksessa (kuva 27) on tehty. Testin tallentamisen jälkeen saadaan näkyviin välilehdet Test steps (testivaiheet), Test Results (testitulokset) ja Mutually Exclusive Tests (toisensa poissulkevat testit). Testivaiheita, eli toiminnallisuuksia ja validointeja, ei voi lisätä ennen kuin testi on tallennettu ensimmäisen kerran.

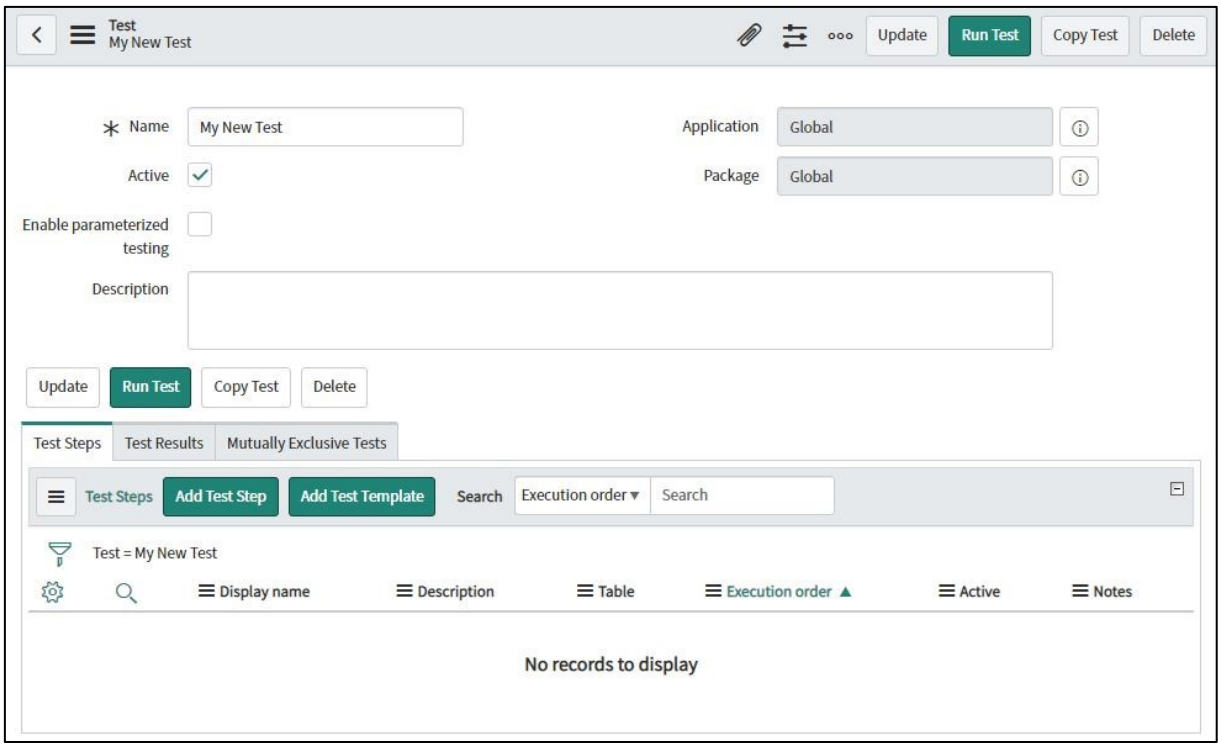
Kuva 23 Uusi ATF-testi



Kun testi on ensimmäisen kerran tallennettu, siihen voidaan lisätä testivaiheita. Testissä testataan vain yhtä asiaa kerrallaan, joten ensimmäisessä testissä ei tulisi olla kuin yksi testivaihe. Kehitystiimille järjestetyn kyselyn perusteella toimeksiantajan ensimmäiseksi automatisoitavaksi testiksi valikoitui peruspalvelupyynnön prosessi. Toimeksiantajan tapauksessa, palvelupyynnöillä on aina asiakas, käsittelijä ja palvelupyyntö. Palvelupyyntö voi olla tilaus tai kysymys. Tilaus kattaa fyysisen tuotteen, sovellusoikeuden tai muun palvelun, esim. käännöspalvelu tai asiakirjan toimitus.

Testin välilehdet näkyvät sivun alaosassa ensimmäisen tallennuksen jälkeen (kuva 24). Testivaihe lisätään välilehdeltä Test Steps ja valitsemalla Add Test Step -painike. Test Results -välilehden tiedot ja sisältö käydään tarkemmin läpi myöhemmässä luvussa. Mutually Exclusive Tests -välilehti sisältää tiedon testeistä, jotka merkitty toisensa poissulkeviksi. Välilehden listanäkymällä ovat ne testit, joita ei suoriteta samanaikaisesti avoimen testin kanssa. Järjestelmä tunnistaa testit, jotka muokkaavat samaa tietuetta, jolloin on olemassa mahdollinen resurssiristiriita ja ne merkitään toisensa poissulkeviksi. Poissulkemissääntöjä voi myös itse luoda tarvittaessa.

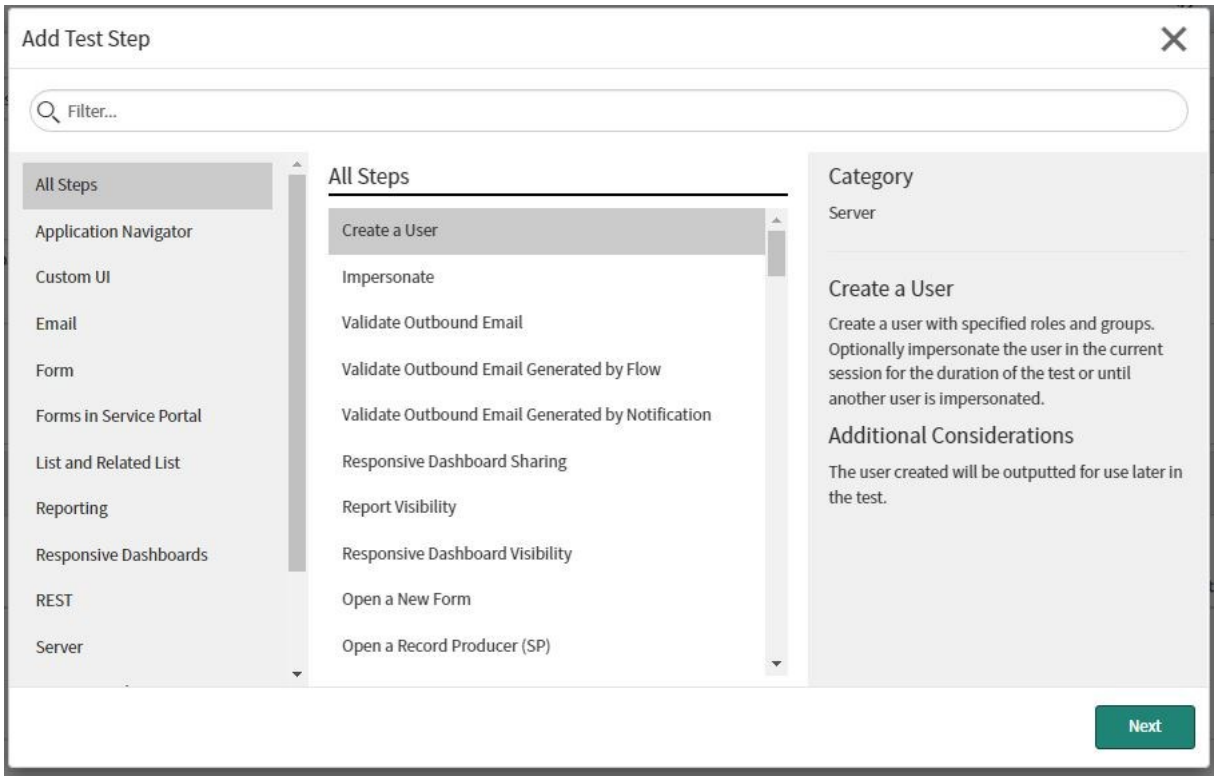
Kuva 24 Testivaiheen lisäys



Vaihe 1. Peruspalvelupyyntöprosessin testin ensimmäinen testivaihe on käyttäjän (asiakkaan) luonti. Testissä luodaan käyttäjä nimeltä ATF Assi Asiakas. Virhetilanteiden ja väärinkäsitysten välttämiseksi, ei ole suositeltavaa luoda tai tekeytyä olemassa olevaksi käyttäjäksi. Toimeksiantajan testeissä luotujen käyttäjien etunimi sisältää aina kirjaimet ATF. Käyttäjien, eli asiakkaiden ja käsittelijöiden nimeämisen takana on syy tunnistaa nämä ATF-testeihin liittyviksi.

Kuva 25 havainnollistaa joitain valittavissa olevia testivaiheita. ATF:n valmiita testivaiheita on 95 kpl ja niitä voi myös itse luoda tarvittaessa. Testivaiheet ovat kategorisoitu, joka voi helpottaa, kun etsii sopivaa vaihetta mitä käyttää testissä. Ensimmäistä testivaihetta varten tarvitaan käyttäjä, joten valitaan testivaiheista Create a User (luo käyttäjä) ja siirrytään eteenpäin valitsemalla Next-painike.

Kuva 25 Testivaiheen valinta, Create a User



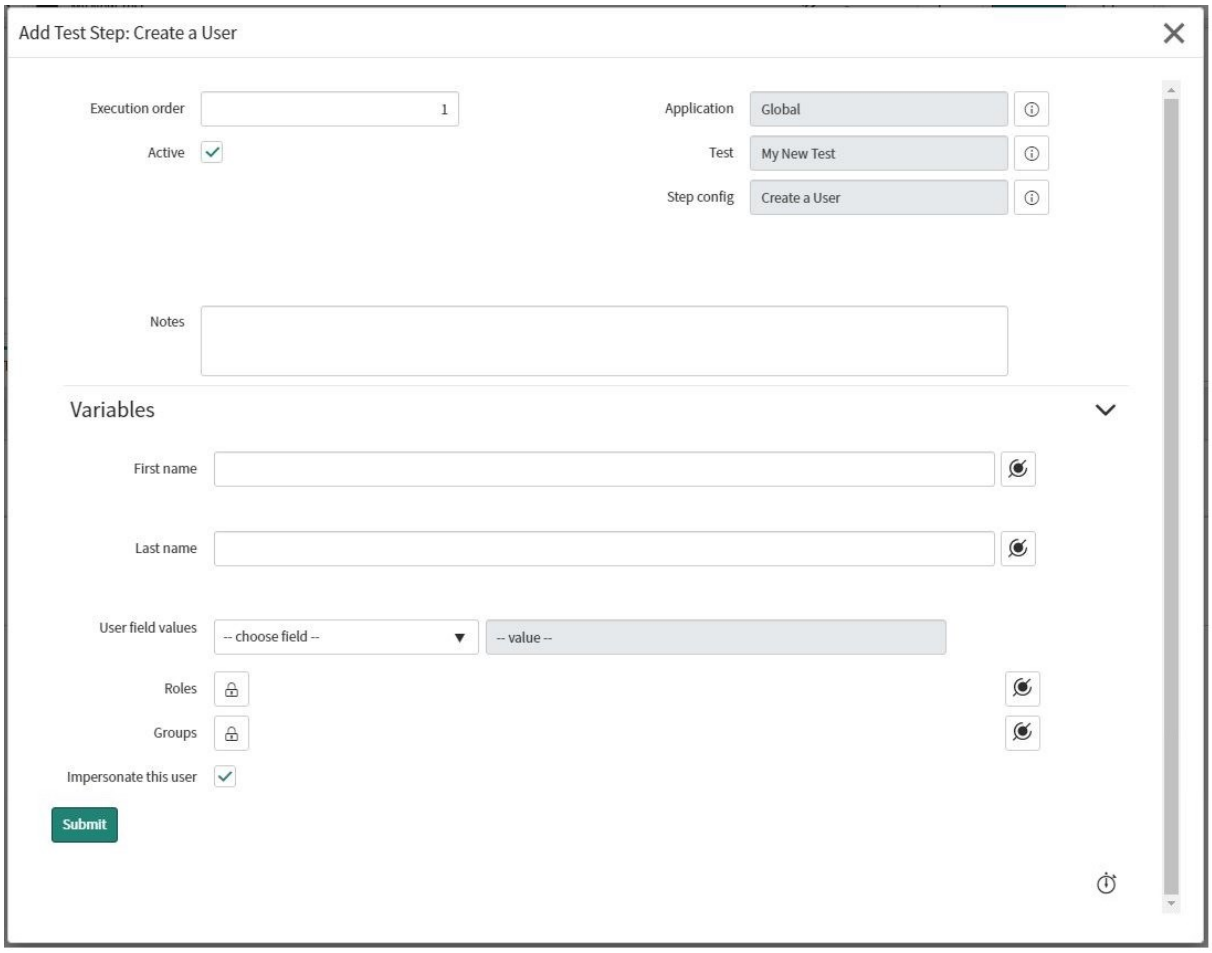
Kuva 26 esittää kenttiä, joita tarvitaan uuden käyttäjän luonnissa. ATF tarvitsee testivaiheiden suoritusjärjestyksen, jotta osaa suorittaa testin oikeassa järjestyksessä. Ensimmäinen testivaihe on oletuksena ensimmäinen suoritettava. Suoritusjärjestystä voi myöhemmin vaihtaa, jos jonkin vaiheen unohtaa lisätä testiä tehdessä. Kohdassa Variables (muuttujat) annetaan käyttäjälle etu- ja sukunimi, sekä rooli, jolla tietyt oikeudet. Tämän jälkeen tallennetaan testivaihe.

Toimeksiantajan testitapauksessa, ensimmäisenä luodaan käyttäjä, joka toimii asiakkaana. Huomaa, että käyttäjää luodessa on oletuksena rastitettu kohta Impersonate this user (tekeydy käyttäjäksi). Tekeytyminen tarvitaan, kun käyttäjän on tarkoitus tehdä järjestelmässä jotakin. Onkin suositeltavaa luoda käyttäjät vasta kun niitä tarvitaan, siten vältytään turhilta testivaiheilta, joissa ainoastaan tekeydytään käyttäjäksi, jos esim. loisi kaikki testissä tarvittavat käyttäjät testin alussa.

Jos asiakkaan sijaan luotaisiin käsittelijä, lisätään näkyvään kenttään Groups (ryhmät) tieto mihin käsittelijäryhmään tai -ryhmiin henkilö kuuluu (kuva 26). Käsittelijä voi kuulua moneen

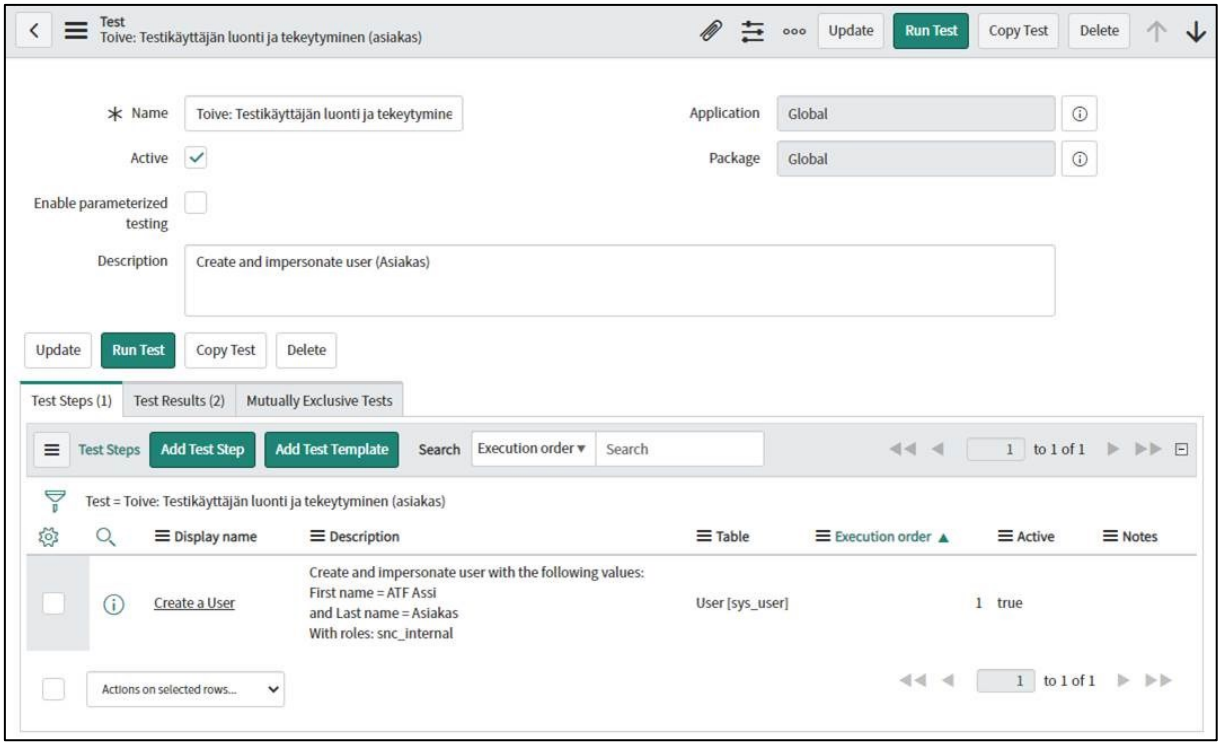
ryhmään ja testin kannalta lisätään kaikki tarvittavat ryhmät. Ryhmällä tarkoitetaan palvelupyyntöjen käsittelyryhmää, tai muuta ryhmää, jolla erityisoikeuksia esim. esihenkilöt.

Kuva 26 Testivaihe, luo käyttäjä



Testin ensimmäinen testivaihe on luotu (kuva 27). Käyttäjä on nimetty toimeksiantajan ATF-testien nimikäytännön mukaisesti ATF Assi Asiakas, jolla on rooli snc_internal. Roolilla on pääsy asiakasportaaliin ja lomakkeisiin, jotka jaettu kaikille käyttäjille.

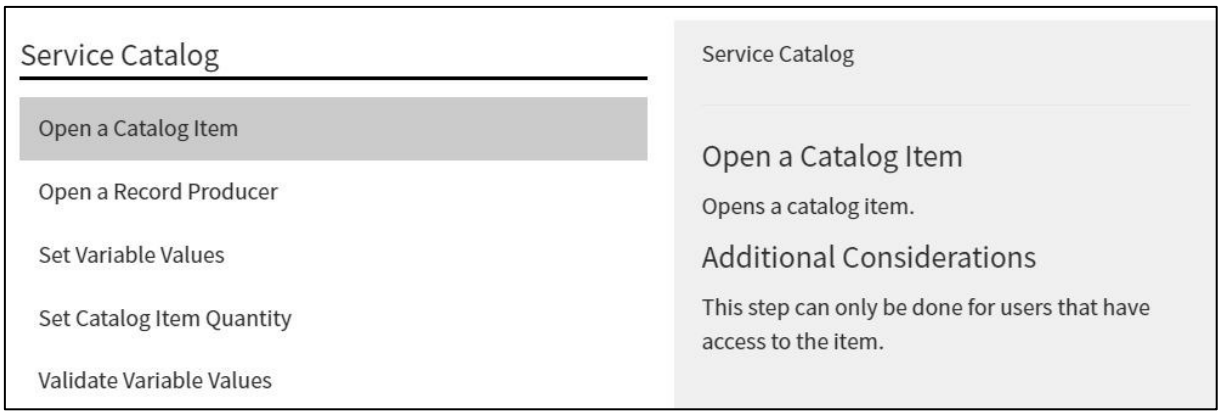
Kuva 27 Ensimmäisen ATF-testin testivaihe



Vaihe 2. Palvelupyyntöprosessin toisessa testivaiheessa testataan, että asiakas saa roolillaan avattua lomakkeen Asiakirjapyynnöt. Kyseinen lomake on portaalilomake ja käyttäjät tekevät sitä koskevan palvelupyynnön tavallisesti portaalin kautta. Testissä ei kuitenkaan testata portaalia, vaan lomakkeen toiminnallisuutta. Ero lomakkeen toiminnallisuuden ja portaalilomakkeen välillä on ulkoasu. Portaalilomakkeelle on määritelty tietty ulkoasu, mutta lomake itsessään sisältää vain määritellyt kentät ja käyttää ServiceNow OOB ulkoasua.

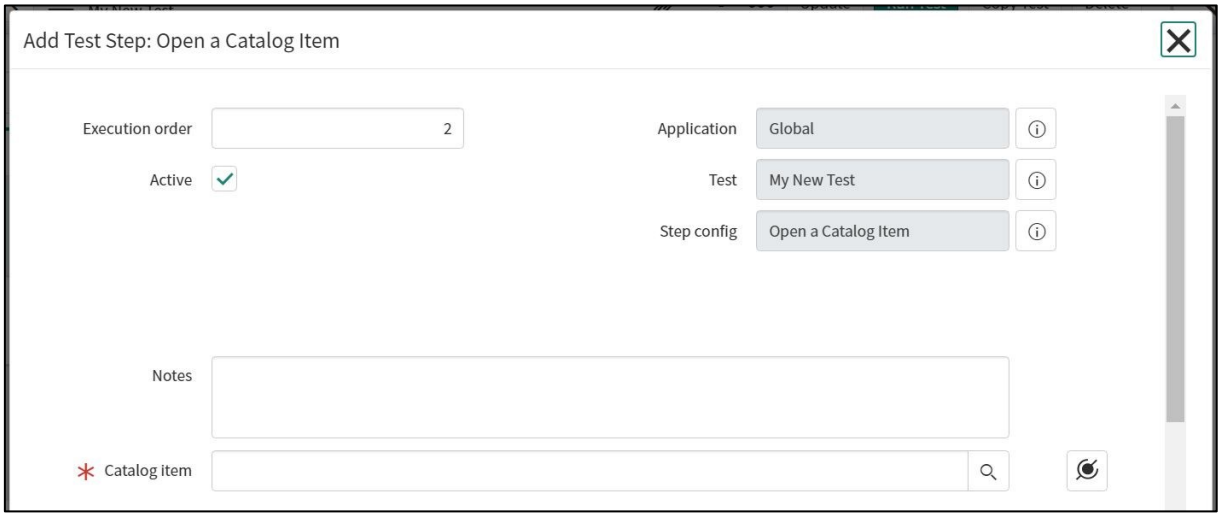
Avataan ensimmäinen testi, jossa luotiin asiakas (kuva 27). Valitaan Copy Test -painike (kopioi testi), jolloin saadaan ensimmäisen testin vaihe toisen testin pohjaksi. Annetaan kuvaava nimi kopioidulle testille ja tallennetaan. Lisätään testivaihe Open a Catalog Item (avaa luettelokohde) (kuva 28), jolla varmistetaan, että asiakas saa lomakkeen auki (kuva 30). Kun testivaihe lisätään testiin, jossa on yksi tai enemmän vaiheita jo olemassa, kysytään käyttäjältä minkä aiemman vaiheen jälkeen uusi lisätään (kuva 29).

