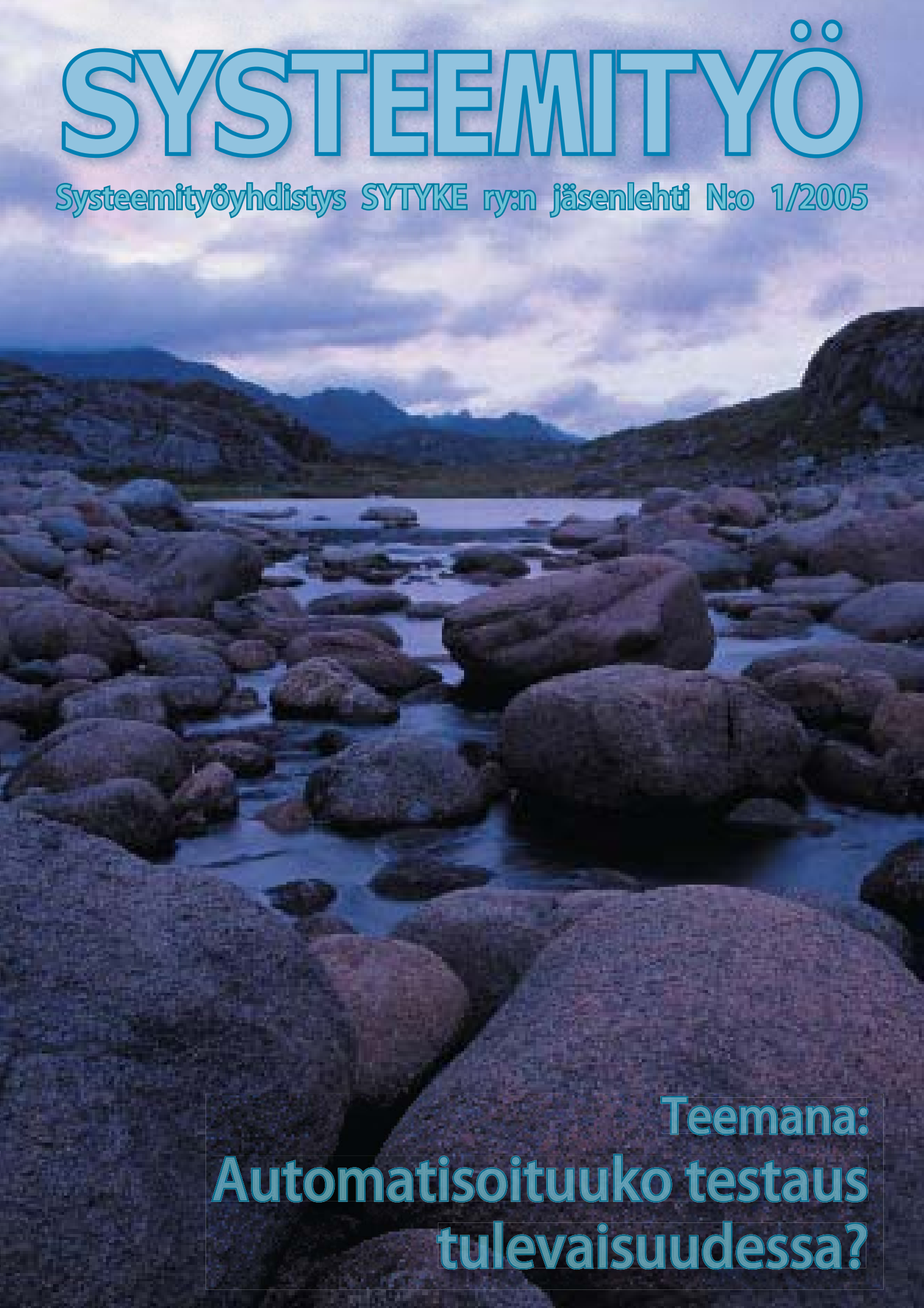
SYSTEEMITYÖ

Systeemityöyhdistys SYTYKE ry:n jäsenlehti N:o 1/2005



Teemana:
Automatisoituuko testaus tulevaisuudessa?

Tiesitkö että 95E bensan hinta vaihteli Suomessa vuonna 2004 runsaasti?

Alimmillaan se oli 0,989 euroa/l ja ylimmillään 1,345 euroa/l.

Tiesitkö että suurin osa kuluttajista teki ostopäätöksiä ja jakeluaseman valintoja jopa yhden sentin/l hintaeron perusteella?

Tiesitkö että ohjelmistojen kunnossapidon hinta vaihteli Suomessa vuonna 2004 runsaasti?

Alimmillaan se oli 5 euroa/fp ja ylimmillään 1500 euroa/fp.

Jokainenhan tietää että litra on tilavuuden mittayksikkö. Kai sinä tiedät että toimintopiste on ohjelmiston laajuuden mittayksikkö (ISO/IEC 14143, 1998). Kai tiedät myös että minkä tahansa ohjelmiston laajuuden voi mitata toimintopisteinä (fp). Myös minkä tahansa ohjelmiston kunnossapito-olosuhteet voidaan arvioida numeerisesti, vertailukelpoisesti. Finnish Software Measurement Association FiSMA ry:n jäsenet ovat alkaneet kerätä tällaisia tietoja kokemustietokantaan jo vuosia sitten. Sinäkin voit nyt tehdä niin.

Tiesitkö että suurin osa kunnossapidon maksajista ei edes vilkaissut hintaa, vaan maksoi kiltisti sen mitä pyydettiin?

Tilaa FiSMAn Ohjelmistotuote todistus.

Tällä 2500 euron[1] investoinnilla säästät saman summan moninkertaisesti heti ensimmäisen vuoden aikana.

Ota yhteyttä ja sovi arvioinnista asiantuntijamme kanssa: www.fisma.fi/scopemanager tai pekka.forselius@sttf.fi

PS. Jos ette ole vielä FiSMAn jäsenenä, mutta ohjelmistotyön mittaaminen ja prosessien parantaminen kiinnostaa, nyt vuoden alussa on hyvä aika liittyä.

[1] Hinta on kiinteä, paitsi jos arvioitava ohjelmisto on poikkeuksellisen laaja, tai jos siitä ei ole käytettävissä kunnollista dokumentistoa. Tässä tapauksessa arvioijamme antaa heti tutkimuksen alussa tarkemman hinta-arvion todistuksen laatimisesta.

Finnish Software FiSMA Measurement Association

Tekniikantie 14 • 02150 Espoo • www.fisma.fi

Julkaisija

Systeemityöyhdistys Sytyke ry
Puhelinvastaus- ja sihteeripalvelu VT Oy
Susanna Koskinen
Henrikintie 7 A, 00370 Helsinki
p. 09-5607 5363
f. 09-5607 5365
sytyke@hennax.fi

Päätoimittaja

Lauri Laitinen, Nokia
lauri.laitinen@nokia.com
puh. 050 483 6551 (NRC)

Toimituskunta 1/2005

Maaret Pyhäjärvi, F-Secure Corporation
Juha Itkonen, Teknillinen korkeakoulu
Kari Kakkonen,
Conformiq Software
Tomi Kaleva, TietoKarhu
Mitro Kivinen, Qentinel
Erkki Pöyhönen, Nokia
Matti Vuori, Plenware

Seuraava numero

2/2005 Projektitoiminta
Toimituskunta:
markku.niemi@sttf.fi
Aineisto: ke 04.05.2005
Ilmestyy: pe 27.05.2005

Lisätietoja lehdestä

http://www.sytyke.org/lehti/

Tilaukset

Systeemityölehti sisältyy yhdistyksen Tietotekniikan liiton suositusten mukaiseen yhdistyksen jäsenmaksuun.
Vuositilaus 20 €
Irtonumerot 5 €
Hyvissä ajoin ennen painatusta tehty vähintään 50 kappaleen lisätilaus 2 €/kpl.
Tilaukset yhdistyksen toimistosta.

Painopaikka

T-Print
Ahokaari 1-3
05460 Hyvinkää
Puh. (019) 475 8500

Painos: 2500 kpl

ISSN 1237-0525
12 vuosikerta, no. 1

Pääkirjoitus

Teema-artikkelit

Yhdistys

Tomi Kaleva, Tietokarhu Oy

Automatisoituuko testaus tulevaisuudessa?

Kädessäsi on vuoden ensimmäinen Systeemityölehti, jonka teemana on testauksen tulevaisuuden visiointi automatisoinnin näkökulmasta. Kaikki kirjoittajat ovat olleet pitkään mukana testauksessa ja sinä aikana sekä nähneet että kokeneet kuinka tuskallisen aikaa vievää on suorittaa testausta käsin eli ns. manuaalityönä. Näitä ajatuksia kirjoittajat esittelevät teille eri näkökulmista.

Jo ennen tietokoneita ihmisen teknisen kehityksen tavoitteena on ollut oma työnsä tehostaminen. Höyrykoneet mahdollistivat saman työn tuottamisen nopeasti ja varmasti, joka olisi lihasvoimalla ollut hidasta tai jopa mahdotonta. Testauksen automatisoinnin lähtökohta on samat tavoitteet. Manuaalitestausta pidetään tylsänä ja liiketaloudellisesti kannattamattomana. Automatisoinnin tulisi nopeuttaa testausta ja samalla koko tuotekehitystä, parantaa laatua tehostamalla testaustyötä, tuoda kustannussäästöjä sekä ennen kaikkea helpottaa testaajien työtä. Testauksen automatisointityövälineitä on ollut markkinoilla kohta kaksi vuosikymmentä ja myytyjä lisenssejä on tuhansia, mutta edelleen konkreettiset onnistumistarinat ovat vähissä.

Työskentelin aikaisemmin testauksen automatisoinnin apuvälineiden maahantuojan palveluksessa. Päivittäin perustelin asiakkaille apuvälinehankintojen järkevyyttä samaan tapaan kuin höyrykoneiden käyttöä perusteltiin aikoinaan. Usein myyjien tapana onkin antaa liian ruusuinen kuva testauksen automatisointivälineiden käytettävyydestä ja kustannussäästöistä. Näin useimmat asiakkaat asettavat itselleen epärealistisia tavoitteita. Osaksi näistä syistä testauksen automatisointi on huonossa maineessa ja tekninen tietotaito vielä lapsenkengissä. Oma näkökulmani on muuttunut, kun siirryin nykyiseen työtehtävääni ja neuvottelupöydän toiselle puolelle myyjästä ostajaksi. Olen huomannut kuinka paljon automatisointi aiheuttaa epäilyksiä ja harhaluuloja. Edelleen johto tavoittelee 90 % automatisointiastetta ja kehittäjät pitävät testauksen automatisointia turhana koodaustyönä. Näin ollen edellytykset onnistuneelle automatisointiprojektille ovat vähissä.

Testauksen automatisoinnin tulevaisuuden näkymiä on esitelty toista kymmentä vuotta testausalan seminaareissa ja pääsanoma on ollut koko ajan, että tulevaisuuden ohjelmistokehitysprojektit eivät tule selviämään pelkällä käsin tehtävällä testauksella. Testausapuvälineiden kehittyvät jatkuvasti ja niistä saavutettavat tekniset hyödyt ovat nopeammin saavutettavissa. Silti ihannemalliin, jossa määrittelyistä luodaan automaattisesti testitapaukset, on vielä pitkä matka. Uskonkin, että automaattinen testaus tulee tulevaisuudessa kasvamaan samassa suhteessa kuin kokonaistestausmäärät kasvavat.

Tomi Kaleva toimii osastopäällikkönä Tietokarhu Oy:ssä. Hän on testauksen osaamisyhteisön ohjausryhmän jäsen.

SCRIBONA

IBM Rational tarjoaa ohjelmistokehitysalustan, jonka avulla ohjelmistokehitysprojekteja voidaan merkittävästi nopeuttaa ja tehostaa. Ratkaisut ovat avoimia ja tukevat alan standardointia.

Rationalin ratkaisun osat ovat parhaat käytännöt, niitä tukevat välineet ja näiden menestyksekkään käyttöönoton varmistavat palvelut. Fortune 100 -listan yrityksistä 98 käyttää IBM Rationalin tuotteita.

Rational® software

Scribona on Suomen johtava IBM-ohjelmistotuotteiden jakelija

Scribona on Pohjoismaiden johtava IT-tuotteiden ja -ratkaisujen toimittaja, joka tarjoaa asiakkailleen korkeatasoista tuoteosaamista, alan johtavan sähköisen kauppapaikan, optimaalisen saatavuuden ja laajan valikoiman täydentäviä palveluita.

Scribonan tuotevalikoimista löydät ***kaikki IBM-ohjelmistotuotteet,*** laajan valikoiman IBM eServer -palvelintuotteita sekä myös muut IBM-tuotteet aina kannettavista tietokoneista järeisiin tallennusratkaisuihin saakka.

Scribona toimii kaikissa Pohjoismaissa ja tekee yhteistyötä yli 6 000 jälleenmyyjän ja järjestelmäintegroijan kanssa.

Suomen Scribona on osa Tukholman pörssissä listattua Scribona-konsernia.

IBM Business Partner

Scribona Oy | PL 148 | 02631 Espoo | Puh. (09) 52 721 | www.scribona.fi

Monenlaista automaatiota

Maaret Pyhäjärvi, F-Secure Corporation
TestausOSY:n vetäjä

Testausautomaatio on laaja käsite testauksen tukemisen välineistä testiautomaatioon. Tässä artikkelissa käydään läpi automaation vaihtoehtoja niin prosessin kuin testisuorituksen automatisoinnin osalta. Tiedätkö mihin aika testauksessa kuluu ja minkä tehostamiseen automaatio voisi purra?

