Sertifioitu Testaaja

Perustason sertifikaattisisältö

Versio 2018

Perustuu englanninkieliseen versioon 4.6.2018

International Software Testing Qualifications Board

Muutoshistoria

Versio	Pvm	Huomautukset
ISTQB 2018	4.6.2018	ISTQB:n hyväksymä englanninkielinen versio
Versio 0.1	23.7.2018	Ensimmäinen katselmoitava suomenkielinen versio
Versio 1.0	10.10.2018	FiSTB:n hyväksymä versio

Sisällysluettelo

Tekijänoikeushuomautus 2
Muutoshistoria 3
Kiitokset 7
0 Esittely 9
0.1 *Tämän sertifikaattisisällön tarkoitus* 9
0.2 *Sertifioitu testaaja -perustaso ohjelmistotestauksessa* 9
0.3 *Tentittävät oppimistavoitteet ja kognitiiviset osaamistasot* 9
0.4 *Perustason sertifiointikoe* 10
0.5 *Akkreditointi* 10
0.6 *Tarkkuustaso* 10
0.7 *Sertifikaattisisällön rakenne* 10
1 Testauksen perusteet 11
1.1 *Mitä on testaus?* 12
1.1.1 Testauksen tyypilliset tavoitteet 12
1.1.2 Testaus ja virheenjäljitys 13
1.2 *Miksi testaus on välttämätöntä?* 13
1.2.1 Testauksen merkitys onnistumisen kannalta 13
1.2.2 Laadunvarmistus ja testaus 13
1.2.3 Viat, virheet ja häiriöt 14
1.2.4 Viat, alkuperäissyyt ja vaikutukset 14
1.3 *Seitsemän testauksen perusperiaatetta* 15
1.4 *Testausprosessi* 16
1.4.1 Testausprosessi eri tilanteissa 16
1.4.2 Testaustoimenpiteet ja tehtävät 16
1.4.3 Testauksen tuotokset 20
1.4.4 Jäljitettävyys testauksen pohjamateriaalin ja tuotosten välillä 22
1.5 *Testauksen psykologia* 22
1.5.1 Ihmispsykologia ja testaus 22
1.5.2 Testaajan ja kehittäjän ajatusmallit 23
2 Testaus ohjelmistokehityksen elinkaaressa 24
2.1 *Ohjelmistokehityksen elinkaarimallit* 25
2.1.1 Ohjelmistokehitys ja ohjelmistotestaus 25
2.1.2 Ohjelmistokehityksen elinkaarimallit eri tilanteissa 26
2.2 *Testaustasot* 27
2.2.1 Yksikkötestaus 27
2.2.2 Integraatiotestaus 28
2.2.3 Järjestelmätestaus 31
2.2.4 Hyväksymistestaus 32
2.3 *Testaustyypit* 35
2.3.1 Toiminnallinen testaus 35
2.3.2 Ei-toiminnallinen testaus 35
2.3.3 Lasilaatikkotestaus 36
2.3.4 Muutokseen pohjautuva testaus 36
2.3.5 Testaustyypit ja testaustasot 37
2.4 *Ylläpitotestaus* 38
2.4.1 Ylläpitotestauksen aiheuttajat 38
2.4.2 Ylläpidon vaikutusanalyysi 39
3 Staattinen testaus 40
3.1 *Staattisen testauksen perusteet* 41
3.1.1 Tuotokset, joita voidaan tutkia staattisella testauksella 41
3.1.2 Staattisen testauksen hyödyt 41
3.1.3 Staattisen ja dynaamisen testauksen erot 42
3.2 *Katselmointiprosessi* 42
3.2.1 Tuotosten katselmointiprosessi 43
3.2.2 Muodollisen katselmoinnin roolit ja vastuut 44
3.2.3 Katselmointityypit 44

3.2.4 Katselmointitekniikoiden soveltaminen 46
3.2.5 Katselmointien menestystekijät 47
4 Testaustekniikat 49
4.1 *Testaustekniikoiden luokittelu* 50
4.1.1 Testaustekniikoiden valinta 50
4.1.2 Testaustekniikoiden luokittelu ja ominaisuudet 50
4.2 *Mustalaatikkotekniikat* 51
4.2.1 Ekvivalenssiositus 51
4.2.2 Raja-arvoanalyysi 52
4.2.3 Päätöstaulutestaus 52
4.2.4 Tilasiirtymätestaus 53
4.2.5 Käyttötapaustestaus 53
4.3 *Lasilaatikkotekniikat* 54
4.3.1 Lausetestaus ja -kattavuus 54
4.3.2 Päätöstestaus ja -kattavuus 54
4.3.3 Lause- ja päätöstestauksen merkitys 54
4.4 *Kokemuspohjaiset tekniikat* 54
4.4.1 Virheenarvaus 55
4.4.2 Tutkiva testaus 55
4.4.3 Tarkistuslistoihin pohjautuva testaus 55
5 Testauksen hallinta 56
5.1 *Testausorganisaatio* 57
5.1.1 Riippumaton testaus 57
5.1.2 Testauspäällikön ja testaajan tehtävät 58
5.2 *Testaussuunnittelu ja työmääräarviointi* 59
5.2.1 Testaussuunnitelman tarkoitus ja sisältö 59
5.2.2 Testausstrategiat ja lähestymistavat 60
5.2.3 Aloitus- ja lopetuskriteerit (Työstövalmiin ja Valmiin määritelmät) 61
5.2.4 Testien suoritusaikataulu 61
5.2.5 Testauksen työmäärään vaikuttavat tekijät 62
5.2.6 Työmääräarvioinnin tekniikat 62
5.3 *Testauksen seuranta ja hallinta* 63
5.3.1 Testauksessa käytetyt mittarit 63
5.3.2 Testiraporttien tarkoitus, sisältö ja kohderyhmä 64
5.4 *Kokoonpanonhallinta* 64
5.5 *Riskit ja testaus* 65
5.5.1 Riskin määritelmä 65
5.5.2 Tuote- ja projektiriskit 65
5.5.3 Riskipohjainen testaus ja tuotteen laatu 66
5.6 *Vikojenhallinta* 67
6 Testausta avustavat työkalut 69
6.1 *Testaustyökaluihin liittyviä asioita* 70
6.1.1 Testaustyökalujen luokittelu 70
6.1.2 Testiautomaation hyödyt ja riskit 72
6.1.3 Testien suoritustyökaluihin ja testauksenhallintatyökaluihin liittyviä erityishuomioita 72
6.2 *Työkalujen tehokas käyttö* 73
6.2.1 Työkalun valinnan pääperiaatteet 73
6.2.2 Työkalun tuominen organisaatioon pilottiprojektien avulla 74
6.2.3 Työkalujen menestystekijät 74
7 Viitteet 75
Standardit 75
ISTQB asiakirjat 75
Kirjat ja artikkelit 76
Muut lähteet (ei suoraan viitattu tässä Sertifikaattisisällössä) 76
8 Liite A – Sertifikaattisisällön tausta 77
Tämän dokumentin historia 77
Perussertifikaatin tavoitteet 77
Kansainvälisen sertifioinnin tavoitteet 77
Aloitusvaatimukset tälle sertifikaatille 77