Kuva 28 Testivaiheen valinta, Open a Catalog Item



Kuva 29 Testivaiheen lisääminen, kun testissä jo yksi tai useampi vaihe

Kuva 30 Testivaihe, avaa luettelokohde

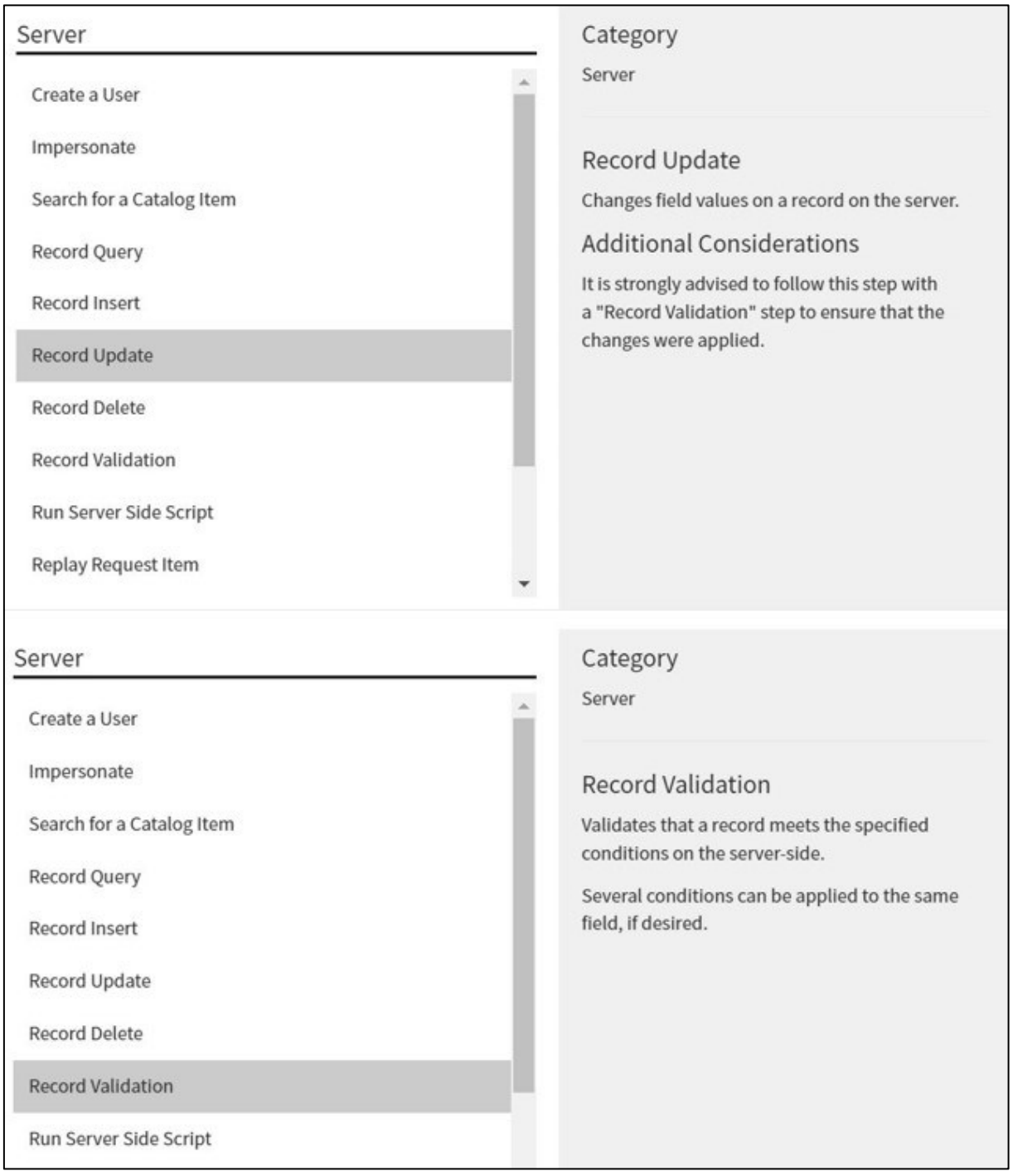


Kuva 30 havainnollistaa testivaiheen, jossa avataan luettelokohde. Lisätään kenttään Catalog item (luettelokohde) lomakkeen nimi, joka toimeksiantajan tapauksessa on Asiakirjapyynnöt.

Testejä 3–6 ei kuvata tässä työssä, mutta ovat toiminnallisuuksiltaan lomakkeelle syötetyt arvot, lomakkeen lähetys, lähetetyn lomakkeen tietojen tarkistus sekä käsittelijän luonti ja tekeytyminen, joka esiteltiin aiemmin (kuva 26). Seitsemäs testi eroaa muista siinä, että

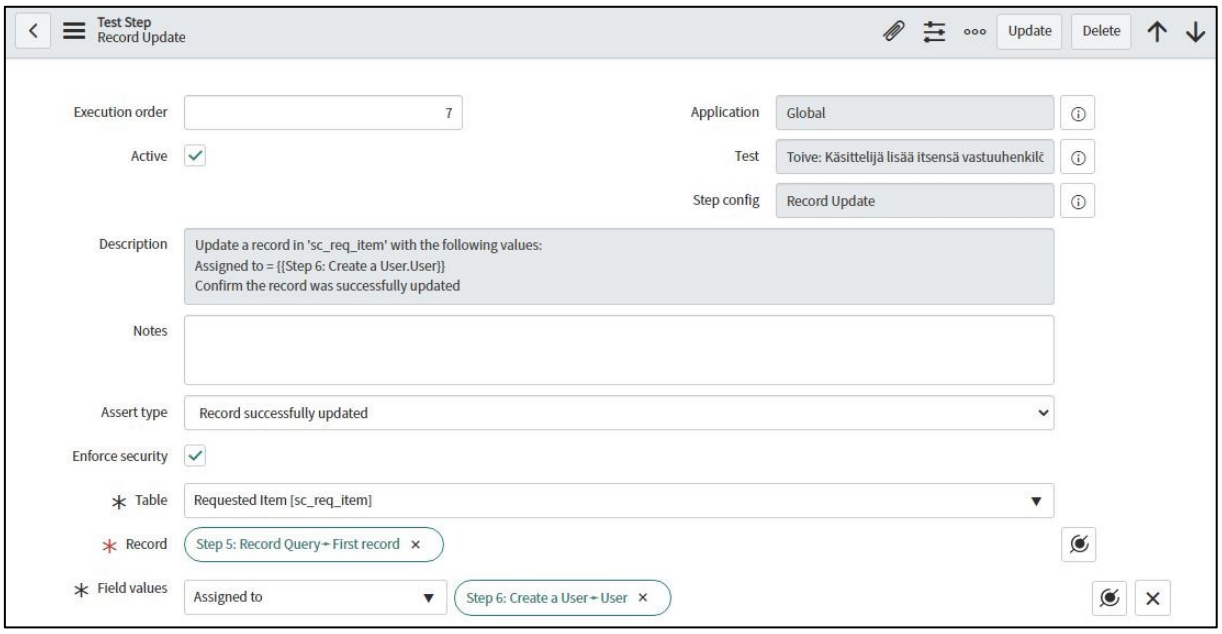
testiin lisätään kaksi testivaihetta yhden sijaan. Syy siihen on ServiceNow'n oma suositus. Kun käytetään Record Update -testivaihetta, tarkistetaan samalla muuttuneet tiedot Record Validation -testivaiheella (kuva 31). Tämä on poikkeus parhaisiin käytäntöihin, sillä ensin testissä päivitetään tietue ja samalla varmistetaan, että tieto muuttui.

Kuva 31 Testivaiheen valinta, Record Update ja Record Validation



Vaihe 7. Peruspalvelupyyntöprosessin seitsemäs vaihe on palvelupyynnön tietueen päivittäminen ja sen validointi. Käytetään Record Update (tietuepäivitys) testivaihetta, jossa käsittelijä lisää itsensä palvelupyynnön vastuuhenkilöksi. Eli tehdään muutos kenttään Assigned to (vastuuhenkilö). Lisätään suosituksen mukaan myös Record Validation (tietueen vahvistaminen) testivaihe, jolla validoidaan, että muutos edellä mainitussa kentässä on tapahtunut. Kuva 32 esittää testivaihetta, jolla tehdään tietuepäivitys. Kentän Assert type (väitetyyppi) vaihtoehdot ovat tietueen päivitys onnistui ja tietuetta ei päivitetty. Koska tarkoituksena on tehdä muutos kenttään, valitaan väitetyypiksi Record successfully updated, eli varmistetaan, että tietueen päivitys onnistui. Kentässä Table (taulu) valitaan palvelupyyntötaulu Requested Item. Kenttään Record (tietue) haetaan viimeisin tieto liittyen palvelupyyntöön. Viimeisin tieto löytyy viidennessä testivaiheessa verifioitu palvelupyynnön sisältö, joten tiedetään että tiedot ovat oikein. Kenttiin Field values (kentän arvot) lisätään ensin tieto kentästä, joka halutaan päivittää. Testin tarkoituksena on lisätä käsittelijä palvelupyynnön vastuuhenkilöksi, jolloin päivitetään Assigned to -kenttä. Seuraavaksi lisätään arvo, joka halutaan syöttää edellä mainittuun kenttään. Haetaan kentän arvoksi testivaiheessa kuusi luotu käyttäjä, eli käsittelijä.

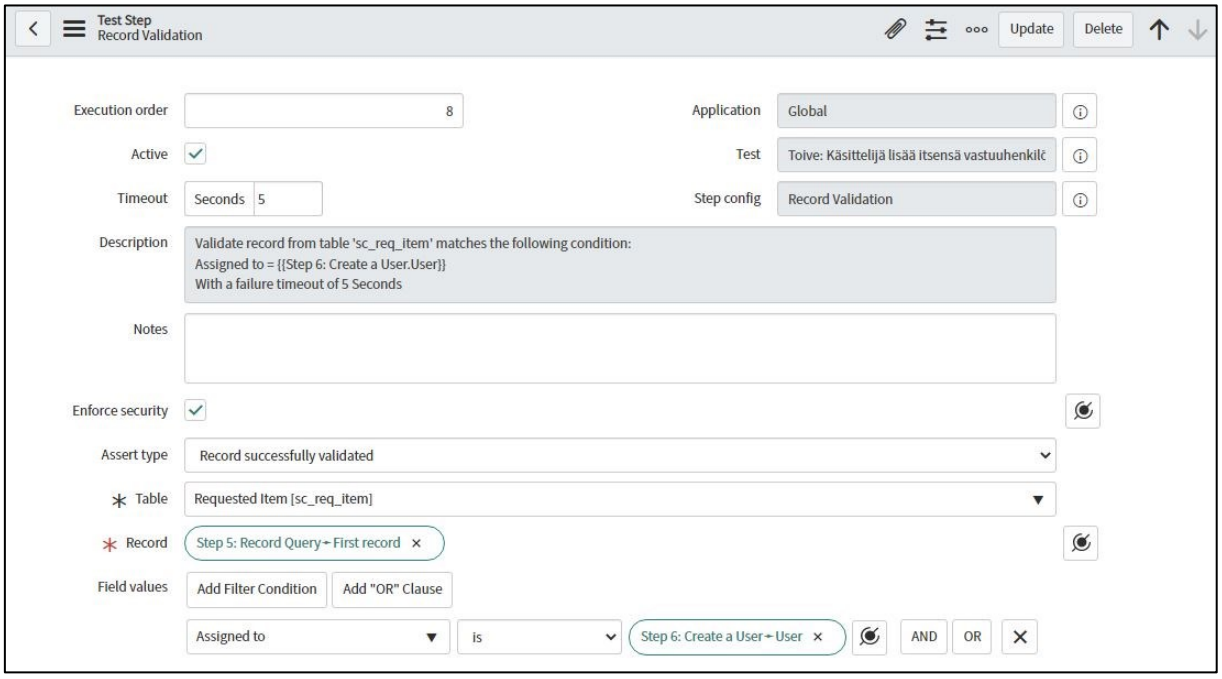
Kuva 32 Testivaihe, tietuepäivitys



Kuva 33 esittää testivaihetta, jolla validoidaan tehty muutos. Testivaiheen kenttiin valitaan arvot peilaten edelliseen vaiheeseen. Väitetyypiksi valitaan, tietueen vahvistaminen onnistui

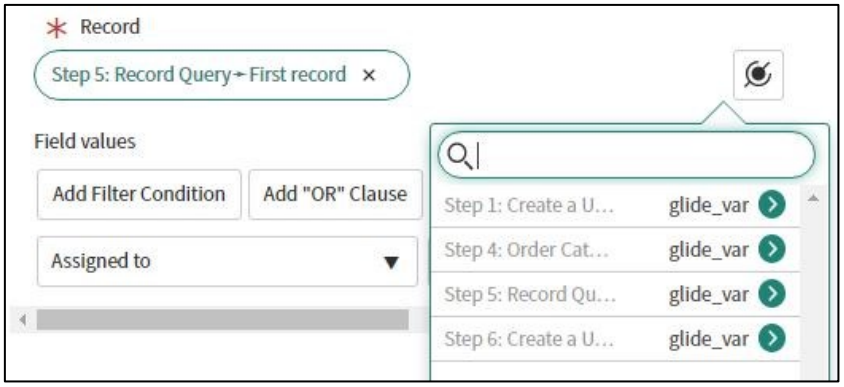
(record successfully validated). Mikäli väitetyyppi olisi edellisessä vaiheessa ollut tietuetta ei päivitetty, validoitaisiin, että tietuetta ei löydy (record not found). Taulu, josta tietoja tarkistetaan, on palvelupyynnöt. Tietueeksi haetaan palvelupyynnön arvot, jotka tarkistettiin viidennessä testivaiheessa. Kenttien arvot mitä validoidaan, ovat vastuuhenkilökenttä ja sen sisältämä syöte. Edelliseen testivaiheeseen verrattuna, validoinnissa suodatetaan valitun taulun sisältämiä kenttiä lisäämällä ehtoja. Ehtoja voi olla useita, jos validoidaan useampia tietoja. Testissä on lisätty vastuuhenkilötieto palvelupyynnölle, joka on ainoa tehty muutos tässä vaiheessa testiä. Lisätään ehdoksi, että vastuuhenkilö (Assigned to) on vaiheessa kuusi luotu käyttäjä, eli käsittelijä.

Kuva 33 Testivaihe, tietueen vahvistaminen



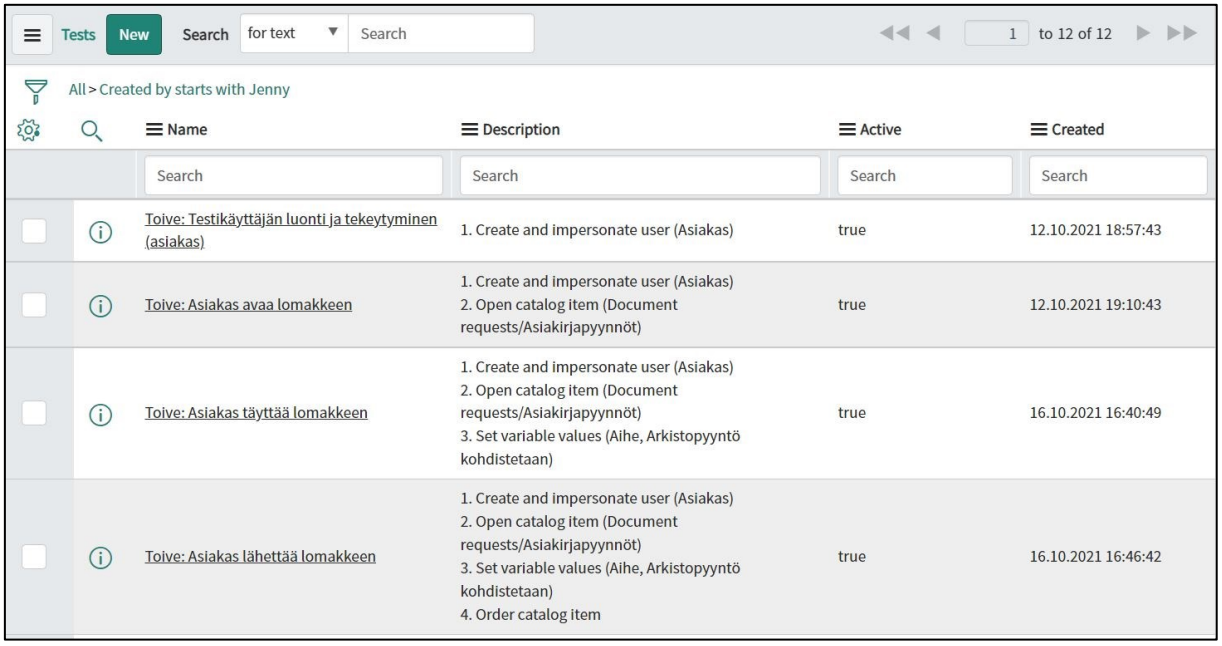
Kaikissa testivaihetyypeissä voidaan hyödyntää, joko aiemmassa testivaiheessa luotua tietoa tai kyseisessä ympäristössä olevaa tietoa, esim. portaalilomaketta tai taulua. Ympäristössä olevalla tiedolla tarkoitetaan ympäristöä, jossa ATF-testi luodaan, esim. kehitysympäristö. Tietojen hyödyntämismahdollisuudet riippuvat testivaiheen tyypistä, kuva 34 liittyy Record Validation -testivaiheen Record-kenttään. Kentässä ei voi hyödyntää muita kuin aiemmissa testivaiheissa luotuja tai varmistettuja tietoja.

Kuva 34 Aiempien testivaiheiden tietojen hyödyntäminen



Toimeksiantajan peruspalvelupyyntöprosessin automatisointiin tarvittiin 12 testiä ja testivaiheita niissä on yhteensä 87 kpl. Kuva 35 näyttää testien neljä ensimmäistä testitietuetta. On suositeltavaa suorittaa testi manuaalisesti jokaisen lisätyn testivaiheen jälkeen, jotta mahdollisen virheen saa heti kiinni testiä luodessa. Silloin välttyy päivittämästä kaikkia seuraavia testejä, jotka käyttävät samoja epäonnistuneen testin testivaiheita omissa testeissään. Esimerkkinä testi, jossa asiakas avaa lomakkeen. Suorittamalla testi varmistetaan, että asiakkaalla on tarvittavat oikeudet kyseiseen lomakkeeseen. Ilman varmistusta, kopioidaan testiä edelleen seuraaviin testeihin, joihin jokaiseen täytyy erikseen muuttaa asiakkaan oikeudet.

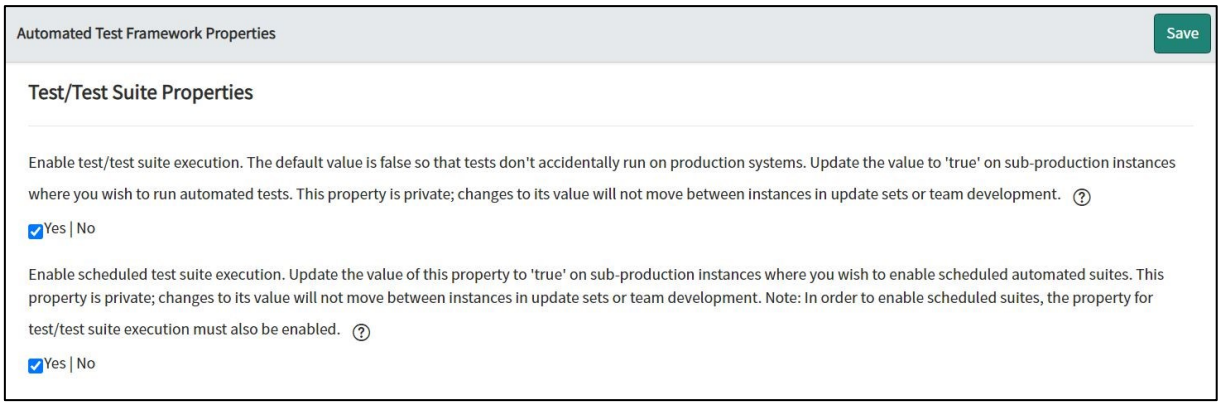
Kuva 35 Peruspalvelupyyntöprosessin testit



6.4.2 Testin suorittaminen

Testien suorittaminen on oletuksena poissa käytössä, jotta testejä ei vahingossa suoriteta tuotantoympäristössä. Testejä suoritetaan vain kehitys-, testi- ja mahdollisissa muissa ei-tuotantoympäristöissä tietojen vioittumisen ja katkosten välttämiseksi. Testien suorittaminen otetaan käyttöön siirtymällä ATF:ssä Administration-moduuliin ja valitaan Properties, rastitetaan molemmat valinnat (kuva 36), jolloin testejä ja testipaketteja voidaan suorittaa manuaalisesti sekä ajastetusti. Nämä asetukset eivät siirry update set:in mukana toiseen ympäristöön. Jos ATF on otettu käyttöön kehitysympäristössä ja halutaan suorittaa testejä myös testiympäristössä tai muussa ei-tuotantoympäristössä, siirretään ensin luodut testit ympäristöjen välillä ja muutetaan asetukset myös toisessa ympäristössä.

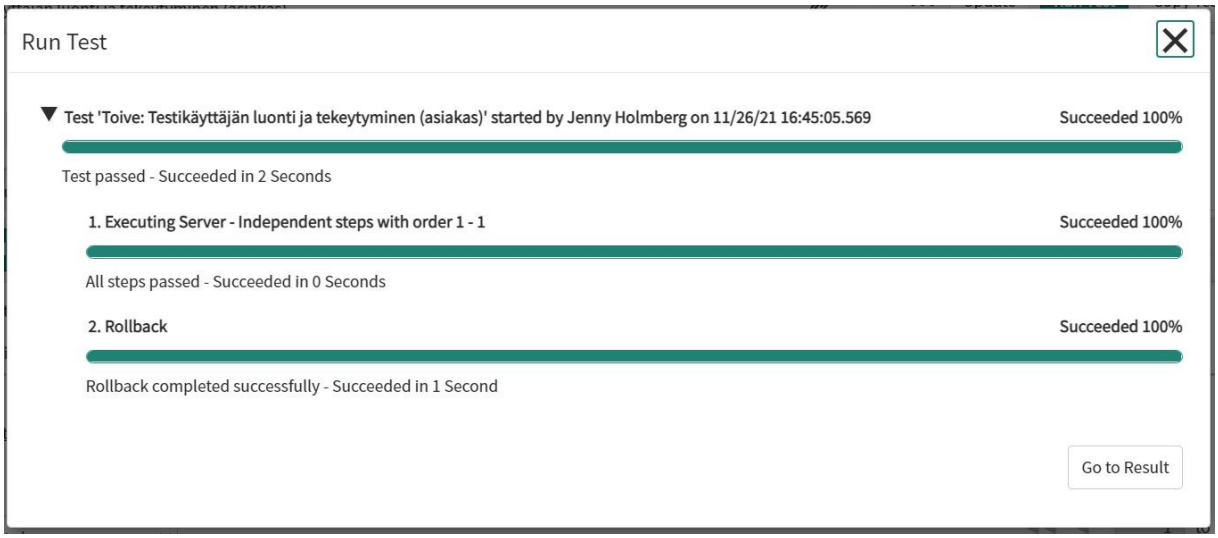
Kuva 36 ATF Test / Test Suite ominaisuudet



Testin suorituksen jälkeen, ATF palauttaa (roll back) tekemänsä muutokset, mutta järjestelmä estää poistamasta tietoja joistain tauluista. Näitä ovat esim. historiaan, sähköposteihin, sekä raporttien suorittamiseen liittyvät taulut. ATF-testeissä luodut käyttäjät nimetään toimeksiantajan nimeämiskäytännön mukaisesti, jotta tehdyt muutokset olisivat helposti jäljitettävissä mahdollisten virhetilanteiden selvittämistä varten. Luettelo kaikista tauluista, joista ATF ei voi palauttaa tehtyjä muutoksia ovat lueteltu liitteessä 4.