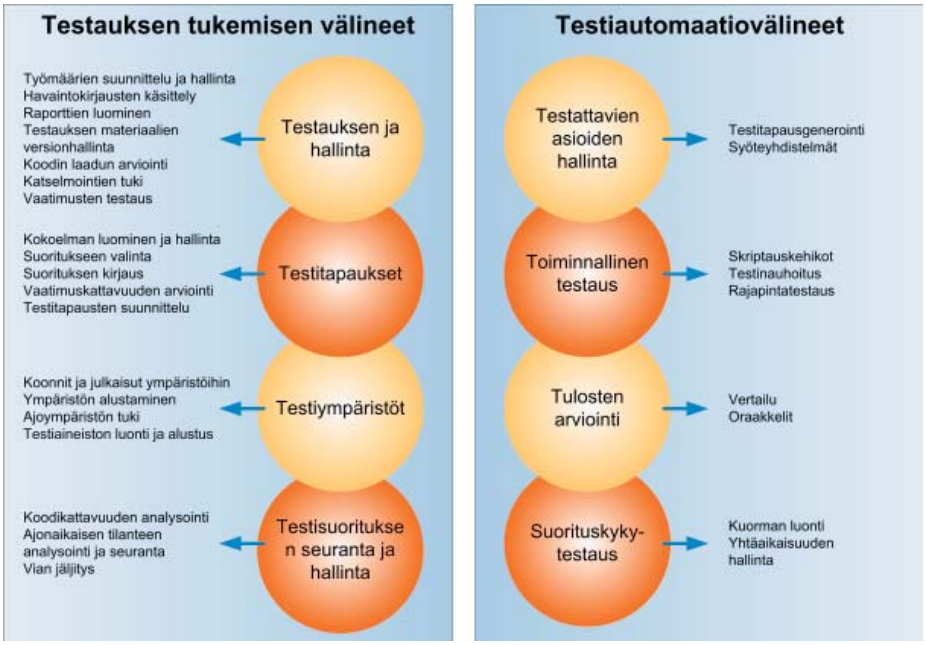
Testauksessa kokonaisuutena on paljon tehtävää. Testaustehtävät haukkaavat merkittävän osan ohjelmistokehityksen kokonaiskustannuksista. Erityisesti toteutuksen loppuvaiheessa testaus on kriittisellä polulla järjestelmän julkaisuun. Halu nopeuttaa ja tehostaa toimintaa erilaisin välinein on luonnollinen suuntaus. Houkuttelevia vaihtoehtoja ovat sekä testauksen suorituksen automatisointi että yleensä välineiden integrointi tukemaan kokonaisuutta paremmin.

Monenlaista automaatiota

Erilaisia välineitä testauksen tukemiseen on tarjolla monipuolisesti. Voi olla myös vaikeaa rajata mikä väline on testausväline – tämä riippuu siitä mitä testaukseen lasketaan.

Korkeimmalla tasolla testauksen välineet voidaan jakaa karkeasti kahteen luokkaan:

- **Testauksen tukemisen välineet** – Välineet, joiden käytöt tukevat sekä käsin suoritettavia että automatisoituja testejä ja joiden käyttöön ei liity oletusta käsin testaamiselta välttymi-



Testausvälineiden käyttöpohjainen ryhmittely

sestä. Testauksen tukemisen välineet ovat testauksen prosessiautomaatiota.

- **Testiautomaatiovälineet** – Välineet, joilla suoritetaan testausta tietokoneavusteisesti. Suoritusautomaatio mielletään varsinaisena testauksen automatisointina.

Nämä kaksi välineluokkaa osaltaan jakaantuvat vielä kumpikin neljään aliluokkaan. Kukin aliluokka sisältää vielä osaltaan tarkemmin määriteltyjä välineluokkia käyttöperusteiseen jaotteluun perustuen. Yksittäinen todellinen testausväline voi sisältää useita testaukselisia käyttötarpeita.

Testauksen tukemisen välineet

Testauksen tukemisen välineet eivät ole automaatiota perinteisimmässä mielessä. Testitapausten suorittamisen asemesta nämä tukemisen välineet on suunnattu tehostamaan ja tukemaan testausprosessia kokonaisuutena.

Testauksen suunnitteluun ja hallintaan liittyviin välineisiin kuuluu erilaisia testauksen projektinhallinnan tukemisen välineitä. Perinteisesti testauksen hallinnan välineiksi kutsutuista välineistä useat tarjoavat osin tukea tällä alueella. Kuitenkin tyypillisesti näiden perinteisten testauksen hallinnan välineiden painopiste on testitapauksissa. On tärkeää erottaa testauksen suunnittelu ja testien suunnittelu toisistaan. Testauksen suunnittelu tarjoaa tukirakenteen testauksen järjestäytymiselle, kun taas testien suunnittelu menee testauksen ytimeen ja yksityiskohtiin.

Testitapauksiin liittyvät välineet sisältävät erilaisia välineitä testitapausten luomiseen, hallinnointiin ja valintaan. Testitapausten käsittelyyn liittyy oleellisesti sekä staattinen (varastointiin liittyvä) ja dynaaminen (suoritukseen liittyvä) ulottuvuus. Testitapausten käsittely ja valinta on testauksen suunnittelun ja hallinnan keskeisimpiä käsitteitä.

Testiympäristöihin liittyviin välineisiin kuuluu erilaiset ympäristöjen pystyttämiseen ja hallintaan keskittyvät välineet. Dynaaminen testaus tapahtuu aina testiympäristössä ja ympäristötekijät vaikuttavat osaltaan testien tuloksiin. Usein testattavana on erilaisia muunnelmia ympäristöstä ja ympäristön tilan hallitseminen voi olla yksi testauksen työläimpiä osia.

Testien ajonaikaisen seurannan ja hallinnan välineet sisältävät tukea testien ajamiseen ja ajonaikaisten tilanteiden diagnosointiin. Usein on vaikea tietää mitä osia testit järjestelmästä itse asiassa ovat kattaneet tai onko järjestelmässä virhe. Kaikki virheet kun eivät ilmene itsestään selvästi.

Testiautomaatiovälineet

Testiautomaatiovälineisiin lasketaan kuuluvaksi erilaiset välineet, jotka automatisoivat osan perinteisesti käsin tehtävästä testaustyöstä tai mahdollistavat testaustyyppejä, jotka käsin eivät olisi saavutettavissa. Yleisesti testauksen tukemisen välineitä yhdistetään näihin testiautomaatiovälineisiin, tehostamaan virhetilanteiden diagnosointia ja yleensäkin testauksen hallittavuutta.

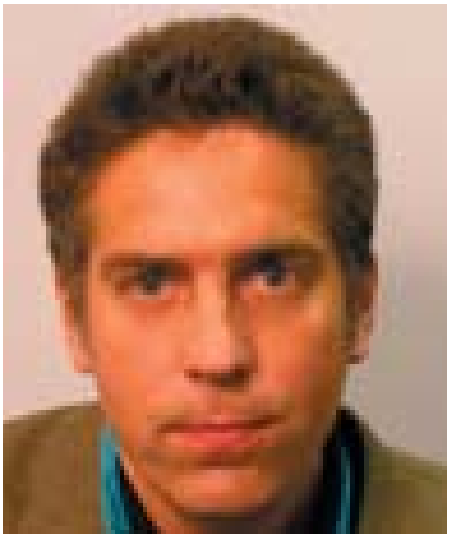
Testattavien asioiden valinnan välineiden tarkoituksena on automatisoida yksityiskohtaisten ja suoritettavissa olevien testitapausten luomiseen liittyviä asioita.

Toiminnallisten testien ajamisen välineet ovat testiautomaatiota perinteisimmillään. Ne tarjoavat keinot käyttää ohjelmistoa automatisoidusti. Niiden jaottelussa kiinnittymiskohta tarjoaa luonnollisimman jaon. Testata voi niin graafisen käyttöliittymän, kuin ohjelmointirajapintojen kautta. Testejä voidaan suorittaa monessa vaiheessa ja monella tasolla.

Testien tuloksen arvioinnin välineet mahdollistavat testin läpimenon automatisoidun arvottamisen. Tuloksen arviointi on erityisen haastava alue niin käsin kuin automatisoidustikin testatessa.

Suorituskykytestien ajamisen välineet muistuttavat monelta osin toiminnallisten testien ajamisen välineitä. Perusajatuksena ajettava testiskripti yhdistää molempia välineitä, mutta tyypillisesti suorituskyvyn testaamiseen liittyy lisäksi yhtäaikaisuuden luomista sekä yhtäaikaisuuden tarkempaa hallintaa.

Maaret Pyhäjärvi toimii testauksen parissa F-Securella ja vetää Systeemityöyhdistyksen testauksen osaamisyhteisöä.

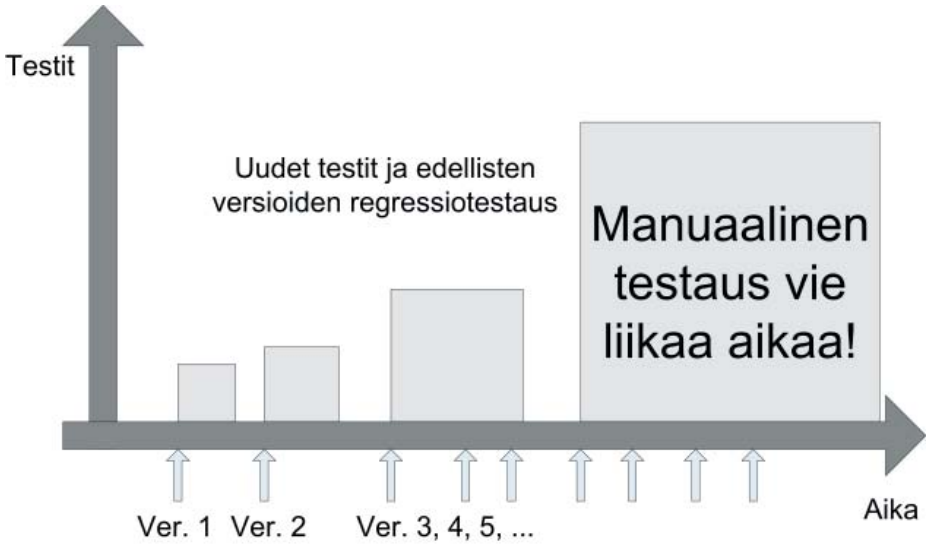


Testauksen automatisointi on haasteellista mutta palkitsevaa

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/.

Testausautomaation kehittäminen ja kehittyminen

Mitro Kivinen, Qentinel Oy

Kirjoitin testauksen osaamisesta IT-Viikon Sytyke-liitteeseen joulukuussa 2004. Tarkkasilmäisimmät kyselivät, missä testausautomaatio on. Vastasin silloin kyselijöille, että testausautomaatiota varten pitää hallita kaikki samat asiat kuin testaustakin varten. Tätä asiaa pitää varmasti täsmentää ja käynkin tässä artikkelissa asiaa tarkemmin läpi.

Testausosaamisen suhteen on mahdotonta sanoa lopullista totuutta, sillä osaamistarpeet vaihtelevat sovelluksen ja sen testaamiseen sopivan automaation mukanaan tuomista vaatimuksista. Automaatio ei suinkaan ole itsestäänselvä ratkaisu kaikkiin testauksen ongelmiin ja sopivan automaatiotason valinta tai automatisoimatta jättäminen vasta osaamista vaatiikin.

Perusongelmana moniulotteisuus

Usein organisaatiossa löytyy monia erilaisia testausautomaation ilmentymiä: on nauhoittimia, työkaluja ja vielä erilaisia skriptejä eri puolilla. Kokonaisvaltainen testiautomaatio strategia puuttuu, mikä johtuu siitä, että organisaatioilla ei ole käytössään riittävästi testausautomaation toimintamallien laaja-alaista osaamista. Tämän vuoksi automaatio ei onnistu ja sen lupaukset jäävät lunastamatta.

> “Onnistuakseen automatisointi vaati hyvää testausosaamista”

Odotuksetkin ovat usein epärealistisia: testausautomaatio nähdään liian helposti kaikenratkaisevana asiana eikä välttämättä ymmärretä, että testausautomaatio voi ratkaista vain osan testaustarpeesta, ei kaikkea kaikilla tasoilla.

Mistä on kysymys

Automatisointi tulee mukaan kuvioon silloin, kun samaa asiaa halutaan toistaa useita kertoja samanlaisena. Jopa näin perustavanlaatuisen totuuden ymmärtäminen voi olla vaikeaa. Automatisointia lähdetään kehit-

Onneksi kehittäjät itse haluavat päästä helpolla ja automatisoivat oman testaustyönsä jopa muiden sitä edes tajuamatta. Kokonaisuutena automaatiokenttä on erittäin haasteellinen hallittava.

tämään miettimättä sitä, mitä halutaan toistaa monta kertaa ja miksi. Lisäksi toistamisen hyötyjen tulee olla suurempia kuin automatisoinnin tuottamat kustannukset.

Toisaalta automatisointi mahdollistaa sellaisia asioita, joita olisi muin tavoin tehtynä mahdotonta toteuttaa millään järkevällä työmäärällä annetussa ajassa. Esimerkeiksi käyvät liikenteen tuottaminen puhelinverkkoon tai miljoonien rivien pohjadatan luominen testauksen tietokantaan.

minen tulee vasta, kun automatisointi on rakennettu ja testiskriptit kirjoitettu.

Manuaalinen testaaminen on käsityötä ja ero on kuin vertaisi tehdasta ja käsityöläisverstasta. Tehdas ei kuitenkaan synny ilman valtavia investointeja. Investointi ja tehtaan rakentaminenkin pitää osata. Samoin on automaation luomisen laita: kyseessä on haastava ja monitahoinen ohjelmankehitysprojekti.

“Pitää tietää mitä halutaan toistaa paljon”

Pienessä mittakaavassa tämä on helppoa ja intuitiivista. Kun samaa komentoketjua antaa Unixissa kolmatta kertaa, tulee asian jo samantien skriptanneeksi. Eipä tarvitse enää näpytellä komentoja ja kas - automaatiota syntyi lähes vahingossa.

Ohjelmien monimutkaistumisen ja kehittämisen haasteet vaativat entistä tehokkaampia tapoja testata laajempia kokonaisuuksia. Tätä haastetta on mahdotonta voittaa vain lisäämällä manuaalista testausta. Siksi automatisoinnin suuntaan kuikuillaan aina uudelleen.

Automatisoinnissa tarvittava osaaminen

Teollisuusautomatisointi on ensisijaisesti erilaisten robottien rakentamista ja niiden ohjelmointia. Sama pätee testaukseen: testaa-

Ohjelmointi- ja skriptaustaidot ovat ehdottoman olennaisia automatisointiprojektissa. Kaikkien ei kuitenkaan tarvitse osata ohjelmointia. Lisäksi ohjelmointitapa riippuu valittavasta välineestä. Keveimmillään ohjelmointi hoituu nauhoittamalla käyttöliittymässä tehtävät hiiren klikkaukset ja syöttökenttiin kirjoitettavat tekstit. Tämän jälkeen nauhoitusrobotti suorittaa annettua toimintoa vaikka loputtomiin. Eri asia on se, kuinka pitkälle tästä on hyötyä. Nauhoitustyökalujen käyttö tehostuu heti, kun mietitään etukäteen, mitä kannattaa nauhoittaa ja toistaa ja nauhoitettuja skriptejä muokataan sen mukaisesti.

Raskaimmillaan kehitetään laaja-alainen automaatioalusta, jonka rakentamiseksi tarvitaan useita erilaisia työkaluja. Olennaisena onnistumisen edellytyksenä on rakenteellisen ohjelmoinnin osaaminen.