Ohjelmistotestauksen perussertifikaatin tausta ja historia 78
9 Liite B - Oppimistavoitteet/osaamistaso 79
Taso 1: Muistaa (K1) 79
Taso 2: Ymmärtää (K2) 79
Taso 3: Soveltaa (K3) 79
10 Liite C - Julkaisuseloste 80

Kiitokset

Tämä dokumentti [sertifikaattisisällön englanninkielinen versio] hyväksyttiin virallisesti julkaistavaksi ISTQB:n Yleiskokouksessa 4.6.2018.

Sen tuotti International Software Testing Qualifications Boardin jäsenistä muodostettu tiimi: Klaus Olsen (puheenjohtaja), Tauhida Parveen (varapuheenjohtaja), Rex Black (projektipäällikkö), Debra Friedenberg, Matthias Hamburg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh ja Eshraka Zakaria,

Tiimi kiittää Rex Blackia ja Dorothy Grahamia heidän tekemästään teknisestä editoinnista sekä katselmointitiimiä, ristiinkatselmointitiimiä sekä kansallisia hallituksia näiden tekemistä ehdotuksista ja panostuksesta.

Seuraavat henkilöt osallistuivat tämän sertifikaattisisällön katselmointiin, kommentointiin ja siihen liittyviin äänestyksiin: Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von Düring, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar ja Karolina Zmitrowicz.

International Software Testing Qualifications Board Foundation-tason työryhmä (Versio 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Dani Almog, Debra Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Müller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria ja Stevan Zivanovic. Ydintiimi kiittää katselmointitiimiä ja kaikkia jäsenyhdistyksiä heidän ehdotuksistaan.

International Software Testing Qualifications Board Foundation-tason työryhmä (Versio 2011): Thomas Müller (puheenjohtaja), Debra Friedenberg. Ydintiimi kiittää katselmointitiimiä (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal), ja kaikkia kansallisia hallituksia heidän ehdotuksistaan.

International Software Testing Qualifications Board Foundation-tason työryhmä (Versio 2010): Thomas Müller (puheenjohtaja), Rahul Verma, Martin Klonk ja Armin Beer. Ydintiimi kiittää katselmointitiimiä (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veendendaal) ja kaikkia kansallisia hallituksia muutosehdotuksista nykyiseen sertifikaattisisällön versioon.

International Software Testing Qualifications Board Foundation-tason työryhmä (Versio 2007): Thomas Müller (chair), Dorothy Graham, Debra Friedenberg ja Erik van Veenendaal. Ydintiimi kiittää katselmointitiimiä (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson ja Wonil Kwon) ja kaikkia jäsenyhdistyksiä heidän ehdotuksistaan.

International Software Testing Qualifications Board Foundation-tason työryhmä (Versio 2005): Thomas Müller (puheenjohtaja), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson ja Erik van Veendendal). Ydintiimi kiittää katselmointitiimiä ja kaikkia jäsenyhdistyksiä heidän ehdotuksistaan.

Vuoden 2018 suomenkielisen version kääntämiseen ja katselmointiin ovat osallistuneet Minna Aalto, Kari Kakkonen, Juha Pomppu ja Tuula Pääkkönen.

Vuoden 2010/2011 päivitysversion kääntämiseen ja katselmointiin ovat osallistuneet Minna Aalto, Harri Honkaharju, Kari Kakkonen, Juha Pomppu, Tuula Pääkkönen ja Matti Vuori.

Vuoden 2009 suomenkielisen version kääntämiseen ja katselmointiin ovat osallistuneet Minna Aalto, Hans Dikert, Harri Honkaharju, Antti Ilmo, Kari Kakkonen Timo Malk, Tuula Pääkkönen, Erkki Pöyhönen, Rami Tuominen ja Matti Vuori.

0 Esittely

0.1 *Tämän sertifikaattisisällön tarkoitus*

Tämä sertifikaattisisältö muodostaa pohjan International Software Testing Qualification -vaatimusten Perustasolle. ISTQB on toimittanut sertifikaattisisällön seuraavia tarkoituksia varten:

1. Jäsenyhdistyksille käännettäväksi paikalliselle kielelle sekä koulutustarjoajien akkreditointia varten. Jäsenyhdistykset voivat muokata sertifikaattisisältöä tietyn kielen tarpeiden mukaisesti sekä muokata viitteitä vastaamaan paikallisia julkaisuja.
2. Sertifiointeja tekeville organisaatioille: Tämän sertifikaattisisällön oppimistavoitteita vastaavien tutkintokysymysten tuottamiseksi paikallisella kielellä.
3. Koulutustarjoajille: koulutusmateriaalin tuottamiseen sekä soveltuvien opetustapojen valitsemiseksi.
4. Sertifiointikokelaille: sertifiointitutkintoon valmistautumista varten (joko osana valmennuskurssia tai itsenäisesti).
5. Kansainväliselle ohjelmisto- ja järjestelmäkehittäjien 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 nämä hakevat ja saavat etukäteen kirjallisen suostumuksen ISTQB:ltä.

0.2 *Sertifioitu testaaja -perustaso ohjelmistotestauksessa*

Perustason sertifiointi on tarkoitettu kaikille, jotka osallistuvat ohjelmistotestaukseen. Tämä tarkoittaa esimerkiksi testaajina, testausanalyytikkoina, testaussuunnittelijoina, testauskonsultteina, testauspäälliköinä, hyväksymistestaajina ja sovelluskehittäjinä toimivia henkilöitä. Tämä perustason pätevyys soveltuu myös jokaiselle, joka tarvitsee perusymmärtämystä ohjelmistotestauksesta, kuten esimerkiksi tuoteomistajat, projektipäälliköt, laatupäälliköt, ohjelmistokehityspäälliköt, liiketoiminta-analyytikot, atk-päälliköt ja johdon konsultit. Perustason sertifikaatin suorittaneet voivat jatkaa ohjelmistotestauksen ylemmän tason pätevyyksiin.