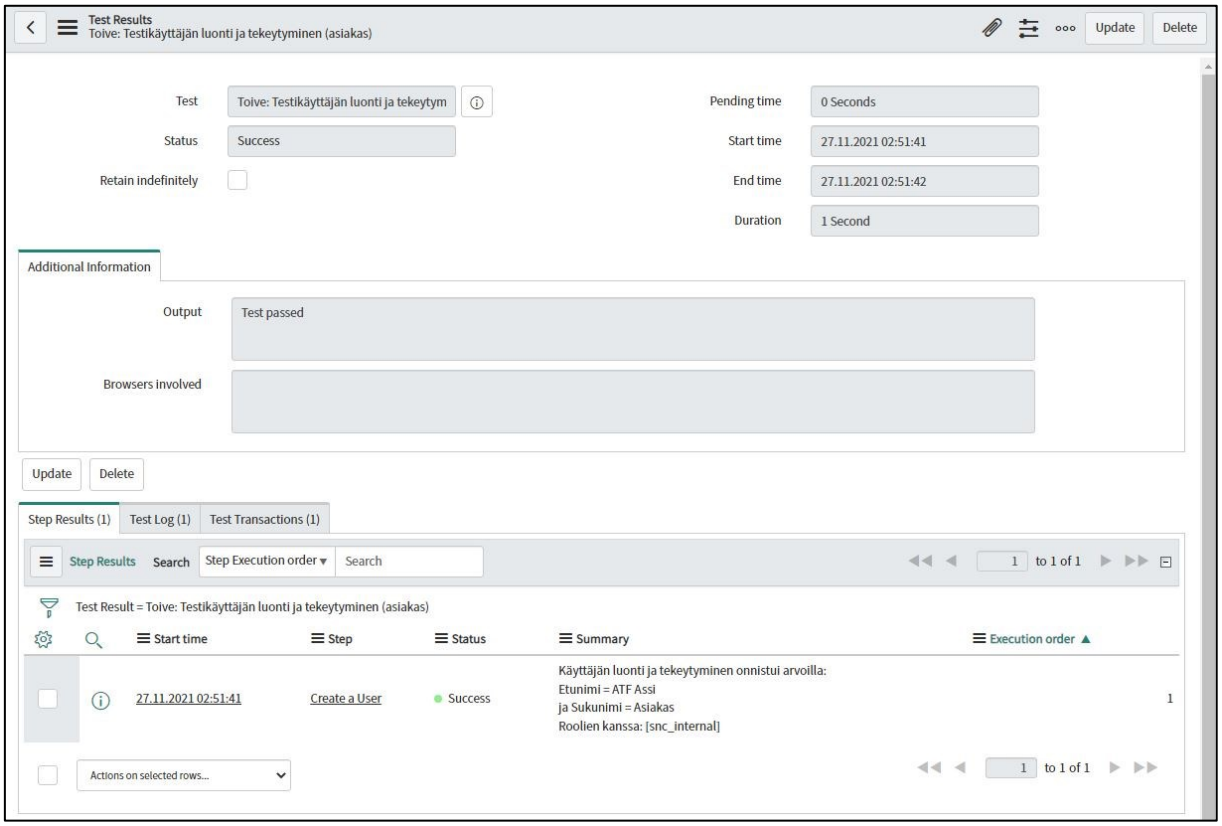
Yksittäinen testi suoritetaan manuaalisesti kyseisen testin sivulla valitsemalla Run test -painike (suorita testi). Testin edistymistä voi seurata suorituskehyksessä (execution frame), josta näkee suorituksen loputtua myös lyhyen yhteenvedon ja tiedon onnistumisesta tai epäonnistumisesta (kuva 37).

Kuva 37 Testin suorituksen yhteenveto, onnistunut



Valitsemalla Go to Result (siirry testitulokseen), näkee tarkemmin testin suorituksen tiedot, esim. testin tila (onnistui/epäonnistui), suorituksen kokonaiskesto, sekä testivaiheiden tiedot. Kuva 38 havainnollistaa testikäyttäjän luonnin testitulosta, josta nähdään, että testi onnistui.

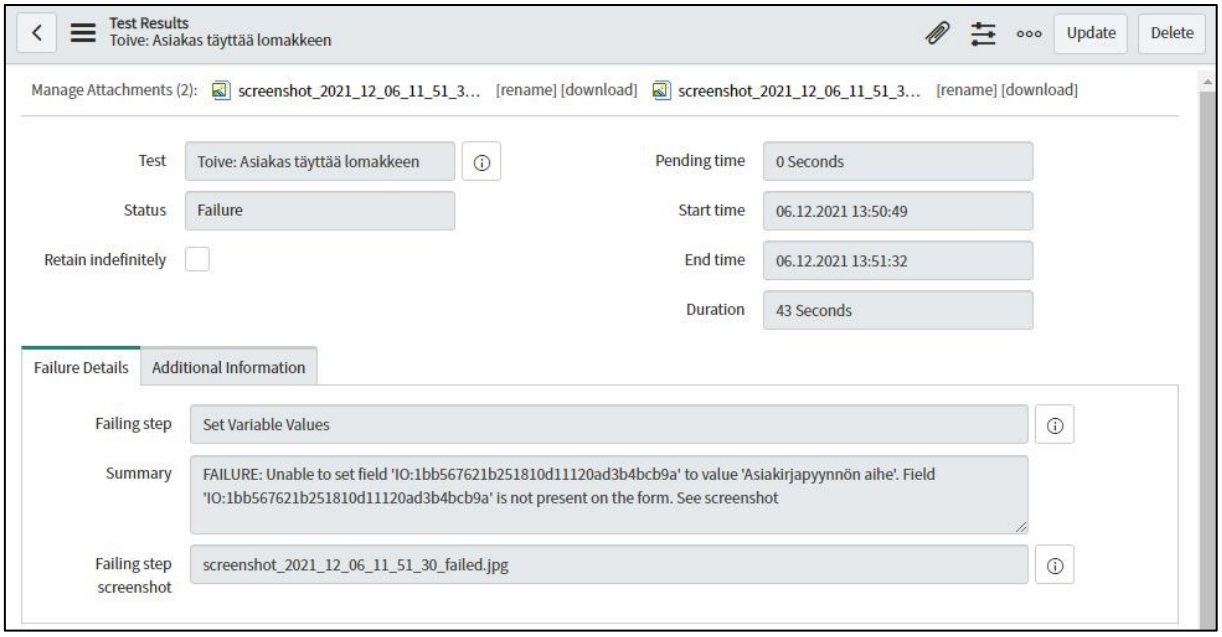
Kuva 38 Testitulos, onnistunut



Epäonnistuneessa testissä tarvitaan vain yksi epäonnistunut testivaihe. Sitten peruspalvelupyyntöprosessin ensimmäisten testien luonnin jälkeen, lomakkeen kenttiin tehtiin pieniä muutoksia. Asiakas täyttää lomakkeen -testin suoritus epäonnistui ja suorituksen yhteenvedosta nähdään, että kyseessä on testivaihe 2 tai 3 (kuva 39). Siirtymällä testitulokseen (kuva 40), havaitaan testin tila epäonnistuneeksi. Testin suoritus kesti 43 sekuntia ja epäonnistunut testivaihe on Set Variable Values (määritä muuttujien arvot).

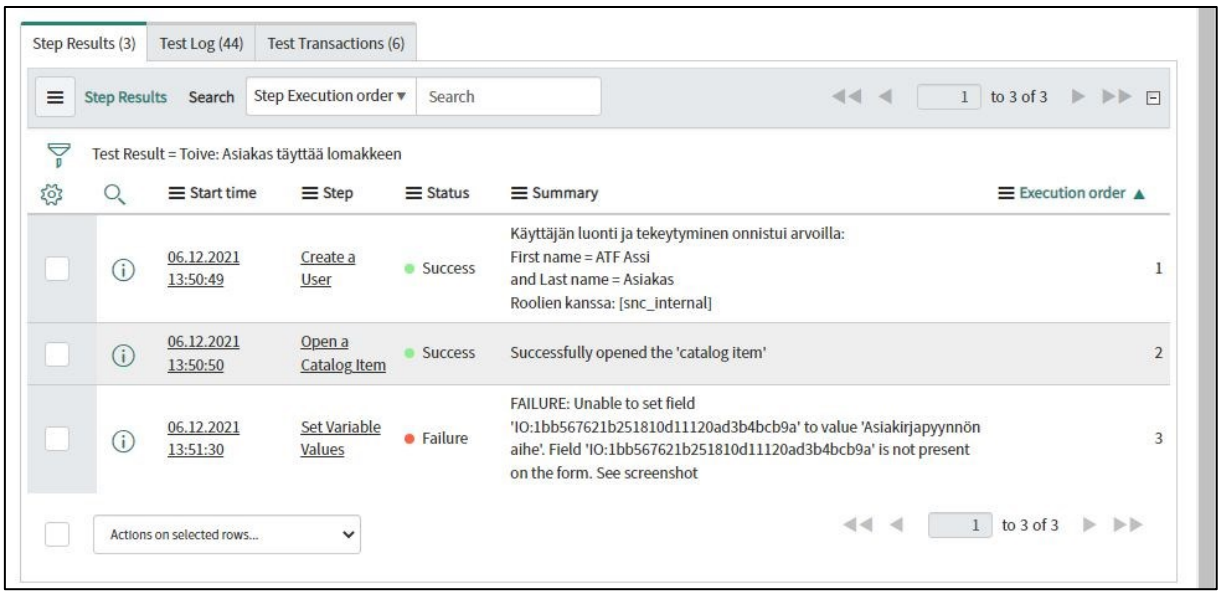
Kuva 39 Testin suorituksen yhteenveto, epäonnistunut

Kuva 40 Testitulos, epäonnistunut

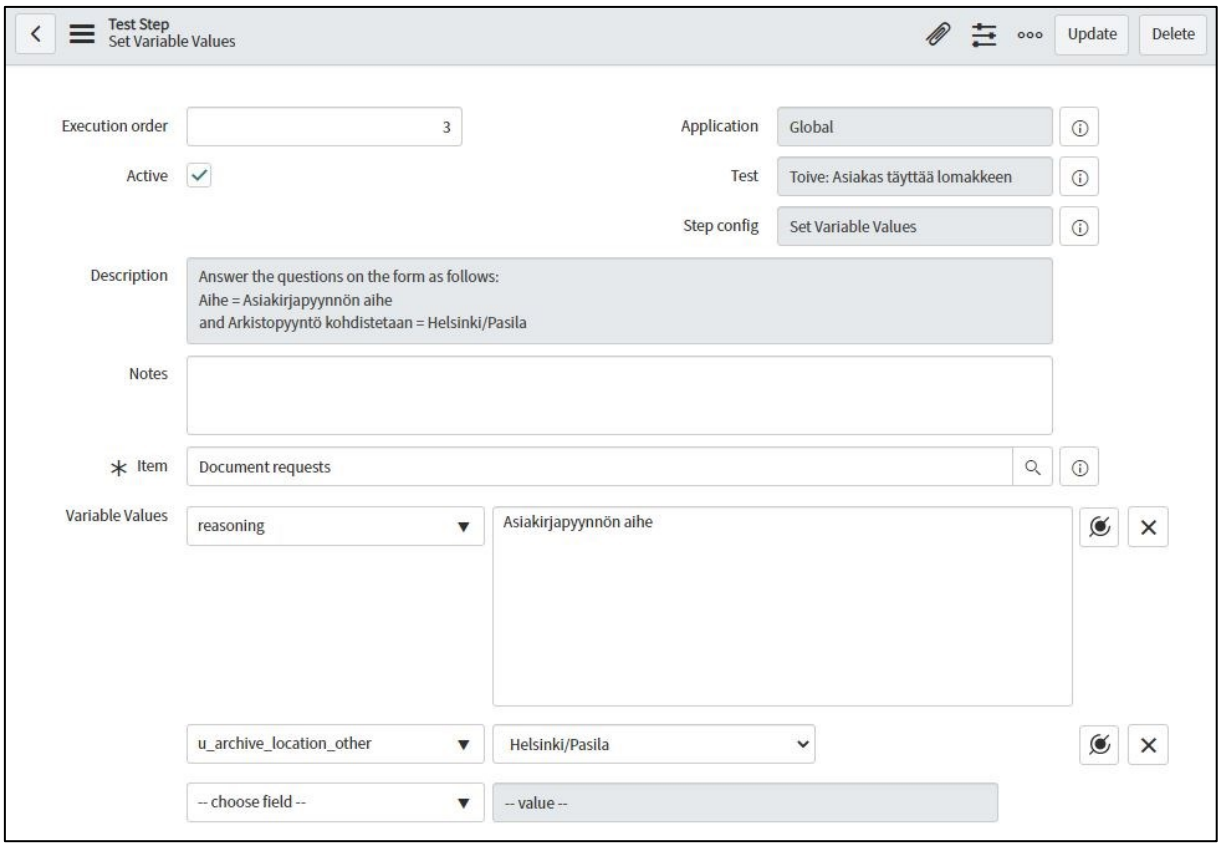


Kuva 41 vahvistaa tiedon, minkä suorituksen yhteenvedosta saatiin, epäonnistunut vaihe on testin kolmas. Epäonnistuneen testivaiheen Summary-saraketta (yhteenveto) tarkastelemalla selviää, että Asiakirjapyynnön aihe -kenttään ei pysty asettamaan arvoa, sillä kenttä ei ole olemassa lomakkeella. Valitsemalla epäonnistuneen vaiheen Step-sarakkeesta, siirrytään vaiheen tarkemmalle sivulle (kuva 42).

Kuva 41 Epäonnistuneen testin testivaiheet



Kuva 42 Epäonnistunut testivaihe

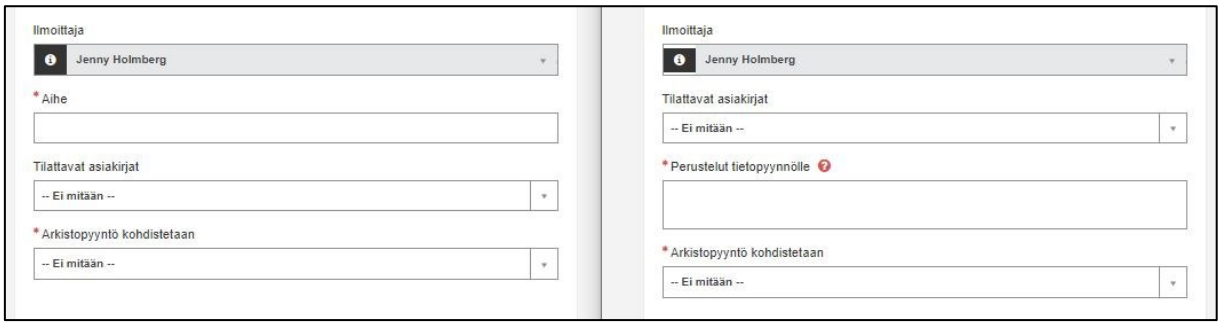


Kuva 42 kertoo Description-kentän perusteella, että kyseisessä testivaiheessa on haluttu asettaa arvot muuttujille Aihe ja Arkistopyyntö kohdistetaan. Testivaiheesta ei kuitenkaan käy tarkemmin ilmi minkä niminen uusi kenttä on, tiedetään vain, että Aihe-kenttää ei ole olemassa. Siirrytään takaisin testituloksiin ja sen liitteeseen, josta löytyy kuvakaappaus epäonnistuneesta vaiheesta (kuva 43). Tarkastelemalla kuvakaappausta, huomataan Aihe-kentän puuttuminen, mutta Arkistopyyntö kohdistetaan -kenttä on edelleen olemassa.

Kuva 43 Testitulos, järjestelmän kuvakaappaus

Varmistetaan vielä lomake-eroavaisuudet, ennen kuin testit päivitetään vastaamaan uutta tilannetta. Kuva 44 osoittaa lomakkeen väliset eroavaisuudet, vasemmanpuoleinen on alkuperäinen ja oikealla päivitetty versio. Aihe-kenttä on poistunut ja tilalle on tullut Perustelut tietopyynnölle. Vaikka lomakkeen kenttien järjestys on myös muuttunut, testi ei epäonnistu sillä niiden järjestystä ei testata. Asiakasportaalin testeissä voisi testata lomakkeen sisältämien kenttien järjestystä, jos sillä on merkitys sen toimivuuteen tai käyttöön. Esim. kenttä 4 tulee näkyviin kentän 3 tehdyn valinnan mukaan, jolloin olisi loogista, että kenttä 4 näkyy lomakkeella vasta kentän 3 jälkeen.

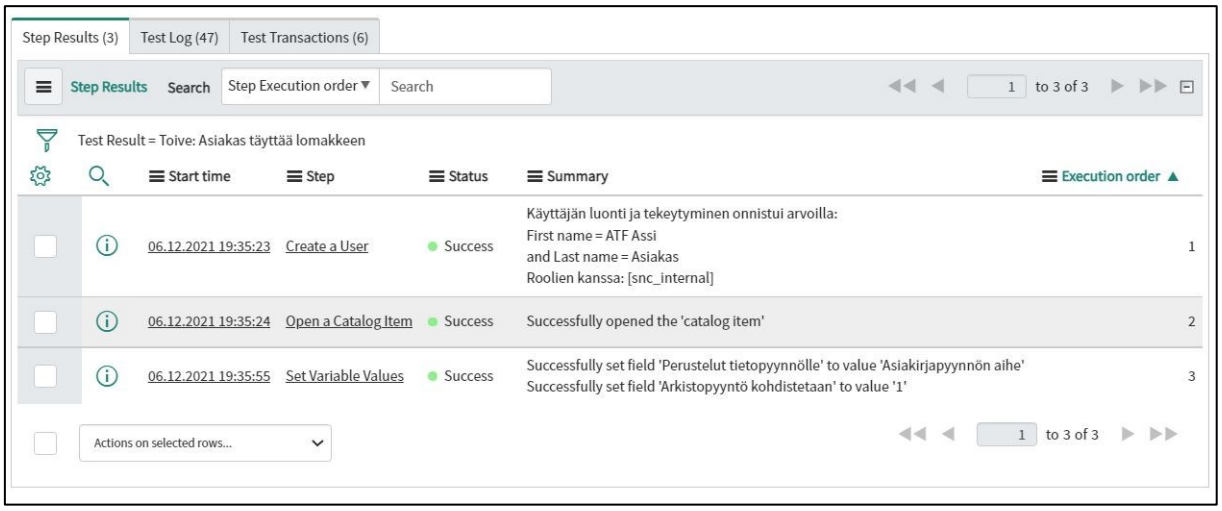
Kuva 44 Lomakkeen kenttien vertailu



Yksinkertainen muutos kentän nimeen, tarkoittaa toimeksiantajan ATF-testien osalta päivitystä 10 testiin. Kaikki peruspalvelupyyntöprosessin testit, jotka sisältävät testivaiheen Set Variable Values oli päivitettävä. Testien ylläpitoon meni aikaa 27 min. Muokkaus sisälsi

10 testin kuvauksen ja -vaiheen päivittämisen. Aiemmin epäonnistunut Asiakas täyttää lomakkeen -testi suoritetaan ATF:ssä onnistuneesti päivityksen jälkeen (kuva 45).

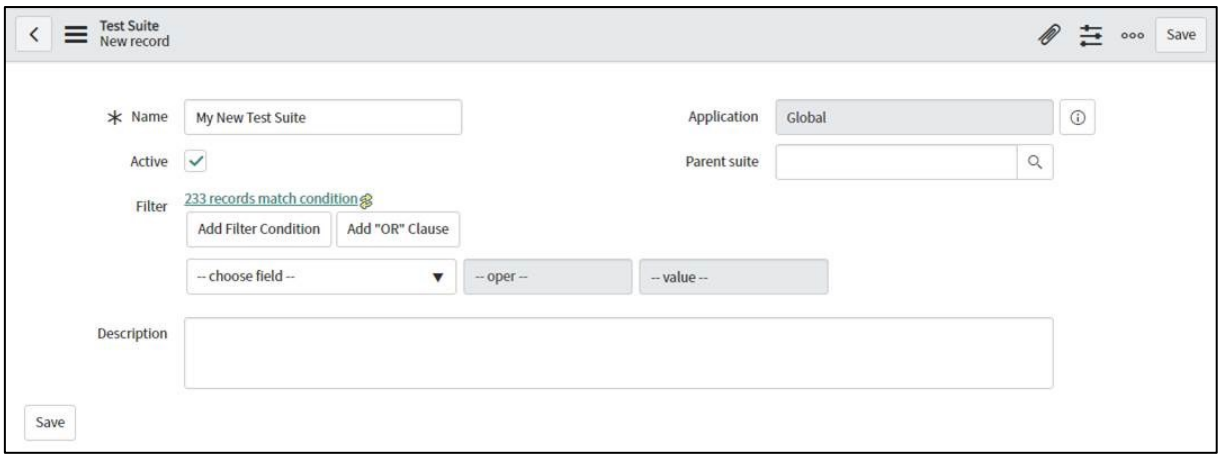
Kuva 45 Päivitetyn testin testivaiheet



6.4.3 Testipaketin luonti

Uusi testipaketti luodaan ATF:n Suites-moduulissa. Testipaketille annetaan nimi ja suodatetaan ehtojen avulla testit, jotka siihen lisätään. Kuva 46 esittää näkymää uuden testipaketin luonnista.

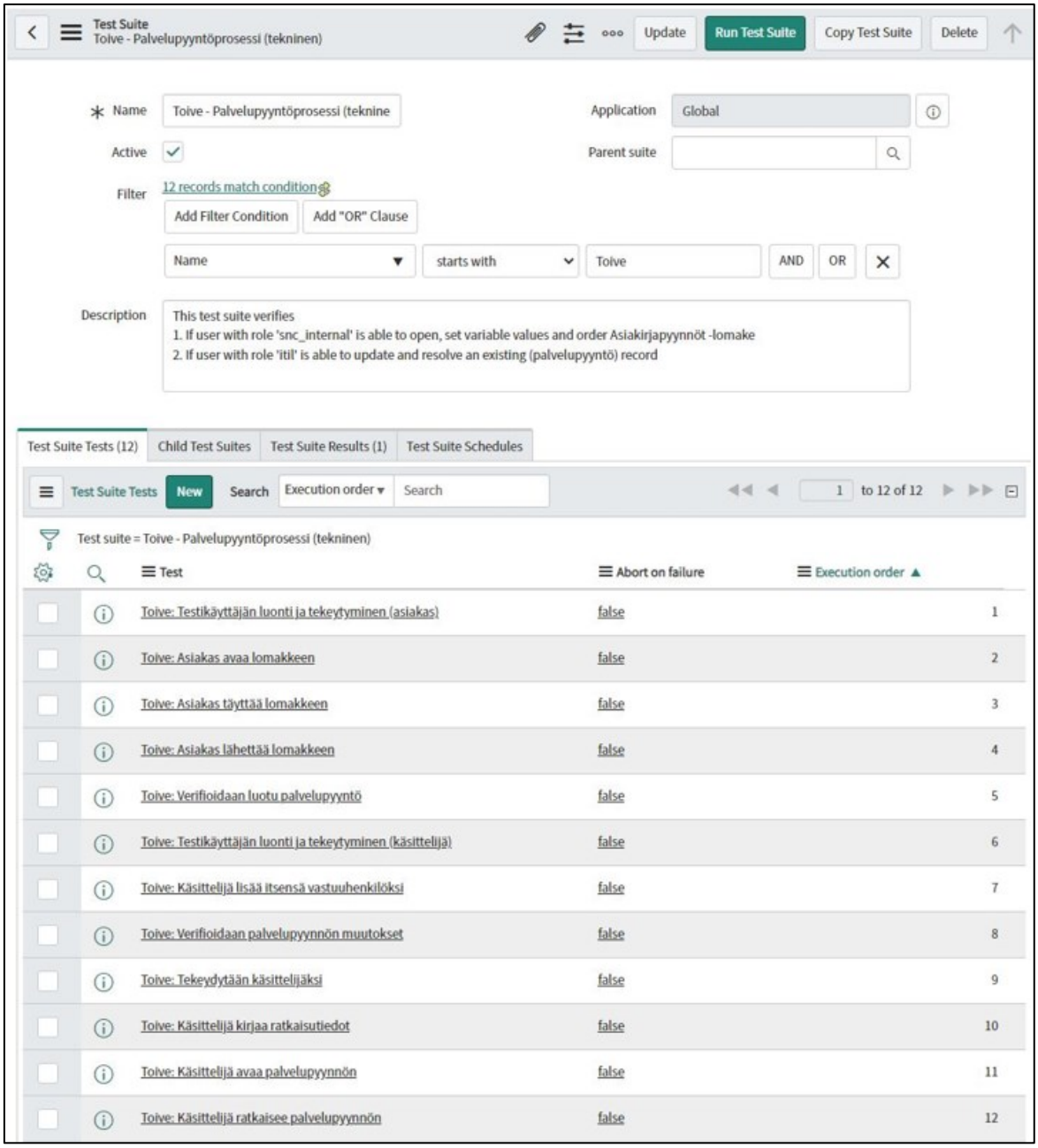
Kuva 46 Uusi ATF-testipaketti



Testien nimeämisen perusteella, toimeksiantajan ensimmäisen testipaketin luonnissa riittää, kun suodatetaan testit, joiden nimi alkaa sanalla Toive. Jatkossa ehtoja tarvitsee lisätä, jotta

saa oikeat testit valikoitua pakettiin. Tallentamisen jälkeen saadaan testipaketin suodatuksen mukaiset testit näkyviin välilehdelle Test Suite Tests (testipakettien testit). Ehtoja voi lisätä, kun testit näkyvät edellä mainitulla välilehdellä, jos niitä on esim. liikaa. Valitsemalla kentän Filter (suodatin) yläpuolella olevan linkin, avautuu uuteen välilehteen [sys_atf_test] -taulun näkymä. Taulunäkymästä on kätevä tarkistaa testivaiheiden kuvaukset, koska niitä ei saa lisättyä omaksi sarakkeeksi testipaketin sivulle. Kun testit on valittu pakettiin, voi niille lisätä halutessaan suoritusjärjestyksen. Suoritusjärjestyksen lisääminen ei ole pakollista, ATF suorittaa tällöin testipaketin testit sattumanvaraisessa järjestyksessä. Virheenselvittely voi tosin olla helpompaa, kun testeillä on suoritusjärjestys. Lisätään testipaketille vielä kuvaus sen sisältämistä testeistä. Kuva 47 esittelee toimeksiantajan ensimmäistä ATF-testipakettia.