Onnistuakseen automatisointi vaatii hyvää testausosaamista. Sitä ei välttämättä tarvitse olla kaikilla automatisointiin osallistuvilla, mutta projektissa tätä osaamista on oltava.

Toki käytettävien välineiden perustarkoitus ja työkalujen tuomat mahdollisuudet pitää osata hyvin. Jokaisella työkalutyypillä on lisäksi omat käyttökohteensa eivätkä ne sovellu kaikkeen. Kaikki palaa alkuperäiseen asetelmaan: pitää tietää, mitä halutaan toistaa paljon.

Testausympäristö vaikuttaa automatisointiin samalla lailla kuin ympäristön pystyttämiseen ja tulosten analysointiin. Automatisointi lähteekin usein käyttöjärjestelmäkomentojen skriptaamisesta. Toinen tyypillinen aloitustapa on skriptata tietokannan datan syöttö ja tulosten tarkistus.

Teknisissä rooleissa testaajan on syytä kehittää yleistä järjestelmäarkkitehtuurien tuntemustaan ja ohjelmointiosaamistaan. Automaation kehittäjät eivät välttämättä ole sovelluksen kohdealueen asiantuntijoita eivätkä toiminnan asiantuntijat välttämättä ymmärrä kaikkia tekniikan hienouksia. Molempia näkökulmia tarvitaan onnistuvan testauksen ja sen automatisoinnin luomiseksi ja siksi testauksesta tulee tiimityötä.

Automaatiota voi lähestyä orgaanisesti pikkuhiljaa kehittämällä tai

suurena projektina. Molemmissa tarvitaan erilaisia osaamisia.

Automaation tilanne tänään

Laajimmin levinneitä testausautomaation ilmentymiä ovat yrityskohtaiset skriptit. Myös erilaiset käyttöliittymän toiminnan nauhoitustyökalut ovat laajalle levinneitä. Ne ovat helpoimmin lähestyttäviä ja suhteellinen hyöty saavutetaan nopeasti. Niiden avulla toteutetaankin valtaosa massatesteistä.

Laajoja automaatioalustoja on vain harvoilla. Onnistuakseen laaja automaatio vaatii organisaatiolta tietynlaista tapaa toimia. Sovelluksen arkkitehtuuri ratkaisee paljon: helpoimmin onnistutaan silloin, kun ohjelman sisäiset ja ulkoiset rajapinnat ovat harkittuja ja vakaita. Automaatio ei onnistu useissa vanhoissa järjestelmissä niissä olevien sekavien rajapintojen takia.

	Testaaja	Automatisoija
Kohteen ja sen toiminnan tuntemus	Välttämätön	Hyödyllinen
Kohteen tuottaman tuloksen analysointitaito	Välttämätön	Välttämätön
Luovuus löytää uutta testattavaa	Välttämätön	Hyödyllinen
Luovuus löytää uusia tapoja testata,	Välttämätön	Välttämätön
Järjestelmällisyys käydä ahkerasti läpi kaikki erilaiset tavat testata	Välttämätön	Hyödyllinen
Priorisointitaito	Välttämätön	Välttämätön
Ohjelmointitaito	Hyödyllinen	Välttämätön
Tietotekniikka-arkkitehtuurit	Hyödyllinen	Välttämätön
Käyttöjärjestelmät	Hyödyllinen	Välttämätön
Tietokannat	Hyödyllinen	Välttämätön
Tietorakenteet ja algoritmit	Hyödyllinen	Välttämätön

Testaajan ja automatisoijan osaamisten erot

Automaation rooli tulevaisuudessa

Matalan tason testausten automatisointi lisääntyy, kun testivetoisen kehittämisen edut alkavat valjeta nykyistä laajemmalle yleisölle. Tämä parantaa ohjelmistojen laatua kautta linjan, mutta se ei yksin riitä.

”Iso osa testaamisesta jää edelleen ihmisten tehtäväksi”

Ketterien kehitysmenetelmien rinnalla on syntynyt iso joukko näitä tukevia kehittäjien työkaluja. Esimerkiksi xUnit –työkalut mahdollistavat aiempaa helpomman yksikkötasoisen testausautomaation. Samoin kehittyneet työkalut mahdollistavat automaattisen jatkuvan integroinnin. Molempia näkee käytössä yhä useammin.

Ohjelmistojen arkkitehtuuria rakennetaan suoraan automatisoitavaksi. Tämä mahdollistaa laajempien ja parhaimmillaan helppokäyttöisempien automaatioalustojen luomisen. Automaatioalustan rakentaminen on kuitenkin suuri investointi, eikä sitä useinkaan voida tehdä. Arkkitehtuurin selkiytyminen on kuitenkin hyvä asia ilman automatisointiakin.

Sovellusten ja järjestelmien entistä laajempi keskinäinen yhteistoiminta asettaa uusia vaatimuksia rajapintojen kuvaamiselle. Tämä tuo eväitä myös automaation toteuttamiselle.

Iso osa testaamisesta jää edelleen ihmisten tehtäväksi. Tutkivaa testausta ei voi luonteensa vuoksi automatisoida, eikä puutteellisesti dokumentoidulle järjestelmälle voi rakentaa laajaa automaatiota. Luovuuttakaan ei voi jättää koneiden tehtäväksi.

Valmistuvan ohjelman testaava käyttäminen paljastaa usein sellaisia asioita, joita ei testiä suunnitellessa osattu arvata. Tätä ei voi nyt eikä ihan pian automatisoida!

Mitro Kivinen toimii testausasiantuntijana Qentinel Oy:ssä ja on mukana Tesstauksen osaamisyhteisön ohjausryhmässä.

Qentinel on riippumaton ohjelmistojen laadunvarmistuspalveluja tarjoava yritys.

Testauksen työvälineet nyt ja tulevaisuudessa - akateeminen näkemys

Mika Katara, Tampereen teknillinen yliopisto

Testauksen arvostus yliopistomaailmassa on perinteisesti ollut varsin heikko. Tämä on näkynyt siten, että testauskoulutukseen ei ole panostettu läheskään yhtä paljon kuin esimerkiksi ohjelmointikursseihin. Tutkimusta testauksen alalla on jonkin verran tehty, mutta tutkimustulosten hyödyntäminen on suurelta osin vasta edessäpäin.

Testauksen osuus ohjelmistoprojektien kustannuksista on merkittävä. Täytyy siis kysyä, miksi yliopistot eivät ole panostaneet samassa suhteessa testauksen koulutukseen ja tutkimukseen. Taustalla on varmasti osaltaan testauksen heikko arvostus yleensäkin. Kenties on lähdetty myös siitä idealistisesta olettamuksesta, että testaus on vain väliaikainen ilmiö; uudet ohjelmistojen kehitysmenetelmät tulevat takaamaan sen, ettei virheitä synny eikä näin testaustakaan merkittävissä määrin tarvita. Tähän mennessä tällaiset kehitysmenetelmät eivät kuitenkaan ole osoittautuneet kovinkaan käytännöllisiksi ja kustannustehokkaiksi. Niin kauan kuin ohjelmistoja tekevät ihmiset, virheitä ilmenee ja siten myös testausta tarvitaan. Iteratiivisten prosessien myötä testauksen tarve tulee vain korostumaan.

”Tulevaisuudessa on selvästi nähtävissä erilaisten apuvälineiden kasvava rooli”

Järjestimme Tampereen teknillisellä yliopistolla (TTY) toukokuussa 2004 testauskoulutusta käsittelevän työpajan. Siihen osallistui testauksen opettajia lähes kaikista testauskoulutusta tarjoavista Suomen yliopistoista. Intoa koulutuksen tason parantamiseen selvästi on. Monesti koulutus ja tutkimus käyvät käsi kädessä, joten tämä voi tarkoittaa myös suomalaisen testaustutkimuksen nousua tulevaisuudessa.

Automaattinen testaus on erityisen mielenkiinnon kohteena yliopistomaailmasta katsottuna. Se tarjoaa mielenkiintoisia tutkimuksellisia haasteita ja toisaalta kiinnostaa opiskelijoita, jotka tulevat testauskursseille opittuaan ohjelmointitaitoja muilla kursseilla. Vaikka manuaalinen testitapausten suorittaminen on edelleen merkittävässä roolissa tehtä-

essä tietyntyyppistä testausta, on tulevaisuudessa selvästi nähtävissä erilaisten apuvälineiden kasvava rooli. Näiden apuvälineiden avulla tullaan automatisoimaan tiettyjä työvaiheita, joita on ennen tehty käsin. Voidaan myös tehdä asioita, joita ei edes pystyttäisi tekemään manuaalisesti; koodikattavuuden mittaaminen on tästä ollut hyvä esimerkki. Kokonaan automaattiseksi testaus ei kuitenkaan muutu. Hyvä testaaja pystyy asettumaan loppukäyttäjän asemaan ja tarkastamaan hänen vaatimustensa toteutumisen paremmin kuin kone ikinä pystyy.

Tässä artikkelissa tarkastelen testauksen ja testiautomaation tulevaisuutta yliopisto-opettajan ja –tutkijan näkökulmasta. Ensin käsittelen testaustutkimusta niin kutsutun mallipohjaisen automaation kannalta ja sitten työkalujen asettamia haasteita testauskoulutukselle.

Testaustutkimus

Siinä missä 1990-luvulla testitapausten automaattiseen suorittamiseen panostettiin voimakkaasti, 2000-luvun teemana testauksen suhteen näyttäisi olevan mallipohjainen testaus. Mallipohjaista testausta käsitellään tässä lehdessä laajemminkin. Sen ideana on, että testitapausten sijaan testaaja laatii testimallin, josta voidaan automaattisesti generoida ääretön määrä erilaisia testitapauksia. Testitapausten laatuun vaikuttaa testimallin lisäksi myös generointiin käytettävät heuristiikat.

”2000-luvun teemana testauksen suhteen näyttäisi olevan mallipohjainen testaus”

Perinteinen testien suorittamisen mahdollistava automaatio löytää harvoin uusia virheitä. Mallipohjaisen lähestymistavan etuna on se, että testimalleista generoidut testitapaukset sisältävät sellaisia tapahtumasekvenssejä, joita kukaan ei ole ennen edes tullut ajatelleeksikaan. Kunhan vain mallin tekemiseen panostetaan kunnolla, sen avulla voidaan löytää uusiakin virheitä. Vahvoin alue mallipohjaiselle testaukselle on rinnakkaisuuden ilmiöiden hallinta, joka ihmisille on vaikeaa erilaisten vaihtoehtoisten suoritusten valtavan määrän takia.

TTY:ssä mallipohjaista testausta on tutkittu viime vuosikymmenen lopulta lähtien lähinnä professori Antti Valmarin tutkimusryhmässä. Mallipohjaisen testauksen taustalla olevat matemaattiset teoriat ovat vuosien saatossa kypsyneet, ja menetelmiä voidaan alkaa soveltaa käytännössä. Nyt tarvitaan mielestäni soveltavaa tutkimusta, joka panee teorian käytäntöön. Tarvitaan käytännön esimerkkejä siitä, kuinka mallipohjainen testaus voidaan tuotteistaa ja ottaa käyttöön teollisuuden projekteissa.

”Perinteinen testien suorittamisen mahdollistava automaatio löytää harvoin uusia virheitä”

Hyvä esimerkki mallipohjaisuutta hyödyntävästä lähestymistavasta on kotimaista tuotantoa edustava Conformiq Softwaren Test Generator –työkalu. Työkalua on tietääkseni tähän mennessä käytetty lähinnä ohjelmointirajapinnan läpi tehtävässä testauksessa. Mallipohjaista testausta voidaan soveltaa myös käyttöliittymän läpi tehtävään testaukseen.

Testauskoulutus

Vastaan TTY:ssä Ohjelmistojen testaus –kurssista, jota suorittaa keväällä 2005 yli sata opiskelijaa. Testiautomaatio on keskeisessä osassa sekä luennoissa että kurssiin liittyvissä harjoitustöissä. Esimerkiksi TTCN-3-testauskielen perusteet käydään luennoilla läpi. Myös mallipohjaista testausta käsitellään, mutta pääpaino on toistaiseksi ollut perinteisissä menetelmissä ja työkaluissa.

Keväällä 2005 opiskelijoiden harjoitustyön aiheena on Mozilla-selaimen URL-jäsentäjän testaa-

minen. Harjoitustyöhön kuuluu V-mallin mukaisesti sekä testauksen suunnittelua että testitapausten ajamista. Yksikkötestaustasolla käytetään automatisointiin CppUnit-sovelluskehystä, joka on suositun JUnit-kehyksen C++-kielinen vastine. Koodista mitataan moniehtokattavuutta Testwellin CTC++-työkalujen avulla. Järjestelmätason testaus tehdään ohjelmointirajapinnan läpi käyttäen esimerkiksi Python-skriptejä. Käyttöliittymän läpi on tarkoitus testata myös manuaalisesti. Luennoilla esitellään tätä varten tutkivan testauksen (exploratory testing) tekniikoita.

havaitsemiaan työkaluja myös jatkossa toimiessaan teollisuuden palveluksessa.

Työkaluvalintoja ei voi tehdä kevyesti, olipa kyseessä sitten yritys tai yliopisto. Kurssin henkilökunnan kouluttautuminen käyttämään jotakin tiettyä työkalua, voidakseen opastaa opiskelijoita, vie aikaa. Tämän takia työkaluja ei voida joka vuosi vaihtaa, vaikka mielenkiintoinen väline löytyisi eikä hintakaan olisi esteenä.

TTY:n ohjelmistotekniikan laitokselle tehdään vuosittain kymmenkunta testausaiheista diplomityötä. Takavuosina suosittu aihe on ollut

”Teollisuudella on selvästi tarve tehostaa testausta ja pienentää siitä syntyviä kustannuksia”

Tulevaisuudessa harkintaan tulee myös muiden työkalujen käyttöönotto. Tämä asettaa kysymyksiä niiden lisensointiin liittyen. Yliopiston kannattaa hyödyntää mahdollisimman paljon aktiivisen kehittäjäyhteisön tukemia avoimen lähdekoodin ohjelmistoja, ja käyttää niitä koulutuksen apuna missä vain mahdollista. Teollisuuden käyttämät kaupalliset välineet ovat myös kiinnostavia esimerkiksi kuormitustestauksessa ja testattaessa käyttöliittymän läpi. Onneksi useimmat työkalujen maahantuojat suhtautuvat suopeasti ei-kaupallisiin yliopistolisensseihin. Nehän ovat investointi tulevaisuuteen; opiskelijat haluavat käyttää hyviksi automatisoida jokin testausvaihe, joka on aikaisemmin tehty manuaalisesti, tai sitten sitä ei ole tehty ollenkaan. Nyt päällimmäisenä näyttäisivät olevan Symbian-maailman testaushaasteet. Toivon näkeväni tulevina vuosina myös mallipohjaisen testauksen soveltamiseen ja käyttöönottoon liittyviä töitä. Laitoksen ensimmäinen mallipohjaiseen testaukseen liittyvä tohtorinväitös on luvassa tämän vuoden aikana.

