Sertifioitu Testaaja

Jatkotason sertifikaattisisältö
Tekninen testausasiantuntija

Versio 2012
Käännösversio 2013

International Software Testing Qualifications Board

Jatkotaso, Tekninen testausasiantuntija alatyöryhmä Graham Bath (puheenjohtaja), Paul Jorgensen, Jamie Mitchell; 2010 - 2012

Muutoshistoria

Version	Date	Remarks
ISEB v1.1	04SEP01	ISEB Practitioner Syllabus
ISTQB 1.2E	SEP03	ISTQB Advanced Level Syllabus from EOQ-SG
V2007	12OCT07	Certified Tester Advanced Level syllabus version 2007
D100626	26JUN10	Incorporation of changes as accepted in 2009, separation of each chapters for the separate modules
D101227	27DEC10	Acceptance of changes to format and corrections that have no impact on the meaning of the sentences.
Draft V1	17SEP11	First version of the new separate TTA syllabus based on agreed scoping document. AL WG Review
Draft V2	20NOV11	NB Review version
Alpha 2012	09MAR12	Incorporation of all comments from NBs received from October release.
Beta 2012	07APR12	Beta Version submitted to GA
Beta 2012	08JUN12	Copy edited version released to NBs
Beta 2012	27JUN12	EWG and Glossary comments incorporated
RC 2012	15AUG12	Release candidate version - final NB edits included
RC 2012	02SEP12	Comments from BNLTB and Stuart Reid incorporated Paul Jorgensen cross-check
GA 2012	19OCT12	Final edits and cleanup for GA release
ISTQB FIN 1.0	19.11.2013	Ensimmäinen hyväksytty versio

Sisällysluettelo

Kiitokset

Tämän dokumentin on tuottanut International Software Testing Qualifications Boardin Jatkotason työryhmän Tekninen testausasiantuntija –alatyöryhmä: Graham Bath (puheenjohtaja), Paul Jorgensen, Jamie Mitchell.

Ydintiimi kiittää katselmointitiimiä ja kansallisia toimikuntia (hallituksia) näiden ehdotuksista ja työpanoksesta.

Jatkotason sertifikaattisisällön valmistumishetkellä Jatkotason työryhmään kuuluivat seuraavat jäsenet (aakkosjärjestyksessä):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (varapuheenjohtaja), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (puheenjohtaja), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

Seuraavat henkilöt osallistuivat tämän sertifikaattisisällön katselmointiin, kommentointiin ja siihen liittyviin äänestyksiin:

Dani Almog, Graham Bath, Franz Dijkman, Erwin Engelsma, Mats Grindal, Dr. Suhaimi Ibrahim, Skule Johansen, Paul Jorgensen, Kari Kakkonen, Eli Margolin, Rik Marselis, Judy McKay, Jamie Mitchell, Reto Mueller, Thomas Müller, Ingvar Nordstrom, Raluca Popescu, Meile Posthuma, Michael Stahl, Chris van Bael, Erik van Veenendaal, Rahul Verma, Paul Weymouth, Hans Weiberg, Wenqiang Zheng, Shaomin Zhu.

Tämä dokumentti julkaistiin virallisesti ISTQB®:n Yleiskokouksessa 19. lokakuuta 2012.

Suomenkielisen version toteutukseen ja katselmointiin ovat osallistuneet seuraavat henkilöt:
Minna Aalto, Kari Kakkonen, Juha Pomppu, Marko Rytkönen, Laura Selonen.

0. Johdatus tähän sertifikaattisisältöön

0.1 Tämän dokumentin tarkoitus

Tämä sertifikaattisisältö muodostaa pohjan Teknistä testausasiantuntijaa koskeville International Software Testing Qualification –sertifioinnin Jatkotason vaatimuksille. ISTQB® on toimittanut sertifikaattisisällön seuraavia tarkoituksia varten:

1. Kansallisille toimikunnille (hallituksille) käännettäväksi paikalliselle kielelle sekä koulutustarjoajien akkreditointia varten. Kansalliset toimikunnat (hallitukset) voivat muokata sertifikaattisisältöä tietyn kielen tarpeiden mukaisesti sekä muokata viitteitä vastaamaan paikallisia julkaisuja.
2. Koelautakunnille: Jokaisen sertifikaattisisällön oppimistavoitteita vastaavien tutkintokysymysten tuottamiseksi paikallisella kielellä.
3. Koulutustarjoajille: koulutusmateriaalin tuottamiseen sekä soveltuvien opetustapojen valitsemiseksi.
4. Sertifiointikokelaille: tutkintoon valmistautumista varten (joko osana valmennuskurssia tai itsenäisesti).
5. Kansainväliselle ohjelmisto- ja järjestelmäkehityksen yhteisölle: ohjelmisto- ja järjestelmätestauksen ammatin edistämiseksi sekä kirjojen ja artikkeleiden pohjamateriaaliksi.

ISTQB® voi antaa myös muille yhteisöille luvan käyttää tätä sertifiointisisältöä muihin tarkoituksiin edellyttäen, että sille haetaan etukäteen kirjallinen suostumus..

0.2 Yleiskatsaus

Jatkotaso muodostuu kolmesta erillisestä sertifikaattisisällöstä:

- Testauspäällikkö
- Testausasiantuntija
- Tekninen testausasiantuntija

Dokumentti Jatkotason yleiskatsaus [ISTQB_AL_OVIEW] sisältää seuraavat tiedot:

- Jokaisen sertifikaattisisällön liiketoiminnalliset tavoitteet
- Jokaisen sertifikaattisisällön yhteenveto
- Sertifikaattisisältöjen väliset yhteydet
- Oppimistasojen (K-tasojen) kuvaus
- Liitteet

0.3 Kuulusteltavat oppimistavoitteet

Oppimistavoitteet tukevat liiketoiminnan tavoitteita ja niitä käytetään laadittaessa koetta Jatkotason Teknisen asiantuntijan sertifioinnin saavuttamiseksi. Yleisesti ottaen kaikki tämän sertifikaattisisällön osat ovat kuulusteltavissa K1-tason kysymyksillä. Se tarkoittaa, että kokelas tunnistaa, muistaa ja pystyy palauttamaan mieleensä termin tai käsitteen. K2-, K3- ja K4-tasojen oppimistavoitteet on kuvattu kyseessä olevan luvun alussa.

0.4 Odotukset

Eräät Tekninen testausasiantuntija-sertifikaatin oppimistavoitteista lähtevät oletuksesta, että kokelaalla on peruskokemus seuraavista alueista:

- Yleiset ohjelmointikäsitteet
- Yleiset järjestelmäarkkitehtuurin käsitteet

1. Teknisen testausasiantuntijan tehtävät riskipohjaisessa testauksessa – 30 min

Avainsanat

riskianalyysi, riskien arviointi, riskien hallinta, riskien tunnistaminen, riskipohjainen testaus, riskitaso, tuoteriski

Oppimistavoitteet – Teknisen testausasiantuntijan tehtävät riskipohjaisessa testauksessa

1.3 Riskien arviointi

TTA-1.3.1 (K2) Esittää lyhyesti yleiset riskitekijät, joita Teknisen testausasiantuntijan yleensä täytyy ottaa huomioon

Yleiset oppimistavoitteet

Seuraavat oppimistavoitteet liittyvät asioihin, joita on käsitelty useammassa kuin yhdessä tämän luvun osassa.

TTA-1.x.1 (K2) Vetää yhteen Tekniselle testausasiantuntijalle kuuluvat testauksen suunnitteluun ja suorittamiseen liittyvät tehtävät riskipohjaista lähestymistapaa käytettäessä.

1.1 Esittely

Testauspäälliköllä on kokonaisvastuu riskipohjaisen testausstrategian käyttöönotosta ja hallinnasta. Testauspäällikkö pyytää yleensä Teknistä testausasiantuntijaa osallistumaan tehtäviin varmistaakseen, että riskipohjaista lähestymistapaa toteutetaan oikein.

Teknisen erityisasiantuntemuksensa vuoksi Tekniset testausasiantuntijat osallistuvat aktiivisesti seuraaviin riskipohjaisen testauksen tehtäviin:

- Riskien tunnistaminen
- Riskien arviointi
- Riskien hallinta

Näitä tehtäviä suoritetaan iteratiivisesti koko projektin ajan esiin nousevien tuoteriskien ja muuttuvien prioriteettien hallitsemiseksi sekä riskien tilan säännöllisessä arvioinnissa ja siitä tiedottamisessa.

Tekniset testausasiantuntijat työskentelevät Testauspäällikön projektille laatiman riskipohjaisen testauksen lähestymistavan puitteissa. He tuovat mukanaan tietämyksensä projektille luontaisista teknisistä riskeistä, joita ovat mm. tietoturvaan, järjestelmän luotettavuuteen ja suorituskykyyn liittyvät riskit.

1.2 Riskien tunnistaminen

Riskien tunnistamisprosessissa löydetään todennäköisimmin suurin mahdollinen määrä merkittäviä riskejä, kun mukaan otetaan laajin mahdollinen joukko eri sidosryhmien edustajia. Koska Teknisillä testausasiantuntijoilla on ainutlaatuisia teknisiä taitoja, he sopivat erityisen hyvin haastattelemaan asiantuntijoita, osallistumaan aivoriihiin muiden työntekijöiden kanssa, sekä analysoimaan nykyisiä ja aiempia kokemuksia todennäköisten tuoteriskejä sisältävien alueiden määrittämiseksi. Erityisen tiiviisti Tekniset testausasiantuntijat työskentelevät teknisten kollegoidensa (esim. kehittäjät, arkkitehdit, operaattorit) kanssa teknisiä riskejä sisältävien alueiden määrittämiseksi.

Mahdollisesti tunnistettaviin riskeihin kuuluvat esimerkiksi:

- Suorituskykyyn liittyvät riskit (esim. vasteaikojen saavuttamatta jääminen korkean kuormituksen alla)
- Tietoturvariskit (esim. arkaluontoisten tietojen paljastuminen tietoturvahyökkäyksissä)
- Luotettavuusriskit (esim. sovellus ei pysty täyttämään palvelutasosopimuksessa määritettyä saavutettavuustasoa)

Tiettyihin ohjelmiston laatuominaisuuksiin liittyviä riskialueita käsitellään tämän sertifikaattisisällön niihin liittyvissä luvuissa.

1.3 Riskien arviointi

Kun riskien tunnistaminen keskittyy mahdollisimman monen oleellisen riskin tunnistamiseen, riskien arviointi tarkoittaa tunnistettujen riskien tutkimista jokaisen riskin luokittelemiseksi sekä riskiin liittyvän todennäköisyyden ja vaikutuksen määrittämiseksi.

Riskitason määrittämiseen kuuluu tyypillisesti riskin toteutumisen todennäköisyyden sekä toteutumisesta seuraavan vaikutuksen arviointi, joka tehdään jokaiselle riskiaiheelle. Riskin toteutumisen todennäköisyydellä tarkoitetaan yleensä todennäköisyyttä, että kyseinen ongelma voi esiintyä testattavassa järjestelmässä.

Tekninen testausasiantuntija auttaa löytämään ja ymmärtämään jokaiseen riskiaiheeseen liittyvät mahdolliset tekniset riskit, kun taas Testausasiantuntija auttaa ymmärtämään ongelman toteutumisesta seuraavia mahdollisia liiketoiminnallisia vaikutuksia.

Tyypillisiä arviointia vaativia tekijöitä ovat:

- Teknologian monimutkaisuus
- Koodirakenteen monimutkaisuus
- Sidosryhmien väliset teknisiä vaatimuksia koskevat ristiriidat
- Kehitysorganisaation maantieteellisestä levinneisyydestä johtuvat viestintäongelmat
- Työkalut ja teknologia
- Aika, resurssit ja johdon taholta tulevat paineet
- Aikaisemman laadunvarmistuksen puute
- Teknisten vaatimusten korkeat muutosmäärät
- Teknisiin laatuominaisuuksiin liittyvien vikojen suuri määrä
- Tekniset liittymä- ja integrointiasiat

Riskeistä saatavilla olevan tiedon perusteella Tekninen testausasiantuntija määrittelee riskien tason Testauspäällikön laatiman ohjeistuksen mukaisesti. Esimerkiksi, Testauspäällikkö voi määrätä, että riskit pitää luokitella käyttäen arvoja väliltä 1 – 10, jossa 1 on suurin riski.

1.4 Riskien hallinta

Projektin aikana Tekninen testausasiantuntija vaikuttaa siihen, miten testaus reagoi tunnistettuihin riskeihin. Yleensä tähän kuuluvat seuraavat asiat:

- Riskien vähentäminen suorittamalla kaikkein tärkeimmät testit ja käynnistämällä testausstrategiassa ja testaussuunnitelmassa määritellyt riskien pienentämistehtävät ja niihin liittyvät varotoimenpiteet.
- Riskien arviointi projektin edetessä esiin tulevan tiedon perusteella ja saadun tiedon käyttäminen aikaisemmin tunnistettujen ja analysoitujen riskien todennäköisyyden tai vaikutuksen pienentämiseen tähtäävissä toimenpiteissä.

2. Rakennepohjainen testaus – 225 min

Avainsanat
atominen ehto, ehtotestaus, kontrollivuotestaus, lausetestaus, moniehtotestaus, oikosulkeminen, polkutestaus, päätösehtotestaus, rakenteeseen pohjautuva tekniikka

Oppimistavoitteet: Rakennepohjainen testaus

2.2 Ehtotestaus
TTA-2.2.1 (K2) Ymmärtää, kuinka ehtokattavuus saavutetaan, ja miksi se ei ehkä ole niin perusteellista testausta kuin päätöskattavuus.

2.3 Päätösehtotestaus
TTA-2.3.1 (K3) Kirjoittaa testitapauksia määritellyn kattavuustason saavuttamiseksi käyttämällä päätösehtotestaus-testisuunnittelutekniikkaa.

2.4 Täydennetty ehtotestaus
TTA-2.4.1 (K3) Kirjoittaa testitapauksia määritellyn kattavuustason saavuttamiseksi käyttämällä täydennetty ehtotestaus-testisuunnittelutekniikkaa.

2.5 Moniehtotestaus
TTA-2.5.1 (K3) Kirjoittaa testitapauksia määritellyn kattavuustason saavuttamiseksi käyttämällä moniehtotestaus-testisuunnittelutekniikkaa.

2.6 Polkutestaus
TTA-2.6.1 (K3) Kirjoittaa testitapauksia käyttämällä polkutestaus-testisuunnittelutekniikkaa.

2.7 API Testaus (ohjelmointirajapintatestaus)
TTA-2.7.1 (K2) Ymmärtää, milloin API-testausta voidaan käyttää, ja minkälaisia vikoja se löytää.

2.8 Rakennepohjaisen tekniikan valinta
TTA-2.8.1 (K4) Valita kulloisenkin projektin tilanteen perusteella siihen soveltuva rakennepohjainen testaustekniikka.

2.1 Esittely

Tässä luvussa esitellään pääasiassa rakennepohjaisia testaustekniikoita, jotka tunnetaan myös lasilaatikkotekniikoina tai koodipohjaisina testaustekniikoina. Nämä tekniikat käyttävät koodia, aineistoa sekä arkkitehtuuria ja/tai järjestelmän toiminnallista kulkua testisuunnittelun pohjana. Jokainen yksittäinen tekniikka mahdollistaa testitapausten järjestelmällisen laatimisen ja keskittyy käsiteltävänä olevaan rakenteeseen tietystä näkökulmasta. Tekniikat tarjoavat kattavuuskriteereitä, joita täytyy mitata ja jotka täytyy yhdistää jokaisen projektin tai organisaation määrittelemiin tavoitteisiin. Täyden kattavuuden saavuttaminen ei tarkoita sitä, että koko testijoukko on valmis, vaan ennemminkin sitä, että käytetty tekniikka ei enää tarjoa hyödyllisiä testejä kohteena olevan rakenteen testaukseen.

Ehtokattavuutta lukuun ottamatta tässä sertifikaattisisällössä käsiteltävät rakennepohjaiset testisuunnittelutekniikat ovat perusteellisempia kuin Perustason sertifikaattisisällössä [ISTQB_FL_SYL] käsitellyt lause- ja päätöskattavuustekniikat.

Tässä sertifikaattisisällössä käsitellään seuraavia tekniikoita:

- Ehtotestaus
- Päätösehtotestaus
- Täydennetty ehtotestaus
- Moniehtotestaus
- Polkutestaus.
- API-testaus

Yllä luetelluista tekniikoista ensimmäiset neljä pohjautuvat päätöselementteihin ja yleisesti ottaen löytävät samantyyppisiä vikoja. Riippumatta siitä, kuinka monimutkainen päätösmuuttuja on, sen arvoksi tulee joko TOSI tai EPÄTOSI, jolloin suoritetaan toinen polku koodin läpi ja toista ei. Vika löytyy silloin, kun ohjelman suoritus ei etene aiottua polkua siksi, että monimutkainen päätösmuuttuja ei saa odotettua arvoa.

Yleisesti ottaen neljä ensimmäistä tekniikkaa on lueteltu perusteellisuusjärjestyksessä; niille tavoitteeksi asetetun kattavuuden saavuttamiseksi tarvitaan enemmän testejä, samoin kuin jotta ne löytävät vaikeammin havaittavia tämän tyyppisiä vikoja.

Viitteet: [Bath08], [Beizer90], [Beizer95], [Copeland03] ja [Koomen06].

2.2 Ehtotestaus

Päätöstestaus (haaratestaus) tutkii koko päätöskohtaa kokonaisuutena ja laskee TOSI- ja EPÄTOSI-tulokset erillisiksi testitapauksiksi, kun taas ehtotestaus tutkii, kuinka päätös tehdään. Jokainen päätösmuuttuja muodostuu yhdestä tai useammasta atomisesta ehdosta, joista jokainen saa diskreetin Boolen arvon. Nämä yhdistetään loogisesti päätöksen lopullisen tuloksen määrittämiseksi. Testitapausten täytyy tutkia jokainen atominen ehto niiden kummallakin tavalla, jotta tämän tason kattavuus saavutetaan.

Sovellettavuus
Ehtotestaus on yleensä kiinnostava vain abstraktina käsitteenä alla kerrottujen vaikeuksien vuoksi. Sen ymmärtäminen on kuitenkin tarpeellista siihen pohjautuvien vaativampien kattavuustasojen saavuttamiseksi.

Rajoitukset/vaikeudet
Kun päätöskohdassa on kaksi tai useampaa atomista ehtoa, testisuunnittelun aikana tehty väärä testiaineiston valinta voi johtaa siihen, että saavutetaan ehtokattavuus mutta ei päätöskattavuutta. Esimerkki: Oletetaan, että päätöstekijä on “A ja B”.

	A	B	A ja B
Testi 1	EPÄTOSI	TOSI	EPÄTOSI
Testi 2	TOSI	EPÄTOSI	EPÄTOSI

100 % ehtokattavuuden saavuttamiseksi suoritetaan yllä olevassa taulukossa esitetyt testit. Vaikka nämä kaksi testiä saavuttavat 100 % ehtokattavuuden, ne eivät saavuta päätöskattavuutta, sillä molemmissa tapauksissa päätöstekijä saa arvon EPÄTOSI.

Kun päätös muodostuu yksittäisestä atomisesta ehdosta, ehtotestaus on identtinen päätöstestauksen kanssa.

2.3 Päätösehtotestaus

Päätösehtotestaus määritellään niin, että testauksen pitää saavuttaa ehtokattavuus (ks. yllä) ja lisäksi vaaditaan, että päätöskattavuus (ks. Perustason sertifikaattisisältö [ISTQB_FL_SYL]) täyttyy. Huolellisesti tehty atomisten ehtojen testiaineiston valinta voi tuottaa tämän kattavuustaso saavuttamisen ilman uusien testitapausten lisäämistä lukuun ottamatta tapauksia, joita tarvitaan ehtokattavuuden saavuttamiseksi.

Alla olevassa esimerkissä testataan samaa päätöstekijää kuin yllä, “A ja B”. Päätösehtokattavuus voidaan saavuttaa samalla testimäärällä valitsemalla testeille eri arvot.

	A	B	A ja B
Testi 1	TOSI	TOSI	TOSI
Testi 2	EPÄTOSI	EPÄTOSI	EPÄTOSI

Siksi tämä tekniikka voi olla tehokkuudeltaan hyödyllisempi.

Sovellettavuus
Tämän tason kattavuutta pitäisi harkita silloin, kun testattava koodi on tärkeää mutta ei kriittistä.

Rajoitukset/vaikeudet
Koska testaus voi vaatia enemmän testitapauksia kuin päätöstason testaus, siitä voi seurata ongelmia, mikäli aikaa on rajoitetusti.

2.4 Täydennetty ehtotestaus

Tämä tekniikka tuottaa voimakkaamman kontrollivuokattavuuden. Jos oletetaan, että atomisten ehtojen määrä on N, täydennetty ehtokattavuus voidaan yleensä saavuttaa N+1:llä yksilöllisellä testitapauksella. Täydennetty ehtokattavuus tuottaa päätösehtokattavuuden, mutta vaatii myös seuraavien ehtojen täyttymistä:

1. Tarvitaan ainakin yksi testi, jossa päätöksen lopputulos muuttuu, jos atominen ehto X on TOSI.
2. Tarvitaan ainakin yksi testi, jossa päätöksen lopputulos muuttuu, jos atominen ehto X on EPÄTOSI.
3. Jokaista atomista ehtoa kohti on testit, jotka täyttävät vaatimukset 1 ja 2.