Kuva 47 Testipaketti, Palvelupyyntöprosessi



6.4.4 Testipaketin suorittaminen

Testipaketti suoritetaan manuaalisesti valitun paketin sivulla valitsemalla painike Run Test Suite (suorita testipaketti). Testipaketin suorituksesta saa lyhyen yhteenvedon ja tiedon sen onnistumisesta tai epäonnistumisesta, kuten yksittäisen testin suorituksestakin. Kuva 48 on esimerkki epäonnistuneesta testipaketin suorituksesta. Kun testipaketin testi tai testit epäonnistuvat, ne voi suorittaa uudestaan Re-run Failed Tests -painikkeella. ATF suorittaa tällöin yksittäiset epäonnistuneet testit uudestaan, koko testipaketin suorittamisen sijaan.

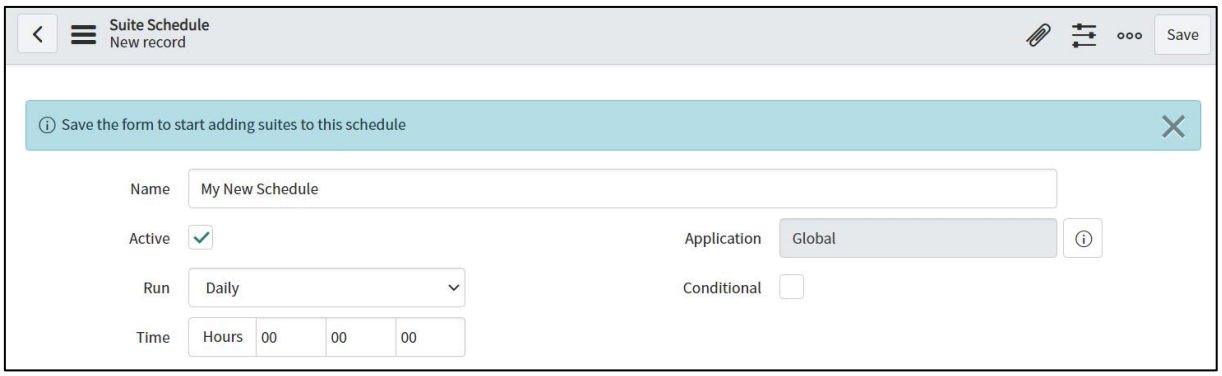
Jos suoritettava testipaketti on ajastettu, on käynnistettävä Scheduled Client Test Runner, eli testin suorittaja, ennen testipaketin ajoitettua käynnistymistä.

Kuva 48 Testipaketin suorituksen yhteenveto

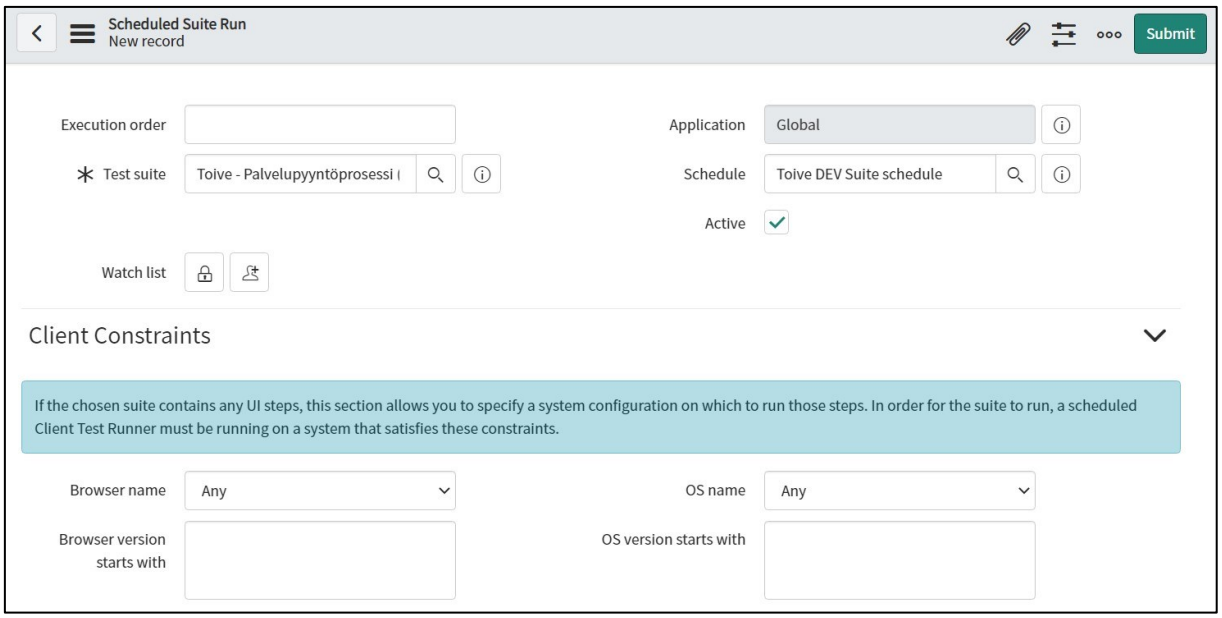
6.4.5 Testipaketin ajastaminen

Testipaketin ajastaminen tehdään Schedules-moduulissa (ohjelmistopaketin aikataulut). Kuva 49 esittää uuden testipaketin ajastamisen näkymää. Testipaketin ajastamiseen tarvitaan tieto, miten usein ja mihin kellonaikaan testipaketti suoritetaan. Testipaketin suorittamisen vaihtoehdot ovat päivittäin, viikoittain, kuukausittain, säännöllisesti, kerran, sekä tarvittaessa. Tallentamisen jälkeen siirrytään näkymälle, jossa lisätään tieto itse testipaketista mikä ajastetaan (kuva 50).

Kuva 49 Uuden ATF-testipaketin ajastus

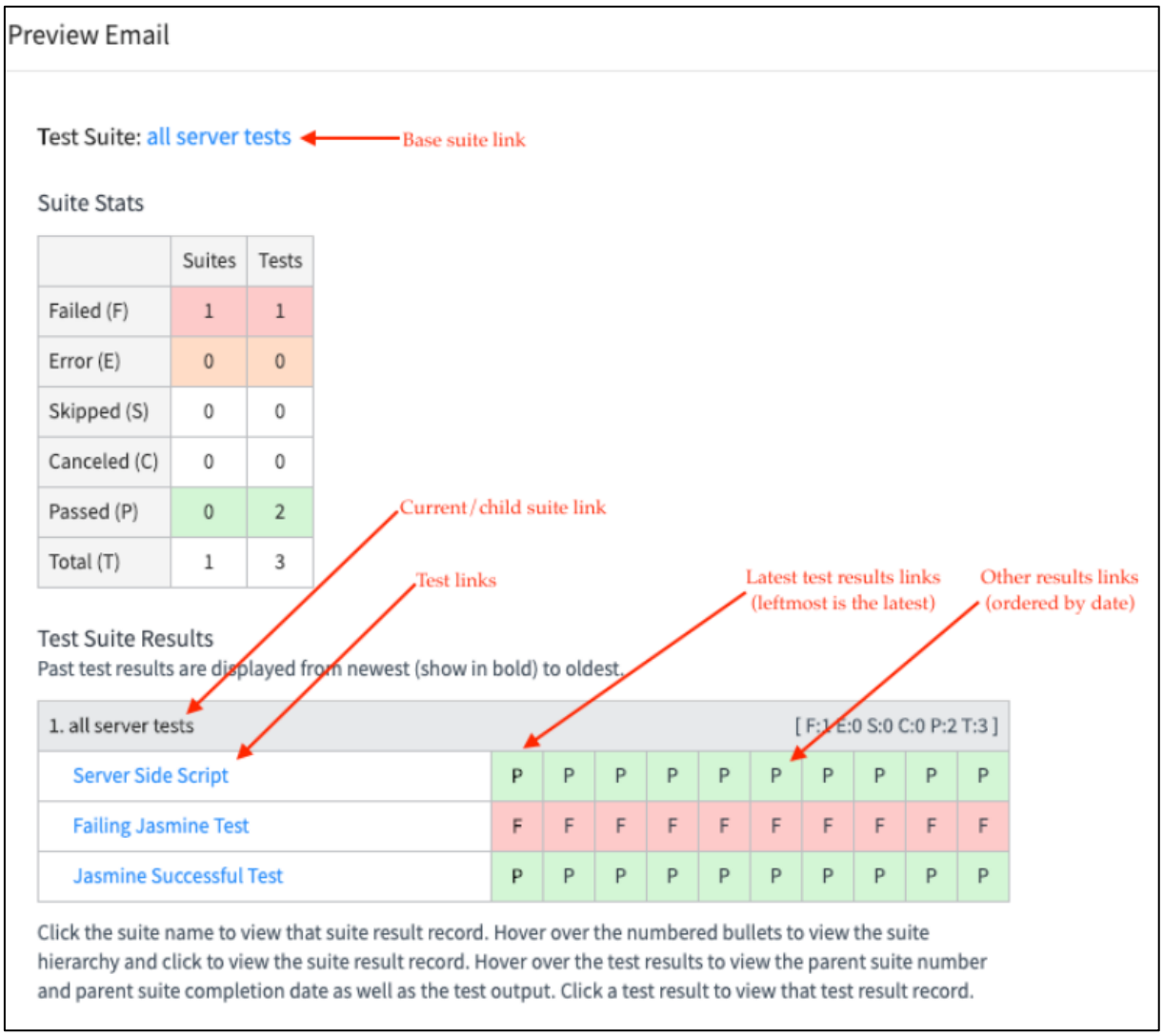


Kuva 50 Ajastetun testipaketin valinta



Sivulla Scheduled Suite Run (ajastettu ohjelmistopaketti) lisätään Test Suite -kenttään testipaketin nimi ja tallennetaan. Huomaa tallennuksen yhteydessä, että järjestelmä täyttää suoritusjärjestys-kenttään arvon 100, jos sen jättää tyhjäksi. Järjestysnumerolla ei ole merkitystä, jos luo vain yhden ajastetun testipaketin. Kenttään Watch list (seurantaluettelo) voidaan lisätä käyttäjiä, joille lähetetään sähköposti-ilmoitus suoritetusta testipaketista. Sähköposti sisältää tiedon testipaketin onnistuneesta tai epäonnistuneesta suorituksesta. Kuva 51 esittää esimerkkiä sähköpostiviestin sisällöstä. Sähköpostin lähetys tulee olla aktivoitu kehitys-, testi-, tai muussa ei tuotanto ympäristössä, jotta järjestelmä voi lähettää sähköposti-ilmoituksen. Testausta varten suositellaan useimmiten jaettua postilaatikkoa ja yhteiskäyttöistä sähköpostiosoitetta.

Kuva 51 Ajastetun testipaketin sähköposti-ilmoitus (ServiceNow, 2021k)



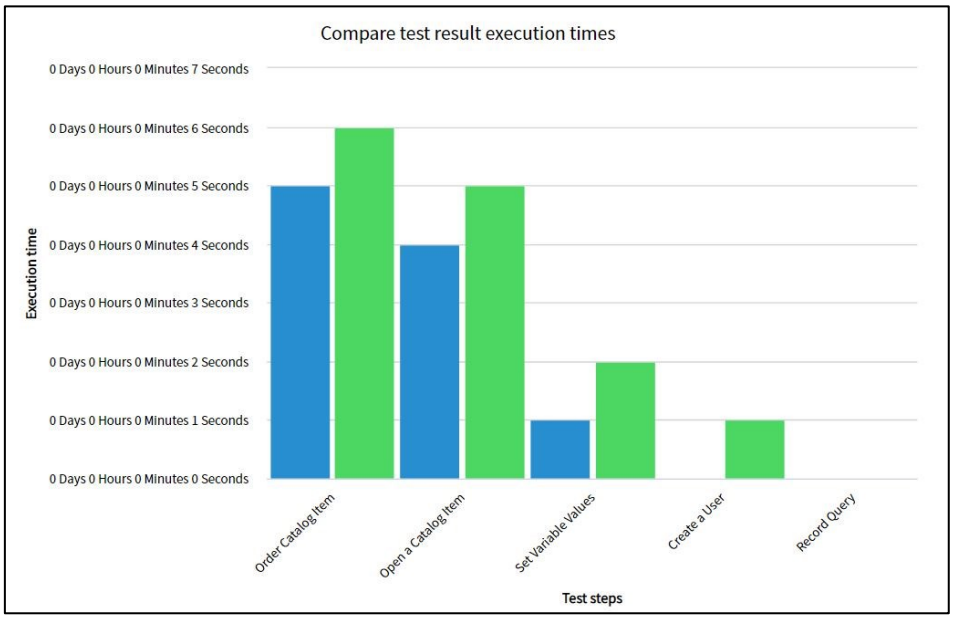
6.4.6 Testitulosten seuranta ja vertailu

Pidemmän ajan testitulosten seurantaa varten kannattaa ottaa itselleen talteen testien ja / tai testipakettien tulokset, sillä järjestelmä poistaa ne automaattisesti 30 päivän kuluttua niiden luonnista. Poikkeukselliset virheilmoitukset kannattaa koota esim. tietämysartikkeliksi myöhempää käyttöä varten, jotta niiden selvittelyyn ei mene tarpeettomasti aikaa niiden toistuessa.

ATF:ssä on mahdollista vertailla automatisoitujen testien tai testipakettien suoritusaikoja keskenään. Yksittäisen testipaketin tuloksia voidaan myös vertailla ajan mittaan. Testi, jonka tuloksia halutaan verrata, valitaan Tests-moduulin kautta tai siirrytään [sys_atf_test] -

tauluun. Testin sivulla valitaan välilehti Test Results (testitulokset), poimitaan ajot, joita halutaan vertailla ja valitaan toiminto Compare test step results (vertaa testivaiheen tuloksia). Järjestelmä luo tämän jälkeen automaattisesti palkkikaavion valituista ajoista (kuva 52).

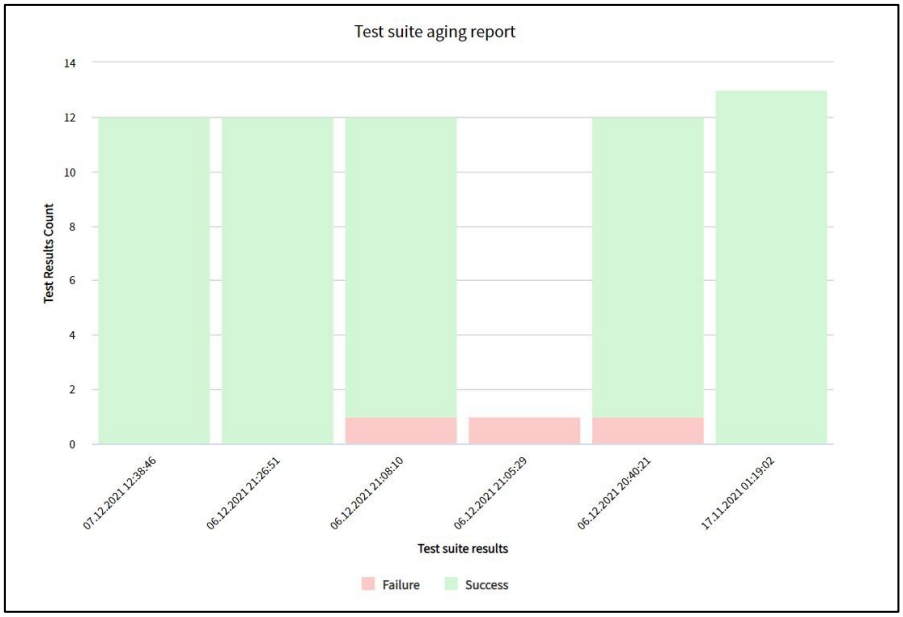
Kuva 52 Testituloksen suoritusajan vertailu



Kuva 52 sininen palkki edustaa aiemmin suoritettua testiä ja vihreä viimeisintä suoritettua testiä. Testin vaiheet eivät ole vertailussa suoritusjärjestyksessä vaan pisimmän suorituskeston mukaisessa järjestyksessä. Testin ensimmäinen vaihe, jossa luotiin käyttäjä on kestänyt sekunnin. Pisin testivaihe valitulta aikaväliltä on testin viimeinen vaihe, jossa verifioitiin luotu palvelupyyntö.

Testipaketin suoritusajan vertailua varten siirrytään Suites-moduuliin, muutoin edetään kuten yksittäisen testin vertailussa. Testipaketin vertailusta voidaan myös tehdä vertailu ajan mittaan. Testipaketin sivulla, valitaan linkki Display aging report (näytä ikääntymisraportti). Järjestelmä luo automaattisesti palkkikaavion onnistuneista ja epäonnistuneista testeistä paketin sisällä (kuva 53). Kaavion palkista voidaan siirtyä edelleen tarkastelemaan esim. epäonnistunutta testiä testituloksen listanäkymään.

Kuva 53 Testipaketin ikääntymisraportti



7 Kokemukset automatisoitavien testien suunnittelusta ja valinnasta

Testien suunnittelussa toimivin ratkaisu oli toiminnallisuuksien priorisointi, testipaketin suunnittelu, toimintojen jakaminen testeihin ja testien käsikirjoitus. Toiminnallisuuksien priorisoinnissa on monta erilaista tapaa. Yksi niistä on laittaa järjestykseen tärkeimmät automatisoitavat kohteet. Kohde voi olla prosessi, toiminnallisuus tai muu tärkeä kokonaisuus. Kohteet arvioidaan toimintojen kriittisyydellä ja virheherkkyyden kautta. Testipaketit suunnitellaan priorisoinnin pohjalta ja jäsennellään automatisoitavat kohteet paketteihin. Yksi paketti voi olla esim. prosessi, portaali tai muu selkeä kokonaisuus. Pohjana voi myös käyttää olemassa olevia manuaalisia testitapauksia. Toimintojen jakaminen testeihin tarkoittaa, että listataan kaikki testipakettiin kuuluvat testattavat asiat, kuten prosessin osat, eri käyttäjien näkökulmat ja erityisesti huomioitavat asiat. Testattavat asiat jaotellaan testeihin. Yhdessä testissä testataan yhtä asiaa, joka on helposti nimettävissä ja kuvailtavissa. Testien käsikirjoituksella tarkoitetaan, että yksittäinen testi kirjoitetaan auki vaihe vaiheelta, ennen sen teknistä toteutusta (kuva 54).

Kuva 54 Esimerkki testin käsikirjoituksesta

Palvelupyyntö-prosessi

1. Testikäyttäjän luonti (asiakas)
2. Asiakas avaa lomakkeen Asiakirjapyynnöt
3. Asiakas täyttää lomakkeen pakolliset kentät
4. Asiakas lähettää lomakkeen portaalista
5. Testikäyttäjän luonti (käsittelijä)
6. Käsittelijä ottaa palvelupyynnön työn alle (vastuuhenkilön lisäys)
7. Käsittelijä ratkaisee palvelupyynnön

Automatisoitavien testien valintaa pohditaan usein toimintojen kriittisyyden ja virheherkkyyden kannalta. Toiminnot, jotka mielletään ja ovat myös toimeksiantajalle kriittiset, ovat asiakaskokemus, lomakkeiden tallennus ja lähetys, automatisoitu asiakasviestintä, prosessin läpikäynti vaiheineen, sekä näkyvyysrajoitukset. Vähemmän kriittisiä ovat esim. coren linkit ja käännökset. Virheherkkiä toimintoja ovat kustomoidut skriptit ja käyttöliittymätoiminnot, sekä kenttien ehdollinen näkyminen tai pakollisuus.

Vakaiksi toiminnoiksi koetaan järjestelmän perustoiminnallisuudet ja staattinen sisältö esim. ohjeteksteissä tai sähköpostiviesteissä.

Tätä kirjoittaessa toimeksiantajalla on järjestelmässä 131 lomaketta, 14 erilaista työnkulkua, 13 prosessia ja 18 toimintoa käytössä. Selvittääkseni mistä testien automatisointi kannattaa aloittaa, päädyin luomaan priorisointimatriisin (kuva 55) yhdessä kollegan kanssa. Kirjasimme prosesseja, toimintoja, integraatioita, lomakkeita ja työnkulkuja Excel-taulukkoon. Pohdimme jokaisen kohteen osalta niiden kriittisyyttä, virheherkkyyttä, muutosherkkyyttä, volyymia, ATF:n hyötyä ja toteutusta. ATF:ään liittyen, pohdintamme olivat täysin mutuntuntumalla tehtyjä, sillä kokemusta testaustyökalusta ei vielä ollut. ATF hyötyihin arvioimme ajansäästön verrattuna manuaaliseen testaamiseen. ATF toteutukseen arvioimme työmäärää testien tekemiseen sekä niiden vaatimaa ylläpitoa tulevaisuudessa.

Pohdimme yleisellä tasolla kohteiden osalta myös, että liittyykö niihin paljon toistoja, onko ajansäästäminen mahdollista automaation avulla verrattuna manuaaliseen testaamiseen. Kohteita vertailtiin niiden stabiliteetin kautta, ovatko ne kriittisiä tai osa kriittistä kokonaisuutta, sekä käyttääkö useampi prosessi samaa toimintoa, jolloin toteutuskustannus olisi suhteessa pienempi. Koitimme myös huomioida ovatko kohteet alttiita inhimillisille virheille ja vaatiiko testin suorittaminen erityistä osaamista.

