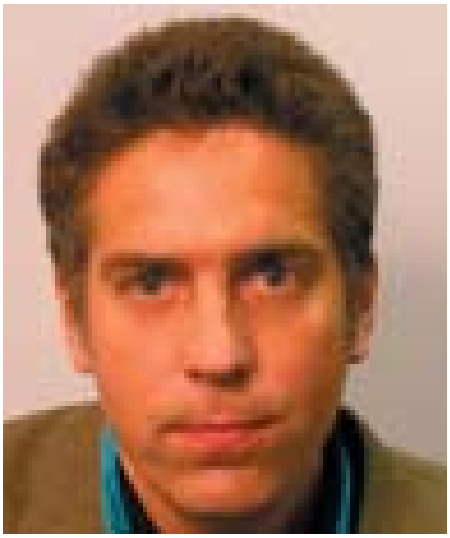
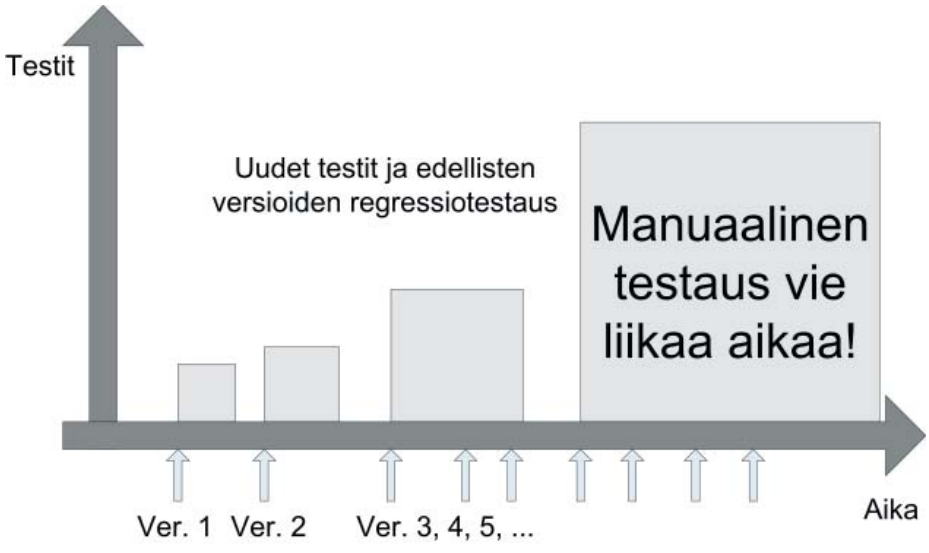


Testauksen automatisointi on haasteellista mutta palkitsevaa

Olli Mensio, IBM Rational Software

Artikkeli keskittyy systeemitestauksen automatisointiin. Yksikkö- ja integrointitestauksessa testaajina toimivat usein miten koodaajat itse, eikä vastaantyyppisiä haasteita automatisoinnin suhteen liiemmin esiinny. Suorituskyky-, stressi- ja kuormitustestaus ovat usein osana systeemitestausta ja niidenkin automatiointi on haasteellista. Tässä artikkelissa keskitytään kuitenkin toiminnallisustestien automatisointiin.

Systeemitestaus on lyhyesti sanottuna uuden tai muokatun järjestelmän toiminnallisuuden ja ominaisuuksien validointia asetettuja vaatimuksia vasten. Miksi testausta olisi mielekästä automatisoida ja mitä automatisointi oikeastaan tarkoittaa? Nykyään vallalla on kehittää järjestelmiä ns. inkrementaalisesti: tuotetaan ensin osa toiminnallisuudesta ja laajennetaan järjestelmää myöhemmin. Tämä on hyvin lähellä käsitteellisesti ns. iteratiivista kehittämistä, mikä niinikään on kehittämispuolella valtavirtaus. Perinteinen ns. vesiputousmallinen kehittämistyö on käytännössä jo lähes historiaa. Perusongelma on esitetty oheisessa kuvassa. Iteratiivisessa ja inkrementaalisessa kehitysprojektissa tulisi testata jokainen tuotettu versio kunkin iteraation lopuksi. Tämän lisäksi tulisi ns. regressiotestata, uudelleentestata, jo aiemmissa versioissa tehty toiminnallisuus ja ominaisuudet. Sillä uutta kehitettäessä ja vanhaa muutettaessa ei välttämättä voida olla varmoja siitä, että vanha, ei koskettu osuus, toimisi edelleen moitteettomati. Tähän ongelmaan on onneksi jo kehitelty apuvälineitä, esim. ohjelmistomoduulien riippuvuusanalyysiohjelmistoja, jotka auttavat kyllä ohjelmoijia huomioimaan riippuvuussuhteita. Nämä eivät kuitenkaan ole normaalisti järjes-