ISTQB Perustason sertifioinnin yleiskatsaus 2018 on erillinen dokumentti, joka sisältää seuraavat tiedot:

- sertifikaattisisällön liiketoiminnalliset hyödyt
- liiketoiminta-tavoitteiden ja oppimistavoitteiden välinen jäljitettävyysmatriisi
- tämän sertifikaattisisällön yhteenveto.

0.3 *Tentittävät oppimistavoitteet ja kognitiiviset osaamistasot*

Oppimistavoitteet tukevat liiketoiminnan tavoitteita ja niitä käytetään testauksen perustason sertifiointikokeita laadittaessa.

Yleisesti ottaen koko tämän dokumentin sisältö on tentittävissä K1-tasolla lukuun ottamatta Esittelyä ja liitteitä. Se tarkoittaa, että kokelasta voidaan pyytää tunnistamaan, muistamaan tai palauttamaan mieleensä avainsanoja tai käsitteitä, jotka on mainittu missä tahansa sertifikaattisisällön kuudesta luvusta. Eri oppimistavoitteiden kognitiiviset tasot on kuvattu jokaisen luvun alussa ja ne on luokiteltu seuraavasti:

- K1: muistaa
- K2: ymmärtää
- K3: soveltaa.

Tarkempia tietoja ja esimerkkejä oppimistavoitteista löytyy liitteestä B.

Kaikkien Avainsanat-otsikon alla välittömästi lukujen otsikoiden alapuolella lueteltujen termien määritelmät on tarkoitettu muistettavaksi (K1), vaikka tätä ei nimenomaisesti ole mainittu oppimistavoitteissa.

0.4 *Perustason sertifiointikoe*

Perustason sertifiointikoe pohjautuu tähän sertifikaattisisältöön. Koekysymyksiin vastaaminen voi edellyttää, että käytetään aineistoa, joka perustuu tämän sertifikaattisisällön useampaan kappaleeseen. Kaikki Sertifikaattisisällön osat ovat tentittäviä lukuun ottamatta Esittelyä ja liitteitä. Standardeja, kirjoja ja muita ISTQB sertifikaattisisältöjä on käytetty viitteinä, mutta niiden sisältö ei ole tentittävää kuin siltä osin, mitä niistä on kuvattu tässä sertifikaattisisällössä.

Tyypiltään koe on monivalintakoe. Kokeessa on 40 kysymystä. Kokeen läpäisemiseksi täytyy vastata oikein vähintään 65 %:iin kysymyksistä (eli 26 kysymykseen).

Kokeen voi suorittaa akkreditoidun valmennuskurssin osana tai erillisessä tenttitilaisuudessa (esim. jossain testikeskuksessa tai kaikille avoimessa koetilaisuudessa). Valmennuskurssille osallistuminen ei ole esivaatimus kokeeseen osallistumiselle.

0.5 *Akkreditointi*

ISTQB:n tunnustama jäsenyhdistys voi akkreditoida koulutuksen tarjoajat, joiden kurssimateriaali on tämän sertifikaattisisällön mukainen. Akkreditoinnin ohjeistus hankitaan akkreditoinnin suorittavalta lautakunnalta tai elimeltä. Akkreditoidun kurssin katsotaan olevan tämän sertifikaattisisällön mukainen, ja ISTQB-koe saadaan järjestää osana kurssia.

0.6 *Tarkkuustaso*

Tämän sertifikaattisisällön yksityiskohtaisuus mahdollistaa kansainvälisellä tasolla yhdenmukaiset kurssit ja kokeet. Tämän päämäärän saavuttamiseksi sertifikaattisisältö koostuu seuraavista osista:

- yleiset ohjaavat tavoitteet, jotka kuvaavat, mihin perustasolla tähdätään
- luettelo termeistä, jotka opiskelijan on kyettävä palauttamaan mieleensä
- tavoiteltavaa kognitiivista oppimistulosta kuvaavat oppimistavoitteet kullekin osaamisalueelle
- avainkäsitteiden kuvaukset viitteineen hyväksyttyihin kirjallisuuslähteisiin tai standardeihin.

Sertifikaattisisältö ei ole kuvaus ohjelmistotestauksen koko osaamissalueesta; se kertoo perustason koulutuskursseilta vaadittavan tarkkuustason. Se keskittyy testauksen käsitteisiin ja tekniikoihin, jotka ovat sovellettavissa kaikkiin ohjelmistoprojekteihin, ketterät projektit mukaan luettuna. Tämä sertifikaattisisältö ei sisällä erityisiä oppimistavoitteita liittyen mihinkään tiettyihin ohjelmistokehityksen elinkaarimalleihin tai menetelmiin, mutta siinä tuodaan esiin, kuinka näitä käsitteitä sovelletaan ketterissä projekteissa, muissa iteratiivisissa ja inkrementaalisissa elinkaarimalleissa sekä peräkkäiselinkaarimalleissa.

0.7 *Sertifikaattisisällön rakenne*

Tässä sertifikaattisisällössä on kuusi sisällöltään tentittävää lukua. Luvun pääotsikko kuvaa kyseiseen lukuun käytettävän ajan; ajankäyttöä ei ole kuvattu tätä alemmilla tasoilla. Akkreditoitujen valmennuskurssien osalta sertifikaattisisältö edellyttää, että opetukseen käytetään vähintään 16.75 tuntia, joka jakautuu kuuden luvun kesken seuraavasti:

- Luku 1: 175 minuuttia Testauksen perusteet
- Luku 2: 100 minuuttia Testaus ohjelmistokehityksen elinkaaressa
- Luku 3: 135 minuuttia Staattinen testaus
- Luku 4: 330 minuuttia Testaustekniikat
- Luku 5: 225 minuuttia Testauksen hallinta
- Luku 6: 40 minuuttia Testauksen työkalutuki.