	A	B	C	(A tai B) ja C
Testi 1	TOSI	EPÄTOSI	TOSI	TOSI
Testi 2	EPÄTOSI	TOSI	TOSI	TOSI
Testi 3	EPÄTOSI	EPÄTOSI	TOSI	EPÄTOSI
Testi 4	TOSI	EPÄTOSI	EPÄTOSI	EPÄTOSI

Yllä olevassa esimerkissä saavutetaan päätöskattavuus (päätöselementin tulos on sekä TOSI että EPÄTOSI) sekä ehtokattavuus (A, B ja C testataan sekä arvolla TOSI että EPÄTOSI).

Testissä 1 A on TOSI ja tulos on TOSI. Jos A muuttuu arvoksi EPÄTOSI (kuten testissä 3, ja muut arvot ovat muuttumattomat), tulos muuttuu arvoksi EPÄTOSI.
Testissä 2 B on TOSI ja tulos on TOSI. Jos B muuttuu arvoksi EPÄTOSI (kuten testissä 3, ja muut arvot ovat muuttumattomat), tulos muuttuu arvoksi EPÄTOSI.

Testissä 1 C on TOSI ja tulos on TOSI. Jos C muuttuu arvoksi EPÄTOSI (kuten testissä 4, ja muut arvot ovat muuttumattomat), tulos muuttuu arvoksi EPÄTOSI.

Sovellettavuus
Tätä tekniikkaa käytetään laajasti ilmailualan ohjelmistoteollisuudessa sekä monissa muissa turvallisuuskriittisissä järjestelmissä. Sitä pitäisi käyttää, kun ollaan tekemissä turvallisuuskriittisten järjestelmien kanssa, joissa häiriö voi johtaa katastrofiin.

Rajoitukset/vaikeudet
Täydennetyn ehtokattavuuden saavuttaminen voi olla monimutkaista silloin, kun tietty muuttuja esiintyy lauseessa useita kertoja; tällaisessa tilanteessa muuttujaa sanotaan ”kytketyksi”. Koodin päätöslauseesta riippuen ei ehkä ole mahdollista muuttaa kytketyn muuttujan arvoja niin, että se yksin aiheuttaisi päätöksen tuloksen muutoksen. Yksi lähestymistapa tähän ongelmaan on määrittää, että vain kytkemättömät atomiset ehdot pitää testata täydennetyllä ehtokattavuustasolla. Toinen lähestymistapa on analysoida tapauskohtaisesti jokainen päätös, jossa kytkentöjä esiintyy.

Jotkut ohjelmointikielet ja kääntäjät on suunniteltu niin, että ne käyttävät oikosulkumenetelmää, kun ne tutkivat monimutkaista päätöslausetta koodissa. Se tarkoittaa, että suoritettava koodi ei välttämättä käy läpi koko lauseketta, jos arvioinnin lopputulos voidaan määritellä jo sen jälkeen, kuin vain osa lausekkeesta on käyty läpi. Esimerkiksi tutkittaessa päätöstä “A ja B” ei ole tarpeen tutkia B:tä, jos jo A saa arvon EPÄTOSI. Mikään B:n arvo ei voi muuttaa lopputulosta, joten koodi voi säästää suoritusaikaa jättämällä B:n tutkimatta. Oikosulkeminen voi vaikuttaa Täydennetyn ehtokattavuuden saavuttamiseen, koska joitain tarvittavia testejä ei välttämättä päästä suorittamaan.

2.5 Moniehtotestaus

Joissain harvinaisissa tapauksissa voidaan joutua testaamaan päätöskohdan mahdollisten arvojen kaikki mahdolliset yhdistelmät. Tätä testauksen täydellistä (täysin kattavaa) tasoa kutsutaan moniehtotestaukseksi. Tarvittavien testitapausten määrä riippuu päätöslauseen sisältämien atomisten ehtojen määrästä ja se voidaan määrittää laskemalla 2^n, missä n on kytkemättömien atomisten ehtojen määrä. Edellä kuvatussa esimerkkitapauksessa tarvitaan seuraavat testit moniehtokattavuuden saavuttamiseksi:

	A	B	C	(A tai B) ja C
Testi 1	TOSI	TOSI	TOSI	TOSI
Testi 2	TOSI	TOSI	EPÄTOSI	EPÄTOSI
Testi 3	TOSI	EPÄTOSI	TOSI	TOSI
Testi 4	TOSI	EPÄTOSI	EPÄTOSI	EPÄTOSI
Testi 5	EPÄTOSI	TOSI	TOSI	TOSI
Testi 6	EPÄTOSI	TOSI	EPÄTOSI	EPÄTOSI
Testi 7	EPÄTOSI	EPÄTOSI	TOSI	EPÄTOSI
Testi 8	EPÄTOSI	EPÄTOSI	EPÄTOSI	EPÄTOSI

Jos ohjelmointikieli käyttää oikosulkemista, todellisten testitapausten määrä pienenee usein riippuen atomisille ehdoille suoritettavien loogisten operaatioiden järjestyksestä ja ryhmittelystä.

Sovellettavuus
Perinteisesti tätä tekniikkaa on käytetty, kun testattiin sulautettuja ohjelmistoja, joiden odotettiin toimivan luotettavasti ja kaatumatta pitkiä ajanjaksoja (esim. puhelinkeskukset, joiden odotettiin toimivan 30 vuotta). Tämän tyyppinen testaus tulee todennäköisesti korvautumaan kaikkein kriittisimmissä sovelluksissa Täydennetyllä ehtotestauksella.

Rajoitukset/vaikeudet
Koska testitapausten määrä voidaan laskea suoraan kaikki atomiset ehdot sisältävästä totuustaulusta, kattavuustaso on helppo määrittää. Testitapausten suuri määrä tekee kuitenkin Täydennetystä ehtokattavuudesta soveltuvamman useimpiin tilanteisiin.

2.6 Polkutestaus

Polkutestauksessa tunnistetaan ensin eri polut ohjelmakoodin läpi ja sen jälkeen suunnitellaan testit kattamaan ne. Periaatteessa olisi hyödyllistä testata jokainen ainutkertainen polku läpi järjestelmän. Kuitenkin kaikissa vähänkin tärkeämmissä järjestelmissä testitapausten määrä voi nousta kohtuuttoman suureksi silmukkarakenteiden luonteen vuoksi.

Jos päättymättömien silmukoiden aiheuttamat ongelmat voidaan sivuuttaa, on kuitenkin järkevää suorittaa jonkin verran polkutestausta. Tämän tekniikan käyttämiseksi [Beizer90] suosittelee laatimaan testejä, jotka seuraava useita polkuja läpi moduulin, alusta loppuun. Mahdollisesti monimutkaisen tehtävän yksinkertaistamiseksi hän ehdottaa asian tekemistä järjestelmällisesti noudattamalla seuraavia toimenpiteitä:

1. Valitse ensimmäiseksi poluksi yksinkertaisin toiminnallisesti järkevä polku alusta loppuun.
2. Valitse lisäpolut niin, että jokainen uusi polku lisää hiukan toiminnallisuutta edelliseen polkuun verrattuna. Yritä muuttaa peräkkäisissä testeissä vain yhtä erilaista polun haaraa kerrallaan. Mikäli mahdollista, suosi lyhyitä polkuja pitkien polkujen sijaan. Suosi toiminnallisuuden kannalta ei-järkevien polkujen sijaan sellaisia polkuja, jotka ovat toiminnallisesti järkeviä.
3. Valitse toiminnallisuuden kannalta ei-järkeviä polkuja vain silloin, mikäli se on tarpeen riittävän kattavuuden saavuttamiseksi. Beizer huomauttaa, että tällaiset polut saattavat olla epäoleellisia ja ne pitäisi siksi kyseenalaisia.
4. Käytä intuitiota polkujen valinnassa (eli mitkä polut suoritetaan todennäköisimmin).

Huomaa, että tätä strategiaa käytettäessä jotkut polkujen osat suoritetaan todennäköisesti useammin kuin kerran. Tämän strategian keskeinen asia on testata jokainen mahdollinen haara koodissa ainakin kerran ja mahdollisesti useitakin kertoja.

Sovellettavuus
Osittaista polkutestausta – niin kuin se on yllä määritelty – suoritetaan usein turvallisuuskriittisille ohjelmistoille. Se täydentää hyvin muita tässä luvussa käsiteltyjä menetelmiä, koska se tarkastelee ohjelman läpi kulkevia polkuja sen sijaan, että se tarkastelisi vain tapaa, jolla päätökset tehdään.

Rajoitukset/vaikeudet
Vaikka kontrollivuokaaviota voidaan käyttää polkujen määrittämiseen, tarvitaan käytännössä työkalua niiden laskemisessa, kun on kyse monimutkaisista moduuleista.

Kattavuus
Mikäli luodaan riittävästi polkuja kaikkien polkujen kattamiseksi (poislukien silmukat), se takaa myös lausekattavuuden ja haarakattavuuden saavuttamisen. Polkutestaus tuottaa perusteellisemman testauksen kuin haarakattavuus ja testien määrä kasvaa siinä suhteellisen vähän. [NIST 96]

2.7 API-testaus (ohjelmointirajapintatestaus)

Ohjelmointirajapinta (API, Application Programming Interface) tarkoittaa ohjelmakoodia, joka mahdollistaa eri prosessien, ohjelmien ja/tai järjestelmien välisen tiedonkulun. APIa käytetään usein asiakas/palvelin-järjestelmissä, joissa yksi prosessi tuottaa jonkinlaisen toiminnallisuuden toisia prosesseja varten.

Tietyssä mielessä API-testaus on hyvin samanlaista kuin graafisen käyttöliittymän (GUI) testaus. Painopiste on syötearvojen ja palautuneiden tietojen tarkastelussa.

Negatiivinen testaus on usein keskeisessä asemassa kun ollaan tekemisissä APIen kanssa. Ohjelmoijat, jotka käyttävät APIa heidän oman koodinsa ulkopuolisten palveluiden saavuttamiseksi voivat yrittää käyttää API-liiittymiä tavoilla, joihin niitä ei ole tarkoitettu. Se merkitsee, että vahva virheidenkäsittely on keskeisessä asemassa väärien toimintojen välttämiseksi. Voidaan myös tarvita monien eri liittymien yhdistelmien testausta, koska APIa käytetään usein yhdessä toisten APIen kanssa ja koska yksittäinen liittymä voi sisältää useita parametreja, jolloin liittymien arvoja voidaan yhdistellä monilla eri tavoilla.

APIen liittymät ovat usein löyhiä, mikä tekee tapahtumien katoamisen tai ajoitusongelmat hyvinkin mahdollisiksi. Tämä tekee toipumis- ja toistamismenetelmien perusteellisen testaamisen tarpeelliseksi. API-liittymiä tuottavien organisaatioiden täytyy varmistaa kaikkien palveluiden erittäin korkea saatavuus; tämä vaatii usein APIn julkaisijalta huolellista luotettavuustestausta samoin kuin infrastruktuuritukea.

Sovellettavuus
API-testauksesta on tulossa yhtä tärkeämpää, kun yhä useampia järjestelmiä hajautetaan tai ne käyttävät etäprosessointia keinona purkaa osa työstä toisille käsittelijöille. Esimerkkejä näistä ovat käyttöjärjestelmäkutsut, palvelu-ohjautuneet arkkitehtuurit (service-oriented architectures, SOA), etäproseduurikutsut (RPC), web-palvelut ja lähes kaikki muut hajautetut sovellukset. API-testaus soveltuu erityisen hyvin ”järjestelmistä koostuvien järjestelmien” testaamiseen.

Rajoitukset/vaikeudet
API:n testaus suoraan vaatii Tekniseltä testausasiantuntijalta yleensä erityistyökalujen käyttöä. Koska APIin ei yleensä liity suoraa graafista käyttöliittymää, työkaluilla voidaan joutua rakentamaan testiympäristö, hallinnoimaan testiaineisto, suorittamaan API-kutsu ja määrittelemään lopputulos.

Kattavuus
API-testaus on yhden testaustyypin kuvaus; se ei ilmaise mitään tiettyä kattavuustasoa. API-testiin pitäisi kuulua vähintään kaikkien API-kutsujen suorittaminen sekä kelvollisten ja järkevien epäkelpojen arvojen testaus.

Vikatyypit
APIen testauksessa mahdollisesti löydettävät viat ovat luonteeltaan varsin erilaisia. Tyypillisiä ovat liittymäongelmat, samoin kuin aineiston käsittelyyn liittyvät ongelmat, ajoitusongelmat sekä tapahtumien katoaminen ja monistuminen.

2.8 Rakennepohjaisen tekniikan valinta

Testattava järjestelmä kokonaisuutena määrittää rakennepohjaisen testauksen kattavuustason, joka pitäisi saavuttaa. Mitä kriittisempi järjestelmä on, sitä korkeampaa kattavuustasoa tarvitaan. Yleisesti ottaen mitä korkeammat kattavuustasovaatimukset ovat, sitä enemmän aikaa ja resursseja tarvitaan kyseisen tason saavuttamiseksi.

Joskus kattavuustasovaatimukset voidaan johtaa ohjelmistojärjestelmää koskevista tilanteeseen soveltuvista standardeista. Esimerkiksi jos ohjelmistoa tultaisiin käyttämään ilmailualuksessa, sen voitaisiin vaatia täyttävän standardin DO-178B vaatimukset (Euroopassa ED12B). Tämä standardi sisältää seuraavat viisi häiriöluokkaa:

A. Katastrofaalinen: häiriö voi aiheuttaa lentokoneen lentämisen tai laskeutumisen kannalta kriittisen toiminnallisuuden puuttumisen.
B. Vaarallinen: Häiriöllä voi olla iso negatiivinen vaikutus turvallisuuteen tai suorituskykyyn.
C. Suuri: häiriö on merkittävä, mutta vähemmän vakava kuin A tai B.
D. Pieni: häiriö on huomattavissa, mutta vaikutukseltaan pienempi kuin C.
E. Ei vaikutusta: häiriöllä ei ole vaikutusta turvallisuuteen.

Jos ohjelmistojärjestelmä on luokiteltu tasolle A, se on testattava täydennetyn ehtokattavuuden tasolla. Jos se on tasoa C, se täytyy testata päätöskattavuuden tasolla, vaikkakin täydennetty ehtokattavuus on vapaavalintainen vaihtoehto. Taso C vaatii vähintään lausekattavuuden.

Vastaavasti IEC-61508 on ohjelmoitavien elektronisten turvallisuuteen liittyvien järjestelmien toiminnallisen turvallisuuden kansainvälinen standardi. Tätä standardia on sovellettu monilla eri aloilla, mukaan luettuna autoteollisuus, raideliikenne, tuotantoprosessit, ydinvoimalat sekä koneteollisuus. Kriittisyys on määritelty käyttämällä asteittain luokiteltua jatkumoa Safety Integrity Level (SIL) (1 on vähiten kriittinen, 4 kaikkein kriittisin), ja kattavuussuositukset ovat seuraavat:

1. Suositellaan lause- ja haarakattavuutta.
2. Suositellaan erittäin vahvasti lausekattavuutta, suositellaan haarakattavuutta.
3. Suositellaan erittäin vahvasti lause- ja haarakattavuutta.
4. Suositellaan erittäin vahvasti täydennettyä ehtokattavuutta.

Nykyaikaisissa järjestelmissä on harvinaista, että yksittäinen järjestelmä suorittaa kaiken prosessoinnin. API-testausta pitäisi käyttää aina silloin, kun osa prosessoinnista tehdään etäprosessointina. Järjestelmän kriittisyyden pitäisi määrittää, kuinka paljon API-testaukseen tulisi panostaa.

Kuten aina, testattavana olevan ohjelmistojärjestelmäkokonaisuuden pitäisi ohjata Teknistä testausasiantuntijaa testausmenetelmien valinnassa.

3. Analyyttiset tekniikat - 255 min

Avainsanat

dynaaminen analyysi, kontrollivuoanalyysi, lähialue-integrointitestaus, muistivuoto, määrittely-käyttö-parit, staattinen analyysi, syklomaattinen kompleksisuus, syöteparien integrointitestaus, tietovirta-analyysi, villi osoitin

Oppimistavoitteet: Analyyttiset tekniikat

3.2 Staattinen analyysi

TTA-3.2.1 (K3) Käyttää kontrollivuoanalyysia paljastamaan, onko koodissa kontrollivuopoikkeamia.
TTA-3.2.2 (K3) Käyttää tietovuoanalyysia paljastamaan, onko koodissa tietovuopoikkeamia.
TTA-3.2.3 (K3) Ehdottaa tapoja parantaa koodin ylläpidettävyyttä staattista analyysiä käyttämällä.
TTA-3.2.4 (K2) Selittää, miten kutsukaavioita käytetään integraatiotestausstrategioiden laatimisessa.

3.3 Dynaaminen analyysi

TTA-3.3.1 (K3) Määritellä tavoitteet, joihin dynaamisen analyysin käytöllä pyritään.

3.1 Esittely

Analyysityyppejä on kaksi: staattinen analyysi ja dynaaminen analyysi.

Staattinen analyysi (kappale 3.2) sisältää analyyttisen testauksen, joka voidaan tehdä ilman ohjelman suoritusta. Ohjelmiston suorittamisen sijaan työkalu tai henkilö tutkii ohjelmistoa ja päättelee, tuleeko se toimimaan oikein, kun se suoritetaan. Tämä staattinen näkymä ohjelmistoon mahdollistaa ohjelmiston yksityiskohtaisen analyysin ilman, että joudutaan luomaan aineistoa ja toteuttamaan esiehtoja, jotka aiheuttaisivat skenaarion suorituksen.

Huomaa, että katselmointien eri muodot, jotka ovat keskeisiä Teknisen testausasiantuntijan näkökulmasta, käsitellään luvussa 5.

Dynaaminen analyysi (kappale 3.3) vaatii ohjelmakoodin suoritusta ja sitä käytetään, kun yritetään löytää ohjelmointivirheitä, jotka paljastuvat helpommin, kun ohjelmakoodia suoritetaan (esim. muistivuodot). Dynaamisessa analyysissa, samoin kuin staattisessa analyysissa, voidaan käyttää työkaluja tai sen voi suorittaa henkilö tarkkailemalla suoritettavaa ohjelmaa erilaisten viitteiden, kuten esimerkiksi nopean muistin kasvun, havaitsemiseksi.

3.2 Staattinen analyysi

Staattisen analyysin tavoitteena on löytää todellisia tai mahdollisia vikoja ohjelmakoodista ja järjestelmäarkkitehtuurista ja pyrkiä parantamaan niiden ylläpidettävyyttä. Staattisessa analyysissä käytetään yleensä apuna työkalua.

3.2.1 Kontrollivuoanalyysi

Kontrollivuoanalyysi on staattinen tekniikka, jossa analysoidaan kontrollivuo läpi ohjelman käyttämällä joko kontrollivuokaaviota tai työkalua. On olemassa lukuisia poikkeamia, joita voidaan löytää järjestelmästä tätä tekniikkaa käyttämällä, ja niihin kuuluvat huonosti suunnitellut silmukat (esim. silmukassa on useita sisääntulokohtia), tulkinnanvaraiset funktiokutsujen kohteet tietyissä ohjelmointikielissä (esim. Scheme), väärät toimintosekvenssit jne.

Yksi tyypillisimmistä kontrollivuoanalyysin käyttökohteista on syklomaattisen kompleksisuuden määrittäminen. Syklomaattisen kompleksisuuden arvo on positiivinen kokonaisluku, joka kertoo vahvasti kytketyssä kaaviossa olevien riippumattomien polkujen määrän silloin, kun jätetään huomiotta silmukat ja iteraatiot sen jälkeen, kun ne on käyty kertaalleen läpi. Jokainen riippumaton polku, lähtöpisteestä päätöspisteeseen, edustaa ainutkertaista polkua moduulin läpi. Jokainen ainutkertainen polku pitäisi testata.

Syklomaattisen kompleksisuuden arvoa käytetään yleensä moduulin yleisen monimutkaisuuden ymmärtämiseen. Thomas McCaben teoria [McCabe 76] oli, että mitä monimutkaisempi järjestelmä, sitä vaikeampi sitä olisi ylläpitää ja sitä enemmän vikoja se sisältäisi. Monet vuosien saatossa tehdyt tutkimukset ovat tuoneet esiin tämän yhteyden monimutkaisuuden ja ohjelman sisältämien vikojen välillä. NIST (National Institute of Standards and Technology) suosittelee monimutkaisuusarvoksi korkeintaan arvoa 10. Mikä tahansa moduuli, jonka monimutkaisuus saa mitattaessa suuremman arvon, voi olla syytä jakaa useampiin moduuleihin.