telmä/systeemitestausihmisten työkaluvalikoimassa, joten varmin tapa on uudelleentestata kaikki. Käytännössä testaajan työmäärä lisääntyy jatkuvasti, kunnes se jo projektin loppupuolella on mahdotonta suorittaa manuaalisesti asetettujen aikataulujen ja henkilöresurssimäärien puitteissa. Tämä on perussyy ja -oikeutus automatisoinnille. Olisihan se hienoa, jos jokaisen versiotoimituksen yhteydessä päästäisiin testauksen osalla keskittymään vain muuttuneeseen toiminnallisuuteen, suunnittelemaan niihin uusia testitapauksia, automatisoimaan näitä ja ajamaan jo aiemmissa vaiheissa tehdyt testit automaattisesti. Tästä kertyisi projektille todellisia säästöjä!

Hyvin usein käydessäni asiakkaalla, tai tässä vaiheessa pitäisi kaiketi sanoa prospektilla, totean saman asetelman: testausta pitäisi tehostaa ja halu automatisoida on suuri. Kyvykkyys automatisointiin on kuitenkin monessa tapauksessa olematon. Monta muuta perusongelmaa pitäisi ratkaista ennen tätä. Poikkeuksiakin on matkan varrelle sattunut. Selkeästi fokusoiden hankitun testaustuotteen opiskeluun ja käyttöönottoon on joissakin tapauksissa tuottanut hedelmällisen lopputuleman. Näille tapauksille yhteistä on ollut selkeä halu päästä hyvään lopputulokseen ja tähän valjastetut resurssit ovat olleet erittäin motivoituneita ja sitoutuneita automatisointihankkeeseen. Hankkeetkin ovat lähes aina lähteneetkin liikkeelle ko. henkilöiden ajamina. Organisaatioissa ylhäältäpäin lähtevissä kehittämishankkeissa on sen sijaan törmätty useimmiten liian optimistisiin haavekuviin. Onko testiorganisaatiossa oikean osaamistason omaavia henkilöitä tai edes yhtä? Onko testausprosessit muuten kunnossa, esim. miten testitapauksia hallitaan? Tietävätkö testaajat oikeasti mitä pitäisi testata, eli onko vaatimukset selviä? Tässä joitakin peruskysymyksiä ensi alkuun.

Testauksen automatisointi tarkoittaa melkein aina myös koodausta

Tähän väliin muutama totuus automatisoinnista niillekin tiedoksi, jotka eivät asiaan ole vihkiytyneitä. Automatisointi ei tapahdu automaattisesti. Automatisointityökalut vaativat osaavat käyttäjänsä. Syvällisen osaamisen saavuttaminen merkitsee tähän työkaluun kouluttautumista ja normaalisti käytännössä, tällä tehokkuuden aikakaudellä, tämä tarkoitaa työkalun opettelemista käynnissä olevassa projektissa. Jo tämä saattaa estää automatisoinnin tekemistä, sillä oikeissa projekteissa ei ole aikaa epäonnistumisiin. Epäonnistumiset ja kokeilut ovat kuitenkin oppimisen hedelmiä. Siis jotta automatisointi onnistuisi tulisi tähän varata riittävästi resursseja, jotka eivät alkuvaiheessa suinkaan ole tuottavia. Resurssien tulisi myös olla oikean tyyppisiä. Automatisoinnissa, oli sitten kyse toiminnallisten testien automatisoinnista käyttöliittymätasolla tai suorituskykytestauksen automatisoinnista, olisi eduksi jos henkilöllä olisi koodaustaustaa, eli hän ymmärtäisi ohjelmistokoodauksen alkeet ja pystyisi tekemään joitakin pieniä lisäyksiä automaattisesti generoituun koodirunkoon. Vaikka ohjelmistoedustajat esittelevätkin ratkaisujaan lähinnä nauhoittaen testattavan järjetelmän toimintaa ja toistaen generoituja nauhoituksia, niin todellisuudessa testaaja joutuu hyvin nopeasti muuttamaan nauhoituksia käsin – tämä tarkoittaa koodausta! Ja tästä herääkin ajatus, että eikö olisi hyvä, jos organisaation jo olemassa olevaa koodaajakaartia voisi käyttää silloin tällöin hyväksi myös testauksen automatisointitöissä. Esteenä hyvälle idealle on aika usein erilaiset työkalut ja koodauskieli. Java kehitysalustana on saavuttanut selkeän 'de-facto' –asema. Esimerkiksi Java-pohjainen testaustyökalu on Javaa käyttävissä organisaatioissa ehdottomasti järkevin vaihtoehto ottaen huomioon edellä mainitut resurssiongelmat. Monet toimittajat, mukaan lukien IBM, loiventavat asiakkaidensa oppimiskynnystä tarjoamalla testaustyökalujen käyttöönoton yhteydessä apua projektitasolla. Tämä vähentää oleellisesti epäonnistumisen toteutumisriskiä.