Kuva 55 Kehitystiimin priorisointimatriisi

						(28.9.2021)	ATF hyöty	ATF toteutus		
			Kriittisyys	Virheherkkyys	Muutosherkkyys	Volyymi	Ajansäästö vs. man.	Työmäärä	Ylläpito	yht.
	Prosessi / toiminto		1=suuri - 3=pieni	1=suuri - 3=pieni	1=pieni - 3=suuri	1=suuri - 3=pieni	1=suuri - 3=pieni	1=pieni - 3 suuri	1=pieni - 3-suuri	
4	Häiriönhallinta (perusprosessi)	P	1	1	1	2	1	2	1	9
5	Palvelupyyntöjen hallinta (perusprosessi)	P	1	2	1	1	1	1	1	8
6	Ongelmanhallinta	P	3	3	1	3	1	3	1	15
7	Muutoksenhallinta	P	3	3	1	3	1	3	1	15
8	Tietämyksenhallinta	P	3	3	1	3	3	1	1	15
19	Robotiikka (BluePrism)	T	1	1	3	3	1	3	3	15
22	Näkyvyysrajoitteet - portaali (lomakkeet)	T	2	3	1	3	3	1	1	14
23	Käännöstaulut	T	2	1	3	1	1	1	1	10
24	Työnkulut	T	1	2	2	1	1	3	2	12
62	**Portaalilomakkeet**		1	2	2	1	1	3	2	12
63	Häiriöilmoitus		1	1	1	2	1	2	2	10
64	Yleinen palvelupyyntö		2	3	1	1	2	1	1	11
65	Palvelupalaute		3	3	1	3	3	1	1	15
66	**Asiakirjat ja kirjaamo (3)**									0
67	Asiakirjapyynnöt		3	3	1	1	2	1	1	12
68	Kirjaamo ja IDA		3	3	1	3	2	2	1	15
69	Tietopyynnöt vanhoista järjestelmistä		3	3	1	3	2	2	1	15

Priorisointimatriisissa määriteltiin automatisointiin liittyvät kriteerit sekä arvioitiin kohteet kriteereitä vastaan. Kriteerit ovat kriittisyys, virheherkkyys, muutosherkkyys, volyymi, hyöty ja toteutus. Kriittisyydellä tarkoitetaan kohteen tärkeyttä liiketoiminnan kannalta. Virheherkkyydellä pyritään ennustamaan, kuinka usein kohteessa esiintyy virheitä,

huomioiden sen monimutkaisuuden. Muutosherkkyys tarkoittaa arviota, kuinka usein kyseiseen kohteeseen tehdään päivityksiä. Volyymi vastaa kohteen käyttömäärään. Hyöty yrittää ennustaa automatisoinnista saatavaa hyötyä verrattuna manuaalitestaukseen. Toteutus pyrkii arvioimaan työmäärää, onko testin tekeminen työlästä toteuttaa, sekä omana arviona testin ylläpitoon menevä aika, onko sitä työlästä ylläpitää.

Kohteet pisteytettiin 1–3 kriteerien mukaan, esim. jos jokin kohde on kriittinen, sai se pisteen 1. Matriisin viimeinen sarake laskee kohteiden saamat pisteet yhteensä. Mitä pienemmät yhteispisteet, sitä todennäköisempi kohde on automatisoinnille. Kun kohteen kriteerien yhteispisteet ovat isot, tarkoittaa se useimmiten, että se ei ole kriittinen, se ei ole virhe- tai muutosherkkä, volyymi on pieni, ei tuo ajansäästöä, eikä se ole toteutuksen kannalta järkevä automatisoida.

Kehitystiimin priorisointimatriisin tukena käytimme kuvaa, josta oli apua kohteiden pisteyttämiseen (kuva 56). Vaikka kuvan lähde käsittelee tuotekehityksen priorisointia Lean Startup -ajattelun mukaan, sopii se erittäin hyvin käytettäväksi myös testien automatisoinnin pohdinnan avuksi.

Kuva 56 Lean Startup priorisointimatriisi (Hietaniemi, 2019)

Kun saimme kollegan kanssa priorisointimatriisin tehtyä, totesimme että ihanteellinen automatisoitava kohde ATF-sovelluksella on kriittinen, virheherkkä, ei altis muutoksille, usein toistuva, hidas testata manuaalisesti, nopea toteuttaa ja helppo ylläpitää.

Priorisointimatriisia käytettiin hyödyksi, kun kehitystiimille luotiin Microsoft O365 Forms-sovelluksella kysely, jonka tarkoituksena oli selvittää heidän toiveitansa ja odotuksia testiautomaation hyödyntämisestä. Matriisiin perustuen kyselyssä huomioitiin kohteet, jotka olivat kokoluokaltaan ajallisesti mahdollisia toteuttaa opinnäytetyötä varten. Tutkimuskysely esitellään kokonaisuudessaan luvussa 9.

Automatisoitavien testien lopulliseen valintaan vaikuttivat kyselyn tulokset sekä priorisointimatriisin kohteiden yhteispisteet. Kyselyn vastausten perusteella, tiimi piti hyödyllisimpänä automatisoida volyymiltaan suurin prosessi, joka toimeksiantajalla on palvelupyyntöprosessi. 83 320 kpl palvelupyyntöä oli kirjattu järjestelmään 11 kuukautta sen käyttöönottamisesta. Palvelupyyntöjen perusprosessi on liiketoiminnan kannalta kriittinen. Pidimme prosessin virheherkkyyttä mahdollisena ja se on stabiili, sillä muutoksia siihen ei varsinaisesti ole kohdistunut. Priorisointimatriisissa palvelupyyntöjen perusprosessi sai 8 pistettä, joten se päätettiin toteuttaa.

Tiimi toivoi lomakkeista automatisoitavaksi ensimmäiseksi sen, jota asiakkaat eniten käyttävät portaalissa. Suurin volyymi yksittäisellä lomakkeella oli asiakirjapyynnöillä, 11 536 kpl. Asiakirjapyynnöt-lomake ei ole kriittinen eikä virheherkkä, käyttöönoton jälkeen lomakkeen kenttiin on tehty muutoksia, jotka huomioitiin automatisoinnin yhteydessä. Ajansäästö testaamisen kannalta ei ole suuri, mutta ei se ole pienikään. Lomake on yksinkertainen, mutta siinä on pudotusvalikkoja, joiden valinnat tulee testata läpi päivityksen tai muutoksen myötä. ATF-testin työmäärä arvioitiin pieneksi, juuri sen yksinkertaisuuden vuoksi, samoin testin ylläpitoon kuluva aika. Priorisointimatriisin yhteispisteet asiakirjapyynnöt-lomakkeelle on 12. Työmäärällisesti lomakkeen testauksen automatisointi oli mahdollista toteuttaa tämän työn puitteissa.

8 ATF:n käyttöönottosuunnitelma

Tässä luvussa käsitellään toimeksiantajan ATF:n käyttöönottosuunnitelmaan vaikuttavia perusteita. Yksittäinen vaikuttava asia on säännölliset versiopäivitykset, jotka aiheuttavat laajan järjestelmän manuaalisen testaustarpeen kehitystiimille. Versiopäivityksen tarkoitus ja sen vaikutus ATF:ään esitellään alaluvussa 8.1. Toinen vaikuttava näkökulma on sovelluksen hyödyntäminen jatkuvassa kehittämisessä, jota varten haastattelin kolmea eri roolissa toimivaa kehitystiimin jäsentä. Haastattelun tarkoituksena oli muodostaa kokonaiskuva eri rooleissa toimivien jäsenten näkökulmista. Haastattelu oli puolistrukturoitu, joka tarkoittaa, että kysymykset olivat etukäteen mietittyjä ja kaikille samoja. Haastattelun tuloksia esitellään alaluvuissa 8.2, 8.3, 8.4 ja 8.5.

8.1 ServiceNow-versiopäivitys

ServiceNow julkaisee kaksi versiopäivitystä vuodessa ja toimeksiantaja on päättänyt ottaa käyttöön viimeisimmän version mahdollisimman pian julkaisun jälkeen. ServiceNow-pohjainen toiminnanohjausjärjestelmä on ollut toimeksiantajalla tuotantokäytössä tätä kirjoittaessa vuoden. Kehitystiimillä on kokemusta kolmesta versiopäivityksestä. Ensimmäinen versiopäivitys tehtiin jo ennen käyttöönottoa, toinen puolen vuoden tuotantokäytön jälkeen ja kolmas päivitys ensimmäisten ATF-testien luonnin jälkeen.

Versiopäivitykseen valmistautumista varten kloonataan tuotantoympäristö alempiin ympäristöihin. Koska ATF-testit suoritetaan vain alemmissa ympäristöissä, tulee muistaa siirtää testit update setissä ensin tuotantoon, jotta ne eivät häviä kloonauksen takia. Kun kloonaus on tehty, aloitetaan versiopäivitys alimmassa ympäristössä, esim. kehitysympäristössä. Versiopäivitystä varten voi luoda erillisen testaussuunnitelman ja testitapauksia. Kehitysympäristö testataan ja varmistetaan versiopäivityksen jälkeen, että toiminnot toimivat edelleen. Kloonauksen ja onnistuneen päivityksen jälkeen ATF-testejä voi jälleen suorittaa ei-tuotantoympäristöissä.

8.2 Automatisoitujen testien hyödyntäminen kehitystiimissä

Tämä luku taustoittaa kolmen kehitystiimin jäsenten vastauksia haastattelussa esitettyihin kysymyksiin. Testaustyökalu ei ole kehitystiimin varsinaisessa käytössä vielä. Vastaukset perustuvat enemmän kuvitteelliseen tilanteeseen, jossa työkalu on käytössä ja testejä olisi keskeisimmistä ja haastavimmista toiminnoista, lomakkeista ja työnkuluista. Haastattelun kysymykset ja vastaukset kokonaisuudessaan ovat esitelty luvussa 10.

Kehittäjän mielestä testituloksia tulisi hyödyntää kehitystyön tukena, niin että testit suoritetaan säännöllisesti ja onnistuneesti. Scrum master, joka vastaa kehitystiimin päivittäisestä tekemisestä, näki että huolella rakennettu ATF tehostaisi tekemistä, varmistaen paremman laadun ja resurssien käytön kohdistamisen. Testausvastaavan mukaan testituloksia voisi hyödyntää samalla tavalla kuin manuaalisen testauksen tuloksia. Automatisoitujen testien avulla havaittaisiin testausjakson aikana nopeammin toiminnot, jotka poikkeavat määrityksistä. Testausvastaava toi esille myös oivalluksen, että testitulokset auttaisivat kehittämään myös ATF-testitapauksia, sekä kehitysprosessia kokonaisuutena.

Roolipohjaiset eroavaisuudet käyvät ilmi vastauksista, kun haastateltavilta kysyttiin millä tavalla ATF-sovellus auttaisi heidän omassa työssään. Kehittäjän mielestä ATF toisi varmuutta työhön, kun automaatio löytäisi isoimmat ongelmat ja heijastusvaikutukset epäonnistuneiden testien muodossa. Scrum master pohti, että ATF lisäisi resursseja kehittämiseen, kun sovellus hoitaisi osan testauksesta. Testausvastaavaa ATF auttaisi keskittymään enemmän testauksen suunnitteluun ja koordinointiin. Hänen mielestään kriittiset ja monimutkaiset prosessit ovat vaikeita automatisoida, ja niihin voisi käyttää enemmän aikaa manuaalisen testauksen muodossa.

Haastatteluista kävi ilmi samoja ajatuksia sovelluksen testitulosten hyödyntämisestä mitä itsellä oli. Testejä tulee suorittaa säännöllisesti ja mahdollisuuksien mukaan kehitystyön yhteydessä. Uuden kehityksen aikana ei välttämättä tarvitse luoda testiä kyseiselle muutokselle tai toiminnallisuudelle, mutta tulisi suorittaa kaikki testit, jotka käyttävät samaa toiminnallisuutta ja ne millä voisi olla heijastusvaikutuksia. Sovelluksen avulla varmistetaan parempi laatu ja nopeutetaan testausprosessia. Olen samaa mieltä kehittäjän kanssa, että testit kannattaisi rajata portaalin ja taustajärjestelmän välillä. Portaalitestit käsittelisivät

käyttöliittymää ja lomakkeita. Taustajärjestelmän testit käyvät läpi toiminnallisuuksia, jotka tapahtuvat esim. portaalilomakkeen lähettämisen jälkeen, kuten sen työnkulkua. Testausvastaava mainitsi haastattelussa, että ATF olisi hyvä bulkkitestauksessa, esim. lomakekenttien vertailussa ympäristöjen välillä. Sovelluksessa ei ole testivaihetta, jolla tarkistetaan lomakekentän nimi. Sen sijaan, voidaan testata syötteen lisäämistä tiettyyn kenttään Set Field Values -testivaiheella. Suorittamalla testit kehitys- ja testiympäristössä, voidaan tehdä ympäristöjen välistä vertailua. Lomaketestit tulevat epäonnistumaan testiympäristössä, mikäli kenttien nimet eroavat kehitysympäristön lomakkeista.

8.3 Käyttöönoton hyödyt ja mahdolliset haitat

Tämä luku esittelee kehitystiimin näkökulmasta ATF:n hyödyt ja haitat, joita käytiin läpi haastattelun muodossa. Kehitystiimiä edustavat kolme jäsentä, jotka toimivat eri rooleissa tiimissä.

Kehittäjälle sovellus toisi hyötyä lisäämällä testikattavuutta, helpottaen jatkokehitystä ja tuotantoon vientiä, kun manuaalitestauksen voisi jättää vähemmälle. Scrum masterin näkökulmasta katsottuna sovelluksen tuoma hyöty on resurssien kohdentaminen. Sovelluksen avulla voidaan tarkistaa laadullisesta näkökulmasta teknistä toteutusta. Testausvastaava toivoo, että kriittisten prosessien manuaalitestaukseen vapautuu resursseja, sovelluksen tehostaessa ja nopeuttaessa perustestauksen. Suoranaisia haittoja kukaan tiimin jäsenistä ei näe sovelluksessa. Haittana ei kehittäjän mielestä ollut kuin testien suorittamattomuus ja testien päivittämättä jättäminen, jos sovellus otetaan käyttöön. Testien päivittäminen tai korjaaminen vasta useamman muutoksen jälkeen tuottaisi enemmän harmia kuin iloa. Scrum master ei myöskään osaa kuvitella haittoja sovelluksesta, muuta kuin että testejä täytyy aika ajoin tarkastella. Kehitystiimi ei hyödy testistä, joka käsittelee harvoin käytettyä lomaketta, sen takia on todella tärkeää tietää mihin ja milloin sovellusta käytetään. Testausvastaava mietti, että sovelluksen puutteena on tietynlaiset E2E-työnkulut, joita ei voida testata yksin automaation avulla.

Kehitystiimin jäseniltä kysyttiin myös heidän mutututuntumaansa, lyhyellä ja ytimekkäällä kysymyksellä, tuoko sovellus enemmän hyötyä vai haittaa kiireiseen arkeen. Kehittäjälle sovellus toisi hyötyä. Testausvastaavalle sovellus toisi enemmän hyötyä kuin haittaa, sillä

kaikki automaatio on hyvästä, kunhan sen ylläpitoon on aikaa ja resursseja. Scrum master toivoi kiireiseen arkeen, että sovellus tarjoiltaisiin valmiina ja toimivana kokonaisuutena. Lyhytnäköisesti ajateltuna siitä on hänen mielestänsä haittaa. Sovellus vaatii aikaa testien luontiin ja hyötysuhteen saavuttaminen satoja testejä. Pidemmällä aikavälillä taasen sovelluksesta on enemmän hyötyä kuin haittaa. Hän arvioi, että sovellus tuottaa kehitystiimille hyötyjä puolen vuoden tai vuoden sisällä.

Haastattelujen perusteella kehitystiimin jäsenten vastauksista on havaittavissa roolipohjaiset eroavaisuudet, mikä on mielestäni hieno asia. Sovelluksen hyödyt ja haitat näyttäytyvät kehitystiimin jäsenille erilaisena, riippuen minkälaista työtä tekee. Yleisellä tasolla testien luonti vie alkuun paljon työaikaa ja voi olla vaivalloista, se ei ole sovelluksen varsinainen haittapuoli vaan osa sen käyttöönottoa. Käyttöönoton jälkeen tulee työn suunnittelussa huomioida sovelluksen vaatima resursointi käyttöön ja ylläpitoon. Olen samoilla linjoilla scrum masterin kanssa, että sovelluksen todelliset hyödyt kehitystiimille tulevat vasta myöhemmin, edellyttäen että testejä luodaan, ylläpidetään ja suoritetaan säännöllisesti. Testejä kannattaa luoda vähitellen, toiminnallisuus tai lomake kerrallaan. Vähitellen tekemällä ja lähes huomaamatta testikattavuus voisi vuodessa hyvinkin olla 50 %, sisältäen käyttöliittymä- ja taustajärjestelmän testejä.

8.4 Sovelluksen hyödyntäminen jatkuvassa kehittämisessä

Kehittäjä oli sitä mieltä, että sovellusta voisi hyödyntää oman kehitystyön tukena, ikään kuin tukiverkkona, varmistaen ettei rikkonut mitään järjestelmässä. Lisäksi hän mainitsi, että sovellusta voisi hyödyntää uutta toiminnallisuutta varten, eli tarkistaa että koodi on oikein tekemällä automaatiotestin. Scrum master toivoi, että sovellusta hyödynnettäisiin ennen kaikkea varmistamaan järjestelmän laatu ja avustamaan muutosten tuotantoon vientiä, kun suurin osa testauksesta suoritettaisiin automaation avustuksella. Testausvastaava näki, että sovellusta voisi hyödyntää enemmän kokonaisuuksissa, tunnistamalla riippuvuudet toisiin järjestelmiin tai prosesseihin, joihin tehdyn muutoksen ei pitäisi vaikuttaa. Sovellusta tarvitaan jatkuvan kehityksen rinnalle pitämään huolta mahdollisista sivuvaikutuksista, uudet osa-alueet testataan manuaalisesti aina hyvin, mutta on mahdotonta huomioida kaikkia riippuvuuksia. Sovellus voisi jopa mahdollistaa nopeamman kehityksen tuomalla

automatisoidun regressiotestauksen avulla esille heti virhetilanteet. Tällöin sykli koodin muutoksesta valmiiksi ja toimivaksi tuotteeksi olisi nopeampaa automaation keinoin.

Aikataulullisesti kehittäjä ja testausvastaava olisivat valmiita ottamaan sovelluksen päivittäiseen käyttöön heti. Kehittäjä totesi testien luonnin olevan alkuun tuskaista niiden määrän takia. Testejä voisi alkuun tehdä uuden kehityksen tai muutoksen mukaan. Testausvastaava ehdottaa alkuun pientä testipakettia, jotta sovelluksesta ja prosessista saadaan ensin käyttökokemuksia. Päivittäinen käyttö kehitystiimissä voisi olla testausvastaavan mielestä vuoden ensimmäisen neljänneksen aikana, kunhan kokonaisuus on pieni. Scrum master totesi testien rakentamisen vaativan alkuun niin paljon aikaa, että sovelluksen käyttö päivittäisessä työssä olisi todennäköistä vasta vuoden päästä. Silloin kehitystiimi voisi hyötyä sovelluksesta jo sprinteittäin.

Sovelluksen käyttöönotto ja sen hyödyntäminen vaatii resurssin tai resursseja, joten haastateltavilta kysyttiin myös kenen tulisi vastata sen käytöstä. Kehittäjä ehdotti muutamaa henkilöä, jotka vastaisivat organisoinnista, loisivat prosessin ja seuraisi testituloksia. Scrum master piti tehtävää sopivana testausvastaan roolissa toimivalle kehitystiimin jäsenelle. Roolin vastuulla olisi automaatiotestauksen kokonaisuuden edistäminen ja testien suunnittelu. Rooli myös seuraisi testituloksia ja pitäisi huolen, että testit päivitetään. Kehitystiimille jalkautettaisiin tekeminen ja vastuu testien luonnista ja päivittämisestä oman kehitystyön ohella. Testausvastaava kertoo testausvastaavan roolin omistavan testausprosessin. Prosessin tulisi vastata miten testitapauksia luodaan, miten niitä päivitetään ja keneltä varmistetaan voiko testin päivittää. Roolin vastuulla voisi olla ajastettujen testipakettien testitulosten seuranta. Lisäksi voisi olla nimetty henkilö tarvittaessa tuuraamassa.

Sovelluksen hyödyntäminen haastateltavien toiveiden ja ajatusten mukaisesti, testipaketit suunnitellaan kattamaan tärkeimmät toiminnallisuudet ja lomakkeet, sekä virheherkät ominaisuudet. Suunnittelua varten kannattaa hyödyntää priorisointimatriisia, jossa edellä mainitut kohdat ovat pisteytetty. Näiden lisäksi monimutkaiset prosessit ja integraatiot, joihin aika ajoin tehdään muutoksia, olisivat hyviä kohteita automatisoinnille jatkuvan kehittämisen tarpeeseen. Testien päivittämisen tulee olla osa järjestelmän kehitys- ja muutostöitä, jotta kasvatetaan sovelluksen käyttöönoton hyödyt. Kuten kehittäjä

haastattelussaan mainitsi, niin testejä suorittaessa voi huomata loogisen virheen, jota ei osannut huomioida, tai sen vaikutuksia muualle järjestelmässä. Testien säännöllinen suorittaminen ja testitulosten seurannalla tulisi välttyä isoimmilta virheiltä, jotka johtuisivat puutteellisesta testauksesta ja ennakoimattomista vaikutuksista. Vastuuta sovelluksesta ja automatisoinnista kokonaisuutena ei kannata jättää yhden henkilön varaan. Haastateltavat olivat samaa mieltä sovelluksen käytön vastuun jakamisesta ainakin kahdelle henkilölle. Itse ajattelin roolituksen myös kahdelle henkilölle, testausvastaavalle ja toiselle tiimin jäsenelle. Henkilöt loisivat sovelluksen käytön prosessin ja jalkauttaisivat kehitystiimille tekemisen. Testausvastaava hoitaisi organisoinnin ja yhdessä tiimin jäsenen kanssa vastaisivat testipakettien säännöllisestä suorittamisesta ja testitulosten seurannasta.

8.5 Kehitystiimin kouluttaminen ja ohjeistus

Osana sovelluksen käyttöönottosuunnitelmaa tuli harkittavaksi koulutusmateriaalin valmistelu tiimin käyttöön. Yhtenä vaihtoehtona olisi ollut Microsoft Teamsin välityksellä järjestetty koulutus, jossa olisi esitelty sovellus ja sen toiminnot esitysmateriaalin ja käytännön kautta. Koulutustallenne olisi jäänyt tiimin jäsenten myöhempää käyttöä varten tai sitä voisi käyttää osana uuden henkilön perehdytystä tiimin töistä. Koulutusmateriaalin ja -tallenteen päivitystarve tulisi tosin tarkastella ServiceNow-versiopäivitysten yhteydessä muutosten osalta. Toisena vaihtoehtona oli tietämysartikkeli sovelluksesta, tai lähinnä sen käyttöön liittyvistä käytänteistä. Tämä olisi helpoin toteuttaa, sekä myös kevyt päivittää.