3.2.2 Tietovirta-analyysi

Tietovirta-analyysi kattaa joukon tekniikoita, joilla kerätään tietoa muuttujien käytöstä järjestelmässä. Tarkastelun kohteena on muuttujien elinkaari (missä ne esitellään, määritellään, missä niitä luetaan, käytetään ja missä ne tuhotaan), sillä poikkeamia voi tapahtua minkä tahansa näiden toimenpiteiden aikana.

Yhtä tyypillistä tekniikkaa kutsutaan määrittele-käytä- (define-use-) tekniikaksi, missä jokaisen muuttujan elinkaari jaetaan kolmeen eri atomiseen tapahtumaan:

- d: kun muuttuja esitellään, määritellään tai alustetaan (engl. declared, defined, initialized)
- u: kun muuttujaa käytetään (engl. used) tai se luetaan joko laskutoimituksessa tai päätöstekijänä
- k: kun muuttuja tuhotaan (engl. killed) tai se jää toimintakentän ulkopuolelle.

Nämä kolme atomista tapahtumaa yhdistetään pareiksi (“määrittely-käyttö-, definition-use-parit”) kuvaamaan tietovirtaa. Esimerkiksi “du-polku” kuvaa koodinpätkää, jossa muuttuja määritellään ja sen jälkeen sitä käytetään.

Mahdollisiin poikkeamiin kuuluvat oikean tapahtuman suorittaminen muuttujalle väärään aikaan tai väärän tapahtuman suorittaminen muuttujan sisältämälle tiedolle. Näihin poikkeamiin kuuluvat seuraavat:

- Muuttujalle annetaan epäkelpo arvo.
- Muuttujalle ei anneta arvoa ennen sen käyttämistä.
- Valitaan väärä polku kontrollimuuttujan väärän arvon vuoksi.
- Muuttujaa yritetään käyttää sen jälkeen, kun se on tuhottu.
- Viitataan muuttujaan, kun se on toimintakentän ulkopuolella.
- Esitellään ja tuhotaan muuttuja käyttämättä sitä.
- Määritellään muuttuja uudelleen ennen kuin sitä on käytetty.
- Dynaamisesti allokoitua muuttujaa ei tuhota (aiheuttaa mahdollisen muistivuodon).
- Muokataan muuttujaa, mikä johtaa odottamattomiin sivuvaikutuksiin (esim. heijastusvaikutukset, kun muutetaan globaalia muuttujaa ottamatta huomioon kaikkia sen käyttötilanteita).

Käytetty ohjelmointikieli voi ohjata tietovirta-analyysissä käytettäviä sääntöjä. Ohjelmointikielet voivat sallia ohjelmoijan tehdä muuttujille tiettyjä toimenpiteitä, jotka eivät ole kiellettyjä, mutta saattavat johtaa siihen, että tietyissä tilanteissa järjestelmä käyttäytyy toisin kuin mitä ohjelmoija odotti. Muuttuja voidaan esimerkiksi määritellä kahdesti ilman että sitä todella käytetään, kun seurataan tiettyä polkua. Tietovirta-analyysi luokittelee tällaiset käytöt usein “epäilyttäviksi”. Vaikka tämä saattaakin olla luvallinen tapa käyttää muuttujan määrittelyominaisuuksia, se voi johtaa jatkossa koodin ylläpidettävyysongelmiin.

Tietovirtatestaus “käyttää kontrollivuokaaviota pyrkiessään tutkimaan niitä mahdottomalta vaikuttavia asioita, joita aineistolle voi tapahtua” [Beizer90] ja siksi se löytää erilaisia vikoja kuin kontrollivuotestaus. Teknisen testausasiantuntijan tulee ottaa tämä tekniikka mukaan suunnitellessaan testausta, sillä monet näistä vioista aiheuttavat epäsäännöllisesti toistuvia häiriöitä, joita on vaikea löytää dynaamisessa testauksessa.

Tietovirta-analyysi on kuitenkin staattinen tekniikka; sitä käytettäessä saattaa jäädä huomaamatta aineistolle tapahtuvia ajonaikaisia ongelmia. Staattinen tietomuuttuja voi esimerkiksi sisältää osoittimen dynaamisesti luotavaan taulukkoon, jota ei edes ole olemassa ennen ohjelman suoritusta. Samanaikainen usean prosessorin käyttö ja ennakoivat yhtäaikaiset tehtävät voivat aiheuttaa kilpailutilanteita, jotka eivät tule esiin tietovirta- tai kontrollivuoanalyysissä.

3.2.3 Staattisen analyysin käyttö ylläpidettävyyden parantamiseksi

Staattista analyysia voidaan käyttää monella tavalla koodin, arkkitehtuurin ja websivustojen ylläpidettävyyden parantamiseen.

Huonosti kirjoitettua, kommentoimatonta ja jäsentämätöntä koodia on yleensä vaikeampi ylläpitää. Kehittäjät saattavat joutua tekemään enemmän työtä paikallistaakseen ja analysoidakseen koodissa olevia vikoja, ja vian korjaamiseksi tai uuden toiminnon lisäämiseksi tehtävät koodimuutokset saattavat johtaa uusien vikojen syntyyn.

Työkaluilla tuettua staattista analyysiä käytetään parantamaan koodin ylläpidettävyyttä todentamalla sen ohjelmointistandardien- ja ohjeidenmukaisuus. Näissä standardeissa ja ohjeissa kuvataan vaadittavat ohjelmointikäytännöt, kuten nimeämistavat, kommentointi, sisentäminen ja koodin modularisointi. Huomaa, että staattisen analyysin työkalut tyypillisesti antavat ennemmin varoituksia kuin virheilmoituksia vaikka koodi olisikin syntaksiltaan oikein.

Modulaarinen suunnittelu johtaa yleensä ylläpidettävämpään ohjelmakoodiin. Staattisen analyysin työkalut tukevat modulaarisen koodin tuottamista seuraavin tavoin:

- Ne etsivät toistuvaa koodia. Tällaiset koodin osat voivat olla ehdokkaita uudelleenjärjestelyn kautta modularisoitaviksi (vaikkakin moduulikutsujen aiheuttamat ajonaikainen kuorma voi olla ongelma reaaliaikaisille järjestelmille).
- Ne tuottavat mittaritietoja, jotka kertovat koodin modularisoinnista tärkeitä tietoja. Näihin kuuluvat kytkentöjen ja koheesion määrä. Helposti ylläpidettävän järjestelmän kytkentäluku (missä määrin moduulit ovat toisistaan riippuvaisia suorituksen aikana) on todennäköisemmin matala ja koheesioluku (missä määrin moduuli on itsenäinen ja keskittynyt yksittäiseen tehtävään) on korkea.
- Oliopohjaisessa koodissa ne ilmaisevat, missä periytyneillä olioilla voi olla liikaa tai liian vähän näkyvyyttä yliluokkaan.
- Ne tuovat koodista tai arkkitehtuurista esiin alueita, joiden rakenteellinen monimutkaisuus on suuri, mitä yleensä pidetään huonon ylläpidettävyyden merkkinä, ja joilla siksi vikojen esiintymisen todennäköisyys on korkeampi. Ohjeisiin voidaan määritellä syklomaattisen kompleksisuuden (ks. kappale 3.2.1) hyväksyttävät tasot sen varmistamiseksi, että koodi toteutetaan modulaarisesti ylläpidettävyys ja vikojen ennaltaehkäisy huomioon ottaen. Koodi, jonka syklomaattinen kompleksisuusluku on korkea, voi olla sopiva ehdokas modularisoitavaksi.

Staattisen analyysin työkaluilla voidaan myös tukea verkkosivuston ylläpitoa. Siinä tavoitteena on tarkistaa, onko sivuston puumainen rakenne hyvin tasapainossa, vai onko se epätasapainossa, mikä saattaa johtaa

- vaikeampiin testaustehtäviin
- lisääntyvään ylläpidon työkuormaan
- käyttäjän kannalta vaikeaan navigointiin.

3.2.4 Kutsukaaviot

Kutsukaaviot ovat tiedonkulun monimutkaisuuden staattinen esitystapa. Ne ovat kohdennettuja kaavioita, joissa solmut esittävät ohjelman osia ja kaaret osien välistä viestintää.

Kutsukaavioita voidaan käyttää yksikkötestauksessa, missä eri funktiot tai metodit kutsuvat toisiaan, integraatio- ja järjestelmätestauksessa, missä eri moduulit kutsuvat toisiaan, tai järjestelmäintegraatiotestauksessa, missä eri järjestelmät kutsuvat toisiaan.

Kutsukaavioita voidaan käyttää seuraaviin tarkoituksiin:

- Tiettyä moduulia tai järjestelmää kutsuvien testien suunnitteluun
- Kuvaamaan, kuinka monesta paikasta ohjelmistossa moduulia tai järjestelmää kutsutaan
- Koodin ja järjestelmäarkkitehtuurin rakenteen arviointiin
- Integrointijärjestysehdotuksien (syöteparien integrointi, lähialue-integrointi) tekemiseen. Näitä käsitellään tarkemmin alla.

Perustason sertifikaattisisällössä [ISTQB_FL_SYL] käsiteltiin kahta eri integrointiryhmää: inkrementaalista (ylhäältä alas eli “top-down”, alhaalta ylös eli ”bottom-up”, jne) ja ei-inkrementaalista (big bang). Inkrementaalisten menetelmien sanottiin olevan suositumpia, koska ne ottavat koodia mukaan inkrementeissä ja tekevät siten vikojen eristämisen helpommaksi, koska tilanteeseen liittyvän koodin määrä on rajallinen.

Tässä Jatkotason sertifikaattisisällössä esitellään kolme ei-inkrementaalista, kutsukaavioita käyttävää tapaa lisää. Nämä saattavat olla suositeltavampia sellaisiin inkrementaalisiin menetelmiin verrattuna,

jotka todennäköisesti vaativat lisäkoonteja testauksen loppuun viemiseksi sekä vain testauksen tukemiseksi käytettävän koodin kirjoittamista. Nämä kolme menetelmää ovat:

- Syöteparien integrointitestaus (jota ei tule sotkea mustalaatikkotekniikkaan “syötteiden pareittainen testaus”), jonka kohteena ovat komponenttiparit, jotka integrointitestausta varten tehdyn kutsukaavion perusteella näyttävät toimivan yhdessä. Vaikka tämä menetelmä vähentääkin koontien määrää vain vähän, se pienentää tarvittavan testikehyskoodin määrää.
- Lähialue-integrointitestauksessa testataan kaikki solmut, jotka yhdistyvät tiettyyn solmuun, ja tämä muodostaa pohjan integrointitestaukselle. Kaikki kutsukaavion määrätyn solmun edeltäjä- ja seuraajasolmut ovat testauksen kohteita.
- McCaben suunnittelumuuttuja-lähestymistapa käyttää syklomaattisen kompleksisuuden teoriaa sovellettuna moduulien kutsukaavioon. Tätä varten on rakennettava kutsukaavio, joka esittää eri tavat, joilla moduulit voivat kutsua toisiaan, mukaan luettuna:
  - Ehdoton kutsu: toisen moduulin kutsu tapahtuu aina
  - Ehdollinen kutsu: toisen moduulin kutsu tapahtuu joskus
  - Molempia puolia rajoittava ehdollinen kutsu: moduuli kutsuu yhtä (ja vain yhtä) moduulia useiden moduulien joukosta.
  - Iteratiivinen kutsu: moduuli kutsuu toista ainakin kerran mutta se voi kutsua sitä myös useita kertoja
  - Iteratiivinen ehdollinen kutsu: moduuli ei kutsu toista lainkaan tai se voi kutsua sitä useita kertoja

  Kutsukaavion laatimisen jälkeen lasketaan integraation monimutkaisuus ja laaditaan testit kattamaan kaavio.

Katso [Jorgensen07] lisätietoja kutsukaavioiden käytöstä ja syöteparien integrointitestauksesta.

3.3 Dynaaminen analyysi

3.3.1 Yleiskatsaus

Dynaamista analyysiä käytetään paljastamaan häiriöitä, joiden oireet eivät ole välttämättä heti nähtävissä. Esimerkiksi muistivuotojen mahdollisuus saattaa olla nähtävissä staattisella analyysillä (löydetään koodia, joka varaa muistia käyttöön mutta ei vapauta sitä), mutta muistivuoto on helposti havaittavissa dynaamisella analyysillä.

Häiriöillä, jotka eivät ole välittömästi toistettavissa, voi olla merkittävät seuraukset testaustyön määrälle ja sille, voidaanko ohjelmisto julkaista tai sitä käyttää tuottavasti. Tällaisia häiriöitä voivat aiheuttaa muistivuodot, osoittimien vääränlainen käyttö sekä muut (esim. järjestelmäpinon) vikaantumistilanteet. Koska nämä häiriöt ovat luonteeltaan sellaisia, että ne voivat aiheuttaa mm. järjestelmän suorituskyvyn asteittaisen heikkenemisen tai jopa järjestelmän kaatumisen, testausstrategioissa täytyy ottaa huomion tällaisiin vikoihin liittyvät riskit ja tehdä järjestelmän kannalta soveltuvin osin dynaaminen analyysi (tyypillisesti työkaluja käyttämällä) niiden vähentämiseksi. Koska nämä häiriöt ovat usein kaikkein kalleimpia löytää ja korjata, on suositeltavaa suorittaa dynaaminen analyysi projektin aikaisessa vaiheessa.

Dynaamista analyysiä voidaan käyttää seuraaviin asioihin:

- Häiriöiden estäminen paljastamalla villejä osoittimia ja järjestelmämuistin häviäminen
- Vaikeasti toistettavien järjestelmähäiriöiden analysointi
- Verkon käyttäytymisen arviointi
- Järjestelmän suorituskyvyn parantaminen tuottamalla tietoa järjestelmän ajonaikaisesta käyttäytymisestä.

Dynaamista analyysia voidaan tehdä kaikilla testaustasoilla ja siinä tarvitaan teknistä ja järjestelmään liittyvää osaamista seuraaviin tehtäviin:

- Dynaamisen analyysin testauskohteiden määrittäminen
- Analyysin sopivan aloitus- ja lopetusajankohdan määrittäminen
- Tulosten analysointi

Järjestelmätestauksessa voidaan käyttää dynaamisen analyysin työkaluja vaikka Teknisen testausasiantuntijan tekninen osaaminen olisikin vain vähäistä; työkalut tuottavat yleensä kattavia lokeja, joiden analysoinnin voivat suorittaa henkilöt, joilla on tarvittava tekninen osaaminen.

3.3.2 Muistivuotojen löytäminen

Muistivuoto syntyy, kun ohjelman varaa sen käytettävissä olevasta muistista (RAM) alueen käyttöönsä, mutta ei vapauta sitä, kun ei enää tarvitse sitä. Tämä muistialue jää varatuksi eikä ole enää uudelleenkäytettävissä. Kun näin tapahtuu usein tai jos muistia on vähän, käytettävissä oleva muisti voi loppua ohjelmalta kesken. Alunperin muistin käytön hallinta oli ohjelmoijan vastuulla. Dynaamisesti muistialueita varaavan ohjelman täytyy vapauttaa varaamansa alueet oikeassa laajuudessa muistivuotojen välttämiseksi. Moniin nykyaikaisiin ohjelmointiympäristöihin kuuluu automaattinen tai puoli-automaattinen ”roskankeräys”, jolloin varattu muisti vapautetaan ilman, että ohjelmoijan tarvitsee vaikuttaa asiaan. Muistivuotojen eristäminen voi olla hyvin vaikeaa silloin, jos automaattinen roskankerääjä vapauttaa varatun muistin.

Muistivuodot aiheuttavat ongelmia, jotka kehittyvät ajan myötä eivätkä aina ole heti huomattavissa. Näin voi tapahtua esimerkiksi silloin, jos ohjelmisto on asennettu hiljattain tai järjestelmä on käynnistetty uudelleen, kuten usein tapahtuu testauksen aikana. Näistä syistä muistivuotojen negatiiviset vaikutukset saattavat usein tulla ilmi vasta siinä vaiheessa, kun ohjelma on tuotantokäytössä.

Muistivuodon oireena on ohjelmiston vasteaikojen tasainen heikkeneminen, mikä voi lopulta johtaa järjestelmän kaatumiseen. Vaikka tällaiset ongelmat voidaankin ehkä ratkaista käynnistämällä järjestelmä uudelleen, se ei aina ole käytännöllistä tai edes mahdollista.

Monet dynaamisen analyysin työkalut tunnistavat koodista alueita, joissa muistivuotoja esiintyy, jotta ne voidaan korjata. Voidaan käyttää myös yksinkertaisia muistimonitoreja, jotta saadaan yleisvaikutelma siitä, pieneneekö käytettävissä olevan muistin määrä ajan myötä, vaikka lisäanalyysiä saatetaan silti tarvita pienenemisen tarkan syyn selvittämiseksi.

On myös muita vuotojen lähteitä, jotka on syytä ottaa huomioon. Esimerkkejä näistä ovat tiedostokahvat, semaforit ja yhteysryhmät.

3.3.3 Villien osoittimien löytäminen

“Villit” osoittimet ovat ohjelmassa osoittimia, joita ei saa käyttää. Villi osoitin on esimerkiksi voinut ”kadottaa” kohteen tai funktion, johon sen pitäisi osoittaa, tai se ei osoita tarkoitettuun muistialueeseen (se esimerkiksi osoittaa alueeseen, joka on käytettävän taulukon rajojen ulkopuolella). Kun ohjelma käyttää villejä osoittimia, siitä voi olla monenlaisia seurauksia:

- Ohjelma voi toimia odotetusti. Näin voi tapahtua, jos villi osoitin osoittaa muistialueeseen, joka kyseisellä hetkellä ei ole ohjelman käytössä ja on ohjelman tietojen mukaan ”vapaa” ja/tai sisältää järkevän arvon.
- Ohjelma voi kaatua. Tässä tapauksessa villi osoitin on voinut aiheuttaa ohjelman suorituksen (esim. käyttöjärjestelmän) kannalta kriittisen muistialueen väärinkäytön.
- Ohjelma ei toimi oikein, koska ohjelman tarvitsemiin objekteihin/olioihin ei päästä käsiksi. Tässä tilanteessa ohjelma saattaa jatkaa toimintaa, vaikkakin se voi antaa virheilmoituksen.
- Muistipaikassa oleva tieto voi korruptoitua väärän osoittimen ja väärien arvojen käytön seurauksena.

Huomaa, että mitkä tahansa ohjelman muistikäyttöön tehdyt muutokset (esim. ohjelmistomuutosta seuraava uusi koonti) voivat aiheuttaa minkä tahansa yllä mainituista seurauksista. Tämä on erityisen kriittistä tilanteessa, jossa ohjelma on alunperin toiminut odotetusti villien osoittimien käytöstä huolimatta, ja kaatuu sitten yllättäen ohjelmistomuutoksen seurauksena (ehkä jopa tuotantokäytössä). On tärkeää huomata, että tällaiset häiriöt ovat usein oire taustalla olevasta viasta (eli villistä osoittimesta). (Ks. [Kaner02], ”Lesson74). Työkalut voivat auttaa tunnistamaan villejä osoittimia, kun

ohjelma käyttää niitä, riippumatta siitä, mikä on niiden vaikutus ohjelman suoritukseen. Joissakin käyttöjärjestelmissä on sisäänrakennettuna toiminto, joka tarkistaa ajonaikaiset muistinkäyttörikkeet. Esimerkiksi käyttöjärjestelmä voi siirtyä poikkeuskäsittelyyn, kun sovellus yrittää käyttää muistialuetta, joka on sovellukselle sallitun muistialueen ulkopuolella.

3.3.4 Suorituskyvyn analysointi

Dynaaminen analyysi ei ole hyödyllinen pelkästään häiriöiden paljastamiseen. Ohjelmiston suorituskyvyn dynaamisen analyysin avulla työkalut auttavat tunnistamaan suorituskyvyn pullonkauloja ja tuottavat paljon erilaista suorituskykyyn liittyvää mittaritietoa, jota toteuttaja voi käyttää ohjelmiston suorituskyvyn säätämiseen. Esimerkiksi saatavilla voi olla tietoa siitä, kuinka monta kertaa moduulia kutsutaan suorituksen aikana. Moduulit, joita kutsutaan usein, voisivat olla hyviä ehdokkaita suorituskyvyn parantamiseen.

Yhdistämällä ohjelmiston dynaamisesta käyttäytymisestä saadut tiedot staattisen analyysin aikana kutsukaaviosta saatuihin tietoihin (ks. kappale 3.2.4), testaaja voi myös tunnistaa moduuleita, jotka saattavat olla ehdokkaita yksityiskohtaiseen ja laajaan testaukseen (esimerkiksi moduulit, joita kutsutaan usein ja joilla on monia liittymiä).

Ohjelman suorituskyvyn dynaaminen analyysi tehdään usein järjestelmätestien suorituksen yhteydessä, vaikka se voidaan tehdä myös testauksen aikaisemmissa vaiheissa, kun testataan yksittäistä alijärjestelmää testikehysten avulla.