Mitä kaikkea voi ja pitäisi sitten automatisoida. Nyrkkisääntönä voisi sanoa, että ikinä ei voi automatisoida kaikkea. Jos puoletkin testeistä pystyttäisiin automatisoida, niin se olisi jo hyvä saavutus. 100%:n tavoittelua ei kannata edes haaveilla. Aina on löydettävissä testitapauksia, joiden automatisointi vie turhan paljon aikaa tai on jopa mahdotonta, esim. käytettävyystestit. Toisaalta on myös hyvä asia, että automatisoinnin rinnalla tehdään manuaalista testausta. Tällöin ihmiset voivat keskittyä uusien luovuutta vaativien testitapausten suorittamiseen ja kone tekee etukäteen luodut deterministisluonteiset testit, tylsät ja tois-

tuvat testit sekä laajat ja aikaavievät toimintoketjujen läpikäyntitestit. Kone ei vain yksinkertaisesti pysty käyttäytymään kaikissa tilanteissa kuten ihminen sopeutuen. Automatisoinnissa testiympäristön tulee olla erittäin hyvin hallinnassa. Skripti, joka toistaa samaa asiaa kerrasta toiseen, luottaa siihen että sen käyttämää data on mahdollisesti samassa tilassa jokaisella toistokerralla. Kaikki ylimääräinen epädeterministisyys joudutaan koodaamaan scriptiin tavalla tai toisella Tämä hidastaa automatisointityötä. Ympäristön hallintatyön ja skriptien joustavuuden aikaansaamisen vaihtoehtona on joissakin tapauksissa on viisaampaa suorittaa testit manuaalisesti. Manuaalinen testaus ei tarkoita hallitsematonta testausta. Tämäkin voidaan suunnitella etukäteen ja suunnitteluun on saatavilla ohjelmistoja. Näissä kannattaa kiinnittää huomioita pystytäänkö niihin kirjaamaan modulaarisia, yleiskäyttöisiä rutiineja. Tämä edesauttaa mahdolloisesti myöhemmin tapahtuvaa automatisointia.

Vaatimuksista testitapauksiin ja niiden hallintaan

Automatisoinnista puhuttaessa unohdetaan usein testauksen hallinta. Testausta voi pienimuotoisesti automatisoida ilman testauksen hallintatyövälinettäkin, mutta kokemuksesta voin sanoa, että ennemmin tai myöhemmin nämä kokeilut johtavat kaaostilanteeseen ja hyvin vahvaan henkilöriippuvuuteen. Vaikka automatisoidaan, tulisi testitapaukset suunnitella ja hallita keskitetysti. Miten testipäällikkö saa kokonaiskuvan testauksen tilanteesta ilman keskitettyä hallintajärjestelmää, joka sisältää suunnitelmien lisäksi myös manuaalisesti ja automaattisesti suoritettujen testien tulostiedot. Kentällä näkee paljon Excel-monitaitureita, suorittamassa haastavia sarakekirjauksia moninaisten paperilappusten avustuksella. Paljonkohan tähän koontityöhön hukkuu työaikaa? Eikö kuulostaisikin houkuttelevalta, jos testaaja kuittaisi testien suoritukset suoraan keskitettyyn hallintajärjestelmään, ja testipäällikkö saisi samaisen järjestelmän kautta reaaliaikaisesti tiedon testauksen tilanteesta.

