LAATU JA TESTAUS

Automatisointi

2/2012

TestausOSY
Finnish Association of Software Testing

JULKAISIJA

Testauksen osaamisyhteisö (TestausOSY)

TOIMITUSKUNTA

Jussi Ahonen

Marko Lappalainen

Antti-Pekka Marjamäki

Agnieszka Nummi

Laura Ojala

Olli-Pekka Puolitaival

Maaret Pyhäjärvi

Erkki Pöyhönen

Soile Sainio

JULKAISUPAIKKA

Verkko: http://www.testausosy.fi

JULKAISUTIHEYS

Kolmesti vuodessa noin neljän kuukauden välein

Lehti on vapaaehtoisvoimin laadittu. Artikkelien oikeudet kuuluvat kirjoittajille.

Sisällysluettelo

Pääkirjoitus

Jussi Ahonen

Testauksen ammattilainen käyttää intensiivisesti aivojaan työssään sillä testaus on abstraktia työtä, jonka lopputuloksena on tietyssä kontekstissa paras mahdollinen tapa testata jokin olettamus.

Tämän työn jälkeen tehdään fyysistä työtä, hiiri liikkuu näytöllä ja kenttiin syötetään tietoa tai päräytetään eräajo käyntiin. Varsinainen testitapauksen suorittaminen on lähinnä erilaista tarkistamista.

Isommassa kontekstissa testauksen avulla saadaan tietoa, jota käytetään päätöksenteossa. Jos tehdään hyviä päätöksiä, tuotteen tai palvelun laatu on paranee ja tällöin saadaan tyytyväisempi loppukäyttäjä ja onnellisempi omistaja. Näin testauksen avulla voidaan tehdä hyvää muille ihmisille.

Kuinka voidaan auttaa testaajaa omassa työssään, voisiko testiautomaatiosta olla apua?

Testiautomaation hyödyntämiseen on eri näkökulmia: joissakin yrityksissä tehdään ensin automaattitestit ja vasta sen jälkeen tutkivalla testauksella etsitään ne hankalimmat bugit.

Toisaalla ensin tehdään perinteiset manuaalitestit ja kun tietojärjestelmä on stabiloitunut, tehdään automaattiset testit. Sama näkökulma ei sovi erilaisiin organisaatioihin.

Olisiko parempi jos voisimme tehdä enemmän testausta, seuraisiko siitä enemmän hyvää muille?

Testiautomaatio lupaa paljon hyvää mutta ainakin käytännössä testiautomaation avulla voidaan automatisoida testien ajaminen ja testitulosten raakadatan analysointi. Automaatiota voi käyttää myös testiaineiston luomiseen, testiympäristöjen luomiseen ja raportointiin

Tässä lehdessä on muhkea paketti testiautomaatiosta eri näkökulmista: lukekaa ja viekää ideoita eteenpäin. Otetaan opiksi muiden kokemuksista ja valjastetaan testiautomaatio auttamaan ihmistä.

Innostavia lukuhetkiä.

Haastattelu: Robotti haltuun

Laura Ojala

Esittelyssä: Marko Voutilainen

Marko Voutilainen vastaanotti vajaa vuosi sitten uuden työtehtävän; edessä oli ketterän kehitystiimin testaajan pesti erityisalueena käyttöliittymätestaus ja automatisoidun hyväksyntätestauksen toteuttaminen. Tässä haastattelussa kyselen Markolta, mitä uusi tehtävä toi tullessaan erityisesti testauksessa käytettäviin työkaluihin liittyen.

Mikä on taustasi testauksessa?

Ikää on 36-vuotta ja työkokemusta IT-alalta on reilu 10 vuotta. Aloitin urani C/C++ koodarina, mutta olen sittemmin liukunut siitä usean sattuman summana tänne testauspuolelle. Kehitin ensin testaussovelluksia ja siirryin siitä ihan kokonaan testaukseen, sillä testaus kiinnosti enemmän kuin puhdas koodaus.

Nykyään asun Helsingissä ja työskentelen Tesnet Group Oy:n ohjelmistotestauskonsulttina. Viimeiset 10 kk olen työskennellyt QA:n roolissa suurille loppukäyttäjämäärille suunnatun finanssialan webbisovelluksen kehitystiimissä. Olennainen osa tehtävää on ollut sprinttien hyväksyntätestauksen automatisointi käyttäen Robot Frameworkia.

Millaista kokemusta sinulla on testauksessa käytettävästä automaatiosta?

Koodaritaustani näkyy testauksessakin siinä, että olen usein vastuussa automaatiopuolesta. Olen usein tehnyt Perl ja Python skriptejä testauksen tueksi, esimerkiksi testidatan generointiin. Lisäksi olen ollut mukana tekemässä tarvekartoituksia automatisoinnille ja esimerkiksi HW-testauslinjojen ohjaukseen käytettävää softaa. Onpa minulla kokemusta myös muutaman Quality Centerillä toteutetun testiajon toteuttamisestakin.

Oliko sinulla kokemusta ATDD-kehityksessä usein käytettävistä automaatiotyökaluista (Cucumber, FitNesse, Robot Framwork, Concordion…) ennen tätä projektia?

Käytännössä tämä projekti oli ensimmäinen, jossa käytin mitään näistä työkaluista. Projektin alussa tiimi teki valinnan Concordionin ja Robot Frameworkin välillä ja valittiin Robot Framework.

Mistä olet saanut oppia Robot Frameworkin käyttämiseen?

Aloitin siitä, että googletin mikä on Robot Framework ja ryhdyin sitten itseopiskelemaan yrityksen ja erehdyksen kautta – positiivisesta palautteesta päätelleen harjoittelu on mennyt hyvin. Robotin käyttöönotto ei siis vaatinut mitään pitkää kurssittamista ja se oli helposti lähestyttävä, varsikin kun minulla oli taustaa automatisoinnissa. Suurin apu opiskelussa oli Robot Frameworkin dokumentaatio sekä kehittäjien sivulta löytyvät valmiit esimerkkipaketit.

Mitä voidaan saavuttaa? Onko ollut panostuksen arvoista?

Näkisin, että tässä projektissa automatisointi on ollut siihen käytetyn panoksen arvoista. Kehityksen laadusta on tullut kautta linjan kiitosta. Missään

vaiheessa ei ole päässyt tulemaan regressiota, sillä regressio-ongelmat ovat jääneet kiinni heti kun uusi muutos lisätään versionhallin.

Toinen selkeä hyöty on ollut toiminnallisuuden todistettavuus: robottitestien myötä tuotteella on todella laaja testisetti, jota ajetaan jokaiselle muutokselle. Softan tämän hetkinen tilanne ja se mikä toimii, on ajantasaisesti kaikkien saatavilla. Toiminnan todistettavuus missä tahansa vaiheessa on siis erittäin hyvällä tolalla.

Ovatko Robotilla toteutetut automatisoidut tarkastukset löytäneet regressio-ongelmia?

Tiimissä ollaan kautta linjan ylpeitä laadusta ja regressiobugeja löytyy vähänlaisesti. Silloin tällöin automaatio löytää kuitenkin ongelmia, joita ei löydettäisi yhtä tehokkaasti käsin testaamalla. Viimeisin esimerkki tästä on se, että kun käyttöliittymän kieltä vaihdettiin, tietyt toiminnot lakkasivat toimimasta. Koska toimintoja löytyy yhdelläkin kielellä testatessa useita, niin on hyvin epätodennäköistä että testauksessa olisi osuttu kovin äkkiä tähän ongelmaan.

Mitä riskejä näet hyväksyntätestien automatisoinnissa?

Robottitestien huumassa ei saa unohtaa ihmisen tekemän testauksen tarvetta. Siihen, että on automatisoidut robottitestit, ei siis voi luottaa 100 %.

Minkä asioiden huomioon ottamista käyttöliittymätestien automatisointi edellyttää?

Testit pitää integroida kehitysympäristöön eikä siis toimia niin, että joku ajelee testit omalla koneellaan silloin kun sattuu huvittamaan. Tässä projektissa CI-palvelimeen integroituminen oli yllättävän haastavaa ja onneksi tiimin Maven-gurut auttoivat siinä.

Testien tekemistä helpottaa se, että käyttöliittymän elementit on nimetty selkeillä tunnisteilla. Testeistä kannattaa lisäksi tehdä modulaarisia, jotta niiden ylläpito on helppoa sitä mukaa kun käyttöliittymä kehittyy – ei siis voi kirjoittaa pötkökoodia ja kannattaa käyttää ylätason avainsanoja, joita voi hyödyntää useissa testeissä.

Millaisia haasteita sinulla on ollut Robot Frameworkin käytössä?

Robot Framework valmiiden Selenium-kirjaston kanssa on helppokäyttöinen, mutta melko rajoittunut. Kun testataan muuta kuin käyttöliittymää, skriptit pitää kirjoittaa itse. Tässä työssä olisi ollut mukava, jos saatavilla olisi ollut valmis moduuli suorille tietokantakutsuille, jotta voi tarkistaa, onko joku tieto tietokannassa.

Lisäksi käyttöliittymän testauksessa piti joissain kohdin olla luova, sillä pelkkä html:n validointi ei kanna pitkälle: valmiilla kirjastoilla pystyy esimerkiksi tarkastamaan, milloin joku elementti ilmestyy sivulle, mutta ei sitä milloin se katoaa. Testasin nämä käyttäen tekemällä actionin, lisäämällä viiveen ja tarkistamalla mikä oli tilanne - viiveiden käyttö aiheutti hitaammissa ympäristöissä ongelmia.

Välillä haasteita aiheutti myös se, että Selenium -kirjasto päivittyy hitaammin kuin selaimet: uuden selainversion ilmestyessä testejä piti päivittää tai jättää uusi selain asentamatta.

Miten tästä eteenpäin?

Siirryn uuteen projektiin ja jatkan edelleen Robot-testien parissa. Robot osaaminen tuntuu olevan tällä hetkellä erittäin kysyttyä.

Testiautomaatio on kuormitustestauksen kaveri

Esa Vaarala

Manuaali vai automaatti?

Ohjelmistotestauksessa yksi eniten puhuttuja aiheita on testiautomaatio ja sen käyttö oikeassa paikassa. Jaetaan seuraavaksi testaus kahteen osioon: ihmisen suorittamaan testaukseen ja koneen suorittamaan testaukseen.

Ihmisen suorittama testaus on ohittamatonta kun testauksen on määrä jäljitellä loppukäyttäjän käyttäytymistä. Aito ihmiskäyttäjä tulee käyttämään tuotetta oikein, väärin ja yllättävästi. Mikä tärkeintä, ihmiskäyttäjä kokee tuotteen jollain tavalla ja se herättää hänessä tunteita. Ihmisen käyttäytymistä voi jäljitellä ja testata vain ihminen. Näitä testituloksia käyttämällä tuotteesta on mahdollista kehittää maailmanluokan tuote joka miellyttää suurta joukkoa ihmisiä.

Testiautomaatio päihittää ihmisen suorittaman testauksen silloin kun tapahtumaa pitää toistaa useaan kertaan. Säännöllisesti toistuvan tapahtuman testaamisessa ihminen on kehno, koska me tylsistymme nopeasti saman toistamiseen. Kun ihmiseltä puuttuu motivaatio, niin tekemiemme virheiden määrä lähtee nousuun niin testauksessa kuin koodauksessakin. Kone ei tylsisty koskaan ja tekee saman asian väsymättä aina uudestaan ja uudestaan, tarvittaessa erittäin nopeasti.