Yhteenveto

Testaus ja sen opettaminen eroaa monessa mielessä ohjelmoinnista. Yksi erottava seikka on se, että siinä missä ohjelmointipuolella teknologian siirtoa tapahtuu lähinnä yliopistoista ja tutkimuskeskuksista teollisuuteen päin, testauksen tapauksessa suunta voi hyvin olla päinvastainen. Meillä yliopistojen opettajilla ja tutkijoilla on paljon opittavaa käytännön työtä tekeviltä.

Testiautomaation alueella pitäisi lisätä kanssakäymistä teollisuuden ja akateemisen maailman välillä. Teollisuudella on selvästi tarve tehostaa testausta ja pienentää siitä syntyviä kustannuksia. Aikajänne näihin muutoksiin on yleensä hyvin lyhyt. Akateemisella puolella taas asioita katsotaan hieman pidemmällä jänteellä, mutta käytettävissä olevat resurssit ovat usein riittämättömät. Synergiaetuja olisi löydettävissä varmasti sekä tutkimus- että koulutuspuolella.

Tekn. toht. Mika Katara työskentelee Tampereen teknillisen yliopiston ohjelmistotekniikan laitoksessa vanhempana tutkijana. Hän vastaa yksikön testauskoulutuksesta ja tutkii mm. mallipohjaista testausta. Kataran tavoittaa sähköpostitse osoitteesta mika.katara@tut.fi tai puhelimitse numerosta (03) 3115 5512.

TTY:n ohjelmistotekniikan laitoksesta valmistuu vuosittain noin sata diplomi-insinööriä. Laitoksessa toimii yhdeksän professoria, muuta henkilöstöä on opetus- ja tutkimustehtävissä noin 90. Merkittävässä roolissa tutkimusrahoituksen osalta ovat erilaiset yhteistyöprojektit yritysten kanssa (Tekes-projektit yms.)

Mallipohjainen testiautomaatio: perusteet ja huomioita käyttöönotosta

Rasa Siegberg, Conformiq Software Oy

Mallipohjaiseksi testaukseksi kutsutaan menetelmää, jossa kohdejärjestelmää koestavat testit ja niiden parametrit luodaan kohdejärjestelmän toimintaa ja sen ympäristöä kuvaavaan malliin perustuen. Todellista testiautomaatiota menetelmästä saadaan lisäämällä testien generoinnin jatkoksi niiden varsinainen suoritus, testitulosten tallentaminen, sekä tulosten arviointi.

Mallipohjaisen testiautomaation potentiaali testauksen ja laadunvarmistuksen tuottavuuden tehostajana on merkittävä, sillä automatisoimalla rutiininomaisia osia testauksen suoritus- ja raportointitöistä se tarjoaa mahdollisuuden vapauttaa testaushenkilöstöä korkeamman tason suunnittelu- ja analyysitehtäviin. Tätä tehostamispotentiaalia selkeyttänee analogia kääntäjiin, ohjelmistokehityksen perustyökaluihin - mallipohjainen testiautomaatio on testaukselle sitä mitä kääntäjäteknologia ohjelmistokehitykselle.

Ohjelmoinnin esihistoriassa ohjelmakoodi tuotettiin matalan tason konekielirutiineina. Tämä menetelmä oli hidas, kankea, ja erittäin virhealtis. Valtava harppaus tuottavuudessa saavutettiin kääntäjien kautta. Kääntäjät mahdollistivat ohjelmistosuunnittelun (eli varsinaisen ohjelman mallinnuksen) korkean tason ohjelmointikielellä (C, Pascal, Fortran). Korkean tason kielen esityksestä (mallista) kääntäjä sitten tuotti varsinaisen konekielisen toteutuksen eli itse ohjelman. Mallipohjaisessa testauksessa mallin voidaan katsoa olevan analogisessa asemassa esimerkin korkean tason ohjelmakoodin kanssa, kun taas itse testit vastaavat esimerkin konekielistä koodia. Mallipohjainen testiautomaatiosovellus on luonnollisesti analoginen vastine kääntäjälle.

Mallipohjainen testiautomaatio on testauksen välinekentässä verrattain uusi tulokas joten tältä kannalta on perusteltua puhua menetelmästä uuden sukupolven testausautomaationa tai uutena automatioparadigmana. Yksi menetelmän edelläkävijöitä on espoolainen Conformiq Software, jonka tuote Conformiq

> ”Mallipohjainen testiautomaatio on testauksen välinekentässä verrattaen uusi tulokas”

Test Generator on ensimmäinen kaupallinen mallipohjainen testiautomaatioratkaisu.

Seuraava kappale esittelee ne peruskomponentit, joista mallipohjainen testiautomaatiosovellus koostuu. Esittely liikkuu yleisellä tasolla eikä ole sinänsä sidoksissa Conformiq Softwaren ratkaisuun.

”Testimalli pyrkii kuvaamaan tuotteen ja ympäristön välistä rajapintaa tai tuotteen liittymää todellisuuteen”

Perusteita

Mallipohjaisella testiautomaatioratkaisulla voidaan ajatella olevan kolme geneeristä osakomponenttia:

- mallinnusformalismi
- analysaatio- ja suorituslogiikka
- liitynnät testattavaan järjestelmään

Seuraavat kappaleet esittelevät nämä komponentit ja joitakin huomioita niiden käytännön toteutuksesta.

Mallinnusformalismi

Testimallin tehtävä mallipohjaisessa testiautomaatiossa on kuvata testattava kohde, ja sen käyttäytyminen siltä osin kuin se on testaamisen kannalta olennaista. Mallin rakentamiseen voidaan käyttää erilaisia tapoja - se voidaan esittää tekstuaalisessa muodossa jonkinlaisella kuvauskielellä tai se voi olla graafinen esitys testattavan järjestelmän eri tiloista ja siirtymistä niiden välillä. Tärkeää on kuitenkin, että testimalli sisältää tiedon järjestelmän erilaisista mahdollisista tiloista ja niistä ehdoista tai lainalaisuuksista, jotka säätelevät tilasta toiseen siirtymistä.

Mallipohjaista testausta tarkastellessa huomio usein kiinnittyy testimallin ja jo olemassaolevien tuotemääritysten samankaltaisuuteen. Käytännön toteutusten käytettävyyden ja mallipohjaisen testiautomaation käyttöönoton helpottamiseksi olisikin edullista, että mallinnusformalismi olisi joku jo entuudestaan toimialalla tunnettu menetelmä. Lisäksi olisi edullista, että jo olemassaolevia tuotemäärityksiä ja -spesifikaatiota voitaisiin hyväksikäyttää testimalleja luotaessa. Conformiq Test Generatorissa käytettävyyttä ja käyttöönottoa on pyritty helpottamaan valitsemalla testimallien mallinnusformalismiksi UML (Unified Modeling Language) -tilakaaviot laajennettuna yksinkertaisella proseduraalisella ohjelmointikielellä.

Vaikka testimallien ja tuotteen määrittelyiden (arkkitehtuurimallien) välillä onkin helppo huomata samankaltaisuuksia on myös tärkeää tunnistaa niiden välinen merkittävä ero. Siinä missä arkkitehtuurimalli pyrkii kuvaamaan tuotteen sisäistä rakennetta ja sen jakoa erilaisiin rakenteellisiin osakokonaisuuksiin, testimalli pyrkii kuvaamaan tuotteen ja sen ympäristön välistä rajapintaa tai tuotteen liittymää todellisuuteen. Tästä syystä usein ehdotettu lähestyminen jonka mukaan testiaineisto ja -tapaukset tulisi voida generoida suoraan testattavan järjestelmän määrittelytiedoista ei tarkemmalla tarkastelulla vaikuta hedelmälliseltä. Testimallien olennaisimpana ominaisuutena voikin pitää niiden sisältämää tietoutta siitä kuinka, ja millaisilla lainalaisuuksilla, testattavan järjestelmän tulisi toimia (ulkoapäin katsottuna), kun taas saman järjestelmän määrittely kertovat lähinnä sen kuinka ko. järjestelmä tuon käytöksen saa aikaiseksi. Pelkistetysti testimallin voidaan ajatella kuvaavan testattavaa järjestelmää ns. ”black-box”-testauksen kannalta, kun taas tuotespesifikaatiot kuvaavat järjestelmää ns. ”glass-box”-lähestymisellä.

Analysaatio- ja suorituslogiikka

Mallipohjaisen testiautomaation suurin työtä säästävä potentiaali syntyy menetelmän tarjoamasta mahdollisuudesta tuottaa suuria määriä testiaineistoa automaattisesti testimalliin perustuen. Testiaineiston ja itse testien generointi asettaa mallipohjaisille testiautomaatioratkaisuille vaatimuksia niin mallien analysoinnin logiikan ja ”älykkyyden” kuin testien suorittamisenkin suhteen, ja sen perusteena oleva teknologia on laskentateoreettisen ja tietojenkäsittely-

tieteellisen tutkimuksen kohteena saapunut todellisen läpimurron kynnykselle 90-luvun aikana.

Käytännön mallipohjaisessa testiautomaatioratkaisussa analysaatio- ja suorituslogiikan tehtävä on:

- analysoida testimalli ja todeta sen mahdollistamat erilaiset testitapahtumat ja -aineisto, sekä huomioida millainen testijärjestelmän käytös tulkitaan läpäistyksi testiksi ja mikä hylätyksi testiksi
- tuottaa, mallin asettamissa rajoissa, "testistimulusta", jota testien suorituksen aikana syötetään testattavalle järjestelmälle
- pitää kirjaa toteutetuista suoritetuista testeistä ja niiden lopputuloksista sekä "testistimuluksina" käytetyistä arvoista
- pitää kirjaa ja seurata testikattavuutta, ja verrata saavutettua kattavuutta testimallin mahdollistamaan "testiavaruuteen"

kattavan, laajuudeltaan valtavan, ja tarkkuudeltaan yksityiskohtaisen testiaineiston luomisen varsin korkealla käsitetasolla liikkuvan testimallin pohjalta.

Liitynnät testattavaan järjestelmään

Mallipohjaisen testiautomaatioratkaisun kytkentä reaalimaailman testiympäristöön mahdollistaa testauksen joko reaaliajassa tai eräajona jonkin ohjelmointikielen kautta. Testijärjestelyn liityntää testattavaan järjestelmään (SUT - System Under Test) kutsutaan testausrajapinnaksi (Test API). Rajapinta on luonnollisesti järjestelmäkohtainen, mutta yleisenä lainalaisuutena voidaan pitää sitä, että rajapinnan tulisi olla testaamisen kannalta merkityksellinen, eli sellainen, että sen kautta voidaan tarkkailla ja vaikuttaa juuri testauksen kohteena oleviin toiminnallisuuksiin. Kun ”perinteisessä” käyttöliittymäpohjaisessa testiautomaatiossa rajapinta rajoittuu useimmiten testattavan sovelluksen käyttöliittymään (ja sen suoranaisesti kontrolloimiin toiminnallisuuksiin), on mallipohjaisen testiautomaation kannalta tehokasta etsiä ”lähempänä” testattavan järjestelmän logiikkaa olevia rajapintoja jolloin testausta eivät rajoita käyttöliittymän ominaisuudet ja konventiot.

Käytännön sovelluksissa mallipohjaisen testiautomaation kytkentä testattavaan järjestelmään toteuttaa seuraavat toiminnat:

- muuntaa testityökalun tuottamat testiaineistot testattavan järjestelmän ymmärtämään muotoon
- välittää testityökalulta testattavalle järjestelmälle testien vaativaa syötettä
- muuntaa testattavan järjestelmän vastaukset testityökalun ymmärtämään muotoon
- välittää testattavan järjestelmän vastauksia testityökalulle
- havainnoi testattavan järjestelmän käytöstä testien aikana

Conformiq Test Generator:in tapauksessa liityntä testattavan järjestelmän ja testityökalun välillä toteutetaan järjestelmäkohtaisella adapterikomponentilla, joka siis kootaan jokaista testiympäristöä ja sen erityispiirteitä vastaavaksi. Vaikka kyseisen kaltainen liityntä olisi mahdollista myös kutakuinkin vakiintuneita teknologioita (CORBA, XML) käyttäen, on valitun menetelmän etuna sen joustavuus. Järjestelmäkohtaisella adaptaatiolla voidaan kytkeytyä myös esimerkiksi sulautettuihin järjestelmiin, joissa suorituskyky on usein rajoitettu ja vaatimukset esim. muistinkulutukselle ovat tiukat.

”Mallipohjaisen testiautomaation käyttöönottoa edelsi menetelmän arviointi- ja pilotointiprojekti””

Conformiq Test Generator:in tapauksessa ratkaisun analyysi- ja suorituslogiikka on rakennettu kykeneväksi niin reaaliaikaiseen "on-the-fly"-testaukseen kuin erilaisten eräajoina suoritettavien testikirjastojen (esimerkiksi Java- tai TTCN-3 kielillä) luomiseenkin. Analyysi- ja suorituslogiikan sisältämä älykkyys mahdollistaa

Havaintoja käyttöönoton tiimoilta

Conformiq Software:n mallipohjaisen testiautomaation ratkaisun asiakaskunta tarjoaa menetelmän ja työkalun käyttöönotto- ja käyttökokemuksia usean vuoden ajalta. Asiakaskunnan ja sen toimialojen heterogeenisyys tarjoaa laaja-alaisen aineiston uuden sukupolven testaustyökalun käyttöönoton tarkastelulle. Seuraavat kappaleet tiivistävät käytännön testiautomaatioprojektien aikana tehtyjä havaintoja.

Valmistautuminen ja suunnittelu

Lähes kaikissa tapauksissa mallipohjaisen testiautomaation käyttöönottoa edelsi (usein varsin perusteellinen) menetelmän arviointi- ja pilotointiprojekti. Tätä lähestymistä on lähes poikkeuksetta pidetty hyvänä sen tarjotessa taloudellisesti riskittömän ja organisaation/testattavan järjestelmän omia erityispiirteitä huomioivan käynnistysvaiheen.