Miten testausta mitataan? Esimerkiksi löydettyjen virheiden lukumäärä per testikierros voisi kertoa jotakin testauksen tehokkuudesta. Tämän tyypisten jälkimittarien rinnalle nostaisin ennakkomittarina kattavuusmittarin. Miten hyvin suunnitellut testitapaukset kattavat asetetut vaatimukset. Jos testaus oli asetettujen vaatimusten validointia, niin eikö juuri kattavuutta pitäisi valvoa. Testauksen hallinnan yhtenä ulottuvuutena tulisikin olla mahdollisuus linkittää asetetut järjestelmävaatimukset testitapauksiin ja tätä kautta pystyä generoimaan kattavuusraportointia. Ihanteellisimmassa tapauksessa vaatimukset ovat helposti ja ristiriidattomasti saatavilla ja testaajat saavat myös tiedon muuttuneista vaatimuksista näppärästi testitapauksiensa kautta. Varsin hyödyllistä tämä olisi silloin kun testaajat pääsisivät käsiksi vaatimuksiin, siis mukaan projektiin, jo hyvin varhaisessa vaiheessa, jolloin myös he voisivat antaa oman näkemyksensä asetettujan vaatimusten suhteen, esim. testattavuus- tai automatisoitavuusnäkökulman. Edellä mainitun mahdollistaa keskenään hyvin integroidut vaatimusten-, muutosten ja testauksen hallintajärjestelmät.

Onnistuneen automatisoinnin edellytyksiä

Automatisoinnin ylläpitokustannukset voivat muodostua ongelmaksi. Esim. käyttöliittymätestien automatisoinnissa käyttöliittymien muuttuminen voimakkaasti eri iteraatioiden yhteyksissä saattaa aiheuttaa paljon ylimääräistä ylläpitotyötä. Tämä työ helpottuu testityökalujen käytön kypsyyden kasvaessa. Aiheesta löytyy kirjallisuutta (mm. Software Test Automation - Mark Fewster & Dorothy Graham). Kypsyystasomallissa alkupää on ‘Nauhoita-Toista’-tasolla. Mitä kypsemmälle tasolle mennään, sitä enemmän tekeminen muuttuu ohjelmointipainotteiseksi. Malli ei välttämättä kaikilta osin pidä kuitenkaan paikkaansa, sillä testityökalut ovat myös kehittyneet ja niihin on tullut ominaisuuksia, jotka vähentävät merkittävästi ylläpitotyötä. Normaalisti käyttöliittymätesteissä käyttöliittymäobjektit tunnistetaan näiden sisäisten attribuuttien arvojen avulla. Mikäli attribuuttien arvot muuttuvat testikertojen välillä, niin testaaja joutuu ylläpitotyöhön. Tämä todetaan usein jälkikäteen testiskriptin epäonnistuneen suorituksen kautta. Esimerkiksi Rationalin testityövälineessä on rakennettun ns. ’Script Assurance’-ominaisuus, joka mahdollistaa onnistuneet suoritukset vaikka attribuuttiarvot syystä tai toisesta muuttuisivatkin testikertojen välillä. Lisäksi nykyvälineillä

ns. dataohjatut testit on helppo suorittaa ilman erillisiä ohjelmoituja kirjastorutiineja, tai että dynaamisen datan on helposti määriteltävissä maskit esimerkiksi tarkistuspisteiden yhteydessä. Tämänkaltaiset ominaisuudet pudottavat organisaatoiden testituotteiden käyttöönoton onnistumisen kynnystä, mutta ei kuitenkaan poista kokonaan mainittua kypsyysvaatimusta.

Testidatan hallintaa tulee kiinnittää erityistä huomioita, kuten jo aiemmin tuli mainittua. Ennen automatisointiprojektin alkua tulisi myös varmistua siitä, pystyykö valitulla testityökalulla ylipäätään testaamaan haluttua järjestelmää. Tämä voidaan tehdä esimerkitse pienen käyttöliittymäprototyypin avulla.