Testiautomaatio on kuormitustestauksen kaveri

Koneen suorittamalle testiautomaatiolle ei oikein edes löydy vaihtoehtoa silloin kun puhumme kuormitustestauksesta. Kuormitustestaustyökalut ovat testiautomaatiotyökaluja. Kuormitustestauksessa on tyypillisesti kyse siitä, että simuloidaan suuri määrä käyttötapauksia testauksen alaiseen palveluun. Puhutaan helposti tuhansista tai jopa miljoonista toistoista suoritettuna yleensä lyhyehkön ajan yli. Tähän ei ihmistestaaja pysty. Testiautomaatiotyökalut ovat oikeita apuvälineitä silloin kun halutaan testata tuotteen käyttäytyminen kuormituksessa etukäteen ennen julkaisua.

Onko palvelualusta säädetty oikein? Kuinka monta käyttäjää kyetään palvelemaan ilman ongelmia? Missä on palvelun pullonkaula? Miten palvelu saadaan palautettua kaatumisen jälkeen? Onko kaatumisen varalle tehty hätäsuunnitelmaa? Tyypillisiä vastauksia näihin suorituskyvyn avainkysymyksiin ovat vastaavasti: Palvelualusta ei ole säädetty oikein. Vain kourallista käyttäjiä voidaan palvella ilman ongelmia. Palvelun pullonkaulaksi muodostuu muisti, kaista, IO, CPU tai usein näiden yhteisvaikutus. Kaatuneen palvelun palauttaminen toimintakuntoon kestää tunteja tai päiviä. Kuormitustestaus antaa tulokset muun muassa näihin kohtiin ja hyvissä ajoin suoritettuna vaadittuja parannuksia ehditään tekemään ennen palvelun julkaisua.

Milloin kuormitustestata?

Edelleenkin kovin usein kuormitustestaus jätetään tekemättä ja palvelu saatetaan loppuasiakkaan käyttöön ilman tietoa siitä, että miten palvelulle

itse asiassa käy kun sitä ryntää yhtä aikaa käyttämään suuri määrä ihmisiä. Viime vuosina olemme saaneet lukea uutisista kuinka moni palvelu on kokenut ongelmia nimenomaan silloin kun palvelua on käyttänyt suuri joukko ihmisiä. Palvelut ovat pääsääntöisesti toimineet pienillä käyttäjämäärillä, mutta miten käy kun ihmiset ryntäävät innolla uutta palvelua käyttämään?

Vuosikymmen sitten kuormitustestaus oli todellista erikoistekniikkaa johon ei kaikilla rahkeet riittäneet. Nykyään tilanne on toinen ja tarjonta kuormitustestauksen työkaluihin on parantunut ja kuormitustestauksen testaamatta jättämistä ei voi enää perustella esimerkiksi teknisillä tai taloudellisilla selitteillä. Markkinoilta löytyy muun muassa seuraavat työkalut: NeoLoad, Soasta, Blazemeter ja Gridinit. Mainituista työkaluista tarkastellaan hieman lähemmin Blazemeteriä.

Blazemeter on palvelu joka tarjoaa palvelinkapasiteettia Jmeter-skriptien ajamiseen. Jmeter on puolestaan avoimen lähdekoodin työkalu suorituskyvyn testaamiseen joka on saavuttanut suorituskyvyn testauksessa de facto -standardin aseman. Kun Jmeter-skriptit ajetaan Blazemeterin palvelussa, niin voidaan puhua oikeasta kuormitustestauksesta ja päästään tuhansiin yhtäaikaisia käyttäjiä simuloiviin testiskenaarioihin.

Kun kyseessä on palvelu jota tulee käyttämään suuri määrä ihmisiä (esimerkiksi viranomaisen tarjoama palvelu) niin vaihtoehtoja on kaksi: riskillä meneminen tai testiautomaatiolla toteutettu kuormitustestaus. Mikäli selkänahka ei ole riittävän parkkiintunut sietämään riskin tuomaa kuumotusta, niin on selkeästi aika pyytää ammattilaisia apuun. Meitä testaajia motivoi aina kun tullaan reilusti kysymään neuvoa yli karien ja kivikoiden.

Testaaja, projekti ja testiautomaatio

Testiautomaatio on yksi testaajan lukuisista työkaluista hänen kookkaassa työkalupakissaan. Vaikka testiautomaatio löytyy testaajan työkalupakista se ei suoraan tarkoita, että sitä pitäisi aina käyttää. Laaja työkalujen kirjo on työkalupakissa siltä varalta, että eteen sattuu tilanne joka tarvitsee hieman erilaista lähestymistapaa kuin aikaisempi tapaus. Työkalujen käytön kokemus lisää varmuutta oikean työkalun valintaan tilanteesta toiseen mentäessä, mutta joitain nyrkkisääntöjä on myös olemassa. Jos projektilla on vain muutama testaaja, niin usein on järkevämpää antaa testaajien suorittaa tutkivaa testausta ja olla koodaajien kaverina, kuin käyttää aikaa testiautomaatin säätämiseen.

Kun testiautomaatio otetaan projektissa käyttöön, niin sen tulee olla koko projektin yhteinen päätös ja toimintamalli. Oli kyseessä sitten automatisoitu yksikkötestaus, toiminnallinen testaus tai kuormitustestaus, niin asian tulee olla kaikkien projektijäsenten tiedossa. Yksikkötestaus ja toiminnallinen testaus vaativat monesti asian huomioon ottamista koodissa kun taas puolestaan kuormitustestaus on kokonaisuus jonka tulokset vaikuttavat koko tuotteen rakenteeseen.

Esa Vaarala on iloinen testaaja, joka nauttii testaushommista jo yhdettätoista vuotta. Hän on erikoistunut kuormitustestaukseen ja pitää hallussaan epävirallista ennätystä 2 500 000 käyttäjän simuloinnista tunnin aikana.

Kokemuksia testiautomaation käyttöönottoprojektista

Sami Luoma

Taustaa

IT-kehitystyö kansainvälisessä rahoitusalan konsernissa on hyvin monimuotoista. Sekä eri toiminta-alueiden tarpeet että muut ympäristötekijät ovat ajan myötä, järjestelmäkehityksen edetessä muokanneet sovelluksia ja ympäristöjä hyvinkin eri suuntiin. Tällaiset järjestelmien eroavuudet aiheuttavat vääjäämättä myös käytettyjen kehitysmallien ja –menetelmien eroavuuksia, ja kun kehitysmallit eroavat toisistaan, myös testauskäytännöille käy samoin. Luonnollisesti myös testiautomaation hyödyntäminen osana testikäytäntöjä vaihtelee huomattavasti konsernin eri toiminta-alueiden kesken.

Testiautomaation kautta saatava potentiaalinen hyöty niin lopputuotoksen laadussa kuin testaustoiminnan tehostumisessakin on tunnistettu Nordea-konsernissa jo jonkin aikaa. Jotta testiautomaatiota hyödynnettäisiin mahdollisimman laaja-alaisesti, on konsernissa käynnissä projekti, jonka tarkoituksena on tukea käynnissä olevia järjestelmä/ sovelluskehitysprojekteja testiautomaation käyttöönotossa. Tämän tukiprojektin kautta kohteena olevan kehitysprojektin oma panos automaation toteutukseen pienenee kun projektin käyttöön osoitetaan uusia asiantuntijaresursseja tukiprojektin toimesta.

Testiautomaation käyttöönoton tuki (TaKT) – projektin organisaatio koostuu sekä Nordean sisäisistä testausasiantuntijaresursseista että konsernin ulkopuolelta hankitusta testiautomaatio-ohjelmoinnin erityisasiantuntijapalveluista. Itse pääsin osalliseksi TaKT -projektiin keväästä 2012 lähtien, jolloin testiautomaatioapua sai tai oli saanut jo toistakymmentä projektia. Tässä artikkelissa kuvaan hieman lähemmin projektissamme käytössä olevaa lähestymistapaa testiautomaation käyttöönottoon, esittelen itse projektitukiprosessin päävaiheet sekä niiden sisällön, ja lopuksi pohdin hieman toiminta-mallimme vahvuuksia, heikkouksia sekä mahdollisia tulevaisuudennäkymiä.

Lähestymistapa

TaKT –projektin osallistujat valitaan kaksivaiheisella menettelyllä. Ensimmäisessä vaiheessa käydään eri alueiden IT-päättäjien eli sponsoreiden kanssa läpi alueella meneillään olevat kehitysprojektit. Sponsorille esitellään myös tarvittaessa testiautomaation ominaispiirteet (mitä edellyttää, millaisissa tilanteissa parhaimmillaan jne.), jotta suoritettava valinta perustuisi oikeanlaiseen ymmärrykseen itse asiasta.

Ensimmäisen vaiheen tuloksena on siis lista sponsorin valitsemia projekteja, joissa testiautomaatiosta saattaisi olla hyötyä ja joiden profiili on sellainen, että testiautomaation käyttöönotto on mahdollista (listalta poistetaan mm. aikataulullisesti ja/ tai resurssien vuoksi kriittiset projektit).

Toisessa vaiheessa valittujen projektien projektipäällikön, testipäällikön ja mahdollisesti myös projektin johtoryhmän kanssa tavataan ja keskustellaan miten yhteistyö voitaisiin käynnistää. Näissä alkutapaamisissa TaKT –projektin edustajat esittelevät sekä testiautomaation pääperiaatteet, että myös itse automaation käyttöönoton tuen prosessin, eli miten asiat tulisivat etenemään, mikäli kohdeprojekti päättää lähteä mukaan (ks. tarkemmin kohta ”prosessikuvaus”).

Alkutapaamisen lopputuloksena voi olla joko kohdeprojektin päätös lähteä mukaan, projektin toive toteutuksen siirtämisestä myöhäisempään ajankohtaan tai yhteinen päätös siitä, että automaatiota ei kyseessä olevan projektin yhteydessä hyödynnetä. Tapaamisen lopputulos viestitään myös projektin alueen sponsorille jotta hän pysyy ajan tasalla sekä siitä, miten laajasti automaatiota hyödynnetään hänen alueellaan että myös niistä tekijöistä jotka on mainittu automaatiosta kieltäytymisen perusteiksi.

Mikäli kohdeprojekti kuitenkin päättää lähteä mukaan joko heti tai hieman myöhemmin, projekti siirtyy automaatiotukiprosessin ensimmäiseen vaiheeseen eli soveltuvuusanalyysiin tai, mikäli tietyt ennakkokriteerit täyttyvät, suoraan pilottiin (ks. tarkemmin luku ”Pilotti”).

Testiautomaatiotuen prosessi

Tässä kappaleessa käydään läpi prosessi, jonka avulla testiautomaation käyttöönotto toteutetaan valituissa (sovellus)kehitysprojekteissa.

Soveltuvuusanalyysi

Automaation käyttöönoton ensimmäinen vaihe on yhden päivän aikana suoritettava soveltuvuusanalyysi, jossa testiautomaatioasiantuntijat selvittävät kohteena olevan sovelluksen/ järjestelmän toimintaympäristön, kehitysnopeuden ym. testiautomaation kannalta merkittäviä tekijöitä. Päivä itsessään koostuu aamulla suoritettavasta 1 – 2 tunnin järjestelmä- ja liiketoiminta-asiantuntijoiden haastattelusta sekä itse analyysiraportin koostamisesta saatujen tietojen pohjalla. Päivän päätteeksi valmistunut analyysi antaa arvion siitä, kannattaako testiautomaatiota ylipäätään harkita kyseisen järjestelmän testaustoiminnan osaksi, ja mikäli kannattaa, myös karkean arvion automaatiolla saavutettavissa olevista rahallisista ja laadullisista hyödyistä.