Teknisen tuote-evaluaation ohella on myös havaittu, että paras tulos uuden testausparadigman soveltamisessa saavutetaan, kun uusi menetelmä otetaan huomioon jo varsin aikaisessa vaiheessa testausta suunniteltaessa, jolloin on syytä kriittisesti tarkastella niitä tavoitteita, joita testauksen automatisoinnilla tavoitellaan. Mitä tarkemmin tavoitteiden asettelu on ennen automaatioprojektin käynnistystä tehty, sen paremmin on tavoitteita kyetty saavuttamaan ja prosessia seuraamaan.

Prosessi ja kompetenssit

Testauksen toteuttamiseen liittyvät työkalujen ja suunnittelun ohella olennaisesti se organisaatio ja henkilöstö, joka itse testausprojektin toteuttaa, ja ne prosessit, joiden mukaan tämä organisaatio toimii. Testauksen automatisoinnin onnistumisen todennäköisyyttä kasvattaa olennaisesti se, että lähtötilanteena on ainakin jollakin tasolla formalisoitu testauksen ja laadunvarmennuksen prosessi.

”Testausautomaatiotyökalut vaikuttavat vaativan henkilöstöltä eri tyyppisiä kompetensseja kuin “perinteinen” manuaalinen testaus”

Yleisesti testausautomaatiotyökalut automaatioparadigmasta riippumatta vaikuttavat vaativan henkilöstöltä eri tyyppisiä kompetensseja kuin “perinteinen” manuaalinen testaus. Tämä pätee myös mallipohjaisen testiautomaation kohdalla. Käyttöönottoprojektien alussa on usein ohjelmointi-intensiivinen vaihe, kun testityökalua sovitetaan ja kytketään testattavaan järjestelmään. Projektin alkuvaiheen jälkeen testausjärjestelyn tuntemuksen sijaan korostuu testattavan järjestelmän tuntemus, kun varsinaisia testimalleja rakennetaan ja ylläpidetään. Kokemusten mukaan projektin alkuvaiheissa on usein tilausta työkalutoimittajan projektiin osallistumiselle. Tämän tarve kuitenkin nopeasti vähenee projektin edetessä ja testijärjestelyn kypsyessä.

Johtopäätelmiä

Mallipohjainen testiautomaatio lupaa merkittävää parannusta testauksen laadun ja kattavuuden saralla. Se myös vaatii käyttöönottajaltaan merkittävää sitoumusta. Mallipohjaisen testiautomaation käyttöönotto merkitseen organisaatiolle panostusta teknologiaan, osaamisen kehittämiseen ja kurinalaiseen ja hyvin organisoituun laadunvarmennukseen. Sillä saavutettavat hyödyt toteutuvat sitä todennäköisemmin mitä tarkemmin automaatioprojektin tavoitteet ja resurssointi on määritelty ja organisaatio on niihin sitoutettu.

Kirjoittajalla toimii testauskonsulttina Conformiq Software Oy:ssä, ja osallistuu aktiivisesti asiakasprojekteihin, joissa mallipohjaista testiautomaatiolähestymistä ja Conformiq Test Generator:ia sovelletaan käytännön ympäristöissä.

Espoolainen Conformiq Software työllistää noin 40 testauksen asiantuntijaa, ja on maan johtava mallipohjaisen testiautomaation kehittäjä ja Conformiq Test Generator –testiautomaatioratkaisun kehittäjä.

Niele Python

Marko Komssi, Mika Eloranta
F-Secure Corporation

Ohjelmistoteollisuudessa kehitysjohtajilta usein kuultava kommentti on, että automaattisesti suoritettavien testien määrää tulisi lisätä. He kuitenkin jättävät huomioimatta, että testausautomaatioprojektit epäonnistuvat usein (Kaner, C., Pitfalls and strategies in automated testing. Computer, 10, 4, 1997). Yksi epäonnistumisen syistä on väärin valittu työkalu. Toinen merkittävä epäonnistumisten syy on testauksen väärä kohde. Esimerkiksi käyttöliittymien kautta tehtävä testien automaattinen suoritus on yleistä, mutta harvoin kustannustehokasta, toisin kuin työkalujen myyjät väittävät.

Jos työkalu ja testauksen kohde ovat sopivia, testien automatisoinnilla on mahdollista saavuttaa merkittäviä hyötyjä. Kokemuksemme Python-ohjelmoinnista testauksen apuna ovat olleet myönteisiä erityisesti F-Securen tuotekomponenttien automaattisessa rajapintatestauksessa, jossa testikoodi kutsuu tuotekomponenttien tarjoamia rajapintafunktioita. Tässä artikkelissa kerromme meidän Python-kokemuksistamme ja sen eduista testausautomaatiossa.

”Python on ilmainen, helppo asentaa ja oppia sekä tehokas käyttää”

Mikä ihmeen Python?

Python (ks. http://www.python.org) on houkutteleva välinevaihtoehto testien automatisointiin. Se on oliopohjainen ohjelmointikieli, joka tukee monin tavoin testien automatisointia. Python on ilmainen, helppo asentaa ja oppia sekä tehokas käyttää. Sen vahvuuksia ja testausautomaatiossa hyödyllisiä ominaisuuksia ovat esimerkiksi:

- sisäänrakennettu yksikkötestaussovelluskehys testien kirjoittamiseen, raportointiin ja virheiden käsittelyyn
- interaktiivinen komentotulkki virheiden paikantamiseen ja suunnitteluideoiden nopeaan kokeiluun
- Pythonin ja testitapausten helppo asennus testiympäristöön
- käyttöjärjestelmäriippumattomuuden ansiosta samat testit voidaan ajaa useissa eri käyttöjärjestelmissä
- helposti toteutettavien C / C++ -laajennusten avulla testattavan moduuleiden rajapintoihin voidaan tehdä Python-testitapauksista kutsuja
- voimallisen syntaksin ja kattavien vakiokirjastojen ansiosta Python-ohjelman pituus on usein merkittävästi lyhyempi kuin vastaava ohjelma esim. C++- tai Java-kielillä kirjoitettuna

Microsoftin tuotteiden kolme arvoa ovat viehättävyys, käytettävyys ja hyödyllisyys. Nämä ovat erittäin tärkeitä ominaisuuksia myös testiautomaatiotyökaluille. Seuraavissa luvuissa kerromme, miten Python täyttää nämä arvot teknisen testaajan, kehitystiimin ja testausarkkitehdin näkökulmista.

Viehättävä työkalu tekniselle testaajalle

Viehättävä testaustyökalu saa teknisesti suuntautuneen testaajan innostumaan. Miltä sinusta tuntuisi testaajana pystyttää esimerkiksi web-palvelin vain kahdella rivillä koodia? Pythonilla se onnistuu seuraavasti:

```
import SimpleHTTPServer
SimpleHTTPServer.test()
```

Tämän jälkeen voisit “surffailla” koneesi tiedostoja selaimellasi osoitteesta ‘http://localhost:8000/’. Aika viileetä, eikö? Itse asiassa voisit myös hakea palvelimesi html-sivujen sisältöä Pythonin avulla esimerkiksi juurihakemistostasi seuraavasti:

```
import urllib
content = urllib.urlopen(”http://localhost:8000/”)
print content.read()
```

Eikö olekin helppoa kuin heinänteko? Testaajana voit käyttää Pythonia helposti myös moniin muihin hyödyllisiin toimintoihin, kuten sähköpostien lähettämiseen ja vastaanottamiseen sekä tietojen etsintään esimerkiksi xml-tiedostoista ja Windowsin rekisteristä.

Monia testaajia viehättävät palkankorotukset ja kollegoiden arvostus. Siksi on hyvä tietää, että Pythonin käyttö antaa uutta pontta palkkaneuvotteluihin. Software Development -lehden (marraskuu 2003) mukaan Pythonia käyttävät ohjelmistoteollisuuden ammattilaiset tienaavat enemmän kuin muita ohjelmointikieliä käyttävät ammattilaiset. Toistaiseksi emme ole huomanneet tällaista ilmiötä palkkapusseissamme, mutta olemme kokeneet saavamme erityistä arvostusta kollegoiltamme. Itse asissa, meidän onnistuneiden kokemuksiemme myötä Pythonin käyttö on jatkuvasti lisääntynyt F-Securella.

Johtajien ja päälliköiden on vaikeampi ymmärtää testausautomaation hyötyjä, jos he eivät pysty näkemään testejä. Vaikka käyttöliittymien kautta tehtävä testien automaattinen suoritus ei yleensä ole optimaalisin tapa, käyttöliittymien testien automatisointia kannattaa pienissä määrin harkita juuri demonstroimista varten. Lisätäksemme työmme näyttävyyttä olemme automatisoineet Pythonilla muutaman käyttöliittymätestitapauksen. Teknisen testaajan on hyvä muistaa, että pomojen hämmästyttämiseen riittää pari testitapausta. Tämä erityisesti siksi, että Python ei ole parhaimmillaan käyttöliittymään kohdistuvassa automaattisessa testauksessa.

Hyödyllinen testityökalu ohjelmistokehitystiimille

Kehitystiimit tuottavat projektin aikana ohjelmistokomponenttien uusia versioita, joiden muutokset mahdollisine sivuvaikutuksineen tuotteessa tulisi varmistaa. Valitettavasti käsin tehtävä regressiotestaus on kallista ja hidasta. Pythonilla toteutettu ohjelmistokomponenttien rajapintatestaus voi nopeuttaa regressiotestaukseen kuluvaa aikaa merkittävästi.

Automaattisella testauksellalöydettyjen vikojen korjaamiseen kuluva aika vähenee, koska Pythonilla kirjoitettujen testitapausten koodia voidaan käyttää virheiden paikallistamiseen. Koska Python-koodi ei vaadi erillistä kääntämistä, testikoodia voidaan muokata suoraan testikoneella ja ajaa testi uudestaan. Lisäksi Pythonin interaktiivista komentotulkkia voidaan käyttää myös käsin tehdyllä testauksella löydettyjen vikojen paikallistamiseen.

Testitapausten ohjelmointi Pythonilla on tehokasta. Kirjoitamme sillä esimerkiksi automaattiset testit lähes kaikkiin tiimimme vastuualueella oleviin muutoksiin uudessa F-Securen Client Security -tuotteessa, koska uskomme voittavamme käytetyn ajan hyvin nopeasti takaisin. Mielestämme Pythonin avulla voidaan rajapintafunktioiden toiminnallisuus, raja-arvot ja poikkeustilanteet kattaa paljon vaivattomammin kuin esimerkiksi Javalla, Perlillä tai C++:lla.

Testaajan tuntemus tuotteesta saattaa rajoittua pitkälti käyttöliittymiin, sillä sisäisten rajapintojen käyttö vaatii esimerkiksi C++-koodin pointtereiden ja kielen monimutkaisen syntaksin tuntemusta. Pelkästään käyttöliittymien kautta tapahtuvalla testauksella ei voi saada tarkkaa kuvaa ohjelman sisäisestä toiminnasta. Pythonilla toteutettu rajapintatestaus kaventaa testaajien ja ohjelmoijien välistä kielimuuria sallimalla testaajan helpomman pääsyn tuotteen sisäisiin rajapintoihin. Esimerkiksi ohjelmoija voi kirjoittaa Pythonilla minimaaliset testitapaukset komponentin tärkeimpiin rajapintafunktioihin ja tämän jälkeen testaaja laajentaa testitapauksia. Tämä toimintatapa lisää mahdollisuuksia löytää ohjelmasta vikoja: testaaja ei tee koodista niin paljon oletuksia kuin ohjelmoija tekee (Kaner, C., Bach, J., Pettichord, B., Lessons Learned from Software Testing - A Context-driven Approach. Wiley Computer Publishing, 2002). Ohjelmoijahan hellii tekemäänsä koodia, toisin kuin (ilkeä) testaaja, joka haluaa rikkoa kaiken käsiin saamansa. Kun testauksella on katettu rajapintafunktioiden tärkeimmät toiminnallisuudet, voi ohjelmoija tehdä melko turvallisesti tarvittavia muutoksia ohjelmakoodiin. Testikoodi toimii lisäksi hyvänä dokumentaationa tuotteen komponentin rajapintafunktioiden eri käyttötavoista ja niiden reunaehdoista.

”Johtajien ja päälliköiden on vaikeampi ymmärtää testausautomaation hyötyjä, jos he eivät pysty näkemään testejä”

Olemme automatisoineet kahden C++-kielellä toteutetun F-Securen ohjelmakomponentin testausta Pythonilla. Ohjelmakomponentit ovat osa laajaa tuoteperhekokonaisuutta. Emme ole ensisijaisesti pyrkineet löytämään uusia vikoja tuotekomponenteistamme, vaan lyhentämään regressiotestaukseen käytettävää aikaa. Tavoitteemme on ollut kattaa tärkeimpiä skenaarioita, joita komponentit tarjovat toisille ohjelmakomponenteille. Voimme varmistaa komponenttien tärkeimmät toiminnallisuudet nopeasti rakentaessamme uuden version. Lisäksi olemme testiympäristöissä korvanneet ulkoisia järjestelmiä simuloimalla niiden testeissä tarvittavat toiminnallisuudet Python-koodilla. Tämä toimintatapa on vähentänyt testiympäristön asentamiseen kuluvaa aikaa ja kohdistanut testauksen pelkästään haluamaamme komponenttiin.

Meillä ei vielä ole käytännön kokemusta Java-koodin testauksesta Pythonin avulla. Pythonista on kuitenkin olemassa Java-kielellä toteutettu sisarversio Jython (ks. http://www.jython.org), jonka avulla Java-koodin testaus on suoraviivaisempaa kuin C++-koodin testaaminen. Jythonilla näet pääsee suoraan käsiksi Java-kirjastojen rajapintoihin ilman ylimääräisten laajennusten kirjoittamista.

```
import urllib2, unittest
class WebServerTest(unittest.TestCase):
    def testNonExistingPage(self):
        try:
            urllib2.urlopen(”http://www.company.com/nonexisting.html”)
            self.fail(”Web page received when expecting error code”)
        except urllib2.HTTPError, error:
            assert error.code == 404, ”Error code received, but a wrong one”

    def testPageTitle(self):
        content = urllib2.urlopen(”http://www.company.com/”)
        for line in content.readlines():
            if ”<TITLE>” in line.upper():
                return
        self.fail(”Title not found in index.html!”)

unittest.main(defaultTest=”WebServerTest”)
```

Käytettävä ja helposti ylläpidettävä työkalu testausarkkitehdille

Kokemuksemme mukaan Python-koodin ylläpito on vaivattomampaa kuin esimerkiksi Perl-kielellä tehdyn koodin. Isohkotkin Python-projektit säilyvät ylläpitokelpoisina, eikä muutosten tekemistä tarvitse välttää koodin rikkomisen pelossa. Oliosuuntautuneisuuden ja selkeän syntaksin lisäksi Pythonissa on mukana unittest-moduuli, jota voidaan käyttää automaattisten testitapausten järkevään ylläpitoon, jaotteluun ja suoritukseen sekä virheiden raportointiin. Alla on yksinkertainen esimerkki unittest-moduulin käytöstä. Esimerkissä testataan aluksi, että yrityksen internet-palvelin lähettää tietyn virhekoodin, kun etsittyä sivua ei löydy. Toisessa testissä varmistetaan, että sivuston pääsivu on olemassa ja että sieltä löytyy haluttu html-elementti. Enempää Python-koodia ei tarvita näiden kahden testitapauksen testaamiseen ja testitulosten raportoimiseen. Koodiesimerkki on muutoksitta ajettavissa esimerkiksi Linuxissa ja Windowsissa.