4. Teknisen testauksen laatuominaisuudet – 405 min.

Avainsanat

analysoitavuus, asennettavuus, hyväksymistestaus, korvattavuus, kypsyys, käyttöprofiili, käyttöön soveltuvuuden muutettavuus, luotettavuuden kasvumalli, luotettavuustestaus, resurssien käytön testaus, robustius, siirrettävyystestaus, sovitettavuus, suorituskykytestaus, testattavuus, tietoturvatestaus, toiminnallinen tehokkuus, toipumistestaus, yhdessätoimivuus, vakaus, ylläpidettävyystestaus

Oppimistavoitteet: Teknisen testauksen laatuominaisuudet

4.2 Yleisiä suunnittelussa huomioon otettavia asioita

TTA-4.2.1 (K4) Analysoida määrättyyn projektiin ja testattavaan järjestelmään liittyvät ei-toiminnalliset vaatimukset ja kirjoittaa testaussuunnitelman vastaavat osat.

4.3 Tietoturvatestaus

TTA-4.3.1 (K3) Määritellä tietoturvatestauksen lähestymistavat ja suunnitella siihen liittyvät korkean tason testitapaukset.

4.4 Luotettavuustestaus

TTA-4.4.1 (K3) Määritellä luotettavuuden laatuominaisuuksiin ja vastaaviin ISO 9126 aliominaisuuksiin liittyvät lähestymistavat ja suunnitella niihin liittyvät korkean tason testitapaukset.

4.5 Suorituskykytestaus

TTA-4.5.1 (K3) Määritellä suorituskykytestauksen lähestymistavat ja suunnitella siihen liittyvät korkean tason käyttöprofiilit.

Yleiset oppimistavoitteet

Seuraavat oppimistavoitteet liittyvät asioihin, joita on käsitelty useammassa kuin yhdessä tämän luvun osassa.

TTA-4.x.1 (K2) Ymmärtää ja selittää syyt sille, miksi ylläpidettävyys-, siirrettävyys- ja resurssien käyttötestit pitäisi sisällyttää testausstrategiaan ja/tai testauksen lähestymistapoihin.
TTA-4.x.2 (K3) Määritellä tiettyä tuoteriskiä koskien soveltuvimmat ei-toiminnalliset testaustyypit.
TTA-4.x.3 Ymmärtää ja selittää sovelluksen elinkaaren vaiheet, joissa ei-toiminnallista testausta pitäisi tehdä.
TTA-4.x.4 (K3) Määritellä tiettyyn skenaarioon liittyen minkälaisia vikoja voisi odottaa löydettävän, kun käytetään ei-toiminnallisia testaustyyppejä.

4.1 Esittely

Yleisesti ottaen Tekninen testausasiantuntija keskittyy testauksessa siihen, “miten” tuote toimii, ennemmin kuin toiminnallisiin piirteisiin, jotka liittyvät siihen, ”mitä” järjestelmä tekee. Tätä testausta voidaan tehdä millä tahansa testaustasolla. Esimerkiksi reaaliaikaisen sulautetun järjestelmän yksikkötestauksessa on tärkeää asettaa suorituskykytestauksen vertailukohta ja testata resurssien käyttö. Järjestelmätestauksen ja käyttöön soveltuvuuden hyväksymistestauksen aikana on sopivaa testata luotettavuuteen liittyvät seikat, kuten toipuvuus. Tällä tasolla testit on suunnattu määrätyn järjestelmän, eli laitteiston ja ohjelmiston yhdistelmien, testaamiseen. Testauksen kohteena olevaan järjestelmään saattaa kuulua erilaisia palvelimia, asiakasohjelmia, tietokantoja, verkkoja ja muita resursseja. Testaustasosta riippumatta testaus pitäisi suorittaa priorisoitujen riskien ja saatavilla olevien resurssien mukaan.

ISO 9126:n sisältämää tuotteen laatuominaisuuksien kuvausta käytetään ohjaamaan ominaisuuksien kuvausta. Muitakin standardeja, kuten ISO 25000-sarja (joka on korvannut ISO 9126:n), voidaan käyttää. ISO 9126 laatuominaisuudet on jaettu ominaisuuksiin, joista jokaisella voi olla aliominaisuuksia. Ne on esitetty alla olevassa taulukossa, johon on myös merkitty, mitkä laatuominaisuudet/aliominaisuudet käsitellään Testausasiantuntija- ja Tekninen testausasiantuntija-sertifikaattisisällöissä.

Ominaisuus	**Aliominaisuus**	**Testausasiantuntija**	**Tekninen testausasiantuntija**
Toiminnallisuus	Tarkkuus, soveltuvuus, yhteentoimivuus, yhdenmukaisuus	X	
	Tietoturva		X
Luotettavuus	Kypsyys (vakaus), vikasietoisuus, toipuvuus, yhdenmukaisuus		X
Käytettävyys	Ymmärrettävyys, opittavuus, käyttökelpoisuus, viehättävyys, yhdenmukaisuus	X	
Toiminnallinen tehokkuus	Suorituskyky (aikakäyttäytyminen), resurssien käyttö, yhdenmukaisuus		X
Ylläpidettävyys	Analysoitavuus, muutettavuus, vakaus, testattavuus, yhdenmukaisuus		X
Siirrettävyys	Sovitettavuus, asennettavuus, yhdessätoimivuus, korvattavuus, yhdenmukaisuus		X

Vaikka tämä työnjako voi vaihdella organisaatioittain, näissä ISTQB sertifikaattisisällöissä noudatetaan yllä esitettyä.

Aliominaisuus ”yhdenmukaisuus” esiintyy jokaisen laatuominaisuuden kohdalla. Eräitten turvallisuuskriittisten tai säädeltyjen ympäristöjen kohdalla voi olla, että jokaisen laatuominaisuuden pitää noudattaa määrättyjä standardeja ja säädöksiä. Koska nämä standardit voivat vaihdella laajasti teollisuudenalan mukaan, niihin ei paneuduta syvällisemmin tässä yhteydessä. Jos Tekninen testausasiantuntija työskentelee ympäristössä, johon yhdenmukaisuusvaatimukset vaikuttavat, on tärkeää ymmärtää kyseiset vaatimukset ja varmistua siitä, että sekä testaus että testausdokumentaatio tulevat täyttämään ne.

Kaikkien tässä kappaleessa käsiteltävien laatuominaisuuksien ja aliominaisuuksien osalta on tunnistettava tyypilliset riskit sopivan testausstrategian laatimiseksi ja dokumentoimiseksi. Laatuominaisuuksien testaus vaatii, että erityistä huomiota kiinnitetään elinkaaren ajankohtaan, tarvittaviin työkaluihin, ohjelmiston ja dokumentaation saatavuuteen sekä tekniseen asiantuntemukseen. Mikäli jokaisen ominaisuuden ja sen yksilöllisten testausvaatimusten käsittelemiseksi ei laadita strategiaa, testausaikatauluun ei ehkä varata testaajalle riittävästi aikaa

suunnitteluun, valmisteluun ja testien suoritukseen [Bath08]. Osa tästä testauksesta, esim. suorituskykytestaus, vaatii laajaa suunnittelua, erityisesti sitä varten tarkoitettua laitteistoa, erityistyökaluja, erikoistunutta testausosaamista ja useimmissa tapauksissa huomattavan paljon aikaa. Laatuominaisuuksien ja aliominaisuuksien testaus täytyy sisällyttää testauksen kokonaisaikatauluun ja siihen pitää varata riittävästi resursseja. Jokaisella näistä alueista on omat erityistarpeensa, jokaisella niistä on tietyt kohteet ja ne saattavat tapahtua eri aikoihin ohjelmiston elinkaaren aikana, kuten edempänä todetaan.

Testauspäällikkö keskittyy laatuominaisuuksiin ja aliominaisuuksiin liittyvien mittaritietojen keräämiseen ja yhteenvetotietojen raportointiin, kun taas Testausasiantuntija tai Tekninen testausasiantuntija (yllä olevan taulukon mukaisesti) kerää tietoja jokaista mittaria varten.

Teknisen testausasiantuntijan ennen tuotantoonsiirtoa tehdyissä testeissä keräämät laatuominaisuuksien mittaritiedot voivat muodostaa pohjan järjestelmän toimittajan ja sidosryhmien (esim. asiakkaat, operaattorit) väliselle palvelutasosopimukselle (Service Level Agreement, SLA). Joissain tapauksissa testaaminen voi jatkua vielä tuotantoonsiirron jälkeenkin, usein erillisen tiimin tai organisaation suorittamana. Tätä pidetään usein toiminnallisen tehokkuuden ja luotettavuuden testauksena, jonka tuotantoympäristössä tuottamat tulokset voivat poiketa testiympäristössä saaduista.

4.2 Yleisiä suunnittelussa huomioon otettavia asioita

Ei-toiminnallisten testien suunnittelematta jättäminen voi aiheuttaa sovelluksen menestymiselle huomattavan riskin. Testauspäällikkö saattaa pyytää Teknistä testausasiantuntijaa tunnistamaan keskeisiin laatuominaisuuksiin liittyvät olennaisimmat riskit (ks. taulukko kappaleessa 4.1) ja ottamaan kantaa esitettyihin testeihin liittyviin suunnitteluongelmiin. Näitä voidaan käyttää kokonaistestaussuunnitelman laatimisessa. Seuraavia yleisiä tekijöitä on mietittävä, kun näitä tehtäviä suoritetaan:

- Sidosryhmien vaatimukset
- Tarvittavat työkaluhankinnat ja koulutus
- Testiympäristöön liittyvät vaatimukset
- Organisaation kannalta huomioon otettavat seikat
- Tietoturvaan liittyvät seikat

4.2.1 Sidosryhmien vaatimukset

Ei-toiminnalliset vaatimukset on usein määritelty huonosti tai niitä ei jopa ole lainkaan. Suunnitteluvaiheessa Teknisen testausasiantuntijan täytyy pystyä selvittämään järjestelmän eri osapuolilta teknisiin laatuominaisuuksiin liittyvät odotukset ja arvioida riskit, joita niihin liittyy.

Yleinen lähestymistapa on olettaa, että mikäli asiakas on tyytyväinen olemassa olevaan järjestelmän versioon, hän on tyytyväinen myös uusiin versioihin, kunhan jo saavutetut laatutasot pysyvät ennallaan. Tämä mahdollistaa järjestelmän olemassa olevan version käyttämisen vertailutasona. Tämä lähestymistapa voi olla erityisen hyödyllinen käytettäväksi joidenkin sellaisten ei-toiminnallisten laatuominaisuuksien kohdalla, joiden osalta sidosryhmien voi olla vaikea määrittää vaatimuksiaan, kuten esimerkiksi suorituskyky.

Ei-toiminnallisia vaatimuksia kerättäessä on syytä ottaa huomioon useita eri näkökulmia. Niitä täytyy koota eri sidosryhmiltä, kuten asiakkailta, käyttäjiltä, operaattorihenkilöiltä ja ylläpitäjiltä; muutoin jotkut vaatimukset jäävät todennäköisesti huomaamatta.

4.2.2 Tarvittavat työkaluhankinnat ja koulutus

Kaupalliset työkalut tai simulaattorit ovat erityisen olennaisia suorituskykytesteissä sekä tietoturvatesteissä. Teknisten testausasiantuntijoiden pitäisi arvioida työkalujen hankintaan, oppimiseen ja käyttöönottoon liittyvät kustannukset ja aikataulut. Jos tullaan käyttämään

erityistyökaluja, suunnittelussa pitää ottaa huomioon uusien työkalujen oppimiskäyrä ja/tai kustannukset, joita syntyy ulkoisten työkaluasiantuntijoiden palkkaamisesta.

Monimutkaisen simulaattorin kehittäminen saattaa muodostaa oman kehitysprojektinsa ja se pitäisi myös suunnitella sellaiseksi. Toteutetun työkalun testaus ja dokumentointi on erityisesti otettava huomioon aikataulussa ja resurssisuunnitelmassa. Simulaattorin päivitykseen ja uudelleentestaukseen pitää varata riittävästi rahaa ja aikaa, kun simuloitu tuote muuttuu. Turvallisuuskriittisten sovellusten testaamisessa käytettävien simulaattoreiden suunnittelussa on otettava huomioon riippumattoman osapuolen simulaattorille suorittama hyväksymistestaus ja mahdollinen sertifiointi.

4.2.3 Testiympäristöön liittyvät vaatimukset

Monet tekniset testit (esim. tietoturvatestit, suorituskykytestit) vaativat tuotannonkaltaisen testiympäristön, jotta niistä saadaan realistisia tuloksia. Testattavan järjestelmän koosta ja monimutkaisuudesta riippuen tällä voi olla merkittävä vaikutus testien suunnitteluun ja rahoitukseen. Koska tällaisten ympäristöjen kustannukset saattavat olla korkeita, voidaan harkita seuraavia vaihtoehtoja:

- Käytetään tuotantoympäristöä.
- Käytetään pienemmäksi skaalattua versiota järjestelmästä. Tällöin on huolehdittava siitä, että saadut testitulokset ovat riittävän edustavia tuotantojärjestelmän kannalta.

Tällaisten testien suorituksen ajoitus täytyy suunnitella huolellisesti ja on varsin todennäköistä, että ne voidaan suorittaa vain tiettyinä aikoina (esim. kun järjestelmän käyttöaste on matala).

4.2.4 Organisaation kannalta huomioon otettavia seikkoja

Teknisiin testeihin saattaa kuulua kokonaisen järjestelmän useiden eri komponenttien käyttäytymisen mittaaminen (esim. palvelimet, tietokannat, verkot). Mikäli nämä komponentit on hajautettu moniin eri paikkoihin ja organisaatioihin, testien suunnitteluun ja koordinointiin tarvittava työpanos saattaa olla huomattava. Esimerkiksi tietyt ohjelmistokomponentit voivat olla käytettävissä järjestelmätestaukseen vain tiettyinä aikoina päivästä tai vuodesta, tai organisaatiot voivat tarjota testaustukea vain tietyn määrän päiviä. Jos jätetään varmistamatta, että toisista organisaatioista olevat järjestelmän komponentit ja henkilöstö (eli ”lainattu” asiantuntemus) ovat tarvittaessa käytettävissä testaustarpeisiin, suunniteltu testausaikataulu saattaa pettää pahasti.

4.2.5 Tietoturvaan liittyvät seikat

Tietyt järjestelmään toteutetut tietoturvaominaisuudet on otettava huomioon testien suunnitteluvaiheessa, jotta voidaan varmistaa, että kaikki testaustehtävät ovat mahdollisia. Esimerkiksi aineiston salaus voi tehdä testiaineiston laatimisesta ja tulosten todentamisesta vaikeaa.

Tietosuojapolitiikka ja lait voivat estää tuotantoaineistoon pohjautuvan testiaineiston laatimisen. Testiaineiston tunnistamattomaksi muokkaaminen on olennainen tehtävä, joka täytyy suunnitella osaksi testien toteuttamista.

4.3 Tietoturvatestaus

4.3.1 Esittely

Tietoturvatestaus eroaa muista toiminnallisen testauksen muodoista kahdella merkittävällä tavalla:

1. Tavalliset testien syöteaineiston valinnassa käytettävät tekniikat voivat aiheuttaa tärkeiden tietoturvaongelmien huomaamatta jäämisen.
2. Tietoturvavikojen oireet poikkeavat suuresti niistä, joita tulee esiin muita toiminnallisia testityyppejä käytettäessä.

Tietoturvatestaus arvioi järjestelmän alttiutta uhkille pyrkimällä vaarantamaan järjestelmän tietoturvamenettelyt. Seuraavassa listassa on esitetty mahdollisia uhkia, joita pitäisi tutkia tietoturvatestauksen aikana:

- Luvaton sovellusten tai aineiston kopiointi
- Pääsyn hallinnan virheet (esim. mahdollisuus suorittaa tehtäviä, joihin käyttäjällä ei ole oikeuksia). Käyttöoikeudet, pääsyt ja valtuudet ovat tämän testauksen keskipiste. Näiden tietojen pitäisi olla saatavilla järjestelmän määrityskuvauksista.
- Ohjelmistosta ilmenee odottamattomia sivuilmiöitä, kun sillä suoritetaan sillä tehtäväksi tarkoitettuja toimintoja. Esimerkiksi mediasoitin voi toistaa äänitiedostot oikein, mutta tallentaa samalla tiedostot salaamattomaan tilapäiseen tietovarastoon ja luo näin ohjelmistopiraateille mahdollisuuden niiden hyväksikäyttämiseen.
- Websivulle syötetty koodi, jonka seuraavat käyttäjät voivat suorittaa (”cross-site scripting”, XSS). Tällainen koodi voi olla tarkoitettu aiheuttamaan vahinkoa.
- Puskurin ylivuoto, joka voidaan aiheuttaa syöttämällä käyttöliittymän syötekenttään merkkijonoja, jotka ovat pidempiä kuin mitä ohjelmakoodi pystyy käsittelemään oikein. Puskurin ylivuodon mahdollistavat haavoittuvuudet luovat mahdollisuuden vahingollisen koodin suorittamiseen.
- Palvelunesto, mikä estää käyttäjiä käyttämästä sovellusta (esim. kuormittamalla web-palvelinta ”häiritsevillä” palvelupyynnöillä).
- Kolmannen osapuolen suorittama tietojen sieppaus, kopiointi ja/tai muuntaminen ja edelleentoimittaminen ilman, että käyttäjä on tietoinen kolmannesta osapuolesta (”Välikäsihyökkäys”).
- Arkaluontoisten tietojen suojaamiseen käytettyjen kryptauskoodien murtaminen.
- Logiikkapommit (kutsutaan myös joskus Pääsiäismuniksi), joita voidaan tahallisesti sijoittaa koodiin ja jotka aktivoituvat vain tietyissä olosuhteissa (esim. tiettynä päivänä). Kun logiikkapommit aktivoituvat, ne voivat suorittaa vahingollisia toimenpiteitä, kuten esimerkiksi poistaa tiedostoja tai alustaa levyjä.

4.3.2 Tietoturvatestauksen suunnittelu

Yleisesti ottaen seuraavat seikat ovat erityisen keskeisiä tietoturvatestauksen suunnittelussa:

- Koska tietoturvaongelmia voi syntyä sekä järjestelmän arkkitehtuuriin, suunnitteluun että käyttöönottoon liittyen, tietoturvatestausta voidaan aikatauluttaa yksikkö-, integraatio- ja järjestelmätestaustasoille. Tietoturvauhkien muuttuvan luonteen vuoksi tietoturvatestejä voidaan aikatauluttaa myös suoritettavaksi säännöllisesti sen jälkeen, kun järjestelmä on otettu käyttöön.
- Teknisen testausasiantuntijan esittämiin testausstrategioihin saattavat kuulua koodikatselmoinnit ja tietoturvatyökalujen avulla tehty staattinen analyysi. Ne voivat olla tehokkaita löytämään arkkitehtuurista, suunnitteludokumenteista ja koodista tietoturvaongelmia, jotka jäävät helposti huomaamatta dynaamisessa testauksessa.
- Teknistä testausasiantuntijaa voidaan pyytää suunnittelemaan ja suorittamaan määrättyjä “tietoturvahyökkäyksiä” (ks. alla), jotka voivat vaatia huolellista suunnittelua ja koordinointia sidosryhmien kanssa. Toiset tietoturvatestit (esim. käyttöoikeuksien, pääsyn ja valtuuksien testaus) voidaan suorittaa yhdessä kehittäjien tai Testausasiantuntijoiden kanssa. Tietoturvatestejä suunniteltaessa on esim. tällaisiin organisatorisiin asioihin kiinnitettävä tarkasti huomiota.
- Tietoturvatestauksen suunnittelun keskeinen osa on valtuuksien hankinta. Teknisen testausasiantuntijan osalta tämä tarkoittaa, että hänen on saatava Testauspäälliköltä yksiselitteisesti määritelty lupa suunniteltujen tietoturvatestien suorittamiseksi. Mitkä tahansa ylimääräiset, etukäteen suunnittelematta suoritetut testit voivat vaikuttaa todellisilta hyökkäyksiltä ja niitä suorittava henkilö voi olla vaarassa joutua oikeudellisten toimenpiteiden kohteeksi. Jos olemassa ei ole mitään kirjallista dokumentaatiota testien tarkoituksesta ja valtuuksista, voi olla vaikea selittää vakuuttavasti puolustusta “Me suoritimme tietoturvatestejä”.
- On syytä huomata, että tietoturvaa parantavat järjestelmän muutokset voivat vaikuttaa sen suorituskykyyn. Tietoturvaan liittyvien parannusten jälkeen on suositeltavaa harkita suorituskykytestien suorittamisen tarvetta (ks. kappale 4.5 alla).

4.3.3 Tietoturvatestien määrittely

Tietyt tietoturvatestit voidaan ryhmitellä [Whittaker02] tietoturvariskin alkuperän mukaan:

- Käyttöliittymään liittyvät – luvaton pääsy ja vahingolliseksi tarkoitetut syötteet
- Tiedostojärjestelmään liittyvät – pääsy käsiksi tiedostojen tai tietovaraston sisältämiin arkaluontoisiin tietoihin
- Käyttöjärjestelmään liittyvät – järjestelmän muistissa säilytetään arkaluontoista tietoa, kuten ei-kryptaamattomia salasanoja, jotka saattavat paljastua, kun järjestelmä kaatuu vahingollisten syötteiden vuoksi
- Ulkoisiin ohjelmistoihin liittyvät – toiminnot, joita voi tapahtua järjestelmän hyödyntäessä ulkoisia komponentteja. Näitä voi olla verkkotasolla (esim. toimitetaan virheellisiä tietopaketteja tai viestejä) tai ohjelmistokomponenttien tasolla (esim. häiriö ohjelmistokomponentissa, josta ohjelmisto on riippuvainen).