1 Testauksen perusteet — 175 minuuttia

Avainsanat

alkuperäissyy, häiriö, jäljitettävyys, kattavuus, kelpuutus, laadunvarmistus, laatu, testattava kohde, testattava tilanne, testauksen hallinta, testauksen pohjamateriaali, testauksen seuranta, testauksen suunnittelu, testauksen tavoite, testaus, testiaineisto, testianalyysi, testien suoritus, testien suoritusaikataulu, testien valmistelu, testijoukko, testimateriaali, testin loppuunsuoritus, testioraakkeli, testiproseduuri, testisuunnittelu, testitapaus, todentaminen, vika, virhe, virheenjäljitys,

Oppimistavoitteet: Testauksen perusteet

1.1 Mitä on testaus?

FL-1.1.1 (K1) Tunnistaa testauksen tyypilliset tavoitteet

FL-1.1.2 (K2) Pystyä selittämään testauksen ja debuggauksen (virheiden jäljityksen) eron

1.2 Miksi testaus on välttämätöntä?

FL-1.2.1 (K2) Kertoa esimerkein, miksi testaus on tarpeellista

FL-1.2.2 (K2) Pystyä kuvaamaan, miksi testaus on osa laadunvarmistusta, ja kertoa esimerkkejä siitä, kuinka testaus myötävaikuttaa laadun parantamiseen

FL-1.2.3 (K2) Erottaa toisistaan virhe, vika ja häiriö

FL-1.2.4 (K2) Erottaa vian alkuperäissyyn vian vaikutuksista

1.3 Seitsemän testauksen perusperiaatetta

FL-1.3.1 (K2) Osata selittää testauksen seitsemän pääperiaatetta

1.4 Testausprosessi

FL-1.4.1 (K2) Osata selittää, miten tilanne vaikuttaa testausprosessiin

FL-1.4.2 (K2) Kuvata testaustehtävät ja niitä vastaavat tehtävät testausprosessissa

FL-1.4.3 (K2) Erottaa tuotokset, jotka tukevat testausprosessia

FL-1.4.4 (K2) Osata selittää testauksen pohjamateriaalin ja testauksen tuotosten välisen jäljitettävyyden ylläpitämisen merkitys

1.5 Testauksen psykologia

FL-1.5.1 (K1) Tiedostaa testauksen onnistumiseen vaikuttavat psykologiset tekijät

FL-1.5.2 (K2) Osata selittää testaustehtävissä tarvittavan ajatusmallin ja toteutustehtävissä tarvittavan ajatusmallin välinen ero

1.1 *Mitä on testaus?*

Tietojärjestelmät ovat olennainen osa jokapäiväistä elämää liiketoimintalähtöisistä sovelluksista (esim. pankkitoiminta) kulutustavaroihin (esim. autot). Useimmilla ihmisillä on kokemusta ohjelmistoista, jotka eivät ole toimineet odotetusti. Väärin toimiva ohjelmisto voi johtaa moniin ongelmiin, kuten rahan, ajan tai liiketoiminnan maineen menetykseen ja jopa loukkaantumiseen tai kuolemaan. Ohjelmistotestaus on keino arvioida ohjelmiston laatua ja pienentää tuotantokäytössä tapahtuvien häiriöiden riskiä.

Yleinen väärinkäsitys testauksesta on, että se käsittää vain testien suorittamisen ohjelmaa käyttämällä ja tulosten tarkistamisen. Kuten on kuvattu alaluvussa 1.4, ohjelmistotestaus on prosessi, johon kuuluu monia eri tehtäviä; testien suoritus (tulosten tarkistaminen mukaan luettuna) on vain yksi näistä tehtävistä. Testausprosessiin kuuluu myös sellaisia tehtäviä kuin testauksen suunnittelu, testien analysointi, suunnittelu ja toteutus, testauksen edistymisestä ja tuloksista raportointi sekä testauksen kohteen laadun arviointi.

Osaan testauksesta kuuluu testattavana olevan komponentin tai järjestelmän käyttäminen; tällaista testausta kutsutaan dynaamiseksi testaukseksi. Osaan testauksesta taas ei kuulu testattavana olevan komponentin tai järjestelmän käyttäminen; tällaista testausta kutsutaan staattiseksi testaukseksi. Näin ollen testaukseen kuuluu myös erilaisten työn aikana syntyneiden tuotosten, kuten vaatimusten, käyttäjätarinoiden ja lähdekoodin, katselmointi.

Toinen yleinen testaukseen liittyvä väärinkäsitys on, että testaus keskittyy yksinomaan vaatimusten, käyttäjätarinoiden tai muiden määrittelyiden todentamiseen. Vaikka testaukseen kuuluukin sen tarkistaminen, täyttääkö järjestelmä sille määritellyt vaatimukset, testaukseen kuuluu myös kelpuutus, joka tarkoittaa sen tarkistamista, täyttääkö järjestelmä käyttäjien ja muiden sidosryhmien tarpeet sen käyttöympäristö(i)ssä.

Testaustehtävät organisoidaan ja suoritetaan eri tavoin erilaisissa elinkaarimalleissa (ks. alaluku 2.1).

1.1.1 Testauksen tyypilliset tavoitteet

Missä tahansa projektissa testauksen tavoitteisiin saattaa kuulua seuraavia asioita:

- arvioida eri tuotoksia, kuten vaatimuksia, käyttäjätarinoita, suunnittelukuvauksia ja koodia
- todentaa, että kaikki kuvatut vaatimukset on täytetty
- kelpuuttaa testauksen kohteen valmius ja toimivuus käyttäjien ja muiden sidosryhmien odotusten mukaisesti
- kasvattaa luottamusta testauksen kohteen laadun tasoon
- ennaltaehkäistä vikoja
- löytää häiriöitä ja vikoja
- tuottaa sidosryhmille riittävästi tietoa, jotta nämä voivat tehdä valistuneita päätöksiä erityisesti testauksen kohteen laadun tason suhteen
- vähentää riittämättömään ohjelmiston laatuun liittyvien riskien tasoa (esim. aikaisemmin havaitsemattomien häiriöiden esiintyminen tuotantokäytössä)
- täyttää sopimuksiin, lakeihin tai muuhun sääntelyyn perustuvat vaatimukset ja/tai todentaa, että testauksen kohde on tällaisten vaatimusten tai standardien mukainen.

Testauksen tavoitteet voivat vaihdella testattavana olevan komponentin tai järjestelmän tilanteen, testaustason sekä käytetyn ohjelmistokehitysmallin perusteella. Näihin eroihin voivat kuulua esimerkiksi seuraavat:

- Yksikkötestauksen aikana yksi tavoitteista voi olla löytää niin paljon häiriöitä kuin mahdollista, jotta niiden taustalla olevat viat tunnistetaan ja korjataan aikaisin. Toinen tavoite voi olla yksikkötestien koodikattavuuden kasvattaminen.
- Hyväksymistestauksen aikana tavoitteena voi olla sen varmistaminen, että järjestelmä toimii odotetusti ja täyttää sille asetetut vaatimukset. Toinen tavoite tämän testauksen aikana voi olla tie-

don tuottaminen sidosryhmille koskien riskejä, jotka liittyvät järjestelmän julkaisemiseen tiettynä ajankohtana.