Tuotettu analyysiraportti käydään läpi ja hyväksytetään sekä itse projektin johdolla että kyseisen alueen sponsorilla. Näin varmistetaan yhteinen käsitys saavutettavissa olevista hyödyistä tai syistä miksi automaatiota ei kyseisessä tapauksessa ehkä kannatakaan ottaa käyttöön.

Pilotti

Automaation käyttöönoton toisessa vaiheessa projektin testauspäällikön ja/ tai projektipäällikön kanssa valitaan projektin koko testauskokonaisuudesta sopiva pienempi osio, jolle projektille nimetty automaatioasiantuntija toteuttaa testiautomaation ”koe-ajon”. Tämän koeajon yhteydessä testiautomaatio-organisaatio tuottaa myös tarkemman analyysin kohteena olevan testauskokonaisuuden automatisointipotentiaalista. Toisin sanoen, koeajon yhteydessä arvioidaan keskimääräinen automatisointiin kuluva aika/ testikokonaisuus, kuinka suuri osa koko projektin testauksesta on mahdollista automatisoida, sekä tarkennetaan alun perin annettua hyötyarviota. Lisäksi pystytään myös arvioimaan kuinka paljon automatisointiasiantuntijat tulevat tarvitsemaan liike-toiminta- ja järjestelmäasiantuntijoiden tukea jos projekti päättää edetä laajempaan automaation käyttöönottoon.

Pilotin tuloksena kohdeprojektin johto sekä toiminta-alueen sponsori saavat tarkemman arvion

automaation käyttöönotolla saavutettavissa olevista hyödyistä, käyttöönottoon tarvittavista resursseista sekä arvion automaation käyttöönoton aiheuttaman investoinnin takaisinmaksuajasta, eli siitä, missä ajassa automaatioon satsatut lisäresurssit maksavat itsensä takaisin saavutettuina laadullisina ja kustannuksellisina hyötyinä. Pilotin tuloksena kohdeprojekti ja/ tai alueen sponsori joko päättävät jatkaa automaation toteutusta koko testausalueeseen tai saavat perusteet automaatioprosessin päättämiselle.

Jos valitun kohdeprojektin toimintaympäristö, käytetty teknologia ja liiketoimintamallit ovat automaatio-organisaatiolle jo entuudestaan tuttuja, voidaan automaation käyttöönotto käynnistää myös suoraan pilotilla. Tällöin on kuitenkin erikseen varmistettava kohdeprojektin avainhenkilöiden tietämys automaation käyttöönottoon liittyvistä järjestelyistä (jotka normaalisti esitellään osana soveltuvuusanalyysiin valmistautumista).

Toteutus

Onnistuneen soveltuvuusanalyysin ja pilotin jälkeen kohdeprojektille valittu automaatioasiantuntija jatkaa työtään osana projektin testaustoimintoja. Testiautomaation luonti- ja ylläpitoprosessit toteutetaan kiinteänä osana projektin testaus-toimintoja, ja automatisoinnin etenemistä seurataan kohdeprojektin ja TaKT –organisaation toimesta. Kohdeprojektin valmistuessa automatisoinnin toteutuneesta tasosta, muotoutuneista toimintamalleista sekä alkuperäisen hyötyarvion ja lopullisen toteutuman ero raportoidaan projektin johdolle sekä alueen sponsorille.

Siirtyminen ylläpitoon

Kohdeprojektin testaukseen liittyvän materiaalin, toimintamallien ja työkalujen siirtyessä ylläpito-organisaatiolle myös käytetty testiautomaatioon liittyvä materiaali siirretään. Erityisesti tässä vaiheessa tulee varmistaa että myös testiautomaatioon liittyvä osaaminen siirtyy ylläpito-organisaatioon.

Tämä ns. testiautomaation hand-over toteutetaan lyhyenä koulutuksena/ perehdytyksenä, jonka yhteydessä automaation alun perin tuottanut asiantuntija ja mahdollisesti muita TaKT –projektin jäseniä perehdyttävät ylläpito-organisaation edustajat sekä olemassa olevan automaatio-kokonaisuuden toimintaan että testiautomaation ylläpidon perusperiaatteisiin. Automatisointiprojektin osuus automatisointityössä päättyy, kun ylläpito-organisaatio hyväksyy automaatio-kokonaisuuden omalle vastuulleen.

Kokemukset tähän mennessä

Seuraavissa kappaleissa annan lyhyesti oman näkemykseni käytössämme olevan toimintamallin vahvuuksista ja heikkouksista, ja lopuksi pohdin hieman tulevaisuuden näkymiä Nordean testiautomaation suhteen.

Vahvuudet

Yksi tärkeimmistä toimintamallimme vahvuuksista on tuottamamme testiautomatisoinnin tuen riippumattomuus kohdeprojektin budjetista. Oli alun perin rohkea päätös toteuttaa TaKT –projekti itsenäisenä tukiprojektina, mutta se on maksanut itsensä takaisin mm. automatisoinnin aloittamisen markkinoinnin helppoutena. Projektipäällikölle on aina helpotus kuulla että projektiin mahdollisesti mukaan tulevat lisätoiminnot eivät aiheuta projektille suuria lisäkustannuksia.

Toinen selkeä vahvuus automaation toteutuksessa kuvatun mallin mukaisesti on mukaanlähdön ”helppous”. Soveltuvuusanalyysin toteutus ei kuormita projektia oikeastaan ollenkaan, ja jo sen perusteella voidaan karkeasti arvioida kannattaako automaatiota lähteä toteuttamaan vai ei. Vaikka päätös edetä pilottiin tehtäisiinkin, vielä tämänkin

vaiheen kuormitus itse projektille on kohtuullisen suppea. Ja pilotin kautta projekti saa jo selkeän arvion saavutettavissa olevista hyödyistä ja voi helposti päättää viedäänkö toteutus loppuun vai ei.

Haasteet

Yksi kohtaamistamme haasteista testiautomaation toteuttamisessa tietynlaisena kehitysprojektin ulkoisena osana on ”momentin säilyttäminen”, toisin sanoen saada asiat etenemään sujuvasti toteutusvaiheesta toiseen. Mikäli soveltuvuus-analyysin tai pilotin tulosten läpikäynti kaikkien tarvittavien sidosryhmien kanssa pitkittyy syystä tai toisesta, projektin ”innostus” asiaan saattaa laantua, avainhenkilöt vaihtua tai jopa projektille asetetut tavoitteet muuttua. Tällaisissa tilanteissa testiautomaation saaminen uudelleen käyntiin voi olla hyvinkin vaikeaa, ja pahimmillaan lopputuloksena voi olla kesken jäänyt automatisointityö. Rautaa on siis taottava sen ollessa kuumaa!

Myös toinen havaitsemani heikkous liittyy toteutusmallimme ulkopuolisuuteen itse projektista. Projektiin mukaan valittavan testiautomaatioasiantuntijan sisäänajo projektin omaan testiorganisaatioon on kriittinen vaihe testiautomaation onnistumiselle. Sisäänajossa selkeä etu saavutetaan kun projektin johto (= projektipäällikkö) ja eritoten testauspäällikkö kokevat testiautomaatiosta saatavat hyödyt realistisiksi ja sitoutuvat ottamaan automaation mukaan osaksi projektin kokonaisuutta. Jos tätä sitoutumista ei tapahdu, automaatiotestausasiantuntija jää helposti organisaation ulkopuoliseksi toimijaksi, jolloin hänen tiedonsaantinsa hankaloituu huomattavasti. Kun tieto ei kulje, jää automaatiolla saavutettavissa ollut hyöty suurelta osalta saavuttamatta. Ja kun hyötyä ei saavuteta oletetusti, sekä kohdeprojektin avainhenkilöiden että alueen sponsorin tyytyväisyys kärsii.

Ajatuksia ja ideoita tulevaisuutta ajatellen

TaKT –projektin takana on tällä hetkellä useita onnistuneita toteutuksia ja muutama kantapään kautta saatu opetus. Edessä taas näyttäisi olevan yhä laajempi kiinnostuksen kenttä, jonka kautta automaatiosta saadaan toivottavasti vieläkin enemmän irti.

Yksi tulevaisuuden haasteista TaKT –projektille tulee olemaan yhteistyön käynnistäminen suoraan ylläpito-organisaatioiden kanssa. Pankkien sovelluskanta kun on tunnetusti suhteellisen pitkäikäistä, on sovellusten ja järjestelmien ylläpitoon liittyvä toiminta hyvinkin laaja-alaista. Nordea ei ole tässä tapauksessa poikkeus, joten ylläpito-toiminnoissa on varmasti vielä paljon potentiaalisia testiautomaatiokohteita.

Odotan mielenkiinnolla millaisia mahdollisuuksia seuraavat pari vuotta tuovat tullessaan. Tukiprojektimme alkuhankaluudet ovat nyt suurelta osalta takanapäin, aika näyttää miltä suunnalta nyt jo lähes vakiintuneet toiminta-mallimme kohtaavat seuraavat isommat haasteensa.

Sami Luoma on Hyvinkäältä Helsinkiin työmatkaava IT-alan sekatyömies, jonka tämänhetkiseen työnkuvaan kuuluu IT-kehitystyön metodit ja menetelmät. Testiautomaatio osana ketteriä (kehitys)menetelmiä kuuluu hänellä myös henkilökohtaisiin kiinnostuksen kohteisiin.

Mallipohjainen testaus voi olla myös helppoa ja nopeaa

Olli-Pekka Puolitaival ja Teemu Kanstren

Mitä on testausautomaatio? Onko se sitä että painetaan nappia ja tietokone ajaa skriptin mikä on vaivalla kirjoitettu sille ajettavaksi? Käytetäänkö skriptin kirjoittamiseen enemmän aikaa kuin mitä se ajaminen olisi vienyt manuaalisesti? Ajetaanko se vain kerran ja sitten järjestelmä onkin jo muuttunut niin paljon ettei sillä enää tee mitään?

Testausautomaatio voi olla kaikkea tätä. Se voi olla myös enemmän ja vähemmän. Nykyisellään eri ympäristöihin löytyy testausautomaatiota tukevia työkaluja joilla erilaisia testitapauksia voidaan luoda ja "automatisoida" eli skriptata. Tällaisia ovat esimerkiksi Selenium webbisoftalle ja Robotium Android ympäristöille.

Testaaja on monesti testattavan ohjelmiston asiantuntija, tai hänestä muokkautuu sellainen. Monesti paras henkilö jolta kysyä miten ohjelmisto toimii ja miten sillä voidaan suorittaa joku toiminto onkin sen testaaja. Tällä täytyy olla kokonaiskuva ohjelmiston toiminnasta ja kokonaisuudesta, siinä missä kehittäjillä on monesti vain kuva omasta osa-alueestaan ja ohjelmiston pienestä palasesta mitä kehittää.

Kun ihminen on asiantuntija jossain asiassa, hän muodostaa mielessään oman mallin siitä miten järjestelmän pitäisi toimia. Testauksen kannalta tämä on hyvä asia, sillä testauksessa on tärkeää miettiä järjestelmää ja sen testausta kokonaisuutena. Modernit järjestelmät ovat kuitenkin hyvin monimutkaisia ja kaikkien olennaisten testipolkujen kattaminen suuressa mittakaavassa on haastavaa. Kaikkien polkujen (testiskriptien) määrittely manuaalisesti vie liikaa aikaa ja rahaa jotta sitä haluaisi, jaksaisi tai kannattaisi tehdä.