Python on yleiskäyttöinen työkalu. Sitä kehitetään jatkuvasti ja uusia versioita julkistetaan usein. Sen monipuoliset kirjastot antavat testien kirjoittajalle mahdollisuuden ratkaista mitä erilaisempia testaukseen automatisointiin liittyviä ongelmia. Tässä lyhyessä artikkelissa rajasimme esimerkkitapauksemme yksinkertaisuuden vuoksi http-protokollaan.

Pythonin valmiita kirjastoja on helppo laajentaa omien tavoitteiden mukaisesti. Olemme esimerkiksi rakentaneet unittest-moduulin ympärille oman sovellustestauskehyksen, joka tarjoaa monipuolisemmat testitapausten hallinnointityökalut ja helppokäyttöiset versiot tuotteidemme perusrajapinnoista ja näin soveltuu myös muiden tuotekomponenttitiimien käyttöön.

> ”Pythonilla toteutettu rajapintatestaus kaventaa testaajien ja ohjelmien välistä kielimuuria”

Python-versioiden ylläpito aiheuttaa testausarkkitehdille vain vähän ylläpitotoimenpiteitä. Testausarkkitehdin pitää uutta versiota päivittäessä kääntää uudestaan C++-kielellä kirjoitetut Python-laajennukset. Tämä on kerrallaan vaatinut meiltä vain noin tunnin verran työtä. Python-versioiden välinen yhteensopivuus on hyvä ja niiden välisten muutosten takia olemme harvoin joutuneet tekemään muutoksia itse Python-koodiin.

Houkutteleva Python pähkinänkuoressa

Python on huokutteleva ohjelmointikieli testausautomaatioprojekteihin, koska sillä on laajat käyttömahdollisuudet, alkuunpääseminen on edullista ja testikoodin ylläpito ei yleensä vaadi paljon resursseja. Valmiiden luokkakirjastojen avulla voidaan kirjoittaa nopeasti koodia. Lisäksi kokemustemme mukaan rajapintatestaus Pythonilla lähentää testaajia ja ohjelmoijia, lisää testaajien tuntemusta testattavasta tuotteesta ja nopeuttaa testaussyklejä. Testaustyökalun valintavaiheessa on tärkeää tunnistaa sen menestystekijät ja sopivuus haluttuun käyttöympäristöön. Emme halunneet lähteä tässä artikkelissa kattavasti vertailemaan Pythonia muihin ohjelmointikieliin ja testausautomaatiotyökaluihin. Python on sopivin testaustyökalu käyttötarkoituksiimme. Koska työkalun arviointi on osin tunneasia, jätämme Pythonin arvioinnin ja vertailun muihin työkaluihin omaksi tehtäväksesi.

Marko Komssilla on noin kahdeksan vuoden työkokemus skriptikielien (esim. Pythonin ja Perlin) käytöstä. Hänen intohimoinaan ovat testauksen ja konfiguroinnin automatisointi sekä ohjelmistojen tarkastukset (engl. software inspections).

Mika Elorannalla on yli kymmenen vuoden työkokemus Windows-ohjelmistokehityksestä sekä automaatio- ja testausohjelmistojen kehityksestä. Hänen erikoisosaamisensa on luontaisen laiskuuden ansiosta rutiinityön välttely automaation avulla, jossa Python on ollut pääasiallisena työkaluna kuutisen vuotta.

Aineisto-ohjattu ja avainsanaohjattu testausautomaatio

Pekka Laukkanen, Qentinel Oy

Tässä artikkelissa käydään läpi mitä ovat aineisto-ohjattu (data-driven) ja avainsanaohjattu (keyword-driven) automatisoitu testaus ja selitetään mitä etuja niistä on. Artikkeli on tarkoitettu kaikille testaajille eikä esimerkiksi

```
SetFocus "Calculator"
Push "2"
Push "+"
Push "4"
Push "0"
Push "="
Check "Result=42"
```

Kuva 1 - Yksinkertainen testiskripti

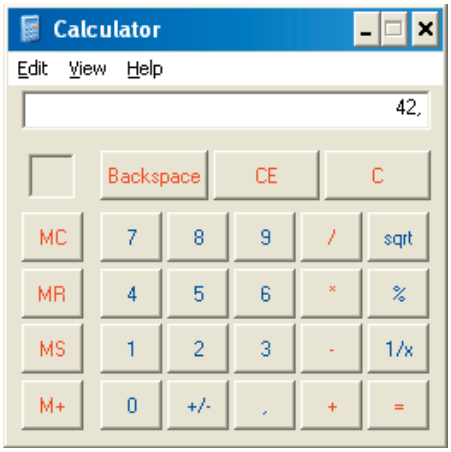
minkäänlaista ohjelmointikokemusta tarvita.

Yksinkertaiset testiskriptit, riippumatta siitä onko ne kirjoitettu käsin vai nauhoitettu, ovat yleisesti ottaen saman tyyppisiä kuin pseudokoodinen esimerkki kuvassa 1. Koodi kyseisessä esimerkissä on tarkoitettu testaamaan kuvassa 2 näkyvää Windows Calculator -ohjelmaa.

Ongelma kuvan 1 tyyppisissä skripteissä on se että käytettävä testidata on kovakoodattuna niiden sisällä. Jos testaaja haluaa testata jotain muuta - esimerkiksi laskun 0+0 tai 1-1 - on näille testeille tehtävä omat testiskriptit. Uuden skriptin tekeminen on periaatteessa helppoa sillä sehän onnistuu kopioimalla vanha ja muokkaamalla sitä hieman. Tosielämässä testiskriptit eivät kuitenkaan ole niin yksinkertaisia kuin kuvassa 1 oleva eikä pientenkään muutosten teko välttämättä onnistu ilman ohjelmointitietämystä. Vielä suurempi ongelma tulee eteen myöhemmin jos (tai paremminkin kun) testattava kohde muuttuu ja kymmenet tai sadat testiskriptit täytyy päivittää. Skriptien ylläpidosta tulee helposti niin iso ongelma että testit on nopeampi koodata tai nauhoittaa uudestaan. Tällöin automaatiolla haetut uudelleenkäytön edut jäävät toteutumatta ja automaation hyödyt ovat kyseenalaisia.

Osittainen ratkaisu näihin ongelmiin on testiaineiston erottaminen testiskripteistä. Testiaineisto vaidaan laittaa erilliseen helposti editoitavissa olevaan tiedostoon ja se esitetään yleensä taulukkomuodossa kuten kuvassa 3.

Hyödyt edelliseen esimerkkiin ovat selvät. Testien editoiminen on helppoa ja ennen kaikkea uusien



Kuva 2 - Testattava ohjelma

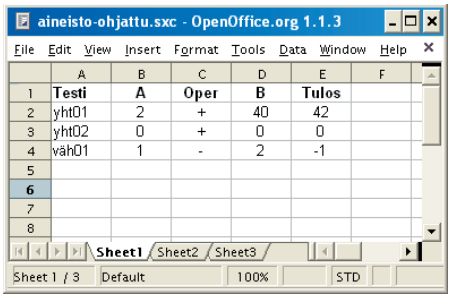
testien lisääminen joukon jatkoksi on vaivatonta. Muutokset testattavassa kohteessa voivat toki edelleen

hajoittaa testijärjestelmän, mutta nyt muutokset rajoittuvat vain yhteen testiskriptiin. Aineisto-ohjattu testaus siis tekee useiden saman tyyppisten testien suunnittelun ja suorittamisen helpoksi.

Itse testiskriptin ei tarvitse olla juurikaan kuvassa 1 olevaa esimerkkiä monimutkaisempi. Taulukkomuotoinen testidata täytyy toki ensin prosessoida ja varsinaisessa testiosassa korvata kovakoodattu testidata muuttujilla. Näiden muutosten tekeminen omiin testiskripteihin ei kuitenkaan ole erityisen vaikeaa ja kaupallisissa testituotteissa tällaiset ominaisuudet ovat yleensä käytössä suoraan.

Kuvan 3 esimerkissä on kuitenkin yksi ikävä ongelma - testien täytyy olla saman tyyppisiä. Jos esimerkiksi kuvan 2 laskinta testattaessa haluttaisiin tarkistaa että myös laskut 1+2+3 tai 2*5-12+2 toimivat ei se onnistuisi kuvan 3 taulukolla. Jos testaajille halutaan antaa vapaus tehdä eri tyyppisiä testejä on syytä viedä aineisto-ohjattu testaus astetta pidemmälle ja siirtyä avainsanaohjattuun testaukseen. Esimerkki tästä on kuvassa 4.

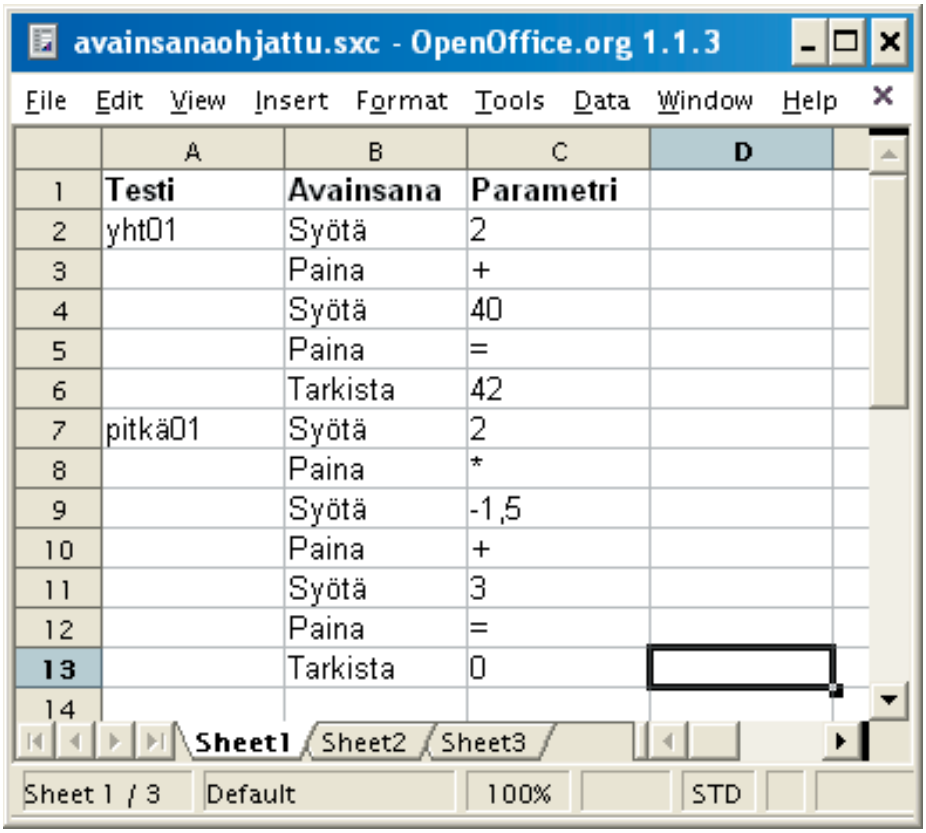
Kuten kuvasta 4 nähdään koostuu yksi avainsanaohjattu testi useasta askeleesta ja yhtä askelta taas kuvaa yksi avainsana sekä sille annettavat parametrit. Testaaja voi koota halumansa testin vapaasti olemassa olevista avainsanoista. Vaikka taulukkoon suunniteltu testi alkaa jo näyttää hiukan samalta kuin testiskripti kuvassa 1 on se kuitenkin huomattavasti helpompi ymmärtää ilman ohjelmointitaitoja. Avainsanat ovat myös yleensä hiukan korkeammalla tasolla kuin elementit koodissa - vertaa esimerkiksi sitä että luvun 20 syöttäminen onnistuu kuvassa 4 yhdellä rivillä kun siihen kuvassa 1 menee kaksi.



Kuva 3 - Esimerkki syötetiedostosta aineisto-ohjatussa testauksessa

Avainsanaohjatun lähestymistavan hyvä puoli on juuri se että erilaisia testejä voi tehdä vapaasti. Avainsanaohjatun testijärjestelmän toteuttaminen on kuitenkin vähintään kertaluokkaa hankalampaa kuin aineisto-ohjatun. Aineisto-ohjattu testidata voi myös olla huomattavasti helpompi tehdä ja ymmärtää - vertaa esimerkiksi samaa asiaa testaavaa testiä "yht01" kuvissa 3 ja 4.

Sekä aineisto- että avainsanaohjattu testaus ovat erittäin käyttökelpoisia menetelmiä ylläpidettävien automaatiorakenteiden toteuttamiseen. Molemmissa on etuna mahdollisuus jakaa testien suunnittelu sekä automaation tekninen toteutus näihin tehtäviin erikoistuneille osaajille kokonaisuuden tehostamiseksi. Se kumpi menetelmä mihinkin tilanteeseen parhaiten sopii riippuu lähinnä testattavasta järjestelmästä sekä käytössä olevista työkaluista sekä osaamisesta.



Kuva 4 - Esimerkki syötetiedostosta avainsanaohjatussa testauksessa

Pekka Laukkanen toimii testaustehtävissä Qentinel Oyssä. Automatisoinnin jäsentäminen ja kehittäminen on hänen keskeisiä kiinnostuksen kohteina.

Testaushallinnan välineen valinta ja käyttöönotto

Sami Kallio, Conformiq Software Oy

Testausprosessin parantaminen alkaa usein sen hallinnan parantamisella. Ja usein testauksen hallintaa voidaan parantaa ottamalla käyttöön tähän tarkoitukseen kehitetty testaushallinnan väline. Testausvälinettä ei kuitenkaan tule ostaa kuin maitoa kaupasta; valinta tulee tehdä suunnitellusti ja harkiten.