1.1.2 Testaus ja virheenjäljitys

Testaus ja virheenjäljitys ovat eri asioita. Testien suorittamisella voidaan tuoda esiin häiriöitä, jotka ovat ohjelmistossa olevien vikojen aiheuttamia. Virheenjäljitys on järjestelmäkehitykseen liittyvä tehtävä, jonka avulla löydetään, analysoidaan ja korjataan tällaiset viat. Tämän jälkeen uudelleentestauksella tarkastetaan, ovatko korjaukset ratkaisseet viat. Joissain tapauksissa testaajat ovat vastuussa ensimmäisistä testeistä ja korjauksen jälkeisestä uudelleentestauksesta, kun taas kehittäjät hoitavat virheenjäljityksen ja siihen liittyvän yksikkötestauksen. Ketterässä kehityksessä ja joissakin muissa elinkaarimalleissa testaajat voivat kuitenkin osallistua virheenjäljitykseen ja yksikkötestaukseen.

ISO-standardista ISO/IEC/IEEE 29119-1 löytyy lisää tietoa ohjelmistotestauksen käsitteistä.

1.2 *Miksi testaus on välttämätöntä?*

Komponenttien ja järjestelmän sekä niihin liittyvien dokumenttien perusteellinen testaus voi auttaa vähentämään tuotantokäytössä tapahtuvien häiriöiden riskiä. Kun vikoja löydetään ja sen jälkeen korjataan, se myötävaikuttaa komponenttien tai järjestelmän laatuun. Ohjelmistotestausta voidaan myös tarvita sopimuksellisten tai lakeihin pohjautuvien tai kyseisen teollisuudenalan standardeihin liittyvien vaatimusten täyttämiseksi.

1.2.1 Testauksen merkitys onnistumisen kannalta

Läpi tietojenkäsittelyn historian on ollut varsin tyypillistä, että ohjelmistoja ja järjestelmiä on otettu käyttöön ja niissä olleiden vikojen vuoksi ne ovat aiheuttaneet häiriöitä tai eivät muuten ole täyttäneet sidosryhmien tarpeita. Sopivia testaustekniikoita käyttämällä voidaan kuitenkin vähentää tällaisten ongelmallisten ohjelmistotoimitusten esiintymistiheyttä, mikäli näitä tekniikoita käytetään oikealla testausasiantuntemuksella, oikeilla testaustasoilla ja oikeissa ohjelmistokehityksen elinkaaren vaiheissa. Esimerkkeihin kuuluvat:

- Testaajien ottaminen mukaan vaatimusten katselmointeihin tai käyttäjätarinoiden tarkentamiseen voi paljastaa vikoja näissä tuotoksissa. Vaatimuksissa olevien vikojen tunnistaminen ja poistaminen vähentää riskiä, että toteutetaan vääränlaisia tai testauskelvottomia ominaisuuksia.
- Kun testaajat työskentelevät järjestelmän suunnitteluvaiheessa tiiviisti yhdessä suunnittelijoiden kanssa, se voi auttaa kaikkia osapuolia ymmärtämään paremmin suunnitelmia ja kuinka niitä tulee testata. Tämä parempi ymmärrys voi vähentää perustavaa laatua olevien suunnitteluvikojen riskiä ja mahdollistaa testien tunnistamisen varhaisessa vaiheessa.
- Kun testaajat työskentelevät koodin toteutusvaiheessa tiiviisti yhdessä kehittäjien kanssa, se voi auttaa kaikkia osapuolia ymmärtämään paremmin koodia ja kuinka sitä tulee testata. Tämä parempi yhteisymmärrys voi vähentää koodissa ja testeissä olevien vikojen riskiä.
- Testaajien tekemä ohjelmiston todentaminen ja kelpuutus ennen sen julkaisua voi auttaa löytämään häiriöitä, jotka muuten olisivat jääneet huomaamatta, ja tukee häiriöitä aiheuttaneiden vikojen poistamisprosessia (virheenjäljitystä). Tämä kasvattaa todennäköisyyttä, että ohjelmisto täyttää sidosryhmien tarpeet ja täyttää vaatimukset.

Näiden esimerkkien lisäksi testauksen tavoitteiden määrittäminen myötävaikuttaa koko ohjelmistokehitykseen ja ylläpidon onnistumiseen.

1.2.2 Laadunvarmistus ja testaus

Vaikka ihmiset usein käyttävät käsitettä laadunvarmistus (tai pelkästään QA) puhuessaan testauksesta, laadunvarmistus ja testaus eivät ole sama asia, mutta ne liittyvät toisiinsa. Isompi käsite, laadunhallinta, sitoo ne yhteen. Laadunhallintaan kuuluvat kaikki toimenpiteet, jotka ohjaavat ja hallitsevat organisaatiota, kun kyse on laadusta. Muiden tehtävien mukana laadunhallintaan kuuluvat sekä laadunvarmistus että laadunvalvonta. Laadunvarmistus keskittyy tyypillisesti oikeiden prosessien noudattamiseen, jotta voidaan luottaa siihen, että riittävä laatutaso tullaan saavuttamaan. Kun prosesseja toteutetaan oikein, prosessien tuloksena syntyvät tuotokset ovat yleensä parempilaatuisia, mikä myötävaikuttaa vikojen ennal-

taehkäisemiseen. Lisäksi perussyyanalyysin tekeminen vikojen aiheuttajien löytämiseksi ja poistamiseksi sekä jälkipalavereissa esiin tuotujen havaintojen oikea käyttö prosessien kehittämiseksi ovat tehokkaan laadunvarmistuksen kannalta tärkeitä.

Laadunvalvontaan kuuluu monia erilaisia tehtäviä, testaustehtävät mukaan luettuna, jotka tukevat riittävän laatutason saavuttamista. Testaustehtävät ovat osa koko ohjelmistokehitys- tai ylläpitoprosessia. Koska laadunvarmistus koskee koko prosessin oikeanlaista toteuttamista, laadunvarmistus tukee oikeanlaista testausta. Kuten on kuvattu alaluvuissa 1.1.1 ja 1.2.1, testaus myötävaikuttaa laadun saavuttamiseen monilla eri tavoilla.