Seuraavia lähestymistapoja [Whittaker04] voidaan käyttää tietoturvatestien kehittämisessä:

- Kerätään tietoja, jotka voivat olla hyödyllisiä testien määrittelyssä, kuten esim. työntekijöiden nimet, fyysiset osoitteet, sisäisiin verkkoihin liittyvät yksityiskohdat, IP-osoitteet, ohjelmiston tai laitteiston yksilöivät tiedot ja käyttöjärjestelmäversiot.
- Suoritetaan haavoittuvuuskartoituksia käyttämällä yleisesti saatavilla olevia työkaluja. Tällaisia työkaluja ei käytetä suoraan järjestelmän saattamiseen uhanalaiseksi, vaan tunnistamaan haavoittuvuuksia, jotka ovat tietoturvapolitiikan rikkomuksia tai saattavat johtaa sellaisiin. Määrättyjä haavoittuvuuksia voidaan tunnistaa myös käyttämällä tarkistuslistoja, joita on laatinut esim. National Institute of Standards and Technology (NIST). [Web-2]
- Laaditaan “hyökkäyssuunnitelmia” (suunnitelmia testaustoimenpiteiksi, joilla pyritään vaarantamaan tietyn järjestelmän tietoturvapolitiikka) hyödyntämällä kerättyjä tietoja. Kaikkein vakavimpien tietoturvavikojen paljastamiseksi on hyökkäyssuunnitelmaan määriteltävä useita eri liittymien (esim. käyttöliittymä, tiedostojärjestelmä) kautta toteutettavia syötteitä. Erilaiset [Whittaker04]:ssä kuvatut “hyökkäykset” ovat arvokas erityisesti tietoturvatestaukseen suunniteltu tekniikoiden lähde.

Tietoturvaongelmia voidaan paljastaa myös katselmointien avulla (ks. luku 5) ja/tai käyttämällä staattisen analyysin työkaluja (ks. kappale 3.2). Staattisen analyysin työkalut sisältävät laajan joukon sääntöjä, jotka liittyvät erityisesti tietoturvauhkiin ja joita vastaan koodi tarkastetaan. Esimerkiksi puskurin ylivuoto-ongelmat, jotka johtuvat siitä, että puskurin kokoa ei tarkasteta ennen tiedon syöttämistä, voidaan löytää vain työkalun avulla.

Staattisen analyysin työkaluja voidaan käyttää web-koodin tarkistamiseen, kun pyritään paljastamaan koodisyötteisiin, evästetietoturvaan, cross site scriptingiin (XSS), resurssien väärinkäyttöön ja SQL-koodisyötteisiin liittyviä tietoturvahaavoittuvuuksia.

4.4 Luotettavuustestaus

ISO 9126 –standardissa tuotteen laatuominaisuuksien luokittelussa määritellään luotettavuudelle seuraavat aliominaisuudet:

- Kypsyys
- Vikasietoisuus
- Toipuvuus

4.4.1 Ohjelmiston kypsyyden mittaaminen

Luotettavuustestauksen tavoitteena on seurata tilastojen avulla ohjelmistokypsyyden kehitystä ajan myötä ja verrata tätä toivottuun luotettavuustasoon, joka voidaan kirjata esimerkiksi palvelutasosopimukseen (Service Level Agreement, SLA). Tällaisia mittareita voivat olla esimerkiksi Häiriöiden välinen keskimääräinen aika (Mean Time Between Failure, MBTF), Korjauksen keskimääräinen kesto (Mean Time To Repair, MTTR) tai mitkä tahansa muut häiriötiheyteen liittyvät

mittarit (esim. tiettyyn vakavuusluokkaan kuuluvien häiriöiden viikoittainen määrä). Näitä voidaan käyttää lopetuskriteereinä (esim. tuotantoon julkaisua varten).

4.4.2 Vikasietoisuuden testit

Sen lisäksi, että toiminnallisilla testeillä arvioidaan ohjelmiston vikasietoisuutta odottamattomien syötearvojen käsittelyn kannalta (niin kutsutut negatiiviset testit), tarvitaan lisätestejä järjestelmän vikasietoisuuden arvioimiseksi, kun kyse on testattavaan järjestelmään ulkoapäin kohdistuvista häiriöistä. Käyttöjärjestelmä raportoi tyypillisesti tällaisista häiriöistä (esim. levy täynnä, prosessi tai palvelu ei ole käytettävissä, tiedostoa ei löydy, muisti ei ole käytettävissä). Erityistyökaluja voidaan käyttää vikasietoisuuden testaamiseen järjestelmätestaustasolla.

On huomattava, että termejä “vakaus” ja “virhesietoisuus” käytetään yleisesti myös silloin, kun puhutaan vikasietoisuudesta (ks. ISTQB sanasto [ISTQB_GLOSSARY]).

4.4.3 Toipuvuustestaus

Luotettavuustestauksen pidemmälle viedyt muodot testaavat ohjelmistojärjestelmän kykyä toipua laitteisto- tai ohjelmistohäiriöistä ennalta sovitulla tavalla, joka lopulta mahdollistaa normaaliin toimintaan palaamisen. Toipuvuustesteihin kuuluvat häiriötilannetestaus sekä varmuuskopioinnin ja palautuksen testaus.

Häiriötilannetestejä suoritetaan, kun ohjelmistohäiriön seuraukset ovat niin negatiiviset, että ohjelmiston toiminnan varmistamiseksi myös häiriötilanteen sattuessa on laadittu erityiset laitteistoon ja/tai ohjelmistoon liittyvät toimenpiteet. Häiriötilannetestejä voidaan suorittaa esimerkiksi silloin, kun taloudellisten menetysten riski on äärimmäisen suuri, tai jos on olemassa kriittisiä tietoturvaongelmia. Jos häiriöt voivat olla seurausta katastrofaalisista tapahtumista, tämän tyyppistä toipuvuustestausta voidaan kutsua myös ”häiriöstäpalautumistestaukseksi”.

Tyypillisiin laitteistohäiriöitä ennaltaehkäiseviin toimenpiteisiin voivat kuulua kuorman tasaaminen useiden prosessoreiden välillä ja palvelinten, prosessoreiden tai levyjen ryhmittäminen (klusterointi) niin, että toinen voi välittömästi alkaa hoitaa tehtäviä, mikäli jossain niistä syntyy häiriötilanne (kahdennetut järjestelmät). Tyypillinen ohjelmistoon liittyvä toimenpide voi olla, että ohjelmistojärjestelmästä (esim. lentokoneen lennonhallintajärjestelmä) asennetaan useampi itsenäinen versio niin kutsuttuihin eriytettyihin kahdennetun järjestelmiin. Kahdennetut järjestelmät ovat tyypillisesti ohjelmisto- ja laitteistotoimenpiteiden yhdistelmä, ja sitä voidaan kutsua kaksinkertaiseksi, kolminkertaiseksi tai nelinkertaiseksi järjestelmäksi, riippuen siitä, kuinka monta itsenäisiä järjestelmiä on laadittu. Järjestelmän eriyttämisnäkökulma saavutetaan, kun samat järjestelmävaatimukset toimitetaan kahdelle (tai useammalle) itsenäiselle ja toisistaan riippumattomalle kehitystiimille, tavoitteena saada aikaan samat palvelut eri ohjelmistojen tuottamana. Tämä suojaa eriytettyjä kahdennettuja järjestelmiä siinä, että sama viallinen syöte tuottaa pienemmällä todennäköisyydellä saman tuloksen. Nämä järjestelmän toipumista parantavat toimenpiteet saattavat vaikuttaa suoraan myös järjestelmän luotettavuuteen ja niitä pitäisi myös harkita, kun suoritetaan luotettavuustestausta.

Häiriötilannetestaus on suunniteltu nimenomaan järjestelmien testaukseen simuloimalla häiriötiloja tai aiheuttamaan todellisia häiriöitä valvotussa ympäristössä. Häiriön jälkeen häiriötilannemenettelyt testataan sen varmistamiseksi, että tietoja ei ole menetetty tai ne eivät ole korruptoituneet ja että toiminta on pysynyt sovitulla palvelutasolla (esim. toimintojen saatavuus tai vasteajat). Lisätietoja häiriötilannetestauksesta: ks. [Web-1]

Varmuuskopioinnin ja palautuksen testaus keskittyy niihin toimenpiteisiin, jotka on laadittu häiriön seurausten minimoimiseksi. Nämä testit arvioivat menetelmiä erityyppisten varmuuskopioiden ottamiseksi (yleensä dokumentoitu ohjekirjaan) sekä kopioitujen tietojen palauttamiseksi, mikäli tiedot on menetetty tai ne ovat korruptoituneet. Testitapaukset suunnitellaan sen varmistamiseksi, että jokaisen proseduurin kriittiset polut on katettu. Teknisiä katselmointeja voidaan käyttää tällaisten tilanteiden “kuivaharjoitteluun” sekä ohjekirjojen oikeellisuuden tarkistamiseen todellisia toimenpiteitä

vasten. Käyttöön soveltuvuuden hyväksymistestauksessa skenaariotilanteet suoritetaan tuotanto- tai tuotannon kaltaisessa ympäristössä niiden todellisen käytön kelpuuttamiseksi.

Varmuuskopioinnin ja palautuksen testaukseen liittyviin toimenpiteisiin voivat sisältyä
- eri tyyppisten varmuuskopioiden (esim. täysi, osittainen) tekemiseen kuluva aika
- tietojen palauttamiseen kuluva aika
- tietojen palauttamiselle luvatut tasot (esim. kaiken alle 24 tuntia vanhan aineiston palautus, tietyntyyppisen korkeintaan tunnin vanhan tapahtuma-aineiston palautus)

4.4.4 Luotettavuustestauksen suunnittelu

Yleisesti ottaen seuraavat seikat ovat erityisen keskeisiä luotettavuustestauksen suunnittelussa:
- Luotettavuuden seuranta voi jatkua vielä sen jälkeen, kun ohjelmisto on otettu tuotantokäyttöön. Ohjelmiston toiminnasta vastaavan organisaation ja henkilöiden kanssa on neuvoteltava, kun kerätään luotettavuusvaatimuksia testaussuunnittelua varten.
- Tekninen testausasiantuntija voi valita luotettavuuden kasvumallin, joka kuvaa odotetut luotettavuustasot tietyn ajan kuluessa. Luotettavuuden kasvumalli voi tuottaa Testauspäällikölle hyödyllistä tietoa, sillä se mahdollistaa odotettujen ja saavutettujen luotettavuustasojen vertailun.
- Luotettavuustestit pitäisi suorittaa tuotannonkaltaisessa ympäristössä. Käytössä olevan ympäristön tulisi pysyä mahdollisimman vakaana, jotta ajan myötä kehittyvien luotettavuustrendien seuraaminen olisi mahdollista.
- Koska luotettavuustestit vaativat usein koko järjestelmän käyttöä, luotettavuustestaus tehdään tyypillisimmin osana järjestelmätestausta. Luotettavuustestaus voidaan kuitenkin suorittaa myös yksittäisille komponenteille samoin kuin integroiduille komponenttijoukoille. Yksityiskohtaisia arkkitehtuurin, suunnittelukuvausten ja koodin katselmointeja voidaan myös käyttää joidenkin käyttöönotetussa järjestelmässä esiintyvien luotettavuusongelmiin liittyvien riskien poistamiseen.
- Tilastollisesti merkittävien testitulosten aikaansaamiseksi luotettavuustestit vaativat yleensä pitkiä suoritusaikoja. Tämä voi aiheuttaa vaikeuksia niiden aikatauluttamisessa muiden suunniteltujen testien kanssa.

4.4.5 Luotettavuustestien määrittely

Luotettavuustestaus voidaan tehdä ennalta määrättyä testijoukkoa toistamalla. Nämä testit voidaan valita satunnaisesti testitapausten joukosta tai ne voivat olla testejä, jotka on luotu tilastollisen mallin pohjalta satunnais- tai puolisatunnaismenetelmiä käyttämällä. Testit voivat pohjautua myös käyttömalleihin, joita joskus kutsutaan ”käyttöprofiiliksi” (ks. kappale 4.5.4).

Joidenkin luotettavuustestien osalta voidaan määritellä, että erityisesti muistia rasittavat toimenpiteet pitää suorittaa toistuvasti mahdollisten muistivuotojen havaitsemiseksi.

4.5 Suorituskykytestaus

4.5.1 Esittely

ISO 9126 –standardin mukaisessa tuotteen laatuominaisuuksien luokittelussa suorituskyky (aikakäyttäytyminen) määritellään toiminnallisen tehokkuuden alaominaisuudeksi. Suorituskykytestaus keskittyy komponentin tai järjestelmän kykyyn vastata käyttäjän tai järjestelmän syötteisiin määrätyn ajan kuluessa määrätyissä olosuhteissa.

Suorituskyvyn mittaus vaihtelee testauksen tavoitteiden mukaan. Yksittäisten ohjelmistokomponenttien osalta suorituskykyä voidaan mitata prosessorin käytön perusteella, kun taas asiakaspohjaisten järjestelmien suorituskykyä voidaan arvioida sen mukaan, kuinka kauan tiettyyn käyttäjätoimintoon vastaaminen kestää. Arkkitehtuuriltaan useista komponenteista (esim. asiakkaat,

palvelimet, tietokannat) koostuvien järjestelmien suorituskykytiedot kerätään yksittäisten komponenttien välisistä tapahtumista, jotta suorituskyvyn ”pullonkaulat” voidaan tunnistaa.

4.5.2 Suorituskykytestauksen tyypit

4.5.2.1 Kuormitustestaus

Kuormitustestaus keskittyy järjestelmän kykyyn selviytyä realistisesta vaiheittain kasvavasta kuormasta, joka syntyy useiden yhtäaikaisten käyttäjien tai prosessien aiheuttamista tapahtumapyynnöistä. Testauksessa voidaan mitata ja analysoida käyttäjien tyypillisissä käyttötilanteissa (käyttöprofiilit) kokemia vasteaikoja. Ks. myös [Splaine01].

4.5.2.2 Rasitustestaus

Rasitustestaus keskittyy järjestelmän tai komponentin kykyyn käsitellä ennakoidun tai määritetyn työkuorman rajalle tai sen yli nousevia kuormahuippuja, tai toimia tilanteessa, jossa resurssien, kuten tietokonekapasiteetin tai kaistanleveyden saatavuus on vähentynyt. Suoritustason pitäisi heikentyä tasaisesti ja ennakoitavasti ilman häiriöitä, kun rasitustasoa nostetaan. Järjestelmän ollessa rasituksen alaisena pitää erityisesti testata järjestelmän toiminnallinen eheys mahdollisten toiminnallisessa prosessoinnissa olevien vikojen tai aineiston epäjohdonmukaisuuksien paljastamiseksi.

Yksi mahdollinen rasitustestauksen tavoite on löytää rajat, jolloin järjestelmä todella kaatuu, jotta “ketjun heikoin lenkki” voidaan selvittää. Rasitustestaus mahdollistaa lisäkapasiteetin lisäämisen järjestelmään sopivalla tavalla (esim. muisti, prosessoriteho, tietokannan koko).

4.5.2.3 Skaalautuvuustestaus

Skaalautuvuustestaus keskittyy järjestelmän kykyyn vastata tulevaisuudessa esiin nouseviin tehokkuusvaatimuksiin, jotka saattavat olla tämänhetkisiä suuremmat. Testien tavoitteena on määritellä järjestelmän kyky kasvaa (esim. enemmän käyttäjiä, enemmän varastoitavaa aineistoa) ilman, että alitetaan kyseisellä hetkellä voimassa olevat suorituskykyvaatimukset tai että järjestelmä kaatuu. Kun skaalautuvuuden rajat tunnetaan, voidaan määritellä mahdollisista ongelmista varoittavat raja-arvot, joita seurataan tuotannossa. Lisäksi tuotantoympäristöä voidaan muokata sopivalla laitteistomäärällä.

4.5.3 Suorituskykytestauksen suunnittelu

Kappaleessa 4.2 kuvattujen yleisten testaukseen suunnitteluun liittyvien seikkojen lisäksi seuraavat asiat voivat vaikuttaa suorituskykytestien suunnitteluun:

- Käytössä olevasta ympäristöstä ja testattavana olevasta ohjelmistosta riippuen (ks. kappale 4.2.3) suorituskykytestaus voi edellyttää koko järjestelmän toteuttamista, jotta testaus voidaan suorittaa tehokkaasti. Tässä tapauksessa suorituskykytestaus aikataulutetaan yleensä tehtäväksi järjestelmätestauksen aikana. Muut suorituskykytestit, jotka voidaan suorittaa tehokkaasti komponenttitasolla, voidaan aikatauluttaa tehtäväksi yksikkötestauksen aikana.
- Yleisesti on suositeltavaa suorittaa ensimmäiset suorituskykytestit niin aikaisin kuin mahdollista, vaikka tuotannonkaltainen ympäristö ei olisikaan vielä käytettävissä. Nämä varhaiset testit voivat löytää suorituskykyongelmia (esim. pullonkaulat) ja vähentää projektiriskejä pienentämällä myöhemmissä ohjelmistokehitysvaiheissa tai tuotannossa tehtävien aikaa vievien korjausten määrää.
- Varsinkin tietokantojen ja komponenttien väliseen yhteistoimintaan sekä virheidenkäsittelyyn keskittyvät koodikatselmoinnit voivat paljastaa suorituskykyongelmia (erityisesti “odota ja yritä uudelleen” -logiikkaan sekä tehottomiin kyselyihin liittyviä) ja ne pitäisi ajoittaa ohjelmiston elinkaaren aikaiseen vaiheeseen.
- Suorituskykytestien ajamiseksi tarvittavat laitteisto, ohjelmisto ja verkon kaistanleveys pitää suunnitella ja budjetoida. Tarpeet riippuvat pääasiassa luotavan kuorman määrästä, joka voi perustua simuloitavien virtuaalikäyttäjien ja heidän luomansa todennäköiseen verkkoliikenteen määrään. Tämän huomiotta jättäminen saattaa aiheuttaa sen, että suorituskyvystä kerätyt mittaritiedot eivät edusta todellista tilannetta. Esimerkiksi paljon käytetyn Internet-sivuston

skaalautuvuusvaatimusten todentaminen saattaa vaatia satojentuhansien virtuaalikäyttäjien simuloimista.
- Suorituskykytestejä varten tarvittavan kuorman luominen voi vaikuttaa merkittävästi laitteistoon ja työkalujen hankintakustannuksiin. Tämä täytyy ottaa huomioon suorituskykytestauksen suunnittelussa sen varmistamiseksi, että riittävä rahoitus on saatavilla.
- Kuorman luomiseen liittyvät kustannukset voidaan minimoida vuokraamalla tarvittava testausinfrastruktuuri. Tämä voi tarkoittaa esimerkiksi, että vuokrataan suorituskykytyökaluja, joiden lisenssimäärää voidaan kasvattaa, tai että käytetään kolmannen osapuolen palveluita laitteistovaatimusten täyttämiseen (esim. pilvipalvelut). Tällöin suorituskykytestien suorittamiseen käytettävissä oleva aika voi olla rajallinen ja siksi sen käyttö täytyy suunnitella huolellisesti.
- Suunnitteluvaiheessa pitää huolehtia siitä, että varmistetaan käytettävien suorituskykytyökalujen yhteensopivuus testattavana olevan järjestelmän käyttämien viestintäprotokollien kanssa.
- Suorituskykyyn liittyvillä vioilla on usein merkittävä vaikutus testattavana olevaan järjestelmään. Jos suorituskykyvaatimukset ovat keskeisen tärkeitä, on usein hyödyllistä tehdä suorituskykytestejä kriittisille komponenteille (hyödyntämällä ajureita ja tynkiä) sen sijaan, että odotetaan järjestelmätestaukseen asti.

4.5.4 Suorituskykytestien määrittely

Testien määrittely erilaisia suorituskykytestaustyyppejä varten, kuten kuormitustestaukseen ja rasitustestaukseen, perustuu käyttöprofiilien määrittelyyn. Profiilit edustavat käyttäjien määrättyjä toimintatapoja sovelluksen eri käyttötilanteissa. Kullakin sovelluksella voi olla useita käyttöprofiileja.

Käyttöprofiilikohtainen käyttäjien lukumäärä voidaan saada selville käyttämällä monitorointityökaluja (silloin, jos varsinainen tai verrattavissa oleva sovellus on jo käytettävissä) tai arvioimalla sovelluksen käyttöastetta. Tällaiset arviot voivat perustua algoritmeihin tai ne voivat olla liiketoimintaorganisaatiolta saatuja. Ne ovat erityisen tärkeitä silloin, kun määritellään skaalautuvuustestauksessa käytettäviä käyttöprofiileja.