Testausta voidaankin tarkastella monesta eri näkökulmasta ja on monenlaisia eri tapoja tehdä testausta. Manuaalisessa testauksessa testit ovat yksilöllisiä ja jokainen suoritus vaatii testaajan henkilökohtaista panosta alusta loppuun. Manuaalinen testaus on parhaimmillaan tutkivaa testausta, jossa ohjelmaa käytetään manuaalisesti, tutkitaan miten se toimii ja kokeillaan erilaisia käyttötapoja. Tämän tyyppistä testausta voidaankin kuvata villiksi ja vapaaksi testaukseksi.

Normaali testausautomaation on meidän näkökulmastamme testiskripteihin pohjautuvaa testausta. Siinä testit kirjoitetaan manuaalisesti uudelleen suoritettavien testiskriptien muotoon pohjautuen erilaisiin skriptaukseen soveltuviin työkaluihin. Tämän kaltaista testausta voidaan kuvata ahkeraksi siinä että tietokone suorittaa ahkerasti testiskriptejä niin kauan kuin tietokoneelle sähkövirtaa riittää ja rauta ei rapistu rikkinäiseksi (melko pitkään siis). Tämän kaltainen testaus on myös rajoittunutta siihen mitä skripti tarkasti ohjeistaa, eli se suorittaa testejä uudestaan ja uudestaan, mutta toistaen aina sitä yhtä ja samaa polkua. Uusien ominaisuuksien tullessa käyttöön tai vanhojen muuttuessa, joudutaan käymään kaikki skriptit läpi ja miettimään mitä uusia skriptejä pitää luoda kattamaan uusien ominaisuuksien eri osa-alueet.

Mallipohjainen testaus on meidän näkökulmastamme näiden kahden asian yhdistelmää. Siinä testaaja kuvaa mielessään olevan kokonaismallin tietokoneelle yhtenä kokonaisuutena. Mallipohjainen työkalu ottaa mallin syötteenä ja saa sitten luvan olla villi ja vapaa mallin puitteissa. Eli työkalu käy mallia läpi ahkerasti, kokeillen erilaisia variaatioita ja arvioiden ohjelmiston toiminnallisuutta tätä mallia vasten. Testattavan järjestelmän muuttuessa voidaan mallia päivittää ja pyytää työkalua jälleen luomaan uudet testitapaukset uutta mallia käyttäen. Uusien ominaisuuksien kohdalla, mallia voidaan laajentaa ja pyytää jälleen työkalua käymään sitä läpi ja luomaan uusia testitapauksia. Työkalun tehtäväksi jää sitten kattaa eri puolia järjestelmän halutusta toiminnallisuudesta ja niiden yhdistelmistä. Mallipohjainen testaus helpottaa erityisesti testien ylläpitoa ja on yksi harvoja testiautomaation tapoja joilla voidaan löytää aidosti uusia virheitä. Se sopii hyvin testaukseen ja järjestelmiin missä luodaan erilaisia testisekvenssejä ja niille parametreja. Eli varsin moneen paikkaan.

Kokemuksemme mukaan mallipohjainen testaus toimii parhaiten kokonaisuuksien eri toimintojen ja niiden yhdistelmien testaamisessa. Mallipohjainen testaus esimerkiksi pystyy luomaan suuria testijoukkoja, testien variointia, kattamaan erilaisia yhdistelmiä. Etuna on myös, että työkalua eivät rajoita ihmisen aikaa myöden itselleen luomat olettamukset, vaikka sitä halutessaan voi niilläkin ohjata. Yksi suurimpia etuja kokonaisuudessaan on myös tiedon jakaminen ja olettamusten tekeminen eksplisiittisemmiksi mallintamisen myötä. Lopuksi kaikelle on hyvä muistaa paikkansa ja esimerkiksi tarkimmat yksityiskohdat kehittäjien testata yksikkötesteillään.

Mallipohjaisen testauksen työkalukenttä

Olemme työssämme kokeilleet ja käyttäneet melkoista kirjoa erilaisia testauksen työvälineitä. Mallipohjaista testausta on tutkittu pitkään ja samoin kuin monen muun tutkimusaiheen kohdalla, pitkään on myös ihmetelty miksi sitä ei ole kattavasti otettu teollisuudessa käyttöön.

Meidän näkökulmastamme yksi keskeisimpiä ongelma on että työkalut ovat sisäänpäin kääntyneitä ja ratkaisevat rajatusti juuri sen ongelman jonka sen kehittelijällä on ollut. Mallinnuspuolella yleensä on joku tiukasti rajattu formaatti millä mallit kuvataan, ja joka sopii juuri siihen pieneen maailmaan mitä kehittäjä on itselleen halunnut rakentaa. Tämä vaatii käytännössä uuden ohjelmointikielen opettelun ja rajaa pois mahdollisuuden hyödyntää olemassa olevaa osaamista, sekä olemassa olevia ohjelmisto- ja testikirjastoja hyödyksi mallin kuvaamisessa.

Se mitä näillä rajatuilla mallien kuvauskielillä monesti haetaan on mahdollisuutta rakentaa niiden päälle pitkälle optimoituja automaattisia testien generoinnin algoritmeja. Kun mallin kuvauskieli on rajattu tiukasti omaan muotoon, voidaan analysoida mitä mahdollisia suorituspolkuja sen läpi voidaan keksiä ja yrittää luoda niistä mahdollisimman tarkkaan tiettyihin kriteereihin sopivat, “optimaaliset” testitapaukset. Valitettavasti tämä rajatun kuvauskielen lisäksi asettaa rajoituksia sille miten mallia voi kuvata jotta työkalu vielä ymmärtäisi sen.

Kun itse kokeilimme ottaa erilaisia mallipohjaisen testauksen työkaluja käyttöön, totesimme että ei ollut ihme että mallipohjaista testausta ei ollut laajemmin haluttu ottaa käyttöön. Suurin syy on että se tuli vain aivan liian kalliiksi niin työkalujen hinnan kuin tarvittavan työmääränkin osalta. Mallipohjainen testaus oli kuitenkin mielestämme oiva keksintö ja halusimme jatkaa sen käyttöä ja tarjota mahdollisuuden myös muille hyödyntää mallipohjaisen testauksen vahvoja puolia.

Totesimme että meille hyvä mallipohjainen testaustyökalu mahdollistaisi olemassa olevien taitojen, työkalujen sekä ohjelmisto- ja testikirjastojen uudelleenkäytön. Sen asemesta että se vaatisi meitä miettimään mallia omien rajoitustensa kautta, työkalu tukisi meitä meidän ehdoillamme. Tästä asetelmasta lähdimme tekemään OSMO nimistä mallipohjaisen testauksen työkalua.

OSMO

OSMOn kehityksessä tavoitteena on ollut tehdä mahdollisimman helppokäyttöinen mallipohjaisen testauksen työkalu. Työkalusta on pyritty tekemään helppo tapa tutustua mallipohjaiseen testaukseen mutta myös oikea tuotantokelpoinen työkalu jota voi laajentaa omien tarpeidensa pohjalta. Sen eri osa-alueita kehitetään jatkuvasti käytännön tarpeiden ja tutkimusmaailman tulosten pohjalta.

OSMO mallit ovat rehellisiä sille mitä mallipohjainen testaus pohjimmiltaan on, eivätkä yritä piilottaa sitä. Mallit tehdään puhtaalla Java-kielellä ja tätä kautta se integroituu saumattomasti kaikkiin Java-maailman kirjastoihin. OSMO mallinnus (kuten kaikki mallipohjainen testaus) vaatii ohjelmointitaitoja, mutta jos sen jo osaa niin ei tarvi opetella juurikaan uutta ja alkuun pääsee tosi helposti.

OSMOn mallianalyysin algoritmit eivät ole aivan viimeisintä tutkimusmaailman kärkeä, mutta oleellisimmat algoritmit testien generoimiseen löytyvät. Halutessaan käyttäjä voi myös kirjoittaa omat algoritmit ja kattavuuskriteerit. Näin on mahdollista tarvittaessa luoda algoritmeja ja kriteerejä jotka on optimoitu juuri omalle domainille ja omille testitarpeille.

Jos mallista luodaan testejä myöhemmin suoritettavaksi (mallipohjaisen testauksen terminologiassa offline-testing), voi OSMO myös luoda suuren joukon testejä ja valikoida niistä pienemmän optimoidun joukon käyttäjälle. OSMO kuitenkin tukee myös suoraan testien ajoa mallista (online-testing), joka mahdollistaa suoraan palautteen saamisen mallia kehitettäessä ja testejä ajettaessa. Se mahdollistaa myös ei-deterministisen järjestelmien testauksen kun seuraavia testisteppejä voidaan määritellä edellisen stepin tuloksen mukaan.

Testejä voi siis joko ajaa lennosta tai kirjoittaa haluttuun formaattiin. Malliin voi kirjoittaa suoraan testisteppejä erilaisia Java-testikirjastoja kuten JUnit ja Selenium WebDriver hyödyntäen. Testiskriptejä saa kirjoittamalla testit tiedostoon haluttuun formaattiin sitä mukaa kun OSMO niitä suunnittelee.

OSMO on ladattavissa osoitteesta http://code.google.com/p/osmo/. Hyvä tapa aloittaa tutustuminen on ladata itse ohjelma, tutoriaalit ja esimerkit. Vaadittavaa perusosaamista on ohjelmointitaito, kokonaisuuksien hallinta (mallintaminen), ja kyky ajatella kokonaisuutta testauksen näkökulmasta.

Mallipohjaisen testauksen kustannuksia tyypillisesti kuvataan käppyröillä, joissa alussa kustannukset ovat korkeammat kun pitää saada rakennettua kokonaiskuva järjestelmästä tai sen testattavasta osasta (jota yleensä ei ole oikeasti siihen mennessä kenelläkään), ja saada testausautomaatio siedettävälle tasolla. Sinänsä siis hyviä asioita tehdä muutenkin. Tämän jälkeen kustannukset käppyröillä taasen menevät yleensä nopeasti pitkällä aikavälillä alemmas, koska mallintamisen kautta kattava testaus ja isojen testijoukkojen ylläpito helpottuu.

Käyttöönotto onkin hyvä aloittaa jostain itselle tutusta asiasta. Jos sinulla on esimerkiksi jo olemassa olevia testitapauksia, voit ottaa niistä jonkin pohjaksi ja koittaa mallintaa sitä. Kun saat sen toimimaan, tarkastele mallia ja mieti miten

saisit sitä jalostettua kattamaan suuremman osan ohjelman toiminnallisuudesta. Kokeile välillä generoida testejä ja katso miten ne sopivat kuvaamaan ja testaamaan eri osia järjestelmästä.

OSMO käytössä

OSMOa on koeponnistettu jo useissa oikeissa tuotanto-olosuhteissa. Käytäntö on osoittanut että OSMO:n käyttö ei ole mikään ongelma eikä vie liikaa aikaa pystyttäessä tai ylläpidettäessä. Mallipohjaisen testauksen käyttöönotto edellyttää että testattava rajapinta on hyvä, koska ei huteralle pohjalle saa rakennettua mitään.

OSMO integroituu helposti jatkuvaan integrointiin (continuous integration) ja tuottaa mitä tahansa tarvittavaa formaattia koska formaatti koodataan itse malliin sisään. Myös kehitysympäristönä voi käyttää mitä tahansa Java-ympäristöä ja kaikkia sen ominaisuuksia, koska OSMO on käytännössä vain jar-tiedosto (eli joukko Java luokkia).