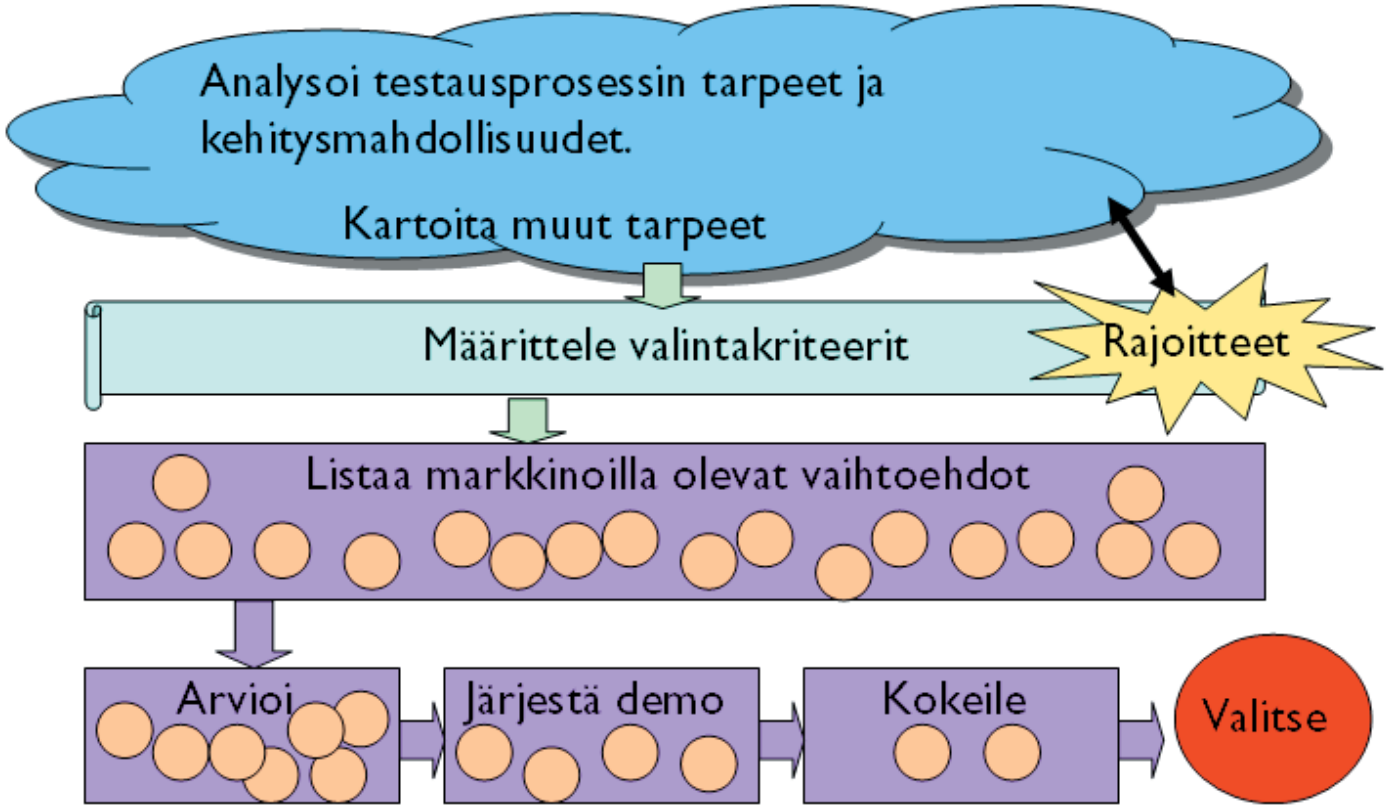
Testauksen hallinnasta

Testauksen tärkeimpiä ja usein vaikeimpia osa-alueita on sen hallinta. Miten pitää testaus budjetissa? Miten mitata testauksen edistymistä ja tuloksia? Miten mitata testattavan kohteen edistymistä ja laatua? Testauksen hallinta on myös erittäin monitahoista: testimateriaalin hallinta, testauksen resurssointi ja resurssien hallinta, testivaatimusten hallinta, testauksen suunnittelu, testitapausarkkitehtuurin ja testitapausten suunnittelu, testitapausten suoritus ja suorituksen seuranta, testauksen tuloksien raportointi ja seuranta, jne. Koska testauksen hallinta on niin monipuolista ja vaikeaa, on sen tueksi syytä harkita apuvälinettä, joko yhtä tai useampaa riippuen tarpeista.

> ”Koska testauksen hallinta on niin monipuolista ja vaikeaa, on sen tueksi syytä harkita apuvälinettä”

Ennen välineen valintaa

Ennen välineen hankintaa on syytä arvioida olemassa olevaa käytäntöä ja tarpeita. Minkä tahansa välineen hankinta on yleensä kallis ja koko testausorganisaatioon vaikuttava asia. Esimerkiksi testausprosessia joudutaan usein muuttamaan välineeseen sopivaksi. Tietenkään ei pidä elää välineen ehdoilla; sen sijaan tulee valita parhaiten omaan prosessiin sopiva väline, ja mukauttaa prosessia vain tarpeen mukaan. Joka tapauksessa, on siis tarpeen selvittää mihin asioihin tarvitaan parannusta ja olisiko jostakin testauksen hallinnan apuvälineestä hyötyä parannuksen teossa. Yleensä parannettavia kohteita löytyy useita, joten ne täytyy laittaa tärkeysjärjestykseen, koska kaikkea ei kuitenkaan voida parantaa kerralla.



Testaushallinnan välineen valinta

Osa parannettavista asioista voidaan korjata olemassa olevia apuvälineitä paremmin hyödyntäen tai yksinkertaisesti kehittämällä testausprosessia. Mutta usein testauksen hallinta tehostuu parhaiten siihen erikoistuneen testausvälineen avulla.

kaan kartoittaa tarjonta ja niiden sopivuus omiin tarpeisiin. Lisäksi on aina mahdollista kehittää omia välineitä testauksen hallinnan tarpeisiin.

Välinehankintaa varten täytyy laatia suunnitelma ja valintakri-

Ennen lopullista valintaa on hyvä suorittaa koekäyttö, jossa varmistuu paitsi yhteensopivuus niin myös käytettävyys ja sopivuus tarpeisiin. Koekäyttöön kannattaa valita 2-3 välinettä, joista saatuja kokemuksia voi sitten vertailla keskenään. Vasta kokeilu todella kertoo sopiiko väline tarpeisiin ja mikä välineistä on paras. Kokeilukäyttö antaa näkemyksen myös välineen käytettävyydestä ja asennuksen helppoudesta, jotka ovat usein merkittäviä valintakriteerejä.

”Välinehankintaa varten täytyy laatia suunnitelma ja valintakriteerit”

Välineen valintaprosessi

Kun nykytilanne on kartoitettu ja parannettavat alueet tunnistettu, voidaan alkaa etsiä sopivaa välinettä korjauksen aikaansaamiseksi. Markkinoilta löytyy valtaisa määrä erilaisia testauksen hallinnan apuvälineitä, sekä ilmaisia että lisenssoituja. Siksi on syytä tarkteerit; kuinka välineen hankintaprosessi toteutetaan ja mitä välineeltä vaaditaan. Kriteereistä päätettäessä tulee huomioida paitsi tarpeet, myös esim. yhteensopivuus olemassa oleviin järjestelmiin tai toisiin harkinnassa oleviin välineisiin.

Yhteenvedettynä välineen valintaprosessi etenee seuraavasti:

1. Tarpeiden ja prosessin kehitysmahdollisuuksien kartoitus
2. Välineen valintakriteerien päättäminen
3. Esikartoitus; kaikkien mahdollisten välineiden etsintä ja arviointi

4. Parhaiten valintakriteerit täyttävien välineiden (2-3 kpl) koekäyttö
5. Välineen valinta
6. Käyttöönotto

Mitä valinnan jälkeen tapahtuu?

Kun testauksen hallinnan väline on valittu ja hankittu, alkaa varsinainen työ: käyttöönotto. Koska muutokset testauksen hallinnassa vaikuttavat koko testausprosessiin ja –organisaatioon, on käyttöönotto suunniteltava huolella. Kaikkia osalliset täytyy sitouttaa ja motivoida uuteen prosessiin ja uuden työvälineen käyttöön. Lisäksi heitä täytyy kouluttaa ja tukea paitsi välineen käytössä niin myös uuden työskentelytavan omaksumisessa.

Jos käyttöönottoa ei toteuteta hyvin, saattaa testaushallinan väline jäädä 'hyllyyn pölyttymään', eli sitä ei pystytä hyödyntämään toivotulla tavalla. Siksi se tulee suunnitella huolellisesti, ja pyrkiä mahdollisimman varhaisessa vaiheessa valintaprosessia sitouttamaan välineen loppukäyttäjät myös sen käyttöönottoon.

Asiakastapaus

Eräällä asiakkaallamme todettiin tarve testauksen hallinnan välineelle. Valintaprosessi aloitettiin kartoittamalla vaatimukset välineelle. Näitä olivat:

- yhteensopivuus käytössä olevan käyttöjärjestelmän kanssa
- testausprosessin järjestelmällinen hallinta
- testitapausten organisointi projekteihin
- testauksen aikataulutus ja seuranta
- testauksen mittarit
- integroituminen versionhallintaan, virheraportointikäytäntöön ja projektinhallinta menetelmään

Arviointiin otettiin mukaan 11 työkalua: Mercury Test Director, Rational Test Manager, Compuware QA Director, Segue SilkRadar, VEReCOMM TestExpert, T-Plan Professional, Hughes Software Systems HyperTest, Quality Systems International TracQA, Bluedawn Computer Services Test Control and Management Database, Software Research SMARTS/MSW ja TestMasters Test Management System. Vertailu toteutettiin yksinkertaisella Excel-taulukolla, johon laitettiin pystyyn eri vaatimukset ja vaakaan eri työkalut, jonka jälkeen kunkin työkalun kohdalle merkittiin, miten hyvin työkalu vastasi kyseiseen vaatimukseen. Arviointi pohjautui työkalusta saatavilla oleviin tietoihin (web-sivut, esitteet, myyjän esittelyt).

> ”Kun testauksen hallinnan väline on valittu ja hankittu, alkaa varsinainen työ: käyttöönotto”

Vaihtoehdoista seitsemän karsiutui pois koska ne eivät vastanneet kaikkiin vaatimuksiin. Jäljelle jääneitä neljää työkalua tutkittiin ja kokeiltiin tarkemmin (TestDirector, Test Manager, TestExpert ja T-Plan Professional). Kokeilujen pohjalta löytettiin parhaiten tarpeisiin sopiva työkalu, Mercury TestDirector, jonka asiakas sitten hankki. Väline asennettiin asiakkaalle, siihen tehtiin tarvittavat mukautukset ja välineen käyttö koulutettiin käyttäjille. Testauksen hallintaprosessit mukautettiin uuteen välineeseen, ja käyttökokemukset ovat olleet hyvät.

Sami Kallio, Senior Testing Consultant, Conformiq Software Oy, sami.kallio@conformiq.com, +358407403137

Olen toiminut testauksen parissa vuodesta 1998 lähtien, ensin L&H Finlandissa, sitten F-Secure Oyj:ssä ja nyt Conformiq Software Oy:ssä. Olen myös toiminut kurssiassistenttina TKK:n Ohjelmistojen testaus ja laadunvarmistus –kurssilla.

Conformiq Software Oy on ohjelmistotestauksen ja laadunvarmistuksen asiantuntija. Yritys toimii testauksen ja laadunvarmistuksen etulinjassa tuodakseen laatutietoisille asiakkailleen lisäarvoa ja kilpailukykyä (www.conformiq.com).

YHDISTYS

Palautetta uudistuneesta ulkoasusta

Pidät kädessäsi ulkoasun puolesta uudistunutta systeemityölehteä – sisältö on toki edelleen totutun laadukkaita ammattilaiselta ammattilaiselle artikkeleita.

Uudistuksen taustalla on lehtitoimikunta, joka mielellään kuulisi myös jäsenistön palautetta. Palaute on tervetullutta päätoimittaja Lauri Laitiselle osoitteeseen Lauri.Laitinen@nokia.com

Sytyttääkö? - Liity jäseneksi

Systeemityöyhdistyksen jäseneksi liitytään Tietotekniikan liiton kautta (http://www.tt-tori.fi/, (09) 4765 8530, jasenasiat@ttlry.fi) valitsemalla jäsenyhdistykseksi Systeemityöyhdistys ry. Nykyinen Tietotekniikan liiton jäsen voi liittyä joko vaihtamalla jäsenyhdistystä tai liittymällä lisäjäseneksi.

Tietotekniikan liiton henkilöjäsenmaksu vuonna 2005 on alkaen 50 €, erityisryhmien hinnoittelusta lisätietoja Tietotekniikan liitosta. Lisäjäsenyys maksaa 11 €/yhdistys.

Jäsen - päivitä yhteystietosi liittoon

Muistathan, että jäsenkirjeemme on nykyisin sähköinen. Pidäthän sähköpostiosoitteesi ajan tasalla. Sait tammikuun alussa Tietotekniikan liitto ry:n lähettämässä jäsenlaskussa ohjeet tietojen päivitykseen TT-torilla. Ongelmatilanteissa saat apua sähköpostitse jasenasiat@ttlry.fi ja puh. (09) 4765 8530.

Osaamisyhteisöt

Systeemityöyhdistyksessä toimitaan niin yhdistystasolla kuin aihepiireittäin erikoistuneissa osaamisyhteisöissä. Monipuolisessa tarjonnassamme löytyy jokaiselle jotakin. Vaihtoehtona on myös perustaa omalle kiinnostukselleen uusi osaamisyhteisö - SYTYKE-hallitus toivottaa toimintaehdotukset tervetulleeksi. Osaamisyhteisön toimintaan pääset mukaan laittamalla postia vetäjälle.

Osaamisyhteisötarjontaan kuuluu:

ProjektiOSY— PrOSY pyrkii yhdistämään Suomen projektitoiminnasta ja sen kehittämisestä kiinnostuneet, vetäjänä Markku Niemi, markku.niemi@sttf.fi.

TestausOSY – FAST on testauksen keskustelu- ja yhteistyöverkosto, vetäjänä Maaret Pyhäjärvi, maaret.pyhajarvi@iki.fi.

JavaOSY— JavaSIG on Javan käyttäjien ja harrastajien intressiryhmä, vetäjänä Simo Vuorinen, simo.vuorinen@tietoenator.com.

DAMA Finland keskittyy tiedon, informaation ja tietämyksen hallintaan. Suomen osaston johtoryhmän (boardin) vetäjänä Pekka Valta, yhteyshenkilönä Minna Oksanen, minna.oksanen@capgemini.com.