1.2.3 Viat, virheet ja häiriöt

Ihminen voi tehdä virheen (erehdyksen), joka voi jota vian (bugin) syntymiseen ohjelmistokoodiin tai muuhun siihen liittyvään tuotokseen. Virhe, joka johtaa vian syntymiseen yhdessä tuotoksessa, voi synnyttää virheen, joka puolestaan johtaa vian syntymiseen toisessa tuotoksessa. Esimerkiksi vaatimuksia laadittaessa tapahtunut virhe voi johtaa vikaan vaatimuksissa, joka sitten johtaa ohjelmointivirheeseen, josta seuraa vika koodissa.

Jos koodissa oleva vika suoritetaan, se saattaa synnyttää häiriön, mutta näin ei välttämättä käy kaikissa tilanteissa. Esimerkiksi jotkut viat vaativat hyvin erityisiä syötearvoja tai esiehtoja aiheuttaakseen häiriön, joka voi tapahtua vain harvoin tai ei milloinkaan.

Virheitä voi tapahtua monista syistä, kuten:

- aikapaine
- ihmisten erehtyväisyys
- kokemattomat tai riittämättömän osaavat projektin jäsenet
- väärinymmärrykset projektin jäsenten välillä, mukaan luettuna vaatimuksiin ja suunnitteluun liittyvät väärinkäsitykset
- koodin, suunnitelmien, arkkitehtuurin, taustalla olevien ongelmien ja/tai käytettävien teknologioiden monimutkaisuus
- väärinymmärrykset liittyen järjestelmän sisäisiin ja ulkoisiin liittymiin, erityisesti silloin kun niitä on paljon
- uudet, vieraat teknologiat.

Koodissa olevien vikojen aiheuttamien häiriöiden lisäksi myös ympäristöolosuhteet voivat aiheuttaa häiriöitä. Esimerkiksi säteily, sähkömagneettiset kentät ja saasteet voivat aiheuttaa vikoja laitteistoissa tai vaikuttaa ohjelman suoritukseen muuttamalla laitteiston toimintaolosuhteita.

Kaikki odottamattomat testitulokset eivät ole häiriöitä. Vääriä positiivisia tuloksia voi syntyä siksi, että testit suoritettiin virheellisesti, testiaineistossa, ympäristössä tai muussa testimateriaalissa olevien vikojen vuoksi, tai muista syistä. Voi tapahtua myös käänteisiä tilanteita, joissa samanlaiset virheet tai viat johtavat vääriin negatiivisiin tuloksiin. Väärät negatiiviset ovat testejä, jotka eivät löydä vikoja, jotka niiden olisi pitänyt löytää; väärät positiiviset raportoidaan vikoina mutta ne eivät oikeasti ole vikoja.

1.2.4 Viat, alkuperäissyyt ja vaikutukset

Vikojen alkuperäissyyt ovat varhaisimpia tapahtumia tai tilanteita, jotka myötävaikuttavat vikojen syntymiseen. Vikoja voidaan analysoida niiden alkuperäissyyn tunnistamiseksi, jotta samankaltaisten vikojen syntymistä jatkossa voidaan vähentää. Keskittymällä kaikkein merkittävimpiin alkuperäissyihin, perussyyanalyysi voi johtaa prosessin parannuksiin, jotka estävät merkittävän määrän uusia vikoja syntymästä jatkossa.

Kuvitellaan esimerkiksi tilanne, jossa yksittäisestä virheellisestä koodirivistä johtuvat väärät koron maksut johtavat asiakasvalituksiin. Virheellinen koodi oli kirjoitettu tulkinnanvaraisen käyttäjätarinan perusteella, ja tulkinnanvaraisuus johtui tuoteomistajan koronlaskentaan liittyvästä väärinkäsityksestä. Jos suuri osuus vioista esiintyy koronlaskennassa, ja jos näiden vikojen alkuperäissyy on samanlaisissa väärinymmärryksissä, tuoteomistajia voitaisiin kouluttaa koronlaskennassa tällaisten vikojen vähentämiseksi tulevaisuudessa.

Tässä esimerkissä asiakasvalitukset ovat vaikutuksia. Väärät koronmaksut ovat häiriöitä. Koodissa oleva väärä laskutoimitus on vika, ja se johtuu ensimmäisestä viasta, käyttäjätarinan tulkinnanvaraisuudesta. Ensimmäisen vian alkuperäissyy oli tuoteomistajan tietämyksen puute, joka johti hänen tekemäänsä virheeseen hänen kirjoittaessaan käyttäjätarinaa. Perussyyanalyysin prosessia käsitellään ISTQB ETM Asiantuntijatason (Testauksen hallinta) sertifikaattisisällössä sekä ISTQB-EITP Asiantuntijatason (Testausprosessin kehittäminen) sertifikaattisisällössä.

1.3 *Seitsemän testauksen perusperiaatetta*

Viimeisen 50 vuoden aikana on ehdotettu muutamia testausperiaatteita, jotka esittelevät kaikelle testaukselle yhteiset yleiset suuntaviivat.

1. Testaus osoittaa virheiden läsnäolon, ei niiden poissaoloa.

Testauksella voidaan osoittaa, että vikoja on olemassa, mutta sillä ei voida osoittaa, että vikoja ei ole. Testaus pienentää todennäköisyyttä, että ohjelmistossa on vielä jäljellä löytämättömiä vikoja, mutta vaikka yhtään vikaa ei löydetä, se ei ole todiste virheettömyydestä.

2. Täydellinen testaus on mahdotonta

Kaiken testaaminen (syötteiden ja esiehtojen kaikki yhdistelmät) ei ole mahdollista lukuun ottamatta triviaaleja tapauksia. Sen sijaan, että pyritään täysin kattavaan testaukseen, tulisi käyttää riskianalyysiä, testaustekniikoita ja priorisointia testaustyön kohdentamiseen.

3. Aikainen testaus säästää aikaa ja rahaa

Jotta viat löydetään aikaisin, sekä staattiset että dynaamiset testaustehtävät pitäisi aloittaa mahdollisimman aikaisin ohjelmistokehityksen elinkaaressa. Aikaiseen testaukseen viitataan joskus termillä ”Shift left” (”siirry vasemmalle”). Testaus aikaisin ohjelmistokehityksen elinkaaressa auttaa vähentämään kalliiksi käyviä kuluja tai poistamaan niitä kokonaan (ks. alaluku 3.1).