Käytännössä OSMO on hyvä työkalu kokeilla sopiiko mallipohjainen testaus omiin tarpeisiisi. OSMO:n saa hyötykäyttöön parissa päivässä jos tarvittava testien ajoympäristö on olemassa. Jos jossain vaiheessa tulee tarve matemaattisesti pätevämmille algoritmeille niin ainahan voi kääntyä kaupallisen työkalun puoleen, jos siltä tuntuu. Siinä vaiheessa toivottavasti on jo käytännön kokemusta mallipohjaisesta testauksesta ja tiedetään mitä halutaan ja tarvitaan, eikä tarvitse ostaa myyntimieheltä sikaa säkissä hienolla puheella varustettuna.

(Paitsi näiltä kahdelta tällä puheella, mutta ainakin hinta on kohdillaan!)

Hyviä mallinnushetkiä

Olli-Pekka Puolitaival, F-Secure

Teemu Kanstren, VTT

Testausta vai automaattisia tarkistuksia

Ville Savolainen

On selvää että testiautomaatio on osa jokaista nykyaikaista IT-projektia. Vähemmän selvää on se mitä testiautomaatiolla tarkoitetaan. Jos termiä lähtee tarkastelemaan, siinä näyttäisi olevan kyse testauksesta ja sen koneellistamisesta. Tehdään siis koneellisesti jotain mitä aiemmin tehtiin käsin.

XBOSOFT testauskonsultit kiteyttävät testiautomaation hyödyt seuraavasti: ”Suorittamalla täsmälleen sama testi joka kerta eliminoidaan riski inhimillisestä virheestä... Kyky ajaa enemmän testejä lyhyemmässä ajassa lisää ohjelmistosi laatua... Automaatiolla saadaan nopeasti tuottoa sijoitukselle ja säästöjä kustannuksiin...”. Voisiko enempää enää toivoa?

Onko yleensä mitään syytä tehdä testausta käsin? Testauksen automatisointi ja sen mukanaan tuomat edut kuulostavat houkuttelevilta. On kuitenkin syytä varmistaa että sanalla ”testaus” tarkoitamme täsmälleen samaa kuin ”testaus” automaatiolla. Oletetaan siis että testaus tarkoittaa molemmissa tapauksessa toimintoja, jotka tehdään tietyssä järjestyksessä. Testauksessa tapahtumat mitä toiminnoista pitäisi seurata, tai ei pitäisi seurata, ovat ennakoitavissa. Vai onko tämä sittenkään testausta?

Tekoälyä odotellessa

Satisfice Inc. yhtiön toimitusjohtaja James Bach ei allekirjoita edellistä määritelmää. Hän kirjoittaa: ”Testaus saattaa vaikuttaa sarjalta toimintoja, mutta hyvä testaus on vuorovaikutteinen ja kognitiivinen prosessi.” Tuo ei kuulosta sellaiselta testiautomaatiolta mitä olen nähnyt, mutta täytyy myöntää että kokemukseni on rajallinen. Ihmiset kyllä pystyvät huomaamaan satoja ongelmakohtia vain vilkaisulla ja erottamaan ne hetkessä harmittomista poikkeamista. Onko tämä todella jotain sellaista mitä voi koneellistaa?

Bachin mukaan testatessa tapahtuva vuorovaikutus on usein monimutkaista, epämääräistä ja altista muutokselle. Näin ollen testaus on prosessi, joka mukautuu vaivattomasti muutokseen ja pystyy tulemaan toimeen monimutkaisuuden kanssa. Tässäkin kohtaa testiautomaatio tuntuisi sopivan huonosti testauksen määritelmään. Kokemukseni perusteella pienienkin järjestelmämuutosten aiheuttamat haasteet testiautomaation ylläpidossa ovat aikaa vieviä ja työläitä.

Kyky toistaa testi täsmälleen samalla tavalla joka kerta, näyttäisi olevan yksi testiautomaation etu. Tosin tämä on vähän kuin kävelisi miinakentässä jonkun toisen jalanjälkiä. Jos tekee sen joka kerta täsmälleen samalla tavalla, pitäisi olla turvassa uusilta miinoilta, tai bugeilta. Tuntuu myös epäilyttävältä väitteeltä että testiautomaatio suojaisi täysin ihmisen tekemältä virheeltä, koska olen itse onnistunut kirjoittamaan virheellisen automaatioskriptin. Seurauksena oli virheettömästi läpi mennyt automaatioskripti.

Testaaminen vs. Tarkistaminen

Onko todella niin, että mahdollisuus ajaa enemmän testejä nopeammin parantaa ohjelmiston laatua? Sekin riippuu siitä mitä testaamisella tarkoitetaan. Testaaja Michael Bolton määrittelee testauksen siten että sen motivaationa on uuden tiedon löytäminen. Tarkistaminen on puolestaan toimintaa, jonka motivaationa on vahvistaa olettamuksia. Jotta automaattisesti suoritettava skripti voidaan tehdä, täytyy sen suorittaminen määritellä ennalta hyvin tarkasti. Tämä puolestaan vaati hyvin tarkkoja olettamuksia, joita skriptillä pyritään vahvistamaan, kerta toisensa jälkeen.

Jos hyväksyy ajatuksen Boltonin määritelmän takana, voisi perustellusti väittää, että testiautomaatiossa on kyse enemmänkin tarkistusautomaatiosta. Tämän takia on vaarallista verrata suoraan tarkistamista ja testaamista keskenään. Voidaan siis sanoa, että siinä ajassa missä voidaan testata viisi toiminnallisuutta, voidaan tarkistaa sata toiminnallisuutta. On kuitenkin mahdollista, että viiden toiminnallisuuden testaaminen tuo enemmän arvokasta tietoa projektille, kuin sadan olettamuksen vahvistaminen. Tämä on syytä ottaa huomioon projektisuunnittelussa.

Ei sinänsä ole yllättävää, jos testaaminen ja tarkistaminen menevät ihmisiltä sekaisin. Testaajat itse puhuvat testauksena toiminnasta, joka selvästikään ei sovi testauksen määritelmiin. Ohjelmoijat ovat kielenkäytössään johdonmukaisempia, eivätkä juurikaan puhu automaattisesta ohjelmoinnista. Puhutaan mieluummin kääntäjistä. Ehkä testaajienkin pitäisi miettiä tarkemmin, milloin puhutaan testaamisesta ja milloin taas automaattisista tarkistuksista.

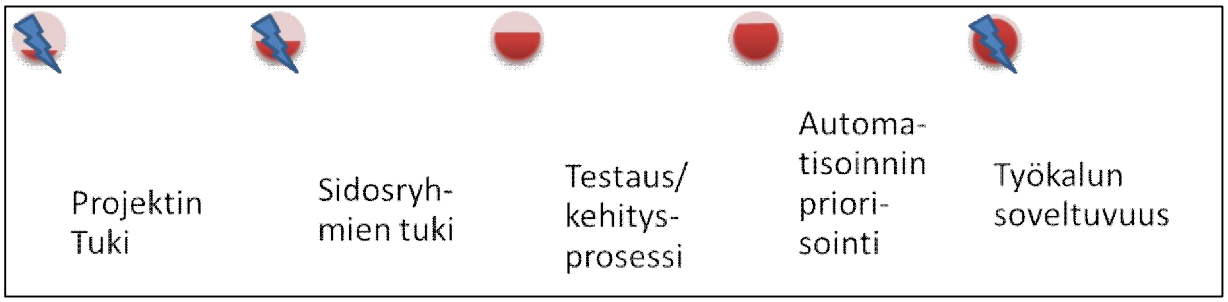
Testauksen automaatio – zombie, vampyyri vai enkeli?

Tuula Pääkkönen

Testauksen automaatio on loistava idea(*). Sillä pystyy helpottamaan a) testiympäristön valmistelua , b) testidatan luomista, c) testitapausten ajoa , ja vaikka vielä d) tulosten jälkikäsittelyä. Tämän artikkelin puitteissa testiautomaatiolla tarkoitetaan kaikkea joka voi testausta auttaa, taustaskripteistä, täysverisiin automaatiotyökaluketjuihin, joten käyttömahdollisuuksia on useita.

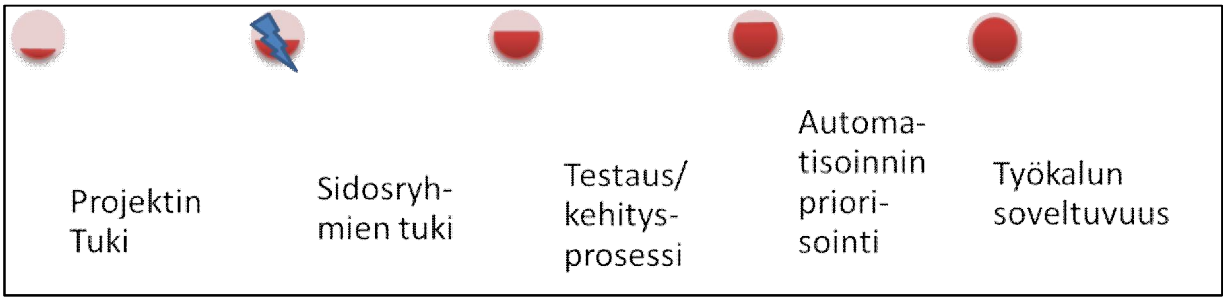
Zombie – elääkö se?

Automaatiota tulee ja automaatiota menee. Innostus uudesta työkalusta (tai menetelmästä) saattaa iskeä zombiarmeijan lailla yllätyksenä. Raa'an työn aikana huomataankin, ettei kaikkea ilmaiseksi saakaan ihan vain työkalun peruskäyttötapauksia seuraamalla, vaan työkalu/menetelmä ja sen paras käyttömalli pitää hallita. Tekeminen hidastuu, kun lähestymisstrategiaa aletaan pohtimaan ja työkalun tehokasta käyttöä mietitään. Eteneminen voi olla myös hortoilevaa – välillä saadaan pikasprintti yhteen suuntaan, ja välillä taas ihmetellään jotakin automaation yksityiskohtaa ja mennäänkin toiseen suuntaan, eikä varsinaista maalia kohti. Projektin ulkopuolinen ei välttämättä tiedä, mitä hiljaisessa automaatioprojektissa tapahtuu, joten viestimistä kannattaa aina yrittää jos sattuma olisi myötämielinen (ref_wiio).



Vampyyri – miksi se vie kaiken?

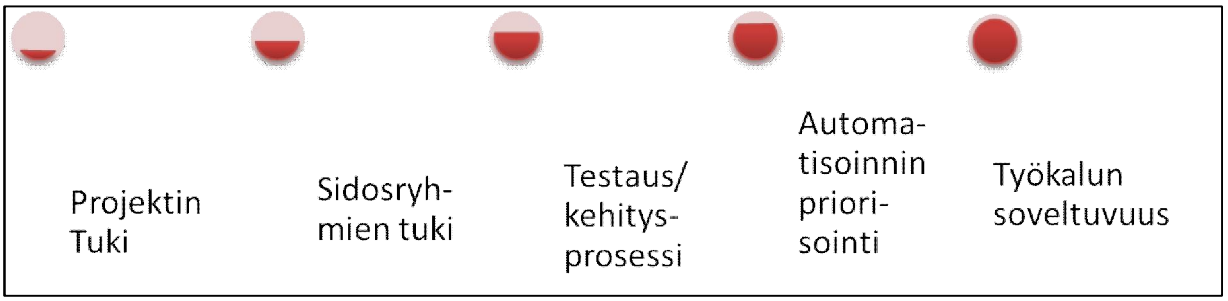
Testauksen automaatio on loistava asia. Pulmaksi usein vain muodostuu se, että tässä kohtaa huomataan: "Oops, automaatiohan on koodia , sitähän pitää ylläpitää, kun tuo testattavakin ohjelmisto on muuttunut" – pulma. Joskus tähän päästään kovinkin äkkiä, joskus voi tämän opin saaminen kestää. Joihinkin juttuihin voi varautua kommunikoimalla kehittäjien kanssa ja/tai jos kehittäjät tekevät myös automaatiota, niin testattavuutta voidaan myös parantaa tarjoamalla automatisoinnille apuvälineitä. Tämä helpottaisi molempia osapuolia.