Käyttöprofiilit muodostavat perustan suorituskykytestauksessa käytettävien testien määrälle ja tyypeille. Näitä testejä hallinnoidaan usein testaustyökalulla, joka luo suuria määriä “virtuaalisia” tai simuloituja käyttäjiä, jotka edustavat testattavana olevaa käyttöprofiilia (ks. kappale 6.3.2).

4.6 Resurssien käyttö

Resurssien käyttö määritellään ISO 9126 –standardissa tuotteen laatuominaisuuksien luokittelussa toiminnallisen tehokkuuden alaominaisuudeksi. Resurssien käyttöön liittyvät testit arvioivat järjestelmäresurssien (esim. muistin, levytilan, verkon kaistanleveyden, yhteyksien) käyttöä ennalta määritettyjä vertailuarvoja vasten. Näitä verrataan sekä normaalissa kuormitustilanteessa että esim. suurien tapahtuma- ja tietomäärien aiheuttamissa rasitustilanteissa, jotta voidaan arvioida, tapahtuuko silloin epänormaalia käytön kasvua.

Esimerkiksi muistin käytöllä (johon joskus viivataan “muistin jalanjälkenä”) on merkittävä rooli reaaliaikaisten sulautettujen järjestelmien suorituskykytestauksessa. Jos muistin jalanjälki ylittää sallitut määrät, järjestelmällä voi olla käytettävissään liian vähän muistia, jotta se pystyisi suorittamaan tehtävänsä määrätyssä ajassa. Tämä voi hidastaa järjestelmän toimintaa ja johtaa jopa järjestelmän kaatumiseen.

Resurssien käytön tutkimiseen (ks. kappale 3.3.4) ja suorituskyvyn pullonkaulojen löytämiseen voidaan käyttää myös dynaamista analyysiä.

4.7 Ylläpidettävyystestaus

Järjestelmän elinajasta kuluu usein merkittävästi suurempi osa sen ylläpitämiseen kuin itse järjestelmän toteutukseen. Ylläpitotestausta tehdään käytössä olevalle järjestelmälle siihen tehtyjen muutosten tai siihen liittyvien ympäristön muutosten vaikutusten testaamiseksi. Ylläpidettävyystestauksella mitataan, kuinka helposti koodi on analysoitavissa, muutettavissa ja testattavissa, jotta varmistetaan, että ylläpitotehtävien tekeminen on niin tehokasta kuin mahdollista.

Tyypillisiin sidosryhmiin (esim. ohjelmiston omistaja tai operaattori) vaikuttaviin ylläpidettävyyden tavoitteisiin kuuluvat:

- Ohjelmiston omistamiseen tai operointiin liittyvien kustannusten minimointi
- Ohjelmiston ylläpitoon tarvittavan alhaallaoloajan minimointi.

Ylläpidettävyystestit pitäisi sisällyttää testausstrategiaan ja/tai testauksen lähestymistapoihin silloin, kun yksi tai useampia seuraavista pitää paikkansa:

- Ohjelmistomuutokset ovat todennäköisiä ohjelmiston tuotantoonsiirron jälkeen (esim. vikojen korjaukset tai suunniteltujen päivitysten toteuttaminen)
- Asianomaisten sidosryhmien edustajat pitävät ylläpidettävyyden tavoitteiden (ks. edellä) saavuttamisesta saatuja hyötyjä suurempina kuin ylläpidettävyystestien suorituksesta ja tarvittavien muutosten tekemisestä aiheutuvia kustannuksia.
- Ohjelmiston huonon ylläpidettävyyden aiheuttamat riskit (esim. pitkät vasteajat käyttäjien ja/tai asiakkaiden raportoimiin vikoihin) ovat peruste ylläpidettävyystestien suorittamiselle.

Ylläpidettävyyden testaamiseen soveltuviin tekniikoihin kuuluvat staattinen analyysi ja katselmoinnit, kuten on todettu kappaleissa 3.2 ja 5.2. Ylläpidettävyyden testauksen pitäisi alkaa heti, kuin suunnitteludokumentit ovat käytettävissä ja sen pitäisi jatkua koko ohjelmakoodin toteutuksen ajan. Koska ylläpidettävyys rakennetaan sisään jokaisen yksittäisen komponentin koodiin ja dokumentaatioon, ylläpidettävyyttä voidaan arvioida jo aikaisessa elinkaaren vaiheessa sen sijaan, että täytyisi odottaa valmista ja toimivaa järjestelmää.

Dynaaminen ylläpidettävyystestaus keskittyy tietyn sovelluksen ylläpitämiseksi laadittuihin dokumentoituihin toimintatapoihin (esim. ohjelmistopäivitysten tekeminen). Erilaisia ylläpitoskenaarioita käytetään testitapauksina sen varmistamiseksi, että vaadittavat palvelutasot ovat saavutettavissa dokumentoiduilla toimenpiteillä. Tämä testauksen muoto on erityisen keskeinen silloin, kun taustalla oleva infrastruktuuri on monimutkainen ja toimintaa tukeviin toimenpiteisiin liittyy monia osastoja/organisaatioita. Tätä testausta voidaan tehdä osana Käyttöön soveltuvuuden hyväksymistestausta. [Web-1]

4.7.1 Analysoitavuus, muutettavuus, vakaus ja testattavuus

Järjestelmän ylläpidettävyyttä voidaan arvioida sillä, kuinka paljon työtä tarvitaan järjestelmään liittyvien ongelmien määrittämiseen (analysoitavuus), koodin muutosten tekemiseen (muutettavuus) ja muutetun järjestelmän testaamiseen (testattavuus). Vakaus liittyy erityisesti siihen, miten järjestelmä vastaa muutokseen. Järjestelmissä, joiden vakaus on heikko, esiintyy paljon eteenpäin kopioituvia ongelmia eli seurannaisvaikutuksia aina, kun muutoksia tehdään. [ISO9126] [Web-1]

Ylläpitotehtävien tekemiseen tarvittava työmäärä riippuu monista tekijöistä, kuten ohjelmiston suunnittelutavasta (esim. olio-ohjautunut) ja käytetyistä ohjelmointistandardeista.

Huomaa, että tässä yhteydessä “vakautta” ei pidä sotkea termeihin “robustius” ja ”vikasietoisuus”, joita käsitellään kappaleessa 4.4.2.

4.8 Siirrettävyystestaus

Yleisesti ottaen siirrettävyystestit liittyvät siihen, kuinka helposti ohjelmisto voidaan siirtää aiottuun käyttöympäristöönsä, joko alunperin tai jo olemassa olevasta ympäristöstä. Siirrettävyystestaukseen

kuuluvat asennettavuuden, yhdessätoimivuuden/yhteensopivuuden, sovitettavuuden ja korvattavuuden testaus. Siirrettävyystestaus voi alkaa yksittäisistä tekijöistä (esim. määrätyn komponentin korvattavuus, kuten esim. siirtyminen tietokannanhallintajärjestelmästä toiseen) ja laajentua kohdealueeltaan sitä mukaa, kun enemmän koodia on saatavilla. Asennettavuutta ei välttämättä voida testata ennen kuin kaikki tuotteen osat ovat toimintakelpoisia. Siirrettävyys pitää suunnitella ja rakentaa sisään tuotteeseen ja siksi se pitää ottaa huomioon jo aikaisin suunnittelu- ja arkkitehtuurivaiheessa. Arkkitehtuurin ja suunnitelmien katselmoinnit voivat olla erityisen hyödyllisiä siinä, että niiden avulla voidaan tunnistaa mahdollisia siirrettävyysvaatimuksia ja ongelmia (esim. riippuvuus määrätystä käyttöjärjestelmästä).

4.8.1 Asennettavuustestaus

Asennettavuustestaus tehdään sekä ohjelmistolle että kirjallisille toimintaohjeille, joita käytetään, kun ohjelmisto asennetaan kohdeympäristöön. Näihin voivat kuulua esimerkiksi ohjelmisto, joka on laadittu käytettäväksi käyttöjärjestelmän asentamisessa prosessoriin, tai ”asennusvelho”, jota käytetään tuotteen asentamisessa asiakkaan PC:lle.

Tyypillisiin asennettavuustestauksen tavoitteisiin kuuluvat seuraavat:

- Todentaa, että ohjelmisto voidaan asentaa onnistuneesti seuraamalla joko asennusohjetta (mukaan luettuna mahdollisten asennusskriptien suorittaminen) tai käyttämällä asennusvelhoa. Tähän sisältyy erilaisiin laitteisto/ohjelmistokokoonpanoihin sekä asennuksen eri vaiheisiin (esim. ensimmäinen asennus tai päivitys) liittyvien vaihtoehtojen testaaminen.
- Testata, käsitteleekö asennusohjelmisto asennuksen aikana esiin tulevat häiriöt (esim. tietyn DLL-tiedoston lataamisen epäonnistuminen) oikein niin, että järjestelmä ei jää määrittelemättömään tilaan (esim. ohjelmiston asentuminen osittain tai väärä järjestelmäkokoonpano).
- Testata, voidaanko osittainen asennus/asennuksen poistaminen suorittaa loppuun.
- Testata, osaako asennusvelho tunnistaa oikein vääränlaiset laitteistoalustat tai käyttöjärjestelmäkokoonpanot.
- Mitata, voidaanko asennusprosessi suorittaa määrätyssä ajassa vai vähemmillä tehtävillä kuin on määritelty.
- Todentaa, että ohjelmisto voidaan onnistuneesti palauttaa aiempaan versioon tai poistaa.

Asennettavuustestauksen jälkeen tehdään yleensä toiminnallisuustestaus mahdollisten asennuksen aiheuttamien vikojen löytämiseksi (esim. väärät kokoonpanot, toimintoja ei ole käytettävissä). Käytettävyystestaus tehdään yleensä rinnakkain asennettavuustestauksen kanssa. (esim. sen todentamiseksi, että käyttäjille annetaan ymmärrettäviä ohjeita ja palaute-/virheilmoituksia asennuksen aikana).

4.8.2 Yhdessätoimivuus-/yhteensopivuustestaus

Toisistaan riippumattomien tietokonejärjestelmien sanotaan olevan yhteensopivia, jos ne voivat toimia samassa ympäristössä (esim. samassa laitteistossa) ilman, että ne vaikuttavat toistensa käyttäytymiseen (esim. resurssiristiriidat). Yhteensopivuustestaus pitäisi suorittaa, kun uusi tai päivitetty ohjelmisto otetaan käyttöön ympäristössä, jossa on jo asennettuja sovelluksia.

Yhteensopivuusongelmia saattaa nousta esiin, mikäli sovellus testataan ympäristössä, jossa se on ainoa asennettu sovellus (jolloin yhteensopivuusongelmat eivät tule esiin) ja sitten se otetaan käyttöön toisessa ympäristössä (esim. tuotanto), jossa ajetaan myös muita sovelluksia.

Tyypillisiin yhteensopivuustestauksen tavoitteisiin kuuluvat seuraavat:

- Sellaisten mahdollisten toiminnallisuuteen kohdistuvien haittavaikutusten arviointi, joita saattaa ilmetä, kun sovelluksia asennetaan samaan ympäristöön (esim. ristiriitainen resurssien käyttö, kun palvelin pyörittää useaa sovellusta).
- Käyttöjärjestelmään liittyvistä korjauksista ja päivityksistä mille tahansa sovellukselle aiheutuvien vaikutusten arviointi.

Yhteensopivuusongelmat pitäisi analysoida tavoitellun tuotantoympäristön suunnittelun yhteydessä, mutta varsinaiset testit suoritetaan yleensä sen jälkeen, kun järjestelmä- ja käyttäjien hyväksymistestaus on suoritettu hyväksyttävästi.

4.8.3 Sovitettavuustestaus

Sovitettavuustestaus tutkii, voiko tietty sovellus toimia oikein kaikissa tarkoitetuissa kohdeympäristöissä (laitteisto, ohjelmisto, väliohjelmistot, käyttöjärjestelmä, jne). Sopeutuva järjestelmä on näin ollen avoin järjestelmä, joka pystyy sopeuttamaan toimintansa ympäristöön tai itse järjestelmän osiin tehtyjen muutosten perusteella. Sovitettavuustestien määrittely edellyttää, että aiotut kohdeympäristöt on tunnistettu ja konfiguroitu, ja että ne ovat testaustiimin käytettävissä. Nämä ympäristöt testataan sitten käyttämällä valikoituja toiminnallisuustestejä, jotka käyttävät kyseisessä ympäristössä olevia eri komponentteja.

Sovitettavuus voi liittyä siihen, kuinka helposti ohjelma voidaan siirtää erilaisiin määrättyihin ympäristöihin ennalta määriteltyä toimintaproseduuria käyttämällä. Testit voivat arvioida tätä proseduuria.

Sovitettavuustestit voidaan suorittaa yhdessä asennettavuustestien kanssa, ja niiden jälkeen suoritetaan tyypillisesti toiminnallisia testejä, joilla pyritään löytämään viat, joita on voinut syntyä, kun ohjelmisto on sovitettu toiseen ympäristöön.

4.8.4 Korvattavuustestaus

Korvattavuustestaus keskittyy siihen, kuinka helposti järjestelmän ohjelmistokomponentteja voidaan vaihtaa toisiin. Tämä voi olla erityisen olennaista, kun kyseessä ovat järjestelmät, jotka käyttävät valmisohjelmistoja määrätyissä järjestelmäkomponenteissa.

Korvattavuustestejä voidaan tehdä yhtä aikaa toiminnallisten integraatiotestien kanssa, jolloin valittavissa on useampi kuin yksi vaihtoehtoinen komponentti integroitavaksi koko järjestelmään. Arkkitehtuuri- ja suunnittelutasolla korvattavuutta voidaan arvioida teknisessä katselmoinnissa tai tarkastuksessa, jolloin painopiste on mahdollisten korvattavien komponenttien rajapintojen selkeässä määrittelyssä.

5. Katselmoinnit - 165 min

Avainsanat
vastamalli

Oppimistavoitteet: Katselmoinnit

5.1 Esittely

TTA-5.1.1 (K2) Selittää, miksi katselmointeihin valmistautuminen on tärkeää Tekniselle testausasiantuntijalle.

5.2 Tarkistuslistojen käyttö katselmoinneissa

TTA-5.2.1 (K4) Analysoida arkkitehtuurikuvauksia ja tunnistaa siitä ongelmia sertifikaattisisällön mukana toimitetun tarkistuslistan avulla.

TTA-5.2.2 (K4) Analysoida koodin tai pseudokoodin osia ja tunnistaa niistä ongelmia sertifikaattisisällön mukana toimitetun tarkistuslistan avulla.

5.1 Esittely

Teknisten testausasiantuntijoiden täytyy osallistua aktiivisesti katselmointiprosessiin ja tuoda esiin ainutlaatuiset näkemyksensä. Heillä pitäisi olla muodollinen katselmointikoulutus, jotta he ymmärtäisivät paremmin kulloisenkin roolinsa teknisessä katselmointiprosessissa. Kaikkien katselmointiin osallistuvien täytyy olla sitoutuneita hyvin toteutetusta teknisestä katselmoinnista saataviin hyötyihin. Täydellinen teknisen katselmoinnin kuvaus sekä useita katselmoinnin tarkastuslistoja löytyy lähteestä [Wiegers02]. Tekniset testausasiantuntijat osallistuvat yleensä teknisiin katselmointeihin ja tarkastuksiin, joihin he tuovat oman toiminnallisen (käyttäytymiseen pohjautuvan) näkökulmansa, joka on saattanut jäädä toteuttajilta huomaamatta. Teknisillä testausasiantuntijoilla on lisäksi tärkeä rooli katselmoinnin tarkistuslistojen ja havaintojen vakavuustietojen määrittelyssä, käyttämisessä ja ylläpidossa.

Käytettävästä katselmointityypistä riippumatta Tekniselle testausasiantuntijalle on annettava riittävästi aikaa valmistautumiseen. Tähän kuuluu vaihetuotteen katselmointiin kuluva aika, ristiviitattujen dokumenttien ja niiden yhdenmukaisuuden varmistamiseen tarvittava aika, sekä aika, jota tarvitaan mahdollisten vaihetuotteesta puuttuvien asioiden tunnistamiseksi. Ilman riittävää valmistautumisaikaa katselmoinnista voi tulla enemmänkin materiaalin muokkaustilaisuus kuin oikea katselmointi. Hyvään katselmointiin kuuluu sen ymmärtäminen, mitä on kirjoitettu, puuttuvien asioiden määrittäminen ja sen todentaminen, että kuvattu tuote on yhdenmukainen jo toteutettujen tai parhaillaan toteutettavana olevien tuotteiden kanssa. Esimerkiksi integrointitestaussuunnitelmaa katselmoidessaan Teknisen testausasiantuntijan on myös pohdittava integroitavina olevia kohteita. Ovatko ne valmiita integroitaviksi? Onko riippuvuuksia, jotka täytyy dokumentoida? Onko saatavilla aineistoa integraatiokohtien testaamiseksi? Katselmointi ei rajoitu vain katselmoitavana olevaan vaihetuotteeseen. Samalla on otettava huomioon myös kyseisen kohteen yhteydet järjestelmän muihin osiin.

Ei ole epätavallista, että katselmoinnin kohteen laatija tuntee olevansa kritisoitavana. Teknisen testausasiantuntijan tulisi lähestyä kaikkia katselmointikommentteja ajatuksella, että tavoitteena on tehdä yhteistyötä katselmoitavan materiaalin laatijan kanssa ja näin saada aikaan paras mahdollinen tuote. Tätä lähestymistapaa noudattamalla kommenteista tulee rakentavia ja ne kohdistuvat itse vaihetuotteeseen eivätkä sen laatijaan. Esimerkiksi, jos lause on tulkinnanvarainen, on parempi sanoa ”En ymmärrä, mitä minun pitäisi testata, jotta saisin todennettua, että tämä vaatimus on toteutettu oikein. Voitko auttaa minua ymmärtämään tämän paremmin?” sen sijaan, että sanoisi ”Tämä vaatimus on tulkinnanvarainen eikä kukaan pysty ymmärtämään sitä”.

Katselmoinneissa Teknisen testausasiantuntijan tehtävänä on varmistaa, että vaihetuotteessa kerrotut tiedot tukevat riittävästi testaustyötä. Jos jotain tietoa ei ole tai tiedot ovat epäselviä, kyseessä on todennäköisesti vika, joka materiaalin laatijan pitää korjata. Kommentit otetaan paremmin vastaan ja kokouksesta tulee tuloksekkaampi, jos kriittisyyden sijaan lähestytään asiaa positiivisesti.

5.2 Tarkistuslistojen käyttö katselmoinneissa

Tarkistuslistoja käytetään katselmoinneissa muistuttamaan osallistujia tiettyjen seikkojen todentamisesta katselmoinnin aikana. Tarkistuslistat voivat myös auttaa tekemään katselmoinnista vähemmän henkilösidonnaisen, esim. ”Tämä on sama tarkistuslista, jota käytämme kaikissa katselmoinnissa, eikä kohteena ole pelkästään sinun vaihetuotteesi”. Tarkistuslistat voivat olla yleisluonteisia ja kaikissa katselmoinneissa käytettäviä, tai ne voivat keskittyä määrättyihin laatuominaisuuksiin tai alueisiin. Esimerkiksi yleisluontoinen tarkistuslista voi keskittyä todentamaan termien ”pitää” ja ”pitäisi” oikean käytön, asiakirjan oikean muotoilun ja muita samantyyppisiä yhdenmukaisuuteen liittyviä asioita. Kohdennettu tarkistuslista voi keskittyä tietoturva- tai suorituskykyasioihin.

Kaikkein hyödyllisimpiä ovat tarkistuslistat, jotka on kehitetty ajan myötä yksittäisessä organisaatiossa, koska ne heijastavat

- tuotteen luonnetta

- paikallista kehitysympäristöä
  - henkilöstö
  - työkalut
  - prioriteetit
- aikaisempien onnistumisten ja vikojen historiaa
- erityisseikkoja (esim. suorituskyky, tietoturva).

Tarkistuslista pitäisi räätälöidä organisaation ja ehkä jopa tietyn projektin käyttöön. Tässä luvussa esitellyt tarkistuslistat on tarkoitettu vain esimerkeiksi.

Jotkut organisaatiot laajentavat tavallisen ohjelmiston tarkistuslistan käsitteen sisältämään myös ”vastamallit”, jotka liittyvät tyypillisiin erehdyksiin, heikkoihin tekniikoihin ja muihin tehottomiin käytäntöihin. Termi on peräisin suositusta ”suunnittelumallin” käsitteestä, jolla tarkoitetaan tyypillisiin ongelmiin löydettyjä uudelleenkäytettäviä ratkaisuja, joiden on osoitettu olevan käytännön tilanteissa tehokkaita [Gamma94]. Näin ollen vastamalli on tyypillisesti tehty erehdys, joka toteutetaan usein tilanteeseen sopivana oikopolkuna.