4. Viat kasaantuvat

Yleensä pieni osa komponenteista sisältää suurimman osan ennen julkaisua tehdyssä testauksessa löydetyistä vioista, tai on vastuussa suurimmasta osasta käytönaikaisia häiriöitä. Ennustetut vikakeskittymät ja testauksen tai tuotantokäytön aikana havaitut todelliset vikakeskittymät ovat tärkeää tietoa riskianalyysissä, jonka avulla pyritään kohdentamaan testauspanostus (kuten mainittiin edellä periaatteessa 2).

5. Varo hyönteismyrkkyparadoksia

Jos samoja testejä toistetaan uudelleen ja uudelleen, lopulta nämä testit eivät enää löydä uusia vikoja. Uusien vikojen löytämiseksi olemassa olevia testejä ja testiaineistoa täytyy ehkä muuttaa ja on ehkä kirjoitettava uusia testejä. (Testit eivät enää toimi tehokkaasti vikojen löytämiseksi samoin kuin hyönteismyrkyt eivät enää tehoa hyönteisiin, kun myrkkyä on käytetty jonkin aikaa.) Joissain tapauksissa, kuten automatisoitujen regressiotestien kohdalla, hyönteismyrkkyparadoksi tuottaa hyödyllisen lopputuloksen, eli regressiovikojen määrä on suhteellisen pieni.

6. Testaus on tilannesidonnaista

Testausta tehdään eri tavalla eri tilanteissa. Esimerkiksi turvallisuuskriittinen teollinen ohjausjärjestelmä testataan eri tavalla kuin verkkokaupan mobiilisovellus. Toinen esimerkki: Ketterissä projekteissa testaus tehdään eri tavalla kuin peräkkäiselinkaarimallia käyttävissä projekteissa (ks. alaluku 2.1).

7. Virheiden poissaolo on harhaluulo

Jotkut organisaatiot odottavat, että testaajat voivat suorittaa kaikki mahdolliset testit ja löytää kaikki mahdolliset viat, mutta periaatteet 2 ja 1 (tässä järjestyksessä) osoittavat tämän mahdottomaksi. Lisäksi on harhaluuloista odottaa, että pelkästään löytämällä ja korjaamalla suuri määrä vikoja taataan järjestelmän onnistuminen. Esimerkiksi, vaikka kaikki järjestelmään liittyvät vaatimukset testattaisiin perusteellisesti ja kaikki löydetyt viat korjattaisiin, tuloksena voi silti iolla järjestelmä, jota on hankala käyttää, joka ei täytä käyttäjien tarpeita ja odotuksia tai on huonompi verrattuna vastaaviin kilpaileviin järjestelmiin.

Lisää esimerkkejä näistä ja muista testauksen periaatteista: ks. Myers 2011, Kaner 2002 ja Weinberg 2008.

1.4 *Testausprosessi*

Ei ole olemassa yhtä yleismaailmallista ohjelmistotestauksen prosessia, mutta on olemassa tyypillisistä testaustehtävistä muodostuvia kokonaisuuksia, joita ilman on epätodennäköistä, että testaus saavuttaa sille määritellyt tavoitteet. Nämä testaustehtävien kokonaisuudet muodostavat testausprosessin. Eri tilanteissa sopiva tietty ohjelmistotestausprosessi riippuu monista tekijöistä. Organisaation testausstrategiassa voidaan ottaa kantaa siihen, mitkä testaustehtävät kuuluvat testausprosessiin, kuinka nämä tehtävät toteutetaan ja milloin ne pitää suorittaa.

1.4.1 Testausprosessi eri tilanteissa

Organisaation testausprosessiin vaikuttaa tilannesidonnaisia tekijöitä, joihin kuuluvat muiden muassa seuraavat:

- käytössä olevat ohjelmistokehityksen elinkaarimalli ja projektimenetelmät
- harkittavissa olevat testaustasot ja testaustyypit
- tuote- ja projektiriskit
- liiketoiminta-alue
- toimintaan liittyvät rajoitteet, joita ovat muiden muassa:
    - budjetti ja resurssit
    - aikataulut
    - monimutkaisuus
    - sopimukselliset ja oikeudelliset vaatimukset
- organisaation politiikat ja käytännöt
- vaaditut sisäiset ja ulkoiset standardit.

Seuraavissa alaluvuissa käsitellään organisaatioiden testausprosessien yleisiä piirteitä seuraavista näkökulmista:

- testaustoimenpiteet ja tehtävät
- testauksen tuotokset
- jäljitettävyys testauksen pohjamateriaalin ja tuotosten välillä.

On erittäin hyödyllistä, jos testauksen pohjamateriaalissa (koskien kaikkia eri testaustasoja tai testaustyyppejä, joita ollaan harkitsemassa) on määritelty mitattavat kattavuuskriteerit. Kattavuuskriteerit voivat toimia tehokkaasti keskeisinä suorituskykyindikaattoreina (Key Performance Indicator, KPI), jotka ohjaavat tehtäviä, joilla osoitetaan testauksen tavoitteiden saavuttaminen (ks. alaluku 1.1.1).

Esimerkiksi jos ajatellaan mobiilisovellusta, testauksen pohjamateriaaliin saattaa kuulua vaatimusluettelo sekä lista tuetuista mobiililaitteista. Jokainen vaatimus on pohjamateriaalin osa. Jokainen tuettu laite on myös pohjamateriaalin osa. Kattavuuskriteereissä voidaan vaatia ainakin yhtä testitapausta jokaista pohjamateriaalin osaa kohti. Kun testit on suoritettu, niiden tulokset kertovat sidosryhmille, ovatko määritetyt vaatimukset täyttyneet ja tuliko tuettuja laitteita käytettäessä esiin häiriöitä.

ISO-standardista ISO/IEC/IEEE 29119-2 löytyy lisää tietoa testausprosesseista.

1.4.2 Testaustoimenpiteet ja tehtävät

Testausprosessi muodostuu seuraavista tehtävien pääryhmistä:

- testauksen suunnittelu
- testauksen seuranta ja hallinta
- testianalyysi
- testien suunnittelu

- testien valmistelu
- testien suoritus
- testauksen päättäminen.

Jokainen päätehtäväryhmä muodostuu toimenpiteistä, jotka on kuvattu edempänä. Jokainen päätehtäväryhmän toimenpide voi puolestaan koostua useista yksittäistä tehtävistä, jotka voivat vaihdella projektin tai julkaisun mukaan.