Kehittäjä vastasi haastattelussa, että koulutusmateriaaliksi riittäisi konseptitason malli, miten testit nimetään ja millä tasolla ne tehdään. Scrum masterin mielestä tiimi ei tarvitse teknisiä ohjeita sovelluksen käyttöön, vaan arvostaisi enemmän vinkkejä käytännön tekemiseen, esim. mitä tulee ottaa huomioon ja parhaat käytännöt. Scrum master tuo myös esille saman kuin kehittäjä, eli ohjeet testien nimeämiseen ja testipakettien rakentamiseen. Testausvastaava peräänkuuluttaa selkeitä ohjeita sovelluksen käytöstä, jotta on selvää millä tavalla testaustyökalua käytetään. Testausvastaavan mielestä tietämysartikkeli käytänteistä riittäisi, sen ylläpidettävyys on parempi kuin esim. koulutustallenne. Pohdittavaksi jäi, riittääkö yksi artikkeli vai tulisiko olla artikkelikokonaisuus.

Haastateltavat olivat yksimielisiä koulutusmateriaalista, tietämysartikkeli riittäisi tiimin ohjeistukseen sovelluksen käytöstä. Varsinaista koulutustilaisuutta ei pidetty tarpeellisena, joten sellaista ei järjestetty. Kehitystiimiä varten kirjoitettiin tietämysartikkeli otsikolla ATF testit, testipaketit ja parhaat käytännöt. Artikkelissa kerrottiin testien ja testipakettien nimeämiskäytännöistä ja molemmista oli lyhyet esimerkit. Testipakettien suunnittelusta oli vinkkejä sekä esimerkkitapaus. Artikkelissa suositeltiin luomaan käyttöliittymä- ja taustajärjestelmätestit erikseen, sillä se selkeyttää testitapauksia. Lopussa oli vinkkejä joihinkin ongelmatapauksiin mitkä saattavat toistua testien luonnin yhteydessä. Kehitystiimin käyttöön jäi lisäksi tekninen ohje, sekä testien käsikirjoitukset luoduista testeistä.

9 Tutkimuskysely

Tämä luku esittelee kehitystiimille tehdyn tutkimuskyselyn ja sen tuloksia.

Automated Test Framework (ATF) käyttöönoton suunnittelun tueksi toteutettiin kehitystiimin jäsenille strukturoitu haastattelu. Haastattelussa käytettiin Microsoft O365 Forms -sovellusta, joka on tarkoitettu sähköisten kyselylomakkeiden tekoon. Kysymykset vastausvaihtoehtoineen esitettiin kehitystiimin jäsenille samassa järjestyksessä. Kyselylomake oli anonymisoitu, eikä kerännyt vastaajista minkäänlaista henkilötietoa. Kyselyyn oli mahdollista antaa yksi vastaus henkilöä kohden. Kehitystiimissä on 16 jäsentä, joista 11 vastasi kyselyyn määräaikaan mennessä. Kyselyssä kysyttiin kehitystiimin toiveita ja odotuksia testiautomaation hyödyntämisestä. Kyselyssä oli kolme kysymystä. Kyselyn kaksi ensimmäistä kysymystä olivat pakollisia. Aineiston analyysissä sovelletaan sisällönanalyysiä. Kysely on kokonaisuudessaan nähtävillä liitteessä 2.

Ensimmäinen kysymys pyrki selvittämään mikä prosessi on kehitystiimin mielestä hyödyllisin automatisoida. Vastausvaihtoehtoina olivat volyymiltaan suurin -, kriittisin -, testitapausten ylläpidon kannalta kevyin prosessi, sekä muu (vapaa tekstikenttä). Vastausten perusteella ylivoimaisin toive automatisoitavaksi prosessiksi oli volyymiltaan suurin prosessi, kahdeksalla äänellä. Muut vastausvaihtoehdot saivat kunkin yhden äänen.

Toinen kysymys käsitteli lomakkeiden automatisointia. Toimeksiantajan järjestelmässä on 131 lomaketta, joista suurin osa ovat loppukäyttäjille tarkoitetussa portaalissa. Lomakkeet niputettiin kolmen kategorian alle ja vastausvaihtoehdot olivat lomake, jota asiakkaat eniten käyttävät portaalissa, - johon tehdään usein päivityksiä, - jossa on paljon täytettäviä kenttiä, sekä muu (vapaa tekstikenttä). Vaihtoehtojen taustalla oli aiemmin työssä tehty priorisointimatriisi, sekä paras arvaus lomakkeista, joista olisi tukea kehitystiimin testaustyöhön. Yhden äänen erolla, eniten ääniä sai vaihtoehto, jossa lomake automatisointi aloitetaan lomakkeesta, jota asiakkaat eniten käyttävät portaalissa. Priorisointimatriisia tehdessä tarkasteltiin järjestelmästä lomakkeiden käyttöastetta, suurin volyymi yksittäisellä lomakkeella oli 11 536 kpl.

Kolmannella kysymyksellä haluttiin ottaa selvää kehitystiimin jäsenten odotuksia ATF-sovelluksen tuomista hyödyistä. Yhdeksän vastaajaa yhdestätoista kirjasivat odotuksiaan hyödyistä. Vaikka vastausvaihtoehtoja ei annettu, on tiimin vastaukset keskenään hyvin samanlaiset. Päällimmäiset odotukset liittyivät manuaalisen testauksen vähenemiseen, tukea versiopäivityksen testaukseen sekä testauksen nopeuttamiseen yleisesti.

Erään kehitystiimin jäsenen vastaus, liittyen odotuksiin ATF-sovelluksen tuomista hyödyistä: ”Toivon että ATF vähentäisi manuaalisen testauksen määrää ja nopeuttaisi siten esim. versiopäivitysten yhteydessä tehtävää regressiotestausta.”

Kyselylomakkeen sekä priorisointimatriisin tulosten perusteella luotiin kehitystiimin ensimmäiset ATF-testit ja -testipaketti.

10 Haastattelu

Kehitystiimistä haastateltiin kolmea eri roolissa toimivaa jäsentä, jotta kysymyksiin saadaan mahdollisimman kattava kokonaiskuva eri näkökulmista. Haastateltavien roolit kehitystiimissä ovat kehittäjä, scrum master ja testausvastaava. Haastateltavilta kysyttiin miten kauan he ovat työskennelleet ServiceNow'n parissa, joka taustoittaa heidän vastauksiaan haastattelun seuraavissa kysymyksissä. Kehittäjä on työskennellyt ServiceNow'n parissa hieman yli vuoden. Kehittäjänä hän on kuitenkin toiminut jo 20 vuotta ja kerryttänyt kokemusta myös Backend, Full stack, teknisen arkkitehdin sekä scrum master rooleista. Kehittäjällä on pitkä kokemus erilaisista testi- ja kehittämisympäristöistä. Scrum master on käyttänyt ServiceNow'ta 2,5 vuotta. Testausvastaava on työskennellyt ServiceNow'n parissa vuoden, saman ajan, kun on ollut osa kehitystiimiä.

Minkälainen kokemus sinulla on ATF:stä? Tällä kysymyksellä tuodaan esille haastateltavien kokemukset sovelluksesta. Vastaukset heijastuvat jokseenkin myös seuraaviin kysymyksiin, sillä jos vastaajalla ei ole kokemusta ATF-sovelluksesta, hänen vastauksensa perustuu todennäköisemmin mielikuviin ja toiveisiin.

Kehittäjän mielestä testien tekeminen ja niiden suorittaminen on todella hauskaa, testien koonti ja raportointi ei ole vielä täysin hallussa. Muista testaustyökaluista on enemmän kokemusta, painottuen enemmän tekniseen- kuin käyttöliittymätestaukseen. Hänen mielestään ATF on erittäin kätevä työkalu käyttöliittymätestaukseen, sillä ei tarvitse kertoa missä mikin painike sijaitsee. Kehittäjää lainatakseni ”odotukset ATF:ää kohtaan ovat varovaisen innostuneet, sovellus vaikuttaa sopivalta kompromissilta siihen, että yritys voi itse tehdä hallittuja automatisoituja testejä.”

Scrum masterilla ei ole kokemusta testaustyökalusta entuudestaan.

Testausvastaava on tutustunut ATF:ään ServiceNow-dokumentaation kautta saatavilla olevaan materiaaliin, lukenut kehittäjille tarkoitettua ServiceNow-keskustelupalstaa ja kysynyt konsulteilta kokemuksia. Hän on ohjeiden avulla harjoitellut ATF-testien tekemistä. Hän arvioi, että on tutkinut ATF:ää dokumentaation ja tekemisen kautta 3–4 viikkoa, mikä on hänen mielestään vähäinen kokemus.

Millä tavalla mielestäsi ATF testituloksia voisi hyödyntää kehitystiimissä? Kysymyksellä tavoiteltiin joko tukea omille ajatuksille, tai niistä eriäviä mielipiteitä, kun haastateltavilta kysyttiin millä tavalla sovelluksen testituloksia voisi hyödyntää kehitystiimissä.

Kehittäjä vastasi kysymykseen ”ATF-testituloksista saadaan kehitystyöhön erityistä iloa siten, että kaikki testit suoritetaan säännöllisesti ja kaikki menevät läpi onnistuneesti. Jos joku testeistä epäonnistuu, niin korjataan joko testi tai koodi, joka on rikkonut testin.”

Scrum masterin mielestä ATF tehostaisi kehitystiimin tekemistä. Hän uskoo, että huolella rakennettu ATF varmistaa paremman laadun ja tehostaa resurssien käyttöä.

Testausvastaavan mukaan ATF-testituloksia voi hyödyntää samalla tavalla kuin manuaalisen testauksen tuloksia. Hän kokee, että ATF-tuloksien etu on niiden nopeus, sillä ATF-testitulokset auttavat havaitsemaan määrityksistä poikkeavan toiminnan nopeammin testausjakson aikana. Jos esimerkiksi havainto on tehty versiopäivityksen testauksen aikana, pystytään suuntaamaan testauksen fokusta nopeammin alueille, joissa vikaantuminen on muita yleisempää. Hän pohtii, että manuaalinen testaus toisi virheen näkyville mahdollisesti vasta myöhemmin. Testausvastaava oivaltaa, että ATF:n tulokset auttaisivat kehittämään sekä ATF-testitapauksia, että kehitysprosessia kokonaisuutena, esim. havaitsemaan nopeammin puutteet määrittelyissä, tukisivat laadukkaan koodin teossa ja toimivaa uuskehitystä nopeammin tuotantoon.

Millä tavalla ATF auttaisi sinun työssäsi? Tällä kysymyksellä haettiin roolipohjaista vastausta haastateltavilta. Vastaukset perustuvat kuvitteelliseen tilanteeseen, jossa työkalu on käytössä ja testejä olisi keskeisimmistä ja haastavimmista toiminnoista, lomakkeista ja työnkuluista.

Kehittäjä ehdottaa, että käyttöliittymä- ja lomaketestaukset rajoitettaisiin niin että ne varmistavat, että ne näyttävät oikealta. Varsinaiset toiminnallisuudet taustajärjestelmässä varmistetaan erillisillä testeillä. Hän perustelee rajausta sillä, että kun tekee paikkaan X muutoksen niin ATF:ssä olisi testipaketti, joka testaa, että kaikki sen sisällä toimivat oikein ja saisi kiinni testit, joihin on heijastusvaikutuksia. Kehittäjänä hän haluaa voida luottaa siihen, että isoimmat ongelmat löytyisivät ATF:llä. Hänen mielestään se toisi kehittämiseen turvallisuutta ja varmuutta, eikä välttämättä tarvitse miettiä oman ongelman ratkaisun

ulkopuolelle mihin kaikkialle se vaikuttaa. Tällä hän tarkoittaa, että ei tarvitse lähteä etsimään riippuvuuksia, vaan ATF osaisi kertoa ne epäonnistuneiden testien muodossa.

Scrum master pohtii, että ATF auttaisi lisäämään resursseja kehittämiseen ei testaamiseen. Häntä lainatakseni ”tiimi voisi keskittyä uuden kehittämiseen ja saisi lisää testattavaa, jolloin myös tuoteomistajan suuntaan laadun näkökulmasta tulee priimalaatua.”

Testausvastaavaa ATF auttaisi keskittymään enemmän testauksen suunnitteluun ja koordinointiin. Hänen mielestään kriittiset ja monimutkaiset prosessit ovat vaikeita automatisoida, ja niihin voisi käyttää enemmän aikaa manuaalisen testauksen muodossa. Hän mainitsee, että ”aikaa menee välillä tarpeettomasti pieneen bulkkitestaukseen kuten kenttien vertailuun ympäristöjen välillä.” Hän näkee, että ATF olisi hyvä bulkkitestauksessa.

Mitkä ovat mielestäsi ATF:n hyödyt työsi kannalta? Kysymyksellä haluttiin tuoda esille roolipohjainen hyöty ja lisäksi mahdollisesti perustelut, miksi sovellus tulisi ottaa kehitystiimin käyttöön.

Kehittäjän mielestä ATF toisi hyötyä, kun testikattavuus olisi jossakin kunnossa. Se helpottaisi jatkokehitystä ja toki myös tuotantoon viennissä, kun manuaalitestauksen voisi jättää vähemmälle. Hänen mielestään testien toistuva ajo tai ajastaminen varmistaisi, että ”rikkinäiset testit jäisivät kiinni.”

Scrum master vastaa kysymykseen lyhyesti ja ytimekkäästi resurssien kohdentamisen. Laatu on hänen mielestään tärkein, että se mitä julkaistaan ja tuotetaan tilaajille, on teknisesti virheetöntä. ATF ei osaa sanoa onko se tilaajan tarpeita vastaava. ATF:llä pystytään tarkistamaan, onko tekninen toteutus oikein, myös laadullisesta näkökulmasta katsottuna.

Testausvastaava toivoo, että ATF tehostaisi ja nopeuttaisi testausta. Vapauttamalla resursseja kriittisten prosessien manuaalitestaukseen, pystyisi keskittymään paremmin niiden testaukseen. Hän lisää, että ATF tuo tavallaan ylimääräisen testausresurssin, kun automatiikka hoitaa osan perustestauksesta.

Mitkä ovat mielestäsi ATF:n haitat työsi kannalta? Kysymyksellä pyritään tuomaan esille roolikohtaiset mielipiteet sovelluksen mahdollisista haitoista. Tällä haettiin myös perusteluita, miksi sovellusta ei tulisi hyödyntää kehitystiimin arjessa.

Kehittäjän mielestä sovelluksen haitat ovat aika vähäiset. Haitta on hänen mielestään ehkä enemmän byrokraattinen, testien luontia voi ensin joutua perustelemaan, koska niiden tekeminen on alkuun vaivalloista. Hän kertoo, että testien tulisi olla yhtä laadukkaita kuin itse koodi ja / tai muutos. Testien tekeminen voi viedä aikaa ja vaivaa, sekä voi jäädä helposti myös tekemättä. Hän toteaa, että jos testejä ei suoriteta eikä päivitetä, niiden korjaaminen tuottaisi enemmän harmia kuin iloa. Hän näkee testien päivittämisen osana muutosta, joka järjestelmään tehdään. Hän ei pidä testien päivittämistä haittana kehittäjän työn kannalta ja tunnustaa että se tietysti lisää itse työmäärää, mutta maksaa itsensä takaisin kyllä.

Scrum master pohtii, ettei sovelluksen turvin voi unohtaa manuaalista testausta kokonaan, jos testit olisivatkin huonosti toteutettu. Hän ei pidä ATF:ää suoranaisena haittana, mutta toteaa, että testejä pitää aika ajoin tarkastella, ettei se käänny itseään vastaan, jos niitä ei ole tehty todella hyvin. Hänen mielestään ATF-testien tekeminen lomakkeelle, jota käytetään kaksi kertaa vuodessa, varmistaisi että se todella toimii niinä harvoina kertoina, mutta ei toisi todellista hyötyä tiimille. Harjoitteleva ATF-testin tekijä voisi edellä mainitut testit tehdä, mutta niiden todellinen hyöty jäisi ohueksi. Hän korostaa, että on äärimmäisen tärkeää tietää mihin ATF:ää käytetään ja milloin.

Testausvastaava toteaa huumorin sävytteisesti sovelluksen haittapuolena sen, ettei kaikkea pysty automatisoimaan. Hän miettii, että sovelluksen puutteena on, ettei tietynlaisia E2E työnkulkuja pysty testaamaan, sellaiset mitkä lähtevät toimittajan tai toisen sidosryhmän järjestelmästä. Hän kertoo, että integraatioyhteyksiä voidaan ServiceNow-dokumentaation perusteella testata, mutta niitä ei ole tutkittu vielä sen enempää. Kaikkia työkalun tuomia mahdollisuuksia ei vielä tiedetä. Hän ei näe suoranaisia haittapuolia ATF:ssä. Hän tietää, että sovelluksen käyttö ja ylläpito tulee viemään resursseja, mitkä pitää huomioida työn suunnittelussa. Resursointiin liittyvä ongelma ei liity suoraan ATF:ään, koska automaation käyttöönotossa tarvitaan myös resursseja ylläpitotehtäviin.

Tuoko ATF enemmän hyötyä vai haittaa kiireiseen arkeen? Roolipohjainen kysymys, jolla haettiin haastateltavien mutututuntumalla suuntaa sovelluksen käyttöönotolle.

Kehittäjä vastasi kysymykseen, että ATF tuo ehdottomasti hyötyä arkeen.

Scrum master toivoisi, että ATF tarjoillaan valmiina ja toimivana kokonaisuutena. Hänen mielestään kysymykseen on vaikea vastata, jos nyt pitäisi rakentaa ATF niin että siitä on heti tiimille arvoa, niin lyhytnäköisesti ajateltuna siitä on haittaa. Alkuun tulee menemään paljon resursseja, kun testit tehdään kunnolla. Hän arvelee, että ATF tuottaa tiimille puolen vuoden tai vuoden sisällä hyötyjä. Hän uskoo, että sovelluksesta on enemmän hyötyä kuin haittaa, kunhan ATF on käytössä.

Testausvastaava miettii, että sovelluksesta on hänelle varmasti enemmän hyötyä kuin haittaa. Häntä lainatakseni ”kunhan ATF:n ylläpitoon on aikaa ja resursseja, niin onhan se hyvä.” Kaikki automaatio on hyvästä. Ylläpitoa tekevä, esim. kehittäjä voisi olla tietysti erimieltä, että onko ATF haitta ja hidaste omalle työlle, mutta testausvastaavan näkökulmasta katsottuna ehdottomasti hyöty.

Miten näet, että ATF:ää voidaan hyödyntää jatkuvassa kehittämisessä? Kysymyksellä haluttiin selvittää haastateltavien näkemys työkalun hyödyntämistä jatkuvassa kehittämisessä. Kysymys liittyy osin myös opinnäytetyön tutkimuskysymykseen, miten automatisoitujen testien tuloksia voidaan hyödyntää kehitystyössä.

Kehittäjän mielestä ATF:ää voidaan hyödyntää parillakin tavalla. Ensimmäiseksi hän mainitsee, että omalle kehitystyölle saa tukiverkon, varmistaa ettei rikkonut muutoksellaan mitään järjestelmässä. Toisena hän mainitsee, että voi tehdä testejä uutta toiminnallisuutta vasten, jolloin automaatiotestausta voi käyttää varmistamaan, että teki koodin oikein. Testiä luotaessa miettii jo mitä muutoksella yrittää tehdä ja mitä sen pitäisi tehdä. Testiä suorittaessa, huomaa helposti siihen liittyvän ajatusvirheen tai muun loogisen asian, jota ei tullut aiemmin ajatelleeksi tai ei osannut huomioida sen vaikutuksia.

Scrum master toivoo, että sovelluksella tehtäisiin testipaketit virheherkkiin ominaisuuksiin, toiminnallisuuksiin ja lomakkeisiin, jotta voidaan varmistaa, että kaikki on kunnossa ennen muutoksen tuotantoon vientiä. Hän arvioi, että monimutkaisiin prosesseihin ja

integraatioihin, joihin aika ajoin tehdään muutoksia, olisi ATF:stä erittäin paljon hyötyä jatkuvassa kehittämisessä. ATF:llä voisi varmistaa järjestelmän laadun, kun suurin osa testauksesta suoritettaisiin automaation avustuksella.

Testausvastaava näkee, että ATF:ää voidaan hyödyntää kokonaisuuksissa, jossa tehdään muutoksia. Sen avulla voidaan tunnistaa riippuvuus toiseen järjestelmään tai prosessiin, johon muutoksen ei pitäisi vaikuttaa. Automatiikalla voidaan varmistaa, että toiminnallisuus on säilynyt muuttumattomana. Tällä hetkellä sitä ei pystytä laajassa mittakaavassa tekemään tai se on hidasta ja manuaalista. Jatkuvassa kehityksessä uudet osa-alueet testataan manuaalisesti hyvin, mutta sen rinnalle tarvitaan ATF varmistamaan, ettei mikään muu mene muutoksen myötä rikki. Kaikkia mahdollisia riippuvuuksia ei manuaalisessa testauksessa tule aina huomioitua. Hän arvioi, että päivittäisessä kehittämisessä ATF mahdollistaisi nopeamman kehityksen, koska se toisi automatisoidun regressiotestauksen kautta heti esille muutoksen mahdolliset negatiiviset vaikutukset. Esim. kehittäjä tekee muutoksen koodiin ja voi välittömästi ajaa regressiotestit alueilla, joissa ei olisi pitänyt tapahtua muutosta siinä yhteydessä. Mahdollisissa virhetilanteissa voitaisiin heti analysoida, onko tarve muuttaa koodia, vai testitapausta. Sykli koodin muutoksesta valmiiksi toimivaksi tuotteeksi on automaation avulla nopeampaa verrattuna manuaaliseen testaukseen.

Millä aikataululla olisit valmis käyttämään ATF:ää päivittäisessä työssä? Tällä kysymyksellä tarkasteltiin haastateltavien valmiutta ottaa sovellus osaksi päivittäistä työtä.

Kehittäjä olisi valmis käyttämään ATF:ää per heti. Häntä lainatakseni ”alkuun se olisi tuskaa, kun testejä ei ole vielä, mutta kun ne ovat tehty niin helpottaa.” Hänen mielestään testejä ei kannata tehdä ”älä korjaa, jos se toimii” -periaatteella, vaan sen mukaan, kun tekee järjestelmään uutta kehitystä tai muutosta. Hän pohtii, että tällä toimintatavalla voisi vuoden päästä tarkastella testikattavuutta ja päättää halutaanko se nostaa.

Scrum master tavoittelisi sovelluksen käyttöä päivittäisessä työssä vuoden päästä. Hän näkisi, että silloin siitä saisi hyötyä sprinteittäin ja se olisi oikeasti osa päivittäistä työtä. Testien rakentaminen vaatii kuitenkin niin paljon alkuun.

Testausvastaava on heti ja hyvin matalalla kynnyksellä valmis ottamaan ATF:n osaksi päivittäistä työtä. Hän toteaa, että kun testien tekemiseen on päässyt sisälle, niitä on helppo

ja nopea tehdä lisää. Lainatakseni testausvastaavaa, ”testipaketin tarvitsee olla alkuun pieni, että saadaan kokemuksia, millaista niitä on päivittäin käyttää ja mikä prosessi sen ylläpitoon voisi olla.” Hän tuumaa, että ATF:n päivittäinen käyttö kehitystiimissä voisi hyvin olla ensimmäisen neljänneksen aikana ensi vuonna, jos kokonaisuus on pieni.

Miten tarpeelliseksi koet ATF:stä koulutusmateriaalin tiimin käytettäväksi ja missä muodossa? Tarkoitus oli selvittää kysymyksellä millainen ja miten tarpeellinen koulutusmateriaali olisi, sekä mahdollisuudet sen valmisteluun.

Kehittäjän mielestä yleinen, enemmän konseptitason malli miten testit nimetään ja millä tasolla ne tehdään, riittäisi. Ehkä ajatelmaa, että millainen on hyvä testi. Hän ehdottaa, että voisi olla erikseen ohjeet käyttöliittymä- ja taustajärjestelmän testausta varten.