Automaatiokin usein päätyy hiljaiseksi taustatyöksi. Sitä tehdään osana niitä ”varsinaisia hommia”, jolloin kuitenkin päivittäisessä työssä joutuu tekemään valintoja mitä tekee – parannanko automaatiokirjastoa, vai teenkö uuden testitapauksen asiaan x, vai ympäristöä valvovan skriptin y. Automaatio imee tietoa, vaivaa ja aikaa, joten jotta varsinainen testauksen kohde pysyy kunnossa, vampyyrin pitää myös luovuttaa maagisia voimiaan, löydöksiään , lopputuloksia projektille, jottei käy huonosti. Esimerkkinä vanha periaate, että automaatiota tehtäessä löytyy paljon löydöksiä, jotka kannattaa korjata, eikä tehdä automaatiota niiden ”ympäri”.

Enkeli?

Joskus tilanne on tämä, kaikki osatekijät ovat kunnossa ja tasapainossa ennakko-odotuksiin.



Lyhyemmin sanottuna, kun tiedetään mitä halutaan, ja tehdään se, automaatio toimii. Tällöin sekä testaus- & softankehitysprosessi tukee myös automaatiota, ennaltaehkäisten automaation sivupolut. Yksinkertainen on kaunista, ja perusratkaisua voi aina kehittää eteenpäin.

```
use Test::More tests=> 1;

#happy day scenario
like( $mech->content(), qr/Thank you/, "Got expected content" );
#add test for field2
#add test for FAIL case
```

KUVA 1 THIS IS AUTOMATION...

KUVA 2 THIS IS AUTOMATION... (REF_TEMA)

Jos vielä löytyy hiukan onnea, että muutama ikävä yllätys vältetään... Enkeli testiautomaatiosta voi tulla, kun ennakko-odotukset ovat kohdallaan. Tarvitaanko projektissa arkkienkeli, joka korjaa kaiken holistisena kokonaisratkaisuna? Vai voisiko riittää pienempi suojelusenkeli, joka auttaa testauksessa oikeissa kohdissa, siten että se on oikeasti hyödyllistä? Riittäisikö pieni kokeilu aloittamisvaiheessa (ref_sitra), jota sitten edelleenkehitettäisiin, jos suunta näyttää oikealta? Odotusten ja automaation vastaavien panostusten pitää olla tasapainossa. Näitäkin on – silloin kun hyvin aseteltu skripti säästää käyttäjältään vaikka tunnin viikossa, estää tylsää toistuvaa tekemistä, ennaltaehkäisee virheitä ja varmistaa, että kaikki vaiheet on tehty samalla tavalla.

Testauksen automaatio on loistava idea*.

*) Edellyttäen, että on opittu aiemmista käyttökokemuksista ja tunnetaan oman kontekstin haasteet. Kokeiluihin suostuminen ja kuunteleminen tepsii myös. Yleensä auttaa myös jos on suunnitelma tai edes hahmottelu lopullisesta tavoitteesta, ja millä etapeilla matkaa tavoitteeseen voi mitata. Lisäksi aina kannattaa suhtautua kriittisesti asiantuntijoihin, jotka tarjoavat hopealuoteja. ☺

Viitteet

- Ref_wiio, http://www.cs.tut.fi/~jkorpela/wiio.html
- Ref_sitra, Kokeilukulttuuri , www.youtube.com/watch?v=1YEM2pQQNEc&
- Ref_tema, Mika Katara, MBT-MOSE Seminar slides, 4.12.2008

Automaatiota automaation vuoksi

Juho Saarinen

"Projektin aikana tulee tehdä myös automaattiset regressiotestitapaukset Seleniumilla". "Haluamme automaatioskripti tehtävän työkalulla XXX". "Regressiotestaus on automatisoitava".

Luultavasti tuttuja lauseita monelle joko asiakkaan suusta tai tarjouspyynnöistä. Ja samalla lauseita, jotka herättävät enemmän kysymyksiä kuin vastauksia. Yleensä herää jopa tunne, että lauseen sanoja ei välttämättä edes tiedä mitä juuri sanoi. Automaattinen kuitenkin kuulostaa aina paremmalta kuin manuaalinen (kuten Scrum ja ketteryys kuulostavat paremmilta kuin vesiputous). Automaatiolla asioita voi tehdä tehokkaammin. Toisaalta voi myös epäonnistua tehokkaammin, kuten Knight Capitals osoitti. Sanojan tulee siis tietää mitä haluaa, ja olla valmis myös oppimaan.

Lähestytäänpä asiaa esimerkkien kautta. Suurimmalle osalle lauseiden käyttäjistä se, että selain vilkkuu ruudulla ja asioita tapahtuu automaattisesti on tarpeeksi. Automaatiosta ymmärtävän kysymykset skriptien ajamisesta automaattisesti, tai edes adaptiivisuudesta datan muutoksiin on turha edes puhua. Katsoja haluaa nähdä uusia sivuja aukeavan selaimeen, ei samoja ajoja uudelleen ja uudelleen, eikä kuulla ongelmista mm. datan alustuksen kanssa. Teknisemmällä tasolla keskustelu asiasta kuulostaa katsojan mielestä liikaa ylläpidolta tai koodaukselta. Käytännön tasolla siis työkalulla X nauhoitetaan jokin ketju, johon valitaan osat jotka eivät voi hajota. Tätä pyörittämällä alussa mainittujen lauseiden käyttäjät ovat tyytyväisiä, koska "nyt meillä on automaatio varmistamassa että kaikki toimii". Ja erityisesti, yksikin skripti varmistaa heidän mielestään yleensä kaiken. Tämä oli siis pahin mahdollinen tapaus, jossa automaatiota tehdään, jotta saadaan automaatio. Ei siis varsinaisesti toimivuuden tarkastusta, vaan vain ja ainoastaan automaatio.

Hieman parempi tilanne on, kun automaatiolle on olemassa jo jokin mielikuva, mitä sen pitäisi tehdä. Tämä voi olla esimerkiksi käyttötapauksia tai käyttäjien prosesseja. Ja tilanne vain paranee, jos automaatio halutaan osaksi kehitystä ja parhaassa tilanteessa jopa automaattisesti ajettavaksi yksikkötestien joukkoon. Tai automaattisesti ajettavaksi ajastetuksi, koska eritoten UI-automaatiotapaukset voivat olla hitaita ajaa. Tässä tilanteessa tuloksena on skriptit, joita ylipäätänsä voi ajaa automaattisesti, ja jotka vastaavat oikeita käyttäjien toimia. Käytännössä saadaan kohtalaisen usein katettua ns. "happy case" -tapaukset, joka on hyvä alku, ja yleensä tarpeeksi (UI-)automaatiolle. Manuaalisesti voidaan sitten vain keskittyä hajottamaan järjestelmää, ja varmistamaan toimintaa erikoistilanteissa. Mutta tässä tilanteessa on myös ongelmansa, jotka liittyvät myös ensimmäisiin lauseisiin. Jos automaatio on ostettu "vain automaationa", ja automaatio rakennettu toimittajan työkaluilla, projektin jälkeen skriptien ajaminen voi olla arpapeliä. Samoin ylläpidettävyys, jos tekijällä on ollut käytössään jokin eksoottisempi väline.

Mikä on sitten paras tilanne? Tehdä automaatio sillä tasolla arkkitehtuurissa jonne se kuuluu, ja unohtaa käsitys että "automaattitestaus"=="UI-

testaus”. Käytännössä aletaan täyttämään yksikkötestien ja UI-testien välistä tyhjää tilaa esim. (kooditasoisilla) integraatiotesteillä. Tämä toki vaatii sekä teknistä osaamista, että projektiin osallistumista sen aikana. Erillinen “rakentakaa automaatio” -projekti taistelee asioiden kanssa usein UI-kerroksessa. Ja vielä sellaisten asioiden kanssa, joiden testaus olisi pitänyt hoitaa jo moneen kertaan alemmissa kerroksissa. Kannattaa myös huomioida, että “automaatiotestaaja” ei välttämättä ole se henkilö, joka voi itse skriptit/testit rakentaa. Hän osaa kertoa, mitä testien pitää tehdä, ja jollain tasolla tulkita tekevätkö testit sen. Testit voi sitten kirjoittaa henkilö, joka omaa taidon tuottaa “pseudo-koodisista” automaatiotestitapauksia. Ja mahdollisesti muutenkin teknistä taustaa, jonka avulla voi valita myös parhaan työkalun ko. tarkoitukseen. Ja aina on muistettava, että kaksi silmäparia huomaa asioita paremmin kuin yksi.

Mutta miksi sitten lauseissa on mukana “Projektin aikana tulee tehdä myös automaattiset regressiotestitapaukset Seleniumilla”? Tässähän toteutuvat parhaan tilanteen vaatimukset. Tehdään projektin aikana, ja käyttäen avoimen koodin ohjelmistoja joiden kautta skriptien ylläpito ja ajaminen on mahdollista ilman lisenssien kanssa taistelua. Ongelma onkin, että niin spesifiltä kuin lause myyjän korvaan kuulostaa, se jättää hyvin paljon auki. Lyhyesti Selenium-testejä voi kirjoittaa esim. Selenesella, Javalla, Pythonilla jne. Tai abstraktoida vielä Robot Frameworkin kanssa avainsanapohjaisiksi. Ko. lauseella lopputuloksena on äkkiä Seleniumin IDEllä nauhoitettu polku. Ja testi hajoaa samoin kuin huonoimmassa tapauksessa ollut skripti, kun asiat yhtään muuttuvat.

Automaatiota kannattaa tehdä, mutta sinne mihin se on järkevää. Koko järjestelmän kuviteltu testaus UI-tason automatisoiduilla testeillä ei tätä ole.

Yhteenvedon esittää edesmennyt viisiosaisen trilogian isä, Douglas Adams:

"I (…) am rarely happier than when spending an entire day programming my computer to perform automatically a task that would otherwise take me a good ten seconds to do by hand."

Juho Saarinen on testauksen parissa jo lähes 10 vuotta toiminut testauskonsultti. Töiden ohella aikaa syövät valokuvaus, taekwon-do sekä "pimeänä" puolena koodaus niin Javalla, PHP:llä, Pythonilla kuin Fortranillakin. Tätä nykyä hän työskentelee Sogeti Finland OY:ssa, ja pitää erityisesti teknisestä testauksesta (automaatio, suorituskyky, tietoturva). Myös avoimen koodin sovellukset ovat lähellä hänen sydäntään. Automaatiosta puhuttaessa hän luottaa tätä nykyä hyvin usein Robot Frameworkiin ja sen lisäkirjas-toihin.