On tärkeää muistaa, että mikäli vaatimus ei ole testattava, eli sitä ei ole määritelty niin, että Tekninen testausasiantuntija voi päätellä, kuinka se pitää testata, kyseessä on vika. Esimerkiksi vaatimusta ”Ohjelmiston pitäisi olla nopea” ei voi testata. Kuinka Tekninen testausasiantuntija voi määritellä, onko ohjelmisto nopea? Jos vaatimuksessa sen sijaan sanottaisiin: ”Ohjelmiston vasteaika saa olla maksimissaan kolme sekuntia määrätyissä kuormitusolosuhteissa”, tämän vaatimuksen testattavuus on huomattavasti parempi, jos määrittelemme ”määrätyt olosuhteet” (esim. yhtäaikaisten käyttäjien määrä, käyttäjien suorittamien toimenpiteiden määrä). Se on myös korkeamman tason vaatimus, koska merkitykseltään tärkeässä järjestelmässä tästä yhdestä vaatimuksesta voi syntyä monia yksittäisiä testitapauksia. Jäljitettävyys vaatimuksesta testitapauksiin on myös kriittistä, koska vaatimuksen mahdollisesti muuttuessa kaikki testitapaukset pitää katselmoida ja niitä on tarpeen mukaan päivitettävä.

5.2.1 Arkkitehtuurikatselmoinnit

Ohjelmistoarkkitehtuuri muodostuu järjestelmän perusrakenteesta, joka pitää sisällään järjestelmän komponentit, niiden keskinäiset suhteet ja suhteet ympäristöön, sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja kehitystä. [ANSI/IEEE Std 1471-2000], [Bass03].

Arkkitehtuurikatselmoinneissa käytettäviin tarkistuslistoihin voi kuulua esimerkiksi seuraavien kohteiden todentaminen [Web-3]:

- Yhteysaltaat/-alueet – vähennetään tietokantayhteyksien muodostamiseen kuluvaan aikaan liittyvää varoaikaa muodostamalla jaettu yhteyksien kokoelma.
- Kuorman tasaus – jaetaan kuorma tasaisesti eri resurssien kesken.
- Hajautettu prosessointi
- Tallennus välimuistiin – käytetään aineiston paikallista kopiota saantiajan lyhentämiseksi.
- Olion (tai jonkin muun) alustaminen käyttöä varten vain tarvittaessa
- Tapahtumien yhdenaikaisuus
- Online Transactional Processing (OLTP) ja Online Analytical Processing (OLAP) –prosessien eriyttäminen
- Aineiston replikointi/monistaminen

Yksityiskohtaisempaa tietoa (sertifikaattikokeen ulkopuolelta) löytyy lähteestä [Web-4], jossa viitataan tutkimukseen, jossa on käyty läpi 117 tarkistuslistaa 24 eri lähteestä. Siinä keskustellaan tarkistuslistan kohtien eri luokista ja annetaan esimerkkejä hyvistä tarkistuslistan kohdista, samoin kuin sellaisista, joita on syytä välttää.

5.2.2 Koodikatselmoinnit

Koodikatselmointien tarkistuslistat ovat pakostakin hyvin yksin yksityiskohtaisia, ja ne ovat hyödyllisimpiä, kun ne ovat ohjelmointikieli-, projekti- ja yrityskohtaisia, kuten arkkitehtuurikatselmointien tarkistuslistatkin. Kooditason vastamallien sisällyttäminen mukaan listoihin auttaa esimerkiksi kokemattomampia ohjelmistokehittäjiä.

Koodikatselmointien tarkistuslistoihin kuuluu usein seuraavat kuusi pääkohtaa (perustuu [Web-5]:een):

1. Rakenne
   - Onko suunnitellut asiat toteutettu koodiin kokonaan ja oikein?
   - Onko koodi keskeisten ohjelmointistandardien mukaista?
   - Onko koodi rakenteeltaan selkeä sekä tyyliltään ja muotoilultaan yhdenmukainen?
   - Sisältääkö koodi kutsumattomia tai tarpeettomia proseduureja tai saavuttamatonta koodia?
   - Onko koodiin jäänyt jäljelle testauksessa käytettyjä tynkiä tai testirutiineja?
   - Voiko koodin osia korvata kutsuilla ulkoisiin uudelleenkäytettäviin komponentteihin tai kirjastofunktioihin?
   - Onko ohjelmassa toistuvia koodinosia, jotka voitaisiin tiivistää yksittäiseksi proseduuriksi?
   - Onko tilankäyttö tehokasta?
   - Käytetäänkö ennemminkin symboleja kuin ”maagista numeroa (“magic number”) tai merkkijonovakioita?
   - Ovatko jotkut moduulit liian monimutkaisia ja pitäisikö niiden rakenne järjestää uudelleen tai jakaa useampiin moduuleihin?

2. Dokumentaatio
   - Onko koodi selkeästi ja riittävästi dokumentoitu helposti ylläpidettävällä kommentointityylillä?
   - Ovatko kommentit yhdenmukaisia koodin kanssa?
   - Onko dokumentaatio soveltuvien standardien mukainen?

3. Muuttujat
   - Onko kaikki muuttujat määritelty oikein ja niin, että niillä on mielekkäät, yhdenmukaiset ja selkeät nimet?
   - Onko ohjelmassa turhia tai käyttämättömiä muuttujia?

4. Aritmeettiset laskutoimitukset
   - Pyritäänkö koodissa välttämään liukulukujen yhtäsuuruuksien vertailua?
   - Onko pyöristysvirheet järjestelmällisesti estetty koodissa?
   - Pyritäänkö koodissa välttämään kokoluokaltaan huomattavasti toisistaan poikkeavien lukujen yhteen- ja vähennyslaskut?
   - Onko jakajien arvona testattu myös nolla sekä tietokoneen laskentatarkkuuden rajalla oleva pieni desimaaliluku, jota saatetaan käsitellä nollana?

5. Silmukat ja haarat
   - Ovatko kaikki silmukat, haarat ja loogiset rakenteet kokonaisia, oikein ja oikealla tavalla sisennettyjä?
   - Onko kaikkein tyypillisimmät tapaukset testattu ensin IF-ELSEIF-ketjuissa?
   - Onko kaikki caset katettu IF-ELSEIF- tai CASE-lohkossa, mukaan luettuna ELSE- tai DEFAULT-oletuslausekkeet?
   - Onko jokaisella case-lauseella oletusarvo?
   - Ovatko silmukoiden päätösehdot ilmeisiä ja poikkeuksetta saavutettavissa?
   - Onko kaikki indeksit tai alaindeksit alustettu oikein juuri ennen silmukkaa?
   - Voidaanko mikään silmukan sisällä olevista lauseista sijoittaa silmukan ulkopuolelle?
   - Välttääkö silmukan koodi indeksimuuttujan muokkausta tai sen käyttöä silmukasta poistuttaessa?

6. Puolustava ohjelmointi

- Testataanko, että indeksit, osoittimet ja viittaukset taulukon tietoihin pysyvät taulukon, tietueen ja tiedoston rajojen osalta?
- Onko järjestelmään tuodun tiedon ja syötemuuttujien kelvollisuus ja täydellisyys testattu?
- Onko kaikki tulosmuuttujat määritelty?
- Käytetäänkö joka lauseessa oikeanlaista tietoelementtiä?
- Onko jokainen muistivaraus vapautettu?
- Onko ohjelmassa aikakatkaisu tai virhetilanteiden käsittely ulkoisten laitteiden käytön aiheuttamia ongelmatilanteita varten?
- Onko tiedostojen olemassaolo tarkistettu ennen kuin niitä yritetään käyttää?
- Jäävätkö tiedostot ja laitteet oikeaan tilaan ohjelman suorituksen päättymisen jälkeen?

Lisäesimerkkejä koodikatselmoinneissa eri testaustasoilla käytettävistä tarkistuslistoista löytyy lähteestä [Web-6].

6. Testaustyökalut ja automaatio - 195 min

Avainsanat

aineisto-ohjattu testaus, avainsanaohjattu testaus, hyperlinkkien testaustyökalu, nauhoitus/toisto-työkalu, staattisen analyysin työkalu, suorituskykytestaustyökalu, testauksenhallintatyökalu, testin suoritustyökalu, virheenjäljitin, vikojenkylvämistyökalu

Oppimistavoitteet: Testaustyökalut ja automaatio

6.1 Työkalujen integrointi ja niiden välinen tiedonkulku

TTA-6.1.1 (K2) Selittää tekniset näkökulmat, jotka on otettava huomioon, kun käytetään yhtä aikaa useita työkaluja

6.2 Testiautomaatioprojektin määrittäminen

TTA-6.2.1 (K2) Kertoa tehtävät, joita Tekninen testausasiantuntija suorittaa perustaessaan testiautomaatioprojektia.
TTA-6.2.2 (K2) Kertoa tieto-ohjatun ja avainsanaohjatun automaation erot.
TTA-6.2.3 (K2) Kuvailla tyypilliset tekniset seikat, jotka voivat saada aikaan sen, että automaatioprojekti ei saavuta suunniteltuja tuottotavoitteita.
TTA-6.2.4 (K3) Luoda määrättyyn liiketoimintaprosessiin pohjautuva avainsanataulukko.

6.3 Testauksen erityistyökalut

TTA-6.3.1 (K2) Kertoa vikojenkylvämis- ja vikojensyöttämistyökalujen tarkoitus
TTA-6.3.2 (K2) Kertoa suorituskykytestaus- ja monitorointityökalujen pääominaisuudet ja niiden käyttöönottoon liittyvät ongelmat.
TTA-6.3.3 (K2) Selittää web-pohjaisessa testauksessa käytettyjen työkalujen yleistarkoitus.
TTA-6.3.4 (K2) Selittää, kuinka työkalut tukevat mallipohjaista testausta.
TTA-6.3.5 (K2) Esitellä yksikkötestauksen ja koontiprosessin tukemiseen käytettyjen työkalujen tarkoitus.

6.1 Työkalujen integrointi ja niiden välinen tiedonkulku

Vaikka työkalun valinta ja integrointi onkin Testauspäällikön vastuulla, Teknistä testausasiantuntijaa voidaan pyytää katselmoimaan työkalun tai työkalujoukon integrointi, jotta varmistetaan eri testausalueilta, kuten staattisesta analyysistä, suoritusautomaatiosta ja kokoonpanonhallinnasta tulevien tietojen tarkka jäljitettävyys. Lisäksi, ohjelmointitaidoista riippuen, Tekninen testausasiantuntija saattaa olla myös mukana laatimassa koodia sellaisten työkalujen integroimiseksi, joiden integroiminen ei tapahdu ”suoraan paketista”.

Ihanteellinen työkalujoukko eliminoi työkalujen sisältämän tiedon kahdentumisen. Vaaditaan enemmän työtä ja virhealttius kasvaa, jos testien suoritusskriptit säilytetään sekä testauksenhallinnan tietokannassa että kokoonpanonhallintajärjestelmässä. On parempi, jos käytössä on testauksenhallintajärjestelmä, joka sisältää kokoonpanonhallintaosion, tai joka voidaan integroida jo organisaation käytössä olevaan kokoonpanonhallintatyökaluun. Hyvin integroidut vianhallinta- ja testauksenhallintatyökalut antavat testaajalle mahdollisuuden luoda vikaraportti testitapauksen suorituksen aikana ilman, että hänen täytyy poistua testauksenhallintatyökalusta. Hyvin integroiduista staattisen analyysin työkaluista pitäisi pystyä raportoimaan kaikki löydetyt havainnot ja varoitukset suoraan vianhallintajärjestelmään (vaikkakin tämän ominaisuuden pitäisi olla konfiguroitavissa, koska se saattaa tuottaa monia varoituksia).

Testityökalujoukon ostaminen samalta välinetoimittajalta ei automaattisesti tarkoita, että työkalut toimivat riittävän hyvin yhdessä. Kun pohditaan tapoja, joilla työkalut integroidaan yhteen, tietokeskeinen lähestymistapa on suositeltava. Tietojen vaihdon tulee tapahtua ilman manuaalisia toimintoja oikea-aikaisesti ja luvatulla tarkkuudella, mukaan lukien vioista toipuminen. Vaikka käyttökokemusten jatkuvasta saannista onkin hyötyä, tietojen keräämisen, varastoinnin, suojaamisen ja esittämisen pitäisi olla työkaluintegraation pääkohde.

Organisaation pitäisi arvioida tiedonkulun automatisoinnista syntyviä kustannuksia verrattuna riskeihin, jotka aiheutuvat tiedon menettämisestä tai siitä, että tiedot muuttuvat epäyhdenmukaisiksi tarvittavien manuaalisten toimenpiteiden vuoksi. Koska integrointi voi olla kallista tai vaikeaa, sen pitäisi olla kokonaistyökalustrategian keskeinen tarkastelukohde.

Jotkut integroidut kehitysympäristöt (integrated development environment, IDE) voivat yksinkertaistaa kyseisessä ympäristössä toimivien työkalujen integrointia. Tämä auttaa yhtenäistämään samassa kehitysrungossa toimivien työkalujen ulkoasun ja käyttötuntuman. Samanlainen käyttöliittymä ei kuitenkaan takaa helposti sujuvaa tiedonvaihtoa eri komponenttien välillä. Integraation loppuun saattamiseksi voidaan tarvita ohjelmointia.

6.2 Testiautomaatioprojektin määrittäminen

Ollakseen kustannustehokkaita testaustyökalut ja erityisesti automaatiotyökalut on organisoitava ja suunniteltava huolellisesti. Testausautomaatiostrategian toteuttaminen ilman kunnon arkkitehtuuria johtaa yleensä työkalujoukkoon, jonka ylläpitäminen on kallista, joka on käyttötarkoitukseensa riittämätön, ja joka ei pysty saavuttamaan sijoitukselle asetettuja tuottotavoitteita.

Testiautomaatioprojektia tulisi pitää ohjelmistokehitysprojektina. Tämä tarkoittaa arkkitehtuuridokumenttien, yksityiskohtaisten suunnitteludokumenttien, suunnitelmien ja koodin katselmointien, komponentti- sekä komponentti-integraatiotestauksen samoin kuin lopullisen järjestelmätestauksen tarvetta. Testaus voi viivästyä tai monimutkaistua tarpeettomasti, jos käytetään epävakaata tai virheellistä testiautomaatiokoodia. Tekninen testausasiantuntija suorittaa useita tehtäviä, jotka liittyvät testiautomaatioon. Näihin kuuluvat

- testien suoritusvastuusta päättäminen
- sopivan työkalun valinta organisaation, aikataulun, tiimin osaamisen ja ylläpitovaatimusten perusteella (huomaa, että tämä saattaa johtaa päätökseen luoda oma väline välineen hankinnan sijaan)

- automaatiotyökalun ja muiden, kuten testauksenhallinnan ja havaintojenhallinnan työkalujen välisten liittymävaatimusten määrittäminen
- sopivan automaatiolähestymistavan – avainsanaohjattu vai tieto-ohjattu - valinta (ks. kappale 6.2.1 edempänä)
- toteutus- ja koulutuskustannusten arvioiminen yhdessä Testauspäällikön kanssa
- automaatioprojektin aikataulutus ja ylläpitoon tarvittavan ajan varaaminen
- testaus- ja liiketoiminta-asiantuntijoiden kouluttaminen käyttämään automaatiota ja toimittamaan tietoa sitä varten
- automatisoitujen testien suoritustavasta päättäminen
- automatisoitujen testien ja manuaalisten testien tulosten yhdistämistavasta päättäminen.

Nämä tehtävät ja niistä syntyvät päätökset vaikuttavat automaatioratkaisun skaalautuvuuteen ja ylläpidettävyyteen. Eri vaihtoehtojen arviointiin, käytettävissä olevien työkalujen tutkimiseen ja organisaation tulevaisuudensuunnitelmien ymmärtämiseen on käytettävä riittävästi aikaa. Jotkut näistä tehtävistä vaativat enemmän pohdintaa kuin toiset, erityisesti päätöksentekoprosessin aikana. Näitä käsitellään tarkemmin seuraavissa kappaleissa.

6.2.1 Automaatiolähestymistavan valinta

Testausautomaatio ei rajoitu pelkästään käyttöliittymän kautta tehtävään testaukseen. On olemassa työkaluja, jotka auttavat automatisoimaan testejä, jotka tehdään API-tasolla, komentoriviltä tai testattavan järjestelmän muiden liittymäkohtien kautta. Yhtenä ensimmäisistä päätöksistä Tekninen testausasiantuntija joutuu valitsemaan testien automatisoinnissa käytettävän tehokkaimman liittymän.

Yksi käyttöliittymän kautta tehtävän testauksen ongelmista on se, että käyttöliittymällä on taipumus muuttua ohjelmiston kehittyessä. Tästä voi aiheutua merkittävästi ylläpitotyötä, riippuen tavasta, jolla testiautomaatiokoodi on suunniteltu. Esimerkiksi testiautomaatiotyökalun nauhoita/toista-ominaisuuden käyttäminen voi johtaa automatisoituihin testitapauksiin (kutsutaan usein testiskripteiksi), jotka eivät enää toimi halutulla tavalla, jos käyttöliittymä muuttuu. Tämä johtuu siitä, että nauhoitettava skripti tallentaa graafisten olioiden väliset tapahtumat, kun testaaja käyttää ohjelmistoa manuaalisesti. Jos käytetyt oliot muuttuvat, voi myös olla tarpeen päivittää nauhoitettu skripti vastaamaan tehtyjä muutoksia.

Nauhoita/toista-työkaluja voidaan käyttää sopivana aloituskohtana automaatioskriptien kehittämisessä. Testaaja nauhoittaa testausistunnon ja tallennettua skriptiä muokataan sitten ylläpidettävyyden parantamiseksi (esim. korvataan osa koodista uusiokäyttöisillä funktioilla).

Testattavasta ohjelmistosta riippuen kussakin testissä käytetty aineisto voi olla erilainen, vaikka suoritettavat testiaskeleet ovatkin lähes identtiset (esim. kun testataan syötekentän virheenkäsittelyä syöttämällä useita epäkelpoja arvoja ja tarkastamalla jokaisesta syntynyt virhe). Automatisoidun testiskriptin kehittäminen ja ylläpitäminen jokaiselle näistä testattavista arvoista on tehotonta. Tyypillinen ratkaisu tähän ongelmaan on aineiston siirtäminen skripteistä ulkoiseen tietovarastoon, kuten esimerkiksi laskentataulukkoon tai tietokantaan. Määrätyn aineiston käyttämiseksi kullakin skriptin suorituskerralla laaditaan funktioita, jotka mahdollistavat sen, että yksittäinen skripti voi käydä läpi kokonaisen testiaineistojoukon, joka sisältää syötearvot ja odotetut tulosarvot (esim. tekstikentässä näytetty arvo tai virheilmoitus). Tätä lähestymistapaa kutsutaan aineisto-ohjatuksi. Tätä lähestymistapaa käytettäessä laaditaan testiskripti, joka prosessoi sille välitetyn aineiston, sekä testikehys ja ympäristö, joita tarvitaan skriptin tai skriptijoukon suorituksen tueksi. Laskentataulukossa tai tietokannassa säilytettävän varsinaisen aineiston laativat Testausasiantuntijat, joille ohjelmiston liiketoiminnalliset tehtävät ovat tuttuja. Tämä työnjako mahdollistaa sen, että ne henkilöt, jotka ovat vastuussa testiskriptien kehittämisestä (eli Tekniset testausasiantuntijat), voivat keskittyä älykkäiden automaatioskriptien toteuttamiseen, kun taas varsinaisen testin omistajuus säilyy Testausasiantuntijalla. Useimmissa tapauksissa Testausasiantuntija on vastuussa testiskriptien suorittamisesta sen jälkeen, kun automaatio on toteutettu ja testattu.

Toinen lähestymistapa, jota kutsutaan avainsana- tai toimisana-ohjatuksi, menee askeleen pidemmälle erottamalla annetulla aineistolla suoritettavaksi tarkoitetut toiminnot testiskriptistä

[Buwalda01]. Jotta tällainen pidemmälle viety eriyttäminen voidaan tehdä, alan asiantuntijat (eli Testausasiantuntijat) laativat korkean tason meta-kielen, joka on enemmänkin kuvaavaa kuin suoraan suoritettavaa. Jokainen tämän kielen lause kuvaa toimialueen kokonaisen tai osittaisen liiketoimintaprosessin, joka saattaa vaatia testausta. Liiketoimintaprosessin avainsanoihin voivat kuulua esimerkiksi ”Login” (”kirjaudu sisään”), ”CreateUser” (”luo käyttäjä”) ja ”DeleteUser” (”poista käyttäjä”). Avainsana kuvaa sovellusalueella suoritettavan ylätason toiminnon. Voidaan määritellä myös alemman tason toimintoja, jotka kuvaavat yhteistoimintaa itse ohjelmiston käyttöliittymän kanssa, kuten ”ClickButton” (”napsauta painonappia”), ”SelectFromList” (”valitse listasta”) tai ”TraverseTree” (”käy läpi puurakenne”), ja näitä voidaan käyttää, kun testataan sellaisia käyttöliittymän ominaisuuksia, jotka eivät vastaa suoraan liiketoimintaprosessien avainsanoja.