ViestintäOSY järjestää yhteistoimintaa viestintäsovellusten alueella, vetäjänä Lea Virtanen, lea.virtanen@isoworks.fi

RELA keskittyy relaatiotietokantoihin, vetäjänä Marja Kärmeniemi, marja.karmeniemi@oracle.com

OlioOSY kehittää, tukee ja voimistaa oliomallilähtöistä sovelluskehittämistä, vetäjänä Jukka Tamminen, jukka.tamminen@pp.inet.fi

SYTYKE ry on vuodesta 1987 toiminut valtakunnallinen systeemityöntekijöiden ammatillinen yhdistys, joka kehittää alan ammattilaisten välistä yhteistyötä ja tutkimustoimintaa.

Teemayhdistyksen jäseneksi voivat liittyä kaikki systeemityöstä kiinnostuneet yksityiset henkilöt, yhdistykset ja yritykset. SYTYKE ry:n toiminta-alueena on koko Suomi. SYTYKE on Tietotekniikan liitto Ry:n jäsenyhdistys.

Lisätietoja SYTYKE ry:stä: http://www.sytyke.org/

Johtokunta 2005

SYTYKE ry:n johtokunnan sähköpostilista: sytyke-hallitus@pcuf.fi

www-sivut: http://www.pcuf.fi/sytyke

Helena Venäläinen (puheenjohtaja)
FD Finanssidata Oy,
Teollisuuskatu 1 B, PL 308,
00101 Helsinki,
Puh. (09) 404 3690; +358 50 568 6690
helena.venalainen@op.fi

Kati Ahlgrén
NOKIA OYJ
Karaportti 2, 02610 Espoo
PL 372, FIN-00045 Nokia Group, Finland
Puh: +358504860036;
Fax:+358718024280
ext-kati.ahlgren@nokia.com

Pirkko Leivo
Pohjolan Systeemipalvelu Oy,
00013 Pohjola
Tel +358 10 559 2786
Mob +358 50 567 9352
Fax +358 10 559 3906
pirkko.leivo@pohjola.fi

Markku Niemi
STTF Oy (Software Technology Transfer Finland Oy)
Tekniikantie 14, 02150 Espoo
Puh. 050 – 51 24 687
Fax 09 – 80 45 13 33
markku.niemi@sttf.fi

Tarja Raussi
Tieturi Oy
HTC Santa Maria,
FIN-00180 Helsinki
Tel. +358 (0)9 431 551
Fax +358 (0)9 4315 5302
tarja.raussi@tieturi.fi

Jori Räty
SysOpen Plc
Hiomotie 19, FIN 00380 Helsinki
mobile: +358 50 551 5152
www.sysopen.fi
jori.raty@sysopen.fi

Simo Vuorinen
TietoEnator Oyj,
PL 40, 02101 Espoo,
Puh. 09- 86 25 27 32,040-52 48 36
simo.vuorinen@tietoenator.com

Minna Oksanen (varajäsen)
Capgemini Finland Oy
Niittymäentie 9
02200 ESPOO
puh: 040-577 6640
Minna.Oksanen@capgemini.com

Erkki Pöyhönen (varajäsen)
Nokia Research Center, Software Technology Laboratory
P.O. Box 407, FIN-00045 NOKIA GROUP, Finland
Phone +358 9 4376 7595,
Fax +358 9 4376 6855
erkki.poyhonen@nokia.com

Liittokokous-edustajat

Lauri Laitinen
lauri.laitinen@nokia.com

Silja Räisänen
silja.raisanen@pohjola.fi

Helena Venäläinen
helena.venalainen@op.fi

Simo Vuorinen
simo.vuorinen@tietoenator.com

Toimisto

Puhelinvastaus- ja sihteeripalvelu VT Oy/Susanna Koskinen
Systeemityöyhdistys Sytyke ry
Henrikintie 7 A, 00370 Helsinki
p. 09-5607 5363
f. 09-5607 5365
sytyke@hennax.fi

Oppeja automaatiotestaus-työpajasta

Testauksen osaamisyhteisön piirissä pidettiin joulukuussa 2004 ensimmäinen ohjelmistotestauksen työpaja, teemalla ”Onnistunut testiautomaatio”. Mukana oli asiantunteva edustus erilaisten organisaatioiden, tarpeiden ja välinekokemusten osalta.

Testiautomaatiotyöpajassa olivat mukana:

- Maaret Pyhäjärvi, testausOSY vetäjä
- Erkki Pöyhönen, Nokia
- Tomi Kaleva, Tietokarhu
- Risto Kumpulainen, F-Secure Corporation
- Tuula Pääkkönen, Nokia
- Pekka Laukkanen, Qentinel
- Juha Saaristo, Nice Business Solutions
- Marko Komssi, F-Secure Corporation
- Mika Katara, Tampereen teknillinen yliopisto
- Petri Kuikka, F-Secure Corporation
- Paavo Häkkinen, Compuware
- Jouni Hynynen, SoftaTest

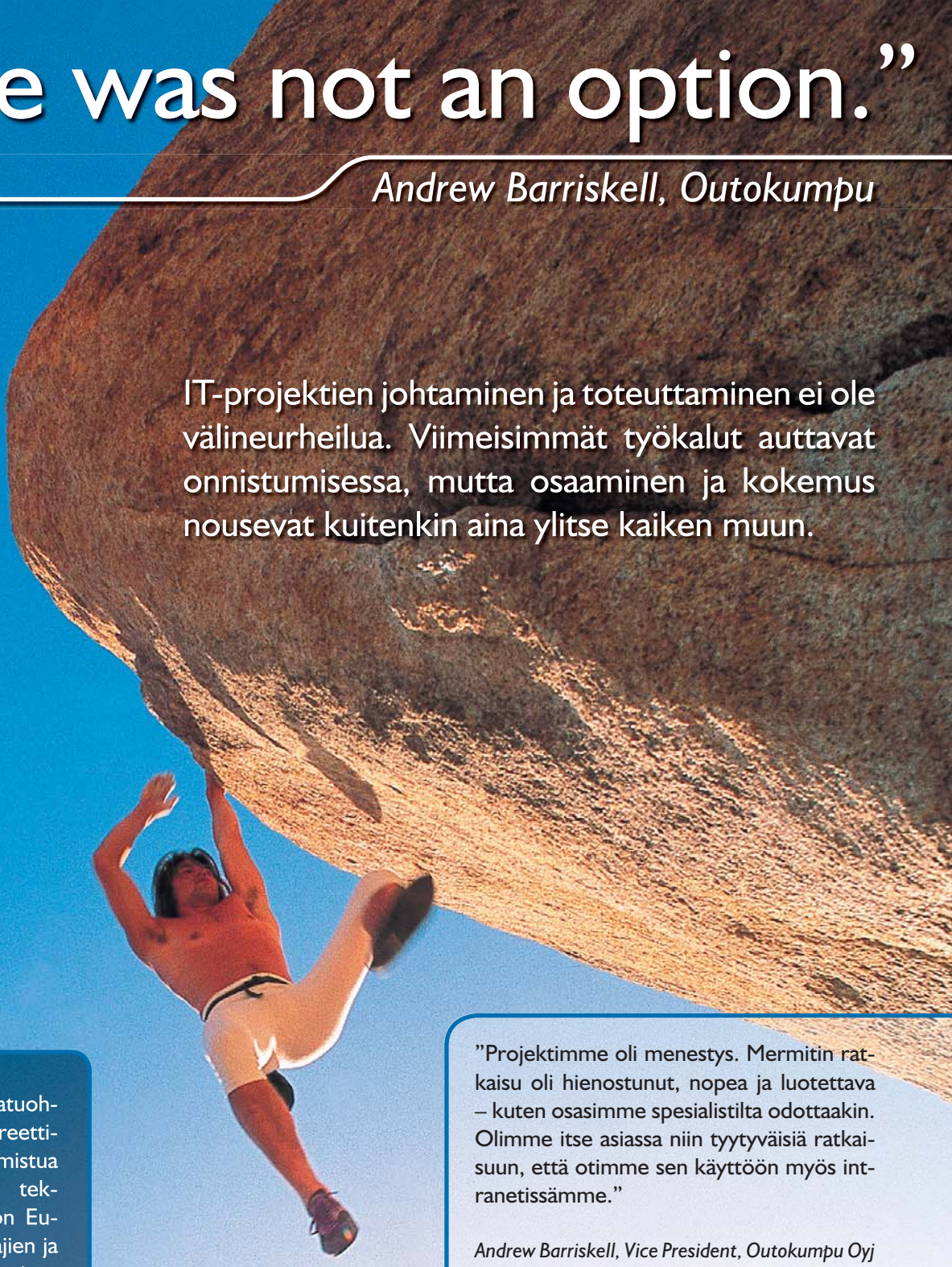
Työpajassa keskityttiin pohtimaan testien suorituksen automatisointiin liittyviä onnistumisen kokemuksia. Ryhmässä löytyi onnistuneita kokemuksia niin käyttöliittymän läpi tehtävän automatisoinnin kuin ohjelmointirajapinta-automatisoinnin osalta, joskin kokemusten sävy tuntui astetta positiivisemmalta ohjelmointirajapinnan läpi tapahtuvan automaation osalta. Erityisen positiivisena asiana pidettiin etenkin kokemusta, että rajapintojen läpi voi automatisoida aikaisemmin kun taas käyttöliittymän pitää olla vakiintunut ennen kuin automatisointi kannattaa. Yleisesti automaatiolla ei juurikaan löydetä virheitä, mutta onnistumisen kriteerinä pidettiin myös sitä että tiedetään että ainakaan tietyn tyyppisiä virheitä ei ole. Onnistumisessa yleensä keskeisenä tekijänä oli järkevä ja käytännönläheinen tekeminen – hyötyjä on saatava lyhyellä aikavälillä on onnistumisen keskeinen edellytys.

Työpajan tulokset ovat jaossa testausOSY:n websivujen kautta, käy siis katsastamassa http://www.sytyke.org/kerhot/testaus/

“Failure was not an option.”

Andrew Barriskell, Outokumpu

IT-projektien johtaminen ja toteuttaminen ei ole välineurheilua. Viimeisimmät työkalut auttavat onnistumisessa, mutta osaaminen ja kokemus nousevat kuitenkin aina ylitse kaiken muun.



Mermitin sertifiointi- ja laatuohjelma tarjoaa sinulle konkreettisen mahdollisuuden varmistua IT-hankkeesi toteuttajan teknisestä tasosta. Mermit on Euroopan kärkeä J2EE –osaajien ja sertifioitujen partnereiden joukossa.

Mermitin laaja-alainen osaaminen kattaa mm. IBM WAS, DB2 ja MQ tuoteteknologiaan perustuvat integraatiot, arkkitehtuurit ja sovelluskehityksen.

”Projektimme oli menestys. Mermitin ratkaisu oli hienostunut, nopea ja luotettava – kuten osasimme spesialistilta odottaakin. Olimme itse asiassa niin tyytyväisiä ratkaisuun, että otimme sen käyttöön myös intranetissämme.”

Andrew Barriskell, Vice President, Outokumpu Oyj

“Työskentely Mermitin kanssa oli joustavaa ja toimivaa läpi koko projektin...Huolimatta ratkaisun suuresta räätälöintiasteesta kehitystyö oli ripeää ja kustannustehokasta. Olemme erittäin tyytyväisiä lopputulokseen.”

Rainer Hiltunen, Ylitarkastaja, Vähemmistövaltuutetun toimisto, Työministeriö

Ota yhteyttä ja kysy kattavasta laatutakuustamme!

Mermit Business Applications Oy
09 - 540 4010
info@mermit.fi

MERMIT
Mermit Business Applications Oy
www.mermit.fi

Testaus vaatii monipuolista osaamista. Usein testauksen ammattilaiset erikoistuvatkin perusasioiden hallinnan lisäksi jollekin testauksen erityisalueelle. Normaalin toiminnallisen testauksen lisäksi tarvitaan mm. suorituskyvyn, käytettävyyden ja tietoturvan testaamista. Erityyppisiä taitoja tarvitaan myös testauksen eri tasoilla yksikkötestauksesta hyväksymistestaukseen asti.

Tieturi tarjoaa testausvalmennuksen lisäksi myös mentorointia ja konsultointia testausmenetelmien kehittämiseen, kuormitustestaukseen ja toiminnalliseen testaukseen. Mikäli kurssikalenterimme ajankohdat eivät osu sopivaan ajankohtaan, ota yhteyttä.

Lisätietoja
www.tieturi.fi/testaus
www.tieturi.fi/systeemityo
Osastopäällikkö Päivi Hietanen
(09) 4315 5664, paivi.hietanen@tieturi.fi
Systeemityön asiantuntija Teppo Heikurinen
050 518 4763, teppo.heikurinen@tieturi.fi
Myyntijohtaja Marita Heinström
(09) 4315 5260, marita.heinstrom@tieturi.fi

Testausmenetelmillä laadukkuutta testaukseen
- Järjestelmän suorituskyvyn testaus 31.3.–1.4.
- Testauksen hallinta 4.4.
- Ohjelmiston käytettävyyden testaus ja arviointi 5.4.
- Testaus järjestelmän tilaajalle 11.4.
- Testaus ohjelmistokehittäjälle 18.4. ja 6.6.
- Sulautetun ohjelmiston testaus 22.4.
- Testauksen valmennusohjelma 9.–11.5.
- Web-sovellusten testaus 16.5.
- JUnit: tehokäyttö testauksessa 7.–8.6.

Testausvälineistä apua testaukseen
- QuickTest-perusteet 9.–10.5. **UUSI!**
- QualityCenter ohjelmistotestauksessa 17.5. **UUSI!**
- Kuormitustestauksen perusteet (LoadRunner) 23.–24.5.
- Kuormitustestien nauhoittaminen (LoadRunner) 25.–26.5.
- QualityCenter-ohjelmiston administrointi 13.6. **UUSI!**
- QuickTest-jatkokurssi 16.–17.6. **UUSI!**

Suunnittelulla laatua ohjelmistokehitykseen
- Komponenttikeskeinen ohjelmistosuunnittelu (UML) 29.–30.3.
- Ketterä ohjelmistokehitys 29.–30.3.
- Oliomäärittely ja -suunnittelu (UML 2.0) 4.–6.4.
- Integroitu ohjelmistokehitys 7.4. **UUSI!**
- Suunnittelijan työn tehostaminen 11.4.

Tieturi
ELÄMÄ ON OPPIMISTA

Ilmoittautumiset: (09) 4315 5333 ■ kurssit@tieturi.fi ■ www.tieturi.fi