Vähemmän napinaa, ohjelmoitavia oraakkeleita

Maaret Pyhäjärvi

Vuosien testauskokemus on tuonut mukanaan enemmän varovaisen kuin positiivisen asenteen testauksen automatisointiin. Kuitenkin lukiessani alkusyksyllä 2012 Cem Kanerin blogiartikkelia oraakkeleista luulen tajunneeni jotain, mitä olen menettänyt keskittäessäni uskoni ja voimavarani pääasiassa älylliseen tutkivaan testaukseen – olen tyytynyt lopettamaan liian aikaisin miettiessäni miten prosessiani voi tehostaa ja miten itse voin kokonaisuutta parhaiten auttaa. Sääntöjä joilla virheitä tunnistetaan pitää voida konkretisoida tasolle, jossa ne tehostavat ja laajentavat testiautomaation käyttöä.

Arvioinnin haaste

Testaamisen keskeisiä haasteita on ohjelmiston toiminnan moniulotteinen arviointi ja mahdollisen virhetilanteen tunnistaminen. Kun virhetilanne ohjelmistoa testaustarkoituksessa käytettäessä tulee kohdalle, kuinka testaaja tunnistaa sen? Miltä osin ja millä mekanismeilla voidaan luoda testiautomaatiota, joka tunnistaa oleellisilta osin oikean käyttäytymisen? Automaatio vähintäänkin helpottaa ihmisvoimin tehtävää testausta säästäen aikaa kun voi rajata huomiotaan kaikesta mahdollisesta asioihin, joita konevoimin on jo voitu etsiä.

Toiminnan puutteiden tunnistamisen ongelmaa kutsutaan oraakkeliongelmaksi. Lähtökohtana on, että testauksen kohteella on moniulotteinen alkutila, testatessa luodaan jonkinlaisia syötteitä ja herätteitä ja moniulotteista lopputilaa tehtyjen toimenpiteiden jälkeen tarkastellaan vertaamalla toteutunutta odotettuun tilanteeseen. Ohjelmistotestauksen oraakkeli on työkalu, joka auttaa päättämään onko ohjelma läpäissyt testin tuottamalla jonkinlaisen odotetun tilanteen johon verrata. Työkalu voi olla ohjelmallinen toteutus, tai se voi olla joukko nyrkkisääntöjä. Oraakkelit ovat heuristisia, ja auttavat tekemään päätöksiä, mutta tuottavat joskus väärän päätöksen.

On harvinaista että yksittäinen oraakkeli, joka vastaa kysymyksiin odotetusta tuloksesta johon verrataan, on käytännössä saavutettavissa tai edes olemassa. Sen sijaan testaajat luottavat osittaisiin oraakkeleihin. Esimerkiksi testaaja saattaa tunnistaa laskennan tuloksen olevan väärä siltä pohjalta että se on suhteettoman suuri vaikka ei tiedäkään tarkkaa tulosta, tai testaaja saattaa tunnistaa ohjelman käyttäytymisen vääräksi vaikka ei tiedä tarkkaan kuinka ohjelman tulisi käyttäytyä.

Oraakkeliymmärryksen monipuolisuus vaikuttaa suoraan testaajan kykyyn tunnistaa ongelmia. Testaaja, joka suorittaa ohjelmiston käytön liikkeet mutta ei tuota tarvittavaa tietoa

ei ole hyödyllinen. Toisaalta, eri ihmiset luontaisesti kiinnittävät huomiota eri asioihin eri järjestyksessä. Esimerkiksi monille on vaikeaa arvioida syvällisesti toiminnallisuutta kun käyttöliittymä on karkea ja täynnä kirjoitusvirheitä. Toiset taas eivät näe kirjoitusvirheitä kuin osoitettaessa. Kyky nähdä asioita monipuolisesti ja arvioida mitä asioita on voinut keskittyä katsomaan tehdyn testauksen puitteissa ovat testaajan perustavaa laatua olevaa osaamista.

Huteja ja vääriä hälytyksiä

Luottaen oraakkeliin saattaa syntyä huti - uskot että ohjelma läpäisi testin vaikka se tekikin jotain väärin - tai väärä hälytys - uskot että ohjelma ei läpäissyt testiä vaikka se toimi tarkoitetusti. Voit väärin perustein päättää että ohjelma läpäisi testin koska sen antamat tulokset vastasivat odotettuja tuloksia, mutta se käyttäytyi väärin jollain muulla tapaa. Esimerkiksi ohjelma joka laskee 2+2=4 on selvästi rikki jos tuloksen aikaansaamiseen kuluu päivä. Voit myös väärin päättää että ohjelma ei läpäissyt testiä koska sen tulokset eivät vastanneet odotettua tulosta, mutta tarkempi tarkastelu paljastaa että et vain tiennyt ohjelman tekevän oikeaa asiaa. Esimerkiksi testatessa tietyn tulosteen tulostamista minuutissa joku toinen lähettää samalle tulostimelle pitkän dokumentin eikä testauksen kohde pääsekään tulostukseen. Monet testaajat, jotka tekevät testausta ihmisvoimin, eivät luultavasti tekisi kumpaakaan virhettä. Mutta automatisoitu testi tekisi molemmat virheet. Samaten sellainen testaaja, joka noudattaa annettuja ohjeita sanasta sanaan eikä käytä harkintaa.

Kumpikaan virhetyypeistä ei ole vältettävissä, sillä kukaan ei pysty täysin määrittämään alkutilaa eikä lopputilaa. Lisäksi, jos lähdet tarkastelemaan tilaa jollain toiminnolla, ko. toiminto muuttaa tilaa, joten tiedät tilanteen vain diagnostiikan ajon jälkeen, et ennen sitä. Oraakkelit ovat siis osittaisia ja heuristisia.

Yhdenmukaisuusheuristiikkoja

Ohjelmien odotetaan käyttäytyvän yleisten sääntöjen puitteissa yhdenmukaisesti. James Bach ja Michael Bolton ovat keränneet joukon hyödyllisiä yhdenmukaisuuden nyrkkisääntöjä, joista poikkeava käyttäytyminen testatessa auttaa tunnistamaan mahdollisen virhetilanteen. Vertailu voi olla tietoista tai tiedostamatonta. Erityisesti ne auttavat selittämään itselle ja sitä kautta muille miksi jokin havainto on oleellinen.

- Yhdenmukaisuus järjestelmän sisäisesti. Toiminto käyttäytyy yhdenmukaisesti muihin saman järjestelmän toimintoihin tai toimintamalleihin verraten. Jos termistö eroaa, vastaavat toiminnallisuudet toimivat eri tavoilla tai näyttävät erilaiselta, näiden erojen nostaminen on useinkin tärkeinä koettuja havaintoja.
- Yhdenmukaisuus vertailtaviin järjestelmiin: Toiminto käyttäytyy yhdenmukaisesti vastaaviin toisten vertailukelpoisten järjestelmien vertailukelpoisten toimintojen kanssa. Jos järjestelmä toimii oleellisesti eri tavalla kuin vastaavat muut järjestelmät, asian tietäminen voi vaikuttaa käyttäjien järjestelmävalintaan. Vertailukohtia voivat olla vaihtoehtoisten järjestelmien lisäksi järjestelmät joissa on ominaisuuksia joita emme halua, tai tietyt vertailukelpoiset

toiminnallisuudet aivan toisen tyyppisistä järjestelmistä.

- Yhdenmukaisuus historiaan. Nykyinen käyttäytyminen vastaa aiempaa käyttäytymistä. Jos jotakin on muuttunut ja kukaan ei ole maininnut muutoksesta, saattaa olla että muutos ei ole tarkoitettu.
- Yhdenmukaisuus imagoon. Käyttäytyminen on yhdenmukaista organisaation haluaman mielikuvan kanssa ulkoisen tai sisäisen asiakkaan ja erilaisten asiakassegmenttien suuntaan. Jos järjestelmässä on piirteitä tai rajoitteita, jotka eivät istu yhteen organisaation itsestään tavoitteleman mielikuvan kanssa, saattaa olla että asiaa halutaan oikaista.
- Yhdenmukaisuus väitteisiin. Käyttäytyminen on yhdenmukaista suhteessa dokumentaatioon, määrittelyihin tai mainoksiin. Jos joku sanoo tai kirjoittaa järjestelmän tekevän jotakin ja se ei sitä tee, asiaan saatetaan kaivata muutosta.
- Yhdenmukaisuus standardeihin ja säädöksiin: Käyttäytyminen on yhdenmukaista ulkoisesti asetettuihin vaatimuksiin.
- Yhdenmukaisuus käyttäjän odotuksiin: Käyttäytyminen on yhdenmukaista suhteessa siihen mitä luulemme käyttäjien haluavan. Käyttäjiä on monenlaisia, ja eri näkökulmista on tarpeen olla tietoinen, esim. loppukäyttäjä ja ylläpitäjä odottavat erilaista toiminnallisuutta ja toimivuutta.
- Yhdenmukaisuus tarkoitukseen: Käyttäytyminen on yhdenmukaista järjestelmän tai toiminnallisuuden käyttötarkoituksen kanssa. Jos ohjelmalla voi tehdä asioita, joita ei pitäisi tai ei voi tehdä asioita joita pitäisi sen tarkoituksen täyttämiseksi, näistä on hyvä tietää.

Englanninkielinen lista hieman eri järjestyksessä muodostaa muistettavan lyhenteen HICCUPPS: History, Image, Comparable products, Claims, User expectations, Purpose, Product, Statutes&Standards.

Ohjelmoitavia oraakkeleita

Yhdenmukaisuusheuristiikat ovat yleisiä ihmisen käyttämiä ongelmaluokitteluja, ja niistä automaatiota tukevien oraakkelien johtamiseen on hieman matkaa. Testiautomaatioon tarvitaan riittävän yksityiskohtaisia ja toteuttamiskelpoisia, silti monipuolisia sääntöjä.

Doug Hoffman ja Cem Kaner ovat keränneet joukon yksityiskohtaisempien ja toteuttamiskelpoisten laajentavaa testiautomaatiota tukevien oraakkelien osalta, jotka auttavat tunnistamaan mahdollisen virhetilanteen.

- Rajoiteoraakkeli. Tarkistetaan mahdottomia arvoja tai suhteita, esim. sähköpostiosoitteen muoto joka ei noudata vakiintuneita rajoitteita ei luultavasti toimi oikein.
- Regressio-oraakkeli. Tarkistetaan nykyisen testin tuloksia suhteessa aiempaan testikertaan samalla testillä järjestelmän aiemmalla versiolla.
- Itsensä tarkistava aineisto. Oikean vastauksen upottaminen testiaineistoon, esim. tarkistussummat.
- Fyysinen malli. Vertailukohta testatessa fyysisen prosessin

ohjelmistosimulointia, esim. painovoiman lait pelihahmon liikkeissä.