Vaikka monet näistä tehtäväryhmistä saattavat näyttää loogisesti peräkkäisiltä, ne toteutetaan usein iteratiivisesti. Esimerkiksi Ketterässä kehityksessä tehdään toinen toisensa jälkeen pieniä ohjelmiston suunnittelun, toteutuksen ja testauksen iteraatioita, joita jatkuva suunnittelu tukee. Näin ollen myös testaustoimenpiteitä tehdään tässä kehitysmallissa iteratiivisesti ja jatkuvasti. Jopa peräkkäismallia noudattavassa kehityksessä tehtävien askelittaiseen, loogiseen ketjuun voi kuulua päällekkäisyyksiä, yhdistämistä, samanaikaisuuksia tai poisjättämisiä, joten näiden päätehtävien räätälöiminen järjestelmän ja projektin tilanteen mukaan on usein tarpeen.

Testauksen suunnittelu

Testauksen suunnitteluun kuuluu tehtäviä, joiden avulla määritellään testauksen tavoitteet ja lähestymistavat näiden tavoitteiden saavuttamiseksi tilanteen aiheuttamien rajoitteiden puitteissa (esim. soveltuvien testaustekniikoiden ja -tehtävien määrittäminen sekä testausaikataulun laatiminen määräaikojen saavuttamiseksi). Testaussuunnitelmiin voidaan palata uudelleen seurannan ja hallinnan tehtävistä saadun palautteen perusteella. Testauksen suunnittelua käsitellään tarkemmin alaluvussa 5.2.

Testauksen seuranta ja hallinta

Testauksen seuranta käsittää todellisen edistymisen jatkuvan seurannan testaussuunnitelmaa vastaan suunnitelmassa määriteltyjä mittareita käyttämällä. Testauksen seurantaan kuuluu tarvittaviin toimenpiteisiin ryhtyminen testaussuunnitelmassa mainittujen tavoitteiden saavuttamiseksi (tavoitteita voidaan päivittää ajan kuluessa). Testauksen seurantaa ja hallintaa tukee lopetuskriteerien arviointi. Lopetuskriteereitä kutsutaan joissakin elinkaarimalleiksi valmiin määritelmäksi (Definition of Done, DoD) (ks. ISTQB-AT Perustason sertifikaattisisällön laajennus, Ketterä testaaja). Esimerkiksi testauksen suorituksen lopetuskriteereiden arviointiin määrätyllä testaustasolla voi kuulua seuraavia tehtäviä:

- testitulosten ja lokien tarkistus määrättyjä kattavuuskriteereitä vastaan
- komponentin tai järjestelmän laatutason arviointi testitulosten ja lokien perusteella
- testitapauksien lisätarpeen määrittäminen (esim. jos tietyn tuoteriskin kattamiseksi suunnitellut testit eivät saavutakaan tavoitteeksi asetettua kattavuustasoa, täytyy suunnitella ja suorittaa lisää testejä)

Testauksen etenemisestä suunnitelmaa vastaan kerrotaan sidosryhmille testauksen edistymisraporteissa, joissa mm. kuvataan poikkeamat suunnitelmasta sekä kerrotaan tietoja, jotka tukevat mahdollista päätöstä lopettaa testaus.

Testauksen seurantaa ja hallintaa käsitellään lisää alaluvussa 5.3.

Testianalyysi

Testianalyysin aikana testauksen pohjamateriaali analysoidaan testattavien ominaisuuksien tunnistamiseksi ja niihin liittyvien testattavien tilanteiden määrittämiseksi. Toisin sanoen, testianalyysi määrittää ”mitä pitää testata” mitattavien kattavuuskriteerien näkökulmasta.

Testianalyysiin kuuluvat seuraavat päätehtävät:

- työn alla olevalle testaustasolle soveltuvan testauksen pohjamateriaalin analysointi, esimerkiksi:
    - vaatimusmäärittelyt, kuten liiketoimintavaatimukset, toiminnalliset vaatimukset, järjestelmävaatimukset, käyttäjätarinat, eepokset, käyttötapaukset tai vastaavanlaiset tuotokset, joissa määritellään komponentin tai järjestelmän toivottu toiminnallinen ja ei-toiminnallinen käyttäytyminen
    - suunnitteluun ja toteutukseen liittyvät tiedot, kuten järjestelmän tai ohjelmiston arkkitehtuurikaaviot tai -dokumentit, suunnittelukuvaukset, kutsukaaviot, mallinnuskaaviot (esim.

UML- tai ER-kaaviot), rajapintamäärittelyt tai muut vastaavat tuotokset, jotka määrittelevät komponentin tai järjestelmän rakenteen

- itse komponentin tai järjestelmän toteutus, mukaan luettuna koodi, tietokannan metatiedot ja kyselyt sekä rajapinnat
- riskianalyysiraportit, joissa voidaan käsitellä komponentin tai järjestelmän toiminnallisia, ei-toiminnallisia ja rakenteellisia ominaisuuksia

- testauksen pohjamateriaalin ja testattavien kohteiden arviointi eri tyyppisten vikojen löytämiseksi, kuten:
  - tulkinnanvaraisuudet
  - puutteet
  - epäyhteneväisyydet
  - epätarkkuudet
  - ristiriitaisuudet
  - turha tieto
- testattavien ominaisuuksien ja ominaisuuksien kokonaisuuksien tunnistaminen
- jokaisen ominaisuuden testattavien tilanteiden tunnistaminen ja priorisointi testauksen pohjamateriaalin analyysin perusteella ottaen huomioon toiminnalliset, ei-toiminnalliset ja rakenteelliset piirteet, muut liiketoiminnalliset ja tekniset seikat sekä riskien tasot.
- kaksisuuntaisen jäljitettävyyden varmistaminen kaikkien testauksen pohjamateriaalin osien sekä niihin liittyvien testattavien tilanteiden välillä (ks. alaluvut 1.4.3 ja 1.4.4).

Mustalaatikko-, lasilaatikko- ja kokemuspohjaisten tekniikoiden soveltaminen voi olla hyödyllistä testianalyysissä (ks. luku 4) tärkeiden testattavien tilanteiden huomiotta jättämiseen todennäköisyyden pienentämiseksi sekä täsmällisempien ja tarkempien testattavien tilanteiden tunnistamiseksi.

Joissain tapauksissa testianalyysi tuottaa testattavia tilanteita, joita käytetään testauksen tavoitteina testausohjeissa. Testausohjeet ovat joidenkin kokemuspohjaisten testaustyyppien tyypillisiä tuotoksia (ks. alaluku 4.4.2). Kun nämä tavoitteet ovat jäljitettävissä testauksen pohjamateriaaliin, voidaan mitata kyseisen kokemuspohjaisen testauksen aikana saavutettu kattavuus.

Vikojen tunnistaminen