Scrum master kokee tarpeellisimmaksi havainnot, miten testit kannattaa tehdä, käytännön tekemiseen vinkkejä, mitä tulee ottaa huomioon ja parhaat käytännöt. Ohjeet testien nimeämisestä ja testipakettien rakentamisesta olisivat myös hyödyllisiä. Hänen mielestään teknisiä ohjeita ei tarvita.

Testausvastaava haluaa, että prosessi ja ohjeet millä tavalla testaustyökalua käytetään tiimissä ovat selvillä, kun ATF otetaan käyttöön ja jalkautetaan päivittäiseen tekemiseen. Koskee niin uusia testejä kuin vanhojen päivittämistä. Hänelle riittää tietämysartikkeli käytänteistä. Ohjeiden tai materiaalin ylläpidettävyyden kannalta tietämysartikkeli on parempi kuin koulutustallenne. Artikkelin laajuus on toinen asia, riittääkö yksi artikkeli vai tulisiko olla artikkelikokonaisuus. Tärkeintä kuitenkin olisi, että ohje on helposti päivitettävä ja ylläpidettävä.

Kenen mielestäsi tulisi vastata ATF:n käytöstä? Tämä kysymys liittyy omalta osaltaan käyttöönottosuunnitelmaan ja resursointiin.

Kehittäjä ehdottaa, että muutama henkilö voisi olla vastuussa siitä, miten testit organisoidaan, miten ne ajetaan, miten testit nimetään ja mihin ne lisätään. Heillä olisi vastuu myös siitä, että testit ajetaan joka yö ja testitulosten tarkistaminen. Hänen mielestään yksittäisten testien tekeminen, lähinnä yksikkötestiä vastaavia (suljetut testit, ei

lähetetä ulkopuolelle mitään), olisivat kehittäjien itsensä vastuulla ja vaikeampiin tapauksiin kysytään tiimiltä apua.

Scrum master toteaa, että kehitystiimissä tulee aina olla testausvastaava. Rooli vastaisi ATF-kokonaisuuden edistämisestä ja suunnittelee, minkälaisia testejä automatisoidaan. Kehitystiimille jalkautetaan tekeminen ja vastuu testien lisäämisestä ja päivittämisestä oman kehittämisen ohella. Testausvastaava seuraa testituloksia ja koordinoi tekemistä testien päivittämisestä. Vastuuta ATF:stä ei kannata jättää yhden henkilön varaan.

Testausvastaava vastaa kysymykseen kertomalla testausvastaavan omistavan prosessin. Se miten testitapauksia tehdään, miten niitä päivitetään ja miten prosessi toimii ATF:n ympärillä, mutta siihen osallistuu muitakin tiimin jäseniä. Prosessin pitäisi vastata siihen keneltä varmistetaan voiko testin päivittää. Epäonnistunut testi tulee tarkistaa tarkistuslistan tai prosessin mukaisesti. Testausvastaavan vastuulla voisi olla testitulosten seuranta, jos testit ovat ajastettuja. Testausvastaavan lisäksi voisi olla nimetty henkilö tuuraamassa tarvittaessa.

11 Johtopäätökset ja pohdinta

Opinnäytetyön teoriaosuudessa käsiteltiin kattavasti automaation näkökulmasta sekä ohjelmistotuotantoa että -testausta. Yleisellä tasolla esiteltiin automaatiotestausta, sen hyötyjä ja haittoja, sekä käytiin läpi mitkä asiat vaikuttavat automatisoitavien testitapausten valintaan. ServiceNow käsiteltiin lyhyesti ja työn painopiste oli sen testaustyökalussa ATF:ssä. Kuten työn teoriaosuudesta käy ilmi, testauksen tarkoitus on antaa tietoa järjestelmän laadusta, varmistaa sen toimivuus ja sitä tulee jatkuvasti kehittää muuttuvan järjestelmän rinnalla. Testaustavasta riippumatta, sillä on suora vaikutus järjestelmän kustannuksiin. Automaatiotestauksella tavoitellaan pidemmän aikavälin hyötyä ja nopeutta testaukseen. Automatisointi säästää aikaa, optimoi toteutusta, lisää testikattavuutta ja edistää järjestelmän laatua. Automatisoinnin suurin hyöty on kustannustehokkuus sekä järjestelmän toimivuuden varmistaminen. Automaation turvin manuaalista testausta ei voi sivuuttaa. Manuaalisessa testauksessa määrittelyä verrataan toteutukseen sekä varmistetaan että se vastaa tilaajan tarpeita, eikä näitä voida automatisoida.

Opinnäytetyön tarkoituksena oli selvittää, voidaanko ATF:n avulla ja vähemmällä työllä varmistaa ei-tuotantoympäristöjen toimivuus muutosten jälkeen, verrattuna manuaaliseen työhön. Teorian, haastattelujen ja käytännön myötä on saatu selville, että ATF:n avulla voidaan varmistaa eri ympäristöjen toimivuus muutosten jälkeen. ATF-sovelluksen käyttöönotto vaatii työtä testien luonnissa. Sovelluksen käyttö päivittäisessä työssä vaatii testien suorittamista, testitulosten seurantaa ja testien ylläpitoa kehittämisen yhteydessä. Järjestelmän toimivuus varmistetaan tärkeimpien ja kriittisten toimintojen sekä ominaisuuksien automatisoiduilla regressiotesteillä, tehdyn muutoksen kattavilla testeillä, sekä testien säännöllisellä ylläpidolla. Loppukäyttäjien luottamus järjestelmän toimivuuteen kasvaa samalla kun kehitystyön laatu paranee kattavammalla testauksella. Olen vakuuttunut ATF-sovelluksella luoduista testeistä ja niiden hyödystä kehitystiimille osana jatkuvaa kehitystä.

Opinnäytetyössä käytiin läpi automatisoinnin perusteet, käytänteitä ja esiteltiin ServiceNow’n ATF-sovellus tarvittavin osin, jotta lukija pystyy ottamaan sovelluksen käyttöön omassa ServiceNow ympäristössään ja luomaan automatisoituja testejä. Tämän opinnäytetyön lopputuloksena oli peruspalvelupyyntöprosessin automatisoidut testit.

Selvitystyön perusteella ei pysty aukottomasti sanomaan, voiko sovelluksen ja vähemmän manuaalisen työn yhdistelmällä varmistaa ympäristöjen toimivuutta, sillä sovelluksen käyttö itsessään vaatii manuaalista työtä. Sen sijaan voidaan verrata testaamiseen kuluvaa aikaa. Manuaaliseen testaamiseen verrattuna, sovelluksella suoritetut testit vievät huomattavasti vähemmän aikaa. Prosessin manuaaliseen testaamiseen kuluu alle 3 minuuttia aikaa. Testipaketin suorittaminen, jossa 12 testiä, kestää sovelluksella 3 minuuttia. Testipaketin viimeinen testitapaus, jota voidaan verrata manuaaliseen testaamiseen, vie sovellukselta aikaa 37 sekuntia. Samassa ajassa, kun ihminen suorittaa testin kerran, ehtii ATF-sovellus suorittaa testin 8 kertaa. On todettava, että monimutkaisen prosessin testin luonti ATF:llä ei ole nopea toteuttaa. Testivaiheiden toimivuus vahvistetaan määrittelyä vastaan, jotta tiedetään testipaketin toimivan oikein. Testien luontiin käytetty aika korreloi testipaketin suoritusaikaan ja siitä saatuun ajalliseen hyötyyn. Sovelluksen avulla voidaan todistetusti vähentää manuaaliseen testaukseen kuluvaa aikaa. Sovellukseen tutustumisen ja kokeilukappaleiden jälkeen, peruspalvelupyyntöprosessin testien tekemiseen arvioin käyttäneeni aikaa tunnin. Testien tekeminen nopeutuu kokemuksen kartuttua erityyppisistä testivaiheista ja kertaalleen tehtyjä vaiheita kannattaa uudelleenkäyttää.

Testien automatisoinnin aloittaminen sovelluksella on helppoa, sillä testivaiheet ja niiden käyttötarkoitukset ovat selkeästi kuvattu. Lisäksi on saatavilla tietoa, jos vaihe edellyttää toisen vaiheen toimiakseen, tarvitsee tietyn käyttöoikeuden tai suosituksen käyttää yhdessä toisen vaiheen kanssa, kuten Record Update -testivaihe. Testejä oli alun perin tarkoitus toteuttaa eri lailla ja vertailla mitkä testit suoritetaan nopeammin tai ovat tarkempia. Kävi kuitenkin nopeasti ilmi, että niitä ei voi suurella variaatiolla tehdä. Rajaus on melkeinpä käyttöliittymän ja taustajärjestelmän välinen. Tarkoittaen, että käyttämällä ainoastaan sovelluksen valmiita testivaiheita, ei käyttöliittymätestejä voi tehdä kuin yhdellä tapaa. Sama koskee taustajärjestelmän testejä. Tämä voi tuntua hyvin rajoitetulta henkilölle, joka on tehnyt aiemmin muilla tavoin automatisoituja testejä. Henkilö, jolle automatisointi ei ole tuttua, pääse helposti ja nopeasti sovelluksen avulla kiinni automatisoituun testaukseen.

ATF-sovelluksen ehdottomiin hyötyihin luen tietojen uudelleen hyödyntämisen ja uudelleenkäyttämisen. Testivaiheissa pystyy hyödyntämään aiemman vaiheen tietoja ja näin käyttämään sitä uudelleen. Testejä voi kopioida ja luoda eri versioita niistä. Esimerkkinä peruspalvelupyyntöprosessin testipaketti, jossa vaaditaan asiakkaalta ja käsittelijältä tietyt

roolit. Testipaketin voi kopioida, vaihtaa asiakkaan tai käsittelijän rooli ja tehdä negatiivinen testaus, eli varmistaa että asiakas ei näe jotakin lomaketta, tai ettei käsittelijä pysty avaamaan palvelupyyntöä roolilla, jolla puutteelliset oikeudet. Selvitin myös mahdollisuutta hyödyntää ServiceNow'n tarjoamia valmiita testejä, jotka löytyvät Quick Start Suites -moduulista. Testit ovat OOB-ominaisuuksia, -toiminnallisuuksia tai -prosesseja varten, eikä niistä pystynyt suoraan hyödyntämään kuin muutaman testivaiheen. Toimeksiantajan tapauksessa, muutamat muutokset painikkeiden nimiin ja toiminnallisuuksiin olivat jo esteenä valmiiden testien hyödyntämiselle. Niiden käyttäminen vaatii pieniä muutoksia testivaiheisiin ja samalla vaivalla tekee itse soveltuvimmat testit omaan ympäristöön.

Yhtenä jatkokehityskohteena näen selvityksen sovelluksen Headless browser -ominaisuuden käyttämisestä testien suorittamisen aikana. Headless browser -automaatiotestaus oli yksi uusista ominaisuuksista ServiceNow'n viimeisimmässä versiopäivityksessä. Headless browser mahdollistaa sovelluksessa käyttöliittymätestauksen ilman manuaalista Client Test Runner:ia (testin suorittaja). Tämä myös siitä syystä, että testipaketteja ei toistaiseksi ajasteta, ne suoritetaan manuaalisesti aina kun testeihin liittyvää kehitystä tehdään. Ajastuksesta ei tällä hetkellä ole hyötyä kehitystiimille, sillä järjestelmä vaatii aktiivisen tietokoneen ja verkkoyhteyden, esim. hyppykoneen, jolla testit suoritettaisiin. Lisää tietoa ja ohjeet miten headless browser otetaan käyttöön, löytyy liitteestä 5.

Perustelut ATF-sovelluksen käyttöönotolle ovat olemassa ja tämä kävi ilmi myös haastattelujen kautta. Jatkotoimenpide kehitystiimille on testiautomaation laajempi käyttö, joka todennäköisesti suunnitellaan edistettäväksi pienemmissä osissa. Tätä varten tarvitaan laadukkaita testitapauksia, joissa on huomioitu liiketoiminnan kannalta kriittisimmät ominaisuudet, toiminnot, ja prosessit. Uusien testien luominen ja vanhojen päivittäminen sidotaan osaksi kehitysprosessia. Testiautomaatio voidaan nähdä jatkuvasti kehittyvänä kokonaisuutena, kuten kehitettävä järjestelmäkin, jota parannetaan yhä kattavammaksi virheiden esiintyessä. Testiautomaation käyttöönoton jälkeen on tärkeää luoda yhteiset toimintatavat ATF:n käytölle sekä vahvistaa testien ylläpidettävyyttä. Hyödyntämällä lisäksi parhaita käytäntöjä on mahdollista saada ATF:llä automatisoiduista testeistä suurimmat hyödyt irti. Ratkaiseva tekijä onnistuneelle automatisoinnin käyttöönotolle on motivoitunut kehitystiimi, joka tekee järjestelmälle parhaiten sopivilla käytänteillä ylläpidettävät automaatiotestit.

Toimeksiantajan palaute työstä oli erittäin positiivinen. Heidän mielestänsä sain nopeasti haltuun ServiceNow’n ATF-sovelluksen ja rakensin tiimille alustan versiopäivityksissä ja yhden prosessin päivityksissä hyödynnettävän ATF-testipaketin. He ovat pystyneet tehostamaan testausta huomattavasti tämän ansiosta. Toimeksiantaja on erittäin tyytyväinen lopputulokseen. Innostukseni aiheeseen välittyi tekemisen kautta ja he huomasivat, että paneuduin siihen huolellisesti. Toimeksiantaja toivoo pystyvänsä jatkamaan aloittamaani ATF-työtä yhtä laadukkaasti.

12 Yhteenveto

Tutkimuskysymyksiin vastaaminen onnistui hyvin. Tutkimuskysymykseni olivat:

- Minkälaisia testejä kannattaa automatisoida?
- Miten automatisoitujen testien tuloksia voidaan hyödyntää kehitystyössä?
- Mitkä ovat ATF-sovelluksen hyödyt ja haitat?

Sain selvitettyä teorian ja käytännön työn kautta minkälaisia testejä kannattaa automatisoida. Käytännön työnä syntyi priorisointimatriisi, josta oli helppo päätellä mistä testien automatisointi kannattaa aloittaa. Kehitystiimiltä kysyin mielipiteitä testien automatisointiin ja kyselyn tulokset ovat esitelty luvussa 9. Puolistrukturoidun haastattelun kautta tuotiin ilmi kolmen kehitystiimin jäsenten ajatuksia ja mielipiteitä, siitä miten ATF:llä toteutettujen automatisoitujen testien tuloksia voidaan hyödyntää. Kysymystä pohdittiin niin jatkuvan kehittämisen osalta kuin myös osana kehitystiimin arkea, jossa jo roolitettiin ATF:n käyttöä. Yhdistämällä teorian ja haastatteluissa annetut vastaukset selvisivät parhaat käytännöt ATF-testien automatisointiin, mistä aloittaa, sekä sovelluksen hyödyt ja haitat.

Opin ymmärtämään syvemmällä tasolla myös teorian kautta mitkä asiat vaikuttavat automaation käyttöönottoon. Huomasin käytännön osuutta tehdessäni, että olisi ollut hyödyllistä, jos olisin ehtinyt tutustumaan paremmin ATF-sovellukseen ennen sen käyttöönottoa ja testien tekemiseen. Ajankäytöllisesti siihen ei ollut kuitenkaan mahdollisuutta. Testien tekeminen ATF-sovelluksella poikkesi hyvin paljon tavasta, jolla olin aiemmin tehnyt automaatiotestejä. Opinnäytetyön kautta sain mahdollisuuden kasvattaa omaa osaamistani automaatiotestauksen parissa sekä voin hyödyntää oppimaani myös työelämässä.

Kehitystiimillä on selkeä tarve ja halu automatisoida testausta laadun varmistamiseksi jatkuvassa kehityksessä sekä versiopäivityksissä manuaalisen testauksen tukena. Työn myötä saatiin luotua alku testien automatisointiin ATF-sovelluksella. Sovelluksen laajempi käyttöönotto tulee vaatimaan aikaa ja resursseja kehitystiimiltä, mutta on varmasti kullan arvoinen tulevaisuudessa, kunhan aika on sille oikea.

Lähteet

AW Academy. (n.d.). *Mikä on suosittu ServiceNow ja millaisen urapolun se tarjoaa IT-alalle?* AW Academy. https://academy.fi/stories-insights/alanvaihto/ura-it-alalla-servicenow

Bose, S. (2020). *Test Planning: A Detailed Guide*. BrowserStack. https://www.browserstack.com/guide/test-planning

Fitzgibbons, L. (2020). *What is ServiceNow and What Does It Do?* TechTarget. https://searchitoperations.techtarget.com/definition/ServiceNow

Garousi, V., & Mäntylä, M. v. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, *76*, 92–117. https://doi.org/10.1016/J.INFSOF.2016.04.015

Hamilton, T. (2021a). *Automation Testing Vs. Manual Testing: What's the Difference?* Guru99. https://www.guru99.com/difference-automated-vs-manual-testing.html

Hamilton, T. (2021b). *Integration Testing: What is, Types, Top Down & Bottom Up Example*. Guru99. https://www.guru99.com/integration-testing.html

Hamilton, T. (2021c). *Test Data Generation: What is, How to, Example, Tools*. Guru99. https://www.guru99.com/software-testing-test-data.html

Hamilton, T. (2021d). *What is Regression Testing? Definition, Test Cases (Example)*. Guru99. https://www.guru99.com/regression-testing.html

Hamilton, T. (2021e). *What is Software Testing? Definition, Basics & Types in Software Engineering*. Guru99. https://www.guru99.com/software-testing-introduction-importance.html

Hietaniemi, J. (2019). *Huono tuoteomistaja on organisaation epätuottavin henkilö*. Gofore. https://gofore.com/huono-tuoteomistaja-on-organisaation-epatuottavin-henkilo/

Hoffman, D. (2019). *The Often Overlooked Test Oracle - The Key to More Powerful Testing*. Association for Software Testing. https://www.associationforsoftwaretesting.org/the-often-overlooked-test-oracle/

Holopainen, J. (2005). *REGRESSIOTESTAUS JA TESTIEN VALINTATEKNIIKAT Pro gradu-tutkielma Tietojenkäsittelytiede Kuopion yliopiston tietojenkäsittelytieteen laitos Huhtikuu 2005*.

Interaction Design Foundation. (n.d.). *What is Ease of Use?* Interaction Design Foundation. https://www.interaction-design.org/literature/topics/ease-of-use

ISTQB. (n.d.). *ISTQB Glossary & Testing Terms explained: Test oracle*. http://istqb-glossary-explanations.blogspot.com/2016/07/test-oracle.html

Jaiswal, S. (n.d.). *Test Plan*. Javatpoint. https://www.javatpoint.com/test-plan

Khimani, B. (2020). *Software Testing Life Cycle*. Infyom. https://infyom.com/blog/software-testing-life-cycle

Kinsbruner, E. (2019a). *Manual Testing vs. Automation Testing*. Perfecto. https://www.perfecto.io/blog/automated-testing-vs-manual-testing-vs-continuous-testing

Kinsbruner, E. (2019b). *What Is Regression Testing?* Perfecto. https://www.perfecto.io/blog/what-is-regression-testing

Mili, A., & Tchier, F. (2015). *Software Testing: Concepts and Operations*. Wiley.

PAT Research. (n.d.). *All about Software Testing: Test Types, Test Cases and Benefits*. PAT Research. https://www.predictiveanalyticstoday.com/all-about-software-testing-test-types-test-cases-and-benefits/

Plat4mation. (n.d.). *All you need to know about ServiceNow releases*. Plat4mation. https://www.plat4mation.com/servicenow/all-you-need-to-know-about-servicenow-releases/

ProfessionalQA. (2020). *What is meant by Stubs and Drivers?* ProfessionalQA.Com. https://www.professionalqa.com/stubs-and-drivers

Pyykkö, T. (2010). *Ohjelmistotestaus siirryttäessä perinteisistä ohjelmistokehitysmenetelmistä Scrumiin*. https://jyx.jyu.fi/bitstream/handle/123456789/22746/1/URN_NBN_fi_jyu-201001181047.pdf

Relevance Lab. (2020). *Intelligent Automation using ServiceNow ATF*. Relevance Lab. https://relevancelab.com/2020/08/12/intelligent-automation-using-servicenow-atf/

Rice, R. W. (n.d.). *What is a Test Plan? Complete Guide With Examples*. PractiTest. https://www.practitest.com/qa-learningcenter/best-practices/write-a-test-plan/#what-is-a-test-plan

Roy, S. C. (2021). *How to Select Correct Test Cases for Automation Testing (and Ultimately Achieve a Positive Automation ROI)*. Software Testing Help. https://www.softwaretestinghelp.com/manual-to-automation-testing-process-challenges/

ServiceNow. (n.d.). *Now Platform: The platform of platforms for 21st century enterprises*. ServiceNow. https://www.servicenow.com/now-platform.html

ServiceNow. (2021a). *Automated Test Framework (ATF) | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/automated-test-framework.html

ServiceNow. (2021b). *Automated Test Framework Best Practices*. https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/success/quick-answer/automated-test-framework-best-practices.pdf

ServiceNow. (2021c). *Automated Test Framework reference | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/atf-ref-overview.html

ServiceNow. (2021d). *Automated Test Framework Use - Customer Success*. ServiceNow. https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/success/quick-answer/automated-test-framework-use.pdf

ServiceNow. (2021e). *IT Service Management | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-it-service-management/page/product/it-service-management/reference/r_ITServiceManagement.html

ServiceNow. (2021f). *Parameterized tests | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/parameterized-tests.html

ServiceNow. (2021g). *Quick start tests | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/quick-start-tests.html

ServiceNow. (2021h). *Roles | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-platform-administration/page/administer/roles/concept/c_Roles.html

ServiceNow. (2021i). *Schedule an automated test suite | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/task/atf-sched-suite-steps.html

ServiceNow. (2021j). *View test results | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/task/atf-view-results-consolidated.html

ServiceNow. (2021k). *Working with scheduled test suites | ServiceNow Docs*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/atf-sched-suites.html#atf-sched-suites

ServiceNow. (2022). *Working with client test runners*. ServiceNow. https://docs.servicenow.com/bundle/rome-application-development/page/administer/auto-test-framework/concept/atf-test-runners.html

Software Testing Help. (2021a). *Grey Box Testing Tutorial With Examples, Tools And Techniques*. Software Testing Help. https://www.softwaretestinghelp.com/grey-box-testing-tutorial/

Software Testing Help. (2021b). *What is Automation Testing (Ultimate Guide to Start Test Automation)*. Software Testing Help. https://www.softwaretestinghelp.com/automation-testing-tutorial-1/

Streyda. (2021). *ServiceNow Automated Test Framework (ATF)*. Streyda. https://www.streyda.eu/post/servicenow-automated-test-framework-atf

Tauriainen, J. (2011). *Auvoisa automaatio*. Ohjelmistotestaus.Fi. https://ohjelmistotestaus.fi/2011/12/02/auvoisa-automaatio/

Taylor, D. (2021). *ServiceNow Tool Tutorial: What is, Use & Reporting Training*. Guru99. https://www.guru99.com/servicenow-tutorial.html

Thelin, R. (2020). *Software Testing 101: Get started with software testing types*. Educative.Blog. https://www.educative.io/blog/software-testing-types-101

Tilton, B. (2021). *What's new with ATF in Rome*. ServiceNow. https://developer.servicenow.com/blog.do?p=/post/rome-atf/

Upadhyay, R. K. (2021). *Test Oracles*. GeeksforGeeks. https://www.geeksforgeeks.org/test-oracles/

Liite 1: Aineistonhallintasuunnitelma

Aineistojen yleinen kuvaus

Aineistoa kerättiin sähköisellä kyselylomakkeella sekä puolistrukturoidulla haastattelulla. Sähköinen kyselylomake lähetettiin koko kehitystiimille, kun puolistrukturoidussa haastattelussa haastateltiin kolmea kehitystiimin jäsentä. Kyselylomake toteutettiin Microsoft O365 Forms-sovelluksella, joka on tarkoitettu sähköisten kyselylomakkeiden tekoon. Kyselylomakkeeseen vastattiin anonyymisti henkilötietoja keräämättä, joten eettistä ennakkoarviointia ei tarvinnut tehdä. Haastattelut toteutettiin Microsoft O365 Teams-sovelluksen kautta. Haastattelussa tuodaan esille vastaajien roolit kehitystiimissä henkilötietoja keräämättä. ATF-sovelluksella ajetut testit järjestelmä poistaa automaattisesti, testitulokset ja testipaketit jäävät järjestelmään kehitystiimin hyödynnettäviksi.