- Liiketoimintamalli. Vertailukohta testatessa liiketoimintaprosessia mallintavaa ohjelmistoa, ehdotettujen ratkaisujen ohjelmistossa tulisi vastata mallia.
- Tilastollinen malli. Kertoo että jokin tietty käyttäytyminen tai käyttäytymisten sarja on hyvin epätodennäköinen suhteessa tiettyyn toimintoon. Käyttäytyminen ei ole mahdoton, mutta siihen pitää suhtautua epäillen. Usein hyödyllinen tapa arvioida oikeellisuutta etsiessä kaavoja suuremmista aineistojoukoista.
- Syöte-vastausjoukon tilastollinen oraakkeli. Järjestelmille joissa syötteillä on tunnettuja tilastollisia ominaisuuksia ja näiden ominaisuuksien tulisi vaikuttaa vastaavin säännöin vastausten joukkoon.
- Tilamallit. Määritellään mitä ohjelma tekee vastineena syötteeseen, joka tapahtuu sen ollessa tunnetussa tilassa.
- Vuorovaikutusmallit. Auttavat testaamaan kahden ohjelman välistä vuorovaikutusta määritellen miten ohjelmat käyttäytyvät vastineena toisen tekemiin toimintoihin.
- Laskentaoraakkelit. Auttavat tarkistamaan laskentatuloksia. Esim. vertaaminen toiseen ohjelmaan joka tekee saman laskennan tai laskennan osittaminen varmistaen että osien poistaminen laskennasta tuottaa edelleen johdonmukaisia tuloksia.
- Käänteisoraakkelit. Erikoistapaus laskentaoraakkelista, esim. listan järjestäminen laskevasta kasvavaan ja takaisin laskevaan suuntaan, saadaanko sama lista kuin alkujaan.
- Referenssiohjelma. Ohjelma luo samat vastaukset joukolle syötteitä kuin testattava ohjelmisto. Joiltain osin referenssiohjelman käyttäytyminen eroaa testattavasta tai ne olisivat sama ohjelma.

Tämän tyyppisiä oraakkeleita voi ohjelmoida, ja niitä voi käyttää automaatiossa. Monipuolisten automatisoitavien oraakkelien tarve nousee erityisesti modernissa testiautomaatiossa, jossa ei koiteta toistaa osaa ihmistestaajan aiemmin tekemistä liikkeistä, vaan laajentaa automaatiota etsimään tarkoituksella virheitä joita on työlästä tai mahdotonta etsiä ihmisvoimin. Eri osaoraakkelit tarkistavat eri käyttäytymisiä ja oireita. Tämä auttaa poistamaan tietyn tyyppisiä ongelmia ja suoraviivaistaa prosessia, jossa ihmistä tarvitaan tarkkailemaan ulottuvuuksia joita automaatio ei kata.

Lopuksi

Cem Kaner on huomannut tarkkaillessaan osaavia ja taitavia opiskelijoita kursseillaan, että jostain syystä kuilu yhdenmukaisuus-heuristiikkojen ja ohjelmoitavissa olevien oraakkelien välillä on erityisen haastava kurottava. Tämä tukee omaa havainnointia minun työstäni testaajana. On liian helppoa piiloutua sen taakse, että tunnistan virheitä erilaisiin heuristiikkoihin ja listoihin perustuen, ja jättää ottamatta se askel että testatessaan ajatus mukanaan syventää ajatustaan askeleen automatisoinnin suuntaan. Jää miettimään:

Miten voisin yksityiskohtaisemmin kuvata miten ongelman tunnistaa ja miten sen voisi opettaa tietokoneelle?

Kolumni: Pekka kirjoittaa lehteen

Antti-Pekka Marjamäki

Myydään vähän käytetty testiautomaatio

Taannoin kuulin erään ison päällikön sanovan Jormalle: ”Laitetaan sun tuloskorttiin, että rakennat tiimillesi testausautomaation.” Eli jos mielit boonusta, rakenna testausautomaatio. No Jorma rakensi. Rakensi päivän ja rakensi toisen. Lopulta hän sai automaatioympäristön valmiiksi. ”Olipa urakka”, sanoi Jorma ja ruksasi tuloskortistaan automaation yli. Bonus tuli. Yhtään testiä Jorma tai hänen tiiminsä eivät automatisoineet.

Kyseessä oli siis integraatiotestaukseen tarkoitettu CI-ympäristö, jossa testit piti ajaa joka päivä kertaalleen. Testien tekeminen vaan ko. tuotteelle ko. ympäristössä oli tuskaista hommaa. Testaajat (tai tässä tapauksessa kehittäjät) eivät ottaneet testiautomaatiota omakseen, koska se oli äärimmäisen tuskallinen lisätyö, joka ei helpottanut testausta juurikaan saatikka tuonut sitä stabiliteetin ja varmuuden mielikuvaa, mitä he sillä toivoivat saavansa.

Eikö kuitenkin olisi kiva, jos me saisimme eliminoitua manuaalisen testaamisen automatisoimalla kaikki testaus? Sillä säästäisi rahaa mukavasti ja päästäisiin eroon ylimääräisistä testaajista. Tiedon valtatien varrella on lukuisia tarinoita epäonnistuneista testiautomaatioista, joiden ainoa tarkoitus on korvata manuaalinen testaus. Tarinoita, joiden opetus on valitettavasti usein sama: Testauksen automatisointi on vaikeaa ja se ei korvaa manuaalista testausta.

Miten siis testausautomaatio pitäisi rakentaa? Minun mielestäni pitäisi ensin miettiä ”miksi testausautomaatio pitäisi rakentaa”. Miettimällä ensin, mitä sen automaation pitäisi palvella, voidaan tehdä paljon pieniä tekoja, jotka mahdollisesti johtavat käyttökelpoisen, onnistuneen testausautomaation luontiin. Jokainen joutuu tietenkin itse miettimään, mikä on se ongelma, joka testausautomaation pitäisi ratkaista. Onko automaatio edes oikea ratkaisu siihen ongelmaan vai muuttuuko ongelma palvelemaan ratkaisua?

Tutki siis oman yrityksesi/tiimisi testausautomaatiota. Onko sellaista edes vielä? Jos on, niin mieti seuraavaa: Mikä on se ongelma, jonka se yrittää ratkaista? Onko se oikea ongelma vai onko se ongelma rakennettu tukemaan testausautomaatiota tai oikeuttamaan sen olemassaolo? Testausautomaation pitäisi tuottaa lisää arvoa testaukselle, tai ainakin minun mielestäni. Kun automaatio on valmis, sitä pitää kehittää jatkuvasti, jotta se ratkaisee jatkuvasti muuttuvia ongelmia. Testausautomaatiota tulisi ehkä ajatella irrotettuna testaajan taitona, joka

kaipaa hiomista ja harjoittamista ettei se ”mene pahaksi”.

Mieti, miksi haluat automaatiosi toteuttaa. Mieti, mikä on se ongelma tai ne ongelmat joita halutaan ratkaista. Mieti, onko testausautomaatio edes oikea ratkaisu ongelmaan. Voisi sanoa, että se on olento, joka vaatii huomiota ja huolenpitoa, jotta se antaa arvoa yritykselle. Yksinään se ei ole kovinkaan vahva. Varsinkaan, jos se ei ratkaise niitä ongelmia, joita sen pitäisi ratkaista.

Kun Jorma sitten miettii sitä testausauto-maatiotaan, hän on jumissa sen kanssa. Sitä ei voi myydä, vaikka Mikrobitissä mainostaisi. Sen rakentamiseen pitää laittaa ajatusta ja aikaa. Tämänkertainen Laatu ja testaus -lehti ehkä antaa muutamia ideoita sen tekemiseen hyvin. Ja voin sanoa, että täällä ei ole myynnissä vähänkäytettyjä testausautomaatioita.

Vääriä oletuksia testiautomaatioon liittyen

Maaret Pyhäjärvi James Bachin Test Automation Snake Oil –klassikkoa mukaillen

1. Testaus on ”sarja toimintoja”

Itse asiassa, testaus on vuorovaikutusta, jota rytmittää ohjelmiston arviointi. Vaikka on hyödyllistä tunnistaa sarja toimintoja joita käydään läpi, on myös tärkeää ymmärtää että testauksen jättäminen tälle tasolle luo liian kevyen joukon testejä. Monet vuorovaikutuksista ovat niin monimutkaisia, moniselitteisiä ja muuttuvia että niitä ei voida kuvata järkevästi etukäteen.

2. Testaus tarkoittaa samojen asioiden toistamista kerta toisensa jälkeen

Jos testitapauksella ei alkujaan löydy virhettä, on todennäköistä että sillä ei koskaan löydy virhettä ellei järjestelmään ilmaannu uutta virhettä. Jos testeissä on vaihtelevuutta, kuten yleensä käsin testatessa on, myös jo olemassa olevien virheiden paljastuminen on mahdollista uusien syntyneiden lisäksi. Bach raportoi Borlandin kehityksessä useiden automatisointivuosien jälkeen yli 80 % virheistä löytyneen edelleen käsin tehtävillä testeillä. Erittäin toistettavat testit voivat itse asiassa minimoida mahdollisuuden löytää tärkeitä ongelmia, vähän niin kuin se että kävelisi miinakentällä toisen jalanjäljissä minimoisi mahdollisuuden räjäyttää miina.

3. Voimme automatisoida testauksen toiminnot

Jotkut asiat ovat vaikeita koneelle, mutta helppoja ihmisille. Erityisesti testauksen tulosten arviointi ”onko tämä oikein” on moniulotteista ja kaikkia ulottuvuuksia ei voi automatisoida. Eikä välttämättä edes ajattele ennen kuin niihin törmää käytännössä.

4. Automatisoitu testi on nopeampi koska se ei tarvitse ihmisen puuttumista asiaan

Kaikki automatisoidut testit vaativat ihmisen puuttumista asiaan, ainakin rikkinäisten testien korjaamisessa ja tulosten analysoinnissa – onko virhe testissä vai testattavassa sovelluksessa.

5. Automaatio vähentää ihmisten tekemiä virheitä

Automaatio vähentää joitain virheitä, mutta toisaalta luo toisenlaisia virheitä. Automaatiolla voi päästä eroon virheistä joita ihmiset tekevät kun annetaan tehtäväksi lista yksinkertaisia ja puuduttavia tehtäviä. Ihmisillä on ainakin periaatteessa paremmat mahdollisuudet käyttää järkeään samalla ja ajatella niissä ulottuvuuksissa joita ei ole tullut automatisoitua.

6. Käsin tehtävän ja automatisoidun testauksen kuluja ja hyötyjä voidaan järkevästi verrata

Käsin testaaminen ja automatisoitu testaaminen ovat hyvin erilaisia prosesseja eivätkä kaksi tapaa suorittaa sama prosessi – ihminen kykenee näkemään asioita moniulotteisesti samaa prosessia suorittaessaan ja tämä ero on merkittävä. Niillä on tapana löytää erilaisia virheitä. Niinpä suora vertailu on mahdollista, mutta merkityksetöntä. Tasan samoja testejä tuskin tehtäisiin käsin tai välttämättä olisi oikeasti tarpeenkaan tehdä käsin.

Automaatio on osa moniulotteista hyvää testausstrategiaa.

7. Automaatio johtaa merkittäviin resurssikustannussäästöihin

Automaation kustannus muodostuu automaation kehittämisestä, käyttämisestä, ylläpitämisestä ja automaation aiheuttaminen uusien tehtävien tekemisen kustannuksesta (testitapauksien tarkempi dokumentointi, automaation testaaminen ja dokumentointi, tulosten läpikäynti, muutosten analysointi automaatiovaikutusmielessä, jaetun automaatiotestijakson kehittämisen koordinointi, testien toimivuus useissa ympäristöissä). Jäljelle jää edelleen paljon käsin tehtävää testausta. Automaatiolla ei yleensä vähennetä työtä.

8. Automaatio ei vaikuta heikentävästi testausprojektiin

On vaarallista automatisoida jotain mitä ei ymmärretä. Testausstrategia ja automaation osuus siinä pitää ymmärtää ennen automaatiota, tai tekninen ihmetys saattaa peittää alleen todellisen tavoitteen eli sovelluksen testaamisen. Vanhoja testejä edellisiltä kehittäjiltä ei uskalleta heittää pois, vaan niitä korjaillaan puolisokkona, koska ne saattavat olla oleellisia.