Automatisoinnista saa eniten hyötyjä, kun työkalujen käyttöasteet ovat korkealla. Mitä lyhyemmät kehityssyklit, sitä parempi tilanne. Paras tilanne saavutettaisiin, jos testattavana olisi joka päivä uusi versio sillä mitä nopeammin virheet löytyvät, sitä helpompi ne on korjata.

Testauksen automatisoinnissa testiohjelmien ja testidatan hallinnan suhteen pätee lähestulkoon samat säännöt kuin varsinaisessa kehitysprojektissa. Hallittavat elementit tulisi olla dokumentoituja, versioitavissa ja kirjastoitavissa esim. versionhallintajärjestelmän avulla. Työtä helpottaisi suunnattomasti, jos työkalu tukisi kaikkia näitä toiminteita.

Edellytyksien listaa voisi jatkaa, mutta viimeisenä otan esille prosessin tärkeyden. Testitiimin tulisi toimia etukäteen määritellyn testiprosessin mukaisesti. Testiprosessin tulisi olla etupainoinen, jolloin virheet löydettäisiin mahdolisimman nopeasti ja täten niiden korjauskustannukset olisivat mahdollisimman pienet.

Automatisoinnin hyödyt

Automatisoinnin hyödyt ovat ilmeiset, mikäli tässä onnistutaan. Ei ole lainkaan mahdotonta varovaisestikin arvioiden säästää keskikokoisessa projektissa kuukausien ihmistyöt automatisoinnin avulla. Lisäksi testauksen kattavuus tulee nousemaan. IBM:n referensseistä löytyy kertomuksia, joissa esim. regressiotestausaika on lyhentynyt 5 viikosta viikoon (Australian Stock Exchange) ja kattavuus 52%:sta 80%:iin tai 2 kuukauden manuaalinen testaus on lyhentynyt 2:een päivään (ALLTEL) (lähteenä IBM:n asiakasreferenssit). Projektit pyrkivät yleensä pitämään aikataulunsa ja se mistä sitten loppumetreillä lähes aina karsitaan on testaus. Automatisoinnin avulla vastaavaa ongelmaa ei pääse syntymään. Tällöin testaukseen voi käyttää muutakin kuin konttoriaikaa.

Mikäli automatisointi edelleen askarruttaa, kannattaa miettiä hiukan seuraavia kysymyksiä:

- Mikä on testauksemme kattavuus tänä päivänä?
- Mikä kustannus syntyy, mikäli järjestelmästä löytyy virhe tekemättömien testien tai huonon testauksen johdosta?
- Mitä projektille merkitsisi ajaa testit päivittäin?
- Mitä kehittäjille merkitsisi saada palautetta tekemisistään päivittäin?
- Mitä projektille ja koko yritykselle merkitsee saada laadukas tuote ulos aikataulussa tai jopa aiemmin?

Mikä mieluisinta, testauksen automatisointi on myös mielekästä ja useimpien tapaamieni automatisoinnista vastaavien henkilöiden mielestä jopa hauskaa tekemistä. Ja varmasti monipuolisempaa tekemistä, kuin samojen manuaalitestien suorittaminen kerrasta toiseen eri iteraatioissa ja eri ajoympäristöissä. Tällaisen ns. teknisen testaajan rooliin kuuluu monipuolisesti ongelmien selvittelyä ja syventämistä jopa manuaalisesti testaamalla, kuin myös haasteellista automatisointityötä ja näiden hedelmistä nauttimista. Nykyaikaisilla työvälineillä, esim. Java-ympäristössä, testaaja pääsee perehtymään tarkemmin, ellei ole aiemmin vastaavia taitoja jo hankkinut, myös ohjelmoinnin haasteisiin. Kuilu kehittäjien ja testaajien välillä pikku hiljaa häviää, testaajien arvostus organisaatiossa kasvaa - testaus saa arvoisensa aseman!

Kirjoittaja, Olli Mensio, toimii IBM Software Groupissa Suomessa tuoteasinatuntijana vastuullaan mm. testauksen, vaatimustenhallinnan ja muutostenhallinnan tuotteet. Ennen IBM:ää hän toimi vastaavissa tehtävissä Rational Finland Oy:ssä. Lisätietoja IBM tarjoamista sovelluskehityksen ja testauksen ohjelmistoratkaisuista saa internetistä sivuilta http://ww.ibm.com/software/rational/.