Kun avainsanat ja käytettävä aineisto on määritelty, testien automatisoija (eli Tekninen testausasiantuntija) kääntää liiketoimintaprosessit avainsanoiksi ja alemman tason toimenpiteet testiautomaatioskriptiksi. Avainsanat ja toiminnot, samoin kuin käytettävä aineisto, voidaan tallentaa laskentataulukkoon tai syöttää käyttämällä erityisiä työkaluja, jotka tukevat avainsanaohjattua testiautomaatiota. Automaatiokehys toteuttaa avainsanat yhden tai useamman suoritettavan funktion tai skriptin joukkona. Työkalut lukevat testitapauksia, joissa avainsanoja on käytetty, ja kutsuvat sopivia testifunktioita tai skriptejä, jotka toteuttavat ne. Suoritettavat lauseet toteutetaan hyvin modulaarisesti, jotta niiden liittäminen määrättyihin avainsanoihin on helppoa. Tällaisten modulaaristen skriptien toteuttamiseen tarvitaan ohjelmointitaitoja.

Liiketoimintalogiikan tuntemuksen erottaminen varsinaisesta testiautomaatioskriptien toteuttamiseen tarvittavasta ohjelmoinnista tarjoaa tehokkaimman tavan testausresurssien käyttöön. Testien automatisoijan roolissa Tekninen testausasiantuntija voi tehokkaasti käyttää ohjelmointitaitojaan ilman, että hänestä täytyy tulla liiketoimintaan liittyvien monien eri alojen asiantuntija.

Koodin erottaminen muutettavasta aineistosta helpottaa eristämään automaation muutoksilta, mikä parantaa koodin yleistä ylläpidettävyyttä ja parantaa automaatioinvestoinnin tuottoa.

Kaikessa testiautomaation suunnittelussa on tärkeää varautua ohjelmistohäiriöihin ja niiden käsittelyyn. Automatisoijan pitää päättää, mitä ohjelmiston tulee tehdä häiriötilanteessa. Pitääkö häiriö kirjata ylös ja testien jatkua? Pitääkö testit keskeyttää? Voidaanko häiriöstä selvitä jollakin määrätyllä toimenpiteellä (kuten napsauttamalla painiketta valintaikkunassa) tai ehkä lisäämällä testiin viive? Käsittelemättömät häiriöt voivat aiheuttaa myöhempien testitulosten korruptoitumisen sekä ongelmia testissä, jota suoritettiin häiriön tapahtuessa.

On myös tärkeää miettiä järjestelmän tilaa testien alussa ja lopussa. Voi olla tarpeen varmistaa, että järjestelmä palaa ennalta määritettyyn tilaan, kun testin suoritus on päättynyt. Tämä mahdollistaa sen, että automatisoitujen testien joukko voidaan ajaa uudelleen ilman järjestelmän tiettyyn tilaan palauttamiseksi tarvittavia manuaalisia toimenpiteitä. Tätä varten testiautomaation on ehkä poistettava aineisto, jonka se loi, tai sen on muutettava tietokannan tietueitten tiloja. Automaatiokehyksen pitää varmistaa, että testien suoritus on päättynyt oikealla tavalla (eli kirjaudutaan ulos testien päätyttyä).

6.2.2 Liiketoimintaprosessien mallintaminen automaatiota varten

Jotta testausautomaatiossa voidaan käyttää avainsanaohjattua lähestymistapaa, testauksen kohteena olevat liiketoimintaprosessit täytyy mallintaa korkean tason avainsanoja sisältävällä kielellä. On tärkeää, että kieli on intuitiivinen sen käyttäjille, jotka ovat todennäköisimmin projektissa työskenteleviä Testausasiantuntijoita.

Avainsanoja käytetään yleensä edustamaan järjestelmän avulla suoritettavia korkean tason liiketoimintoja. Esimerkiksi ”Cancel_Order” (”peruuta tilaus”) voi edellyttää tilauksen olemassaolon tarkistamista, peruutusta pyytävän henkilön oikeuksien tarkistamista, peruutettavan tilauksen näyttämistä ja peruutuksen vahvistuksen pyytämistä. Testausasiantuntija käyttää avainsanojen ryhmiä (esim. ”Login”, ”Select_Order”, ”Cancel_Order” eli ”kirjaudu sisään”, ”valitse tilaus”, ”peruuta tilaus”) ja niihin liittyvää testiaineistoa testitapausten määrittämiseksi. Seuraavassa on yksinkertainen avainsanojen syötetaulukko, jota voitaisiin käyttää, kun testataan ohjelmiston kykyä lisätä, määrittää uudelleen ja poistaa käyttäjätilejä.

Avainsana	Käyttäjä	Salasana	Tulos
Add_User	User1	Pass1	Viesti ”käyttäjä lisätty”
Add_User	@Rec34	@Rec35	Viesti ”käyttäjä lisätty”
Reset_Password	User1	Welcome	Viesti ”salasana vaihdettu”
Delete_User	User1		Viesti ”väärä käyttäjätunnus/salasana”
Add_User	User3	Pass3	Viesti ”käyttäjä lisätty”
Delete_User	User2		Viesti ”käyttäjää ei löydy”

Taulukkoa käyttävä automaatioskripti etsii taulukosta syötearvoja automaatioskriptin käytettäväksi. Esimerkiksi kun se etenee riville, jossa avainsanana on ”Delete_User”, vain käyttäjätunnus on pakollinen. Uuden käyttäjän lisäämiseksi vaaditaan sekä käyttäjätunnus että salasana. Syötearvoina voidaan käyttää myös viittauksia muihin tietovarastoihin, kuten näkyy toisella ”Add_User”-rivillä, missä viittaus tietoon tarjoaa enemmän joustavuutta, kun käsitellään tietoa, joka saattaa muuttua testin suorituksen aikana. Tämä mahdollistaa aineisto-ohjattujen tekniikoiden yhdistämisen avainsanojen käyttöön.

Huomioon otettaviin seikkoihin kuuluvat mm. seuraavat:

- Mitä hienojakoisempia avainsanat ovat, sitä tarkempia skenaarioita voidaan kattaa, mutta korkean tason kielestä voi tulla monimutkaisempaa ylläpitää.
- Jos Testausasiantuntijat voivat määritellä alemman tason toimenpiteitä (”ClickButton”, ”SelectFromList” jne.), avainsanatestit voivat käsitellä eri tilanteita paljon paremmin. Koska nämä toimenpiteet on kuitenkin sidottu suoraan käyttöliittymään, tästä voi seurata, että testit vaativat enemmän ylläpitoa, kun muutoksia tapahtuu.
- Koosteavainsanojen käyttö voi yksinkertaistaa toteutusta mutta monimutkaistaa ylläpitoa. Esimerkiksi voi olla kuusi eri avainsanaa, jotka yhdessä käytettyinä luovat tietueen. Pitäisikö toimenpiteen yksinkertaistamiseksi luoda avainsana, joka kutsuu näitä kaikkia kuutta avainsanaa?
- Riippumatta siitä, kuinka paljon avainsanakieltä analysoidaan sitä laadittaessa, eteen tulee usein tilanteita, jolloin tarvitaan uusia ja erilaisia avainsanoja. Avainsanaan liittyy kaksi eri toimialuetta (avainsanan takana oleva liiketoimintalogiikka ja avainsanan suorittamiseen käytettävä automaatiotoiminnallisuus). Siksi on tarpeen laatia prosessi, jolla huolehditaan molemmista alueista.

Avainsanapohjainen testiautomaatio voi vähentää merkittävästi testiautomaation ylläpitokustannuksia, mutta se on kalliimpaa, vaikeampaa toteuttaa, ja sen oikeanlaiseen suunnitteluun menee enemmän aikaa, jos sillä halutaan saavuttaa panostukselle odotetut tuotot.

6.3 Testauksen erityistyökalut

Tämä kappale sisältää tietoa Teknisen testausasiantuntijan todennäköisesti käyttämistä työkaluista laajemmassa määrin kuin mitä on käsitelty Jatkotason Testausasiantuntijan [ISTQB_ALTA_SYL] ja Perustason [ISTQB_FL_SYL] sertifikaattisisällöissä.

6.3.1 Vikojen kylvämis- ja syöttämistyökalut

Vikojen kylvämistyökaluja käytetään pääasiassa kooditasolla luomaan järjestelmällisesti yksittäisiä tai määrätyn tyyppisiä vikoja koodiin. Nämä työkalut syöttävät tarkoituksella vikoja testattavaan kohteeseen, tarkoituksena testijoukkojen laadun arviointi (kuinka hyvin ne löytävät vikoja).

Vikojen syöttäminen keskittyy testaamaan testattavaan kohteeseen rakennettua vikojenkäsittelymekanismia kohdistamalla siihen poikkeustilanteita. Vikojen syöttämistyökalut syöttävät tarkoituksella ohjelmistoon vääriä syötearvoja varmistaakseen, että ohjelmisto pystyy selviämään vikatilanteesta.

Näitä työkaluja käyttää tyypillisesti Tekninen testausasiantuntija, mutta myös kehittäjä saattaa käyttää niitä testatessaan vasta toteutettua koodia.

6.3.2 Suorituskykytestaustyökalut

Suorituskykytestaustyökaluilla on kaksi päätehtävää:
- Kuorman generointi
- Määrätyn kuormituksen järjestelmälle aiheuttaman vasteen mittaus ja analysointi.

Kuorman generointi tapahtuu toteuttamalla ennalta määritelty käyttöprofiili (ks. kappale 4.5.4) skriptinä. Skripti voidaan ensin tallentaa yksittäistä käyttäjää varten (mahdollisesti käyttämällä nauhoita/toista-työkalua) ja sitten sitä käytetään suorituskykytestaustyökalulla määrätyn käyttöprofiiliin kanssa. Tämän käyttötavan täytyy ottaa huomioon erilaiset aineiston tapahtuma- tai tapahtumajoukkokohtaiset vaihtelut.

Suorituskykytyökalut luovat kuormaa simuloimalla isoa määrää yhtäaikaisia käyttäjiä (“virtuaalikäyttäjiä”) ja toimimalla heidän määritettyjen käyttöprofiiliensa mukaisesti tietyn tyyppisten syöteaineistomassojen luomiseksi. Yksittäisiin testien suoritusautomaatioskripteihin verrattuna monet suorituskykytestausskriptit toistavat käyttäjän järjestelmälle suorittamat toimenpiteet viestiprotokollatasolla eikä simuloimalla käyttäjän toimenpiteitä graafisen käyttöliittymän kautta. Tämä yleensä vähentää testauksen aikana tarvittavien erillisten ”istuntojen” määrää. Jotkut kuorman generointityökalut voivat myös ajaa sovellusta sen käyttöliittymän kautta tarkempien vasteaikojen mittaamiseksi, kun järjestelmä on kuormituksen alaisena.

Suorituskykytestaustyökalu kerää laajalta alueelta mittaritietoja, joita voidaan analysoida testin aikana tai sen jälkeen. Tyypillisiin kerättäviin mittaritietoihin ja tuotettuihin raportteihin kuuluvat:
- testin aikana simuloitujen käyttäjien määrä
- simuloitujen käyttäjien luomien tapahtumien määrä ja tyyppi sekä tapahtumien saapumisnopeus
- käyttäjien tekemien tiettyjen tapahtumapyyntöjen vasteajat
- raportit ja kaaviot kuorman ja vasteaikojen suhteesta
- raportit resurssien käytöstä (esim. käyttö tiettynä aikana maksimi- ja minimiarvoineen)

Merkittäviä seikkoja, joita on harkittava suorituskykytestaustyökalujen käyttöönoton yhteydessä, ovat mm.
- kuorman generoimiseksi tarvittava laitteisto ja verkon kaistanleveys
- työkalun yhteensopivuus testattavana olevan järjestelmän käyttämän viestiprotokollan kanssa
- järjestelmän joustavuus erilaisten käyttöprofiilien toteuttamiseksi helposti
- tarvittavat monitorointi, analysointi- ja raportointiominaisuudet.

Tyypillisesti suorituskykytyökalut ennemminkin hankitaan kuin kehitetään itse niiden kehittämisen vaatiman panostuksen vuoksi. Voi kuitenkin olla hyödyllistä kehittää määrätyntyyppinen suorituskykytyökalu itse, jos tekniset rajoitteet estävät saatavilla olevan työkalun käytön tai jos käytettävä kuormaprofiili ja muut tarpeet ovat suhteellisen yksinkertaisia.

6.3.3 Web-pohjaisen testauksen työkalut

Web-pohjaiseen testaukseen on saatavilla joukko avoimen lähdekoodin työkaluja sekä kaupallisia erityistyökaluja. Seuraavassa listassa esitetään joidenkin tyypillisten web-pohjaisten testaustyökalujen käyttötarkoitus:
- Hyperlinkkien testaustyökaluja käytetään tutkimaan ja tarkastamaan, että web-sivulla ei ole rikkinäisiä hyperlinkkejä tai että linkkejä ei puutu.
- HTML- ja XML-tarkistimet ovat työkaluja, jotka tarkastavat, että web-sivuston luomat sivut noudattavat HTML- ja XML-standardeja
- Kuormasimulaattorit testaavat, miten palvelin reagoi, kun suuri määrä käyttäjiä ottaa yhteyden
- Kevyemmät automaatiosuoritustyökalut, jotka toimivat eri selaimilla
- Työkalut, jotka etsivät palvelimelta orpoja (kytkemättömiä) tiedostoja
- HTML:ään erikoistuneet oikolukuohjelmat
- Cascading Style Sheet:iin (CSS, websivujen tyyliohje) liittyvät tarkistustyökalut
- Standardirikkomuksia etsivät työkalut, esim. USA:n saavutettavuusstandardin kohta 508 tai Euroopassa M/376

- Työkalut, jotka etsivät erilaisia tietoturvaongelmia.

Hyvä avoimen lähdekoodin web-testaustyökalujen lähde on W3C [Web-7]. Tämän websivuston taustalla oleva organisaatio asettaa Internet-standardit ja se tuottaa useita erilaisia työkaluja, joilla voidaan etsiä näiden standardien rikkomuksia.

Jotkut työkalut, joihin sisältyy web-hakukone, voivat myös tuottaa tietoa sivujen koosta ja niiden lataamiseen tarvittavasta ajasta ja siitä, onko sivusto käytettävissä vai ei (esim. HTTP error 404). Tämä tuottaa hyödyllistä tietoa kehittäjälle, webmasterille ja testaajalle.

Testausasiantuntija ja Tekninen testausasiantuntija käyttävät näitä työkaluja pääasiassa järjestelmätestauksen aikana.

6.3.4 Mallipohjaista testausta tukevat työkalut

Mallipohjainen testaus (Model-Based Testing, MBT) on tekniikka, jossa muodollista mallia, kuten tilakonetta, käytetään kuvaamaan ohjelmiston hallinnoiman järjestelmän aiottua suoritusaikakäyttäytymistä. Kaupalliset MBT-työkalut (ks. [Utting 07]) tarjoavat usein koneen, jolla käyttäjä voi ”suorittaa” mallin. Mielenkiintoiset suorituspolut voidaan tallentaa ja niitä voidaan käyttää testitapauksina. Muut suoritettavat mallit, kuten Petri-verkot ja tilakaaviot, tukevat myös MBT:tä. MBT-malleja (ja työkaluja) voidaan käyttää laajojen ainutkertaisten suorituspolkujoukkojen luomiseen.

MBT-työkalut voivat auttaa pienentämään mahdollisten polkujen hyvin suurta määrää, joka usein syntyy malliin.

Näitä työkaluja käyttämällä testaus voi tuoda esiin erilaisen näkökulman testattavaan ohjelmistoon. Tämä voi johtaa sellaisten vikojen löytymiseen, jotka olisivat jääneet toiminnallisessa testauksessa huomaamatta.

6.3.5 Komponenttitestauksen ja koontien työkalut

Vaikka komponenttitestauksen ja koontien automatisointityökalut ovatkin kehittäjien työkaluja, Tekniset testausasiantuntijat käyttävät ja ylläpitävät niitä usein, erityisesti Ketterän kehityksen yhteydessä.

Komponenttitestaus-/yksikkötestaustyökalut ovat usein ohjelmointikielikohtaisia. Esimerkiksi jos ohjelmointikielenä käytetään Javaa, voidaan JUnitia käyttää yksikkötestien automatisointiin. Monilla muilla kielillä on omat erityistyökalunsa; niitä kutsutaan yleisesti nimellä XUnit-kehykset. Tällaiset kehykset luovat jokaiselle luodulle luokalle testattavia kohteita, ja helpottavat näin tehtäviä, joita ohjelmoijan on tehtävä hänen automatisoidessaan yksikkötestausta.

Debuggaus-/virheenjäljitystyökalut helpottavat manuaalista yksikkötestausta hyvin alhaisella tasolla, sillä niiden avulla Tekninen testausasiantuntija voi suorituksen aikana muuttaa muuttujan arvoja ja käydä testatessaan läpi ohjelmakoodia rivi riviltä. Kehittäjät käyttävät myös debuggaustyökaluja helpottamaan koodissa olevien ongelmien eristämistä ja tunnistamista, kun testaustiimi on raportoinut häiriön.

Koontien automatisointityökalut mahdollistavat uuden koonnin luonnin automaattisesti aina, kun komponenttia on muutettu. Kun koonti on valmis, muut työkalut suorittavat automaattisesti yksikkötestit. Koontiprosessiin liittyvää tämän tason automatisointia pidetään yleensä jatkuvan integroinnin ympäristönä.

Oikein asennettuna tällaisella työkalujoukolla voi olla hyvin positiivinen vaikutus testaukseen julkaistavien koontien laatuun. Mikäli toteuttajan tekemien muutosten seurauksena koontiin syntyy regressiovikoja, seurauksena on yleensä joidenkin automatisoitujen testien epäonnistuminen, mikä johtaa välittömästi häiriöiden syiden selvittämiseen ennen kuin koonti julkaistaan testiympäristöön.

7. Viitteet

7.1 Standardit

Seuraavat standardit mainitaan alla luetelluissa luvuissa.

- ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems
  Luku 5

- IEC-61508
  Luku 2

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)
  Luku 4

- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality
  Luku 4

- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992.
  Luku 2

7.2 ISTQB dokumentit

- [ISTQB_AL_OVIEW-FI] ISTQB Jatkotason yleiskatsaus, Versio 2012
- [ISTQB_ALTA_SYL-FI] ISTQB Jatkotason sertifikaattisisältö, Testausasiantuntija, Versio 2012
- [ISTQB_FL_SYL-FI] ISTQB Perustason sertifikaattisisältö, Versio 2011
- [ISTQB_GLOSSARY-FI] ISTQB:n testaussanasto, Versio 2.2, 2012

7.3 Kirjat

[Bass03] Len Bass, Paul Clements, Rick Kazman “Software Architecture in Practice (2nd edition)”, Addison-Wesley 2003] ISBN 0-321-15495-9
[Bath08] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook”, Rocky Nook, 2008, ISBN 978-1-933952-24-6
[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3
[Beizer95] Boris Beizer, “Black-box Testing”, John Wiley & Sons, 1995, ISBN 0-471-12094-4
[Buwalda01]: Hans Buwalda, “Integrated Test Design and Automation” Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
[Copeland03]: Lee Copeland, “A Practitioner's Guide to Software Test Design”, Artech House, 2003, ISBN 1-58053-791-X
[Gamma94] Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994, ISBN 0-201-63361-2
[Jorgensen07]: Paul C. Jorgensen, “Software Testing, a Craftsman’s Approach third edition”, CRC press, 2007, ISBN-13:978-0-8493-7475-3
[Kaner02]: Cem Kaner, James Bach, Bret Pettichord; “Lessons Learned in Software Testing”; Wiley, 2002, ISBN: 0-471-08112-4
[Koomen06]: Tim Koomen, Leo van der Aalst, Bart Broekman, Michael Vroon, "TMap Next for result-driven testing"; UTN Publishers, 2006, ISBN: 90-72194-79-9

[McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976. PP 308-320
[NIST96] Arthur H. Watson and Thomas J. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", NIST Special Publication 500-235, Prepared under NIST Contract 43NANB517266, September 1996.
[Splaine01]: Steven Splaine, Stefan P. Jaskiel, “The Web-Testing Handbook”, STQE Publishing, 2001, ISBN 0-970-43630-0
[Utting 07] Mark Utting, Bruno Legeard, “Practical Model-Based Testing: A Tools Approach”, Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1
[Whittaker04]: James Whittaker and Herbert Thompson, “How to Break Software Security”, Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0
[Wiegers02] Karl Wiegers, "Peer Reviews in Software: A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

7.4 Muut lähdeviitteet

Seuraavat viittaukset kohdistuvat Internetistä saatavilla olevaan tietoon. Vaikka nämä lähdeviitteet tarkistettiin tämän Jatkotason sertifikaattisisällön julkaisuhetkellä, ISTQB ei kuitenkaan vastaa siitä, jos lähdemateriaali ei ole enää myöhemmin saatavilla.

[Web-1] www.testingstandards.co.uk
[Web-2] http://www.nist.gov NIST National Institute of Standards and Technology,
[Web-3] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx
[Web-4] http://portal.acm.org/citation.cfm?id=308798
[Web-5] http://www.processimpact.com/pr_goodies.shtml
[Web-6]http://www.ifsq.org
[Web-7] http://www.W3C.org

Luku 4: [Web-1], [Web-2]
Luku 5: [Web-3], [Web-4], [Web-5], [Web-6]
Luku 6: [Web-7]