Opinnäytetyössä esiintyvät ruutukaappaukset ovat käytössä olevasta järjestelmästä ja siihen luodusta aineistoista. Ruutukaappausten käyttölupa on varmistettu toimeksiantajalta.

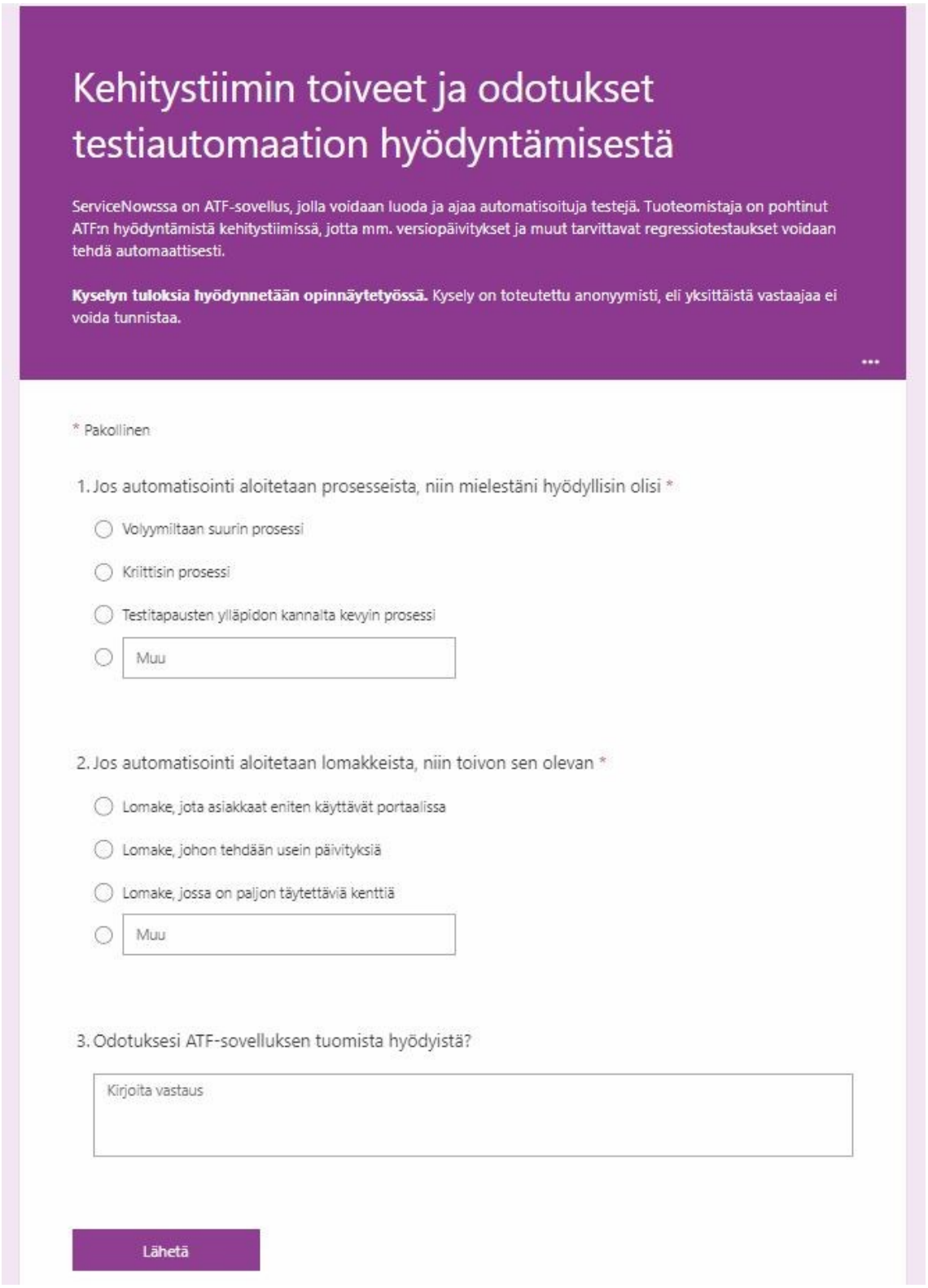
Aineiston käsittely

Kyselylomakkeen vastauksista muodostui kaavioita, jotka esittävät tiedot tiivistettynä numeraalisesti. Saatu aineisto tallentui automaattisesti Microsoft O365 Forms-sovellukselle, joka on ainoastaan opinnäytetyön tekijän käytettävissä. Haastattelut tallennettiin Microsoft O365 Teams-sovelluksella, jotka ovat opinnäytetyön tekijän sekä haastateltavan käytettävissä.

Aineiston säilytys ja tuhoaminen

Kyselylomakkeesta saatu aineisto säilytetään vuoden ajan Microsoft O365 Forms-sovelluksella. Haastattelujen kautta saatu aineisto säilytetään toimeksiantajan organisaation päättämän ajan. Viimeistään vuoden kuluttua opinnäytetyön valmistumisesta kaikki aineistot hävitetään poistamalla ne pysyvästi. Kyselylomake ja haastattelut vastauksineen ovat opinnäytetyön liitteenä.

Liite 2: Tutkimuskysely



Kehitystiimin toiveet ja odotukset testiautomaation hyödyntämisestä

ServiceNow:ssa on ATF-sovellus, jolla voidaan luoda ja ajaa automatisoituja testejä. Tuoteomistaja on pohtinut ATF:n hyödyntämistä kehitystiimissä, jotta mm. versiopäivitykset ja muut tarvittavat regressiotestaukset voidaan tehdä automaattisesti.

Kyselyn tuloksia hyödynnetään opinnäytetyössä. Kysely on toteutettu anonyymisti, eli yksittäistä vastaajaa ei voida tunnistaa.

* Pakollinen

1. Jos automatisointi aloitetaan prosesseista, niin mielestäni hyödyllisin olisi *

- ◯ Volyymiltaan suurin prosessi
- ◯ Kriittisin prosessi
- ◯ Testitapausten ylläpidon kannalta kevyin prosessi
- ◯ Muu

2. Jos automatisointi aloitetaan lomakkeista, niin toivon sen olevan *

- ◯ Lomake, jota asiakkaat eniten käyttävät portaalissa
- ◯ Lomake, johon tehdään usein päivityksiä
- ◯ Lomake, jossa on paljon täytettäviä kenttiä
- ◯ Muu

3. Odotuksesi ATF-sovelluksen tuomista hyödyistä?

Kirjoita vastaus

Lähetä

Kysymykset | Vastaukset 11

Kehitystiimin toiveet ja odotukset testiautomaation hyödyntämisestä

11	05:16	Aktiivinen
Vastaukset	Keskimääräinen vastaamisaika	Tila

Tarkastele tuloksia

Avaa Excelissä

1. Jos automatisointi aloitetaan prosesseista, niin mielestäni hyödyllisin olisi

Lisätietoja

- Volyymiltaan suurin prosessi 8
- Kriittisin prosessi 1
- Testitapausten ylläpidon kann... 1
- Muu 1

2. Jos automatisointi aloitetaan lomakkeista, niin toivon sen olevan

Lisätietoja

- Lomake, jota asiakkaat eniten ... 5
- Lomake, johon tehdään usein ... 4
- Lomake, jossa on paljon täytet... 1
- Muu 1

3. Odotuksesi ATF-sovelluksen tuomista hyödyistä?

Lisätietoja

9
Vastaukset

Uusimmat vastaukset

"Ajan säästö mm. versiopäivitysten testauksessa ja laadun paranemin..."

"Vähentää testaamiseen käytettävää työmäärää"

"Manuaalinen testaus vähenee."

1. Jos automatisointi aloitetaan prosesseista, niin mielestäni hyödyllisin olisi

11 Vastaukset

Tunnus ↑	Nimi	Vastaukset
1	anonymous	Volyymiltaan suurin prosessi
2	anonymous	Volyymiltaan suurin prosessi
3	anonymous	Volyymiltaan suurin prosessi
4	anonymous	Volyymiltaan suurin prosessi
5	anonymous	Testitapausten ylläpidon kannalta kevyin prosessi
6	anonymous	Siitä prosessista, johon voidaan odottaa kohdistuvan eniten muutoksia
7	anonymous	Volyymiltaan suurin prosessi
8	anonymous	Volyymiltaan suurin prosessi
9	anonymous	Kriittisin prosessi
10	anonymous	Volyymiltaan suurin prosessi
11	anonymous	Volyymiltaan suurin prosessi

2. Jos automatisointi aloitetaan lomakkeista, niin toivon sen olevan

11 Vastaukset

Tunnus ↑	Nimi	Vastaukset
1	anonymous	Lomake, jota asiakkaat eniten käyttävät portaalissa
2	anonymous	Lomake, johon tehdään usein päivityksiä
3	anonymous	Lomake, jota asiakkaat eniten käyttävät portaalissa
4	anonymous	Lomake, jota asiakkaat eniten käyttävät portaalissa
5	anonymous	Lomake, johon tehdään usein päivityksiä
6	anonymous	Siitä lomakkeesta, johon voidaan odottaa kohdistuvan eniten muutoksia
7	anonymous	Lomake, jossa on paljon täytettäviä kenttiä
8	anonymous	Lomake, johon tehdään usein päivityksiä
9	anonymous	Lomake, jota asiakkaat eniten käyttävät portaalissa
10	anonymous	Lomake, johon tehdään usein päivityksiä
11	anonymous	Lomake, jota asiakkaat eniten käyttävät portaalissa

3. Odotuksesi ATF-sovelluksen tuomista hyödyistä?

9 Vastaukset

Tunnus ↑	Nimi	Vastaukset
1	anonymous	Toivon että ATF vähentäisi manuaalisen testauksen määrää ja nopeuttaisi siten esim. versiopäivitysten yhteydessä tehtävää regressiotestausta.
2	anonymous	Huomioitaisiin testauksen suunnittelu ja toteuttaminen ennen muutosten toteuttamista. Tulisiko ATF:n käyttämät testitapaukset osaksi julkaistavaa kokonaisuutta?
3	anonymous	Versiopäivityksen testauksen apuna, lomakepäivitysten apuna.
4	anonymous	Vähemmän manuaalista testausta, virheet saadaan nopeammin kiinni, kriittisiin asioihin ei livahda tuotantoon virheitä
5	anonymous	Helpottaa ja nopeuttaa testaamista
6	anonymous	Varmasti tulee monessa tilanteessa helpottamaan ja nopeuttamaan versiopäivitysten yms. eri tilanteiden vaatimaa testaustarvetta.
7	anonymous	Manuaalinen testaus vähenee.
8	anonymous	Vähentää testaamiseen käytettävää työmäärää
9	anonymous	Ajan säästö mm. versiopäivitysten testauksessa ja laadun paraneminen

Liite 3: ATF-testitulosten säilytyskäytännön muokkaus

Modify data retention policy for ATF test results

Rome

Modify the Auto Flush data retention policy, which designates how long the system retains data, and referencing data, for test and test suite results. You can change the frequency of flushing for the sys_atf_test_result or sys_atf_test_suite_result base tables. This setting controls how far back in time test result data is available.

Before you begin

Role required: atf_test_admin

About this task

The system regularly flushes data in the sys_atf_test_result and sys_atf_test_suite_result base tables, (and optionally, referencing data). By default, the system deletes test and test suite results data 30 days after creation. This task enables you to modify the Auto Flush retention policy for data stored in a specific base table (sys_atf_test_result or sys_atf_test_suite_result).

Procedure

1. Navigate to **Automated Test Framework** > **Administration** > **Table Cleanup**.

   The system displays a list of the retention policies (Auto Flushes) it maintains for automated testing results tables.
2. Select the retention policy (sys_atf_test_result or sys_atf_test_suite_result) to modify.

   The system displays the record for this retention policy.
3. The **Tablename** field displays the name of the table to which the selected Auto Flush retention policy applies. Skip this field to accept the default.

   Note: Selecting another tablename in this field compromises the integrity of the Auto Flush record. Leave the tablename on existing ATF policies at the base system (default) value so it does not adversely affect ATF data retention behavior.
4. Specify how the system should determine the length of time for retention of data, and referencing data.
   a. In the **Matchfield** field, enter the field you want the system to use to monitor duration. For example, to specify that you want to delete data *x* amount of time after the system created it, leave **Matchfield** set to its default value of **sys_created_on**.
   b. In the **Age in seconds** field, enter the amount of time (in seconds) the system must wait before deleting the associated data and referencing data.
5. If you want to apply the policy to the specified data (for example, sys_atf_test_result), AND any data that references it, select **Cascade delete** (default value).

   Affected referencing data is stored the following tables: sys_atf_test_result_item, sys_atf_test_result_step, and sys_attachment (when table_name = sys_atf_test_result). If you want the policy to simply flush data in the selected table (for example, sys_atf_test_result), and skip flushing of the referencing data, then clear **Cascade delete**.
6. In the **Conditions** field, specify the filter conditions to use for selection of data (and optionally, referencing data) for this Auto Flush retention policy.

   The default is **Retain indefinitely is false**, because the Test results record also contains a **Retain indefinitely** check box that allows opting out of the auto flushes for specific test results.
7. Click **Update**.

Liite 4: Taulut, joista ATF ei voi palauttaa tehtyjä muutoksia

Tables excluded from rollback after running an automated test

Rome

The Automated Test Framework tracks data created by running tests and rolls back changes after testing. The system excludes certain tables from being tracked during testing.

The system excludes certain tables from being tracked or rolled back:

- The History [sys_history_line] table
- The ECC Queue table [ecc_queue].
- The Email [sys_email]Email Log [sys_email_log] tables
- The Report Executions [report_executions] and ReportStats [report_stats] tables.
- The Execution Tracker [sys_execution_tracker] tables
- The Progress Worker [sys_progress_worker] table
- The Schema Change [sys_schema_change]
- The Upgrade History [sys_upgrade_history] and Upgrade History Log [sys_upgrade_history_log] tables
- The Mutex [sys_mutex] table
- The Plugin Log [sys_plugin_log] table
- The Status [sys_status] table
- The Number Counter [sys_number_counter] table
- The AMB Channel Presence [sys_amb_channel_presence] table
- Any table that extends an excluded table.
- Any table starting with the following prefixes are also excluded:
    - syslog
    - sys_amb_message
    - sys_cluster
    - cmdb_metric
    - ts_
    - v_
    - sys_delete_recovery

If your test run changes (inserts/updates/deletes) any record on these excluded tables, the system does not roll back the change after testing.

Liite 5: ATF-käyttöliittymätestien ajo Headless Browser:illa

Headless Browser for Automated Test Framework

Rome

In the Rome release, ServiceNow® improved UI testing by automating the creation of browsers to process Automated Test Framework (ATF) User Interface (UI) tests. This feature is known as a headless browser, and helps you test UI functionality without having to manually open a browser to the Client Test Runner, which is what processes the UI tests in your local browser.

Background

ServiceNow customers use Automated Test Framework (ATF) to test applications and instances. When developing in the ServiceNow platform, your changes both have the desired behavior and don't break existing features.

Since the ServiceNow Orlando release, customers can automate the testing and deployment of their applications via the Continuous Integration/Continuous Delivery (CI/CD) API.

Overview

The automation provided by the ServiceNow Headless Browser for ATF skips the step of manually opening a browser during testing. There are several sequential procedures to follow in the one-time setup. Below are the instructions for both Linux and Windows setup.

Headless browser setup for Linux

The ServiceNow® headless browser for Automated Test Framework provides automation so you can skip having to manually open a browser during testing. The headless browser setup is available in both Linux and Microsoft Windows. This topic covers the setup for Linux.

Prerequisites

Role required: admin on your ServiceNow® instance and local administrator on the host machine.

- Docker application installed
- Java keytool installed
- OpenSSL
- Two-way communication
  - There must be two-way communication between the instance URL and your server.
  - Find the IP address of your server and get your hostname. You can use one or both of them, but you need at least one.

  Note: If you don't have a hostname and are connecting via the IP address, you can enter the IP address and put "localhost" in the Hostname environment variable.

  Tip: To make remembering these easier, set the following environment variables:
    - export PASSWORD="<password to generate the certificates with>"
    - export SERVERIP="<THIS SERVER'S IP ADDRESS>"
    - export HOSTNAME="<HOSTNAME OF THIS SERVER>"

- Port
  - Use Port 2376 or your own default port for this procedure. Make sure your firewall rules allow inbound requests on this port.

- To learn more, see Use TLS (HTTPS) to protect the Docker daemon socket.

1. Generate certificates for Headless Browser setup for Linux
   Step 1 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework: Generate TLS/SSL certificates to secure the Docker REST API and authenticate HTTP requests.

2. Configure Docker for Headless Browser setup in Linux
   Step 2 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework: Configure Docker Server to authenticate all requests.

3. Create the Docker image and containers for Headless Browser setup in Linux
   Step 3 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework.

4. Add secrets to Docker for Headless Browser setup in Linux
   Step 4 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework. Docker Secrets is a feature for securely storing the passwords that will be used in containers. You will be creating a Docker secret, which stores the password of the ServiceNow user who will log into the instance to execute the tests.

5. Set up your instance for Headless Browser in Linux
   Step 5 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework. Set up your instance so it can support the Headless Browser.

6. Configure ATF for Headless Browser in Linux
   Step 6 in the Linux setup for the ServiceNow® Headless Browser for Automated Test Framework.

Headless Browser setup for Microsoft Windows

The ServiceNow® Headless Browser for Automated Test Framework provides automation so you can skip having to manually open a browser during testing. The Headless Browser setup is available in both Linux and Microsoft Windows. This topic covers the setup for Windows.

Prerequisites

Role required: admin on your ServiceNow instance and local administrator on the host machine.

> ⚠ The only ServiceNow-supported version of Microsoft Windows as a host is: Windows Server 2019 v10.0.17763.737. No other versions are supported. If you are unable to meet these requirements, a Linux host is recommended.

- Make sure that the following programs are installed on your Windows server:
  - Install Docker for Headless Browser setup for Windows
  - Java keytool Chocolatey tool for javaruntime
  - OpenSSL Chocolatey tool for openssl
- Two-way communication
  - There must be two-way communication between the instance URL and your server.
  - Find the IP address of your server and get your hostname. You can use one or both of them, but you need at least one.

Note: If you don't have a hostname and are connecting via the IP address, you can enter the IP address and put "localhost" in the Hostname environment variable.

Tip: To make remembering these easier, set the following environment variables:

- set PASSWORD=<password>
- set SERVERIP=<server ip>
- set HOSTNAME=<hostname>

- Port
  - Use Port 2376 or your own default port for this procedure. Make sure your firewall rules allow inbound requests on this port.
- To learn more, see Use TLS (HTTPS) to protect the Docker daemon socket.

1. Install Docker for Headless Browser setup for Windows
   Before you begin setting up your ServiceNow® Headless Browser for Automated Test Framework for Microsoft Windows, you must install Docker.

2. Generate certificates for Headless Browser setup for Windows
   Step 1 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework: Generate TLS/SSL certificates to secure the Docker REST API and authenticate HTTP requests.

3. Configure Docker for Headless Browser setup in Windows
   Step 2 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework: Configure Docker Server to authenticate all requests.

4. Create the Docker image and containers for Headless Browser setup in Microsoft Windows
   Step 3 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework: Pull the Docker image from the Public Registry.

5. Add secrets to Docker for Headless Browser setup in Microsoft Windows
   Step 4 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework. Docker Secrets is a feature for securely storing the passwords that will be used in containers. You will be creating a Docker secret that stores the password of the ServiceNow user who will log into the instance to execute the tests.

6. Set up instance for Headless Browser in Microsoft Windows
   Step 5 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework: Set up your instance so it can support the Headless Browser.

7. Configure Automated Test Framework (ATF) for Headless Browser in MicrosoftWindows
   Step 6 in the Microsoft Windows setup for the ServiceNow® Headless Browser for Automated Test Framework (ATF): Configure ATF with properties.

Headless Browser image-instance version compatibility

The instance-to-image compatibility makes sure that the automation script inside the Docker image is compatible with the instance code. Elements such as the user interface might change over time to support new features or upgrades.

ServiceNow Headless Browser system properties

Below is a table of the properties you must have as you set up the ServiceNow®Headless Browser for Automated Test Framework.

System properties

Property name	Type	Default value
sn_atf.headless.browser_options	string	"--no-sandbox,--disable-gpu"
sn_atf.headless.default_browser	string	Chrome
sn_atf.headless.default_os	string	Linux
sn_atf.headless.docker_image_name	string	
sn_atf.headless.docker_window_seconds	int	60
sn_atf.headless.enabled	true/false	false
sn_atf.headless.heartbeat_enabled	true/false	true
sn_atf.headless.heartbeat_uri	string	/api/now/atf_agent/online
sn_atf.headless.images_check.enabled	true/false	false
sn_atf.headless.login_button_id	string	sysverb_login
sn_atf.headless.login_page	string	login.do
sn_atf.headless.password_field_id	string	user_password
sn_atf.headless.request_timeout_sec	int	200
sn_atf.headless.retry_count	int	10
sn_atf.headless.runner_banner_id	string	test_runner_banner
sn_atf.headless.runner_url	string	atf_test_runner.do? sysparm_nostack=true&sysparm_scheduled_tests_only=true&sysparm_he
sn_atf.headless.secret_gid	string	1000
sn_atf.headless.secret_id	string	
sn_atf.headless.secret_name	string	
sn_atf.headless.secret_path	string	/run/secrets/<secret_name>
sn_atf.headless.secret_uid	string	1000
sn_atf.headless.service_clean_exclude_list	CSV	
sn_atf.headless.service_stop_deletes	true/false	false
sn_atf.headless.timeout_mins	int	1440

sn_atf.headless.user_field_id	string	user_name
sn_atf.headless.username	string	
sn_atf.headless.validation_id	string	headless_vp_validation
sn_atf.headless.validation_page	string	atf_headless_validation_page
sn_atf.headless.vp_has_role_id	string	headless_vp_has_role
sn_atf.headless.vp_success_id	string	headless_vp_success

Headless Browser troubleshooting

These tips can help you troubleshoot your Linux or Microsoft Windows setup of the ServiceNow® Headless Browser for Automated Test Framework.

There are three basic areas to examine when troubleshooting your Headless Browser setup.

Docker container

Error logs: When the headless test completes, if there are errors they are generated in the Docker container. Whether the operation fails or succeeds, the container's **stdout/stderr** logs are placed in the **sn_atf_docker_service** table.

Headless client test runner did not start in the time allotted message: This message generally means an error occurred in the Docker container while it was initializing or starting up. This may indicate something is incorrect in your Docker container setup. Navigate to the **sn_atf_docker_service** table to read the logs and see the error message.

Instance

Error caught running Docker flow when starting the ATF tests messages: Follow the URL address to find the flow context, which contains the error logs. For example, the above error message can display when the instance can't access the Docker host.

Network errors

Firewalls: Make sure your firewalls are set up so that the instance can access the host and port and the server can access the instance.

ServiceNow Headless Browser action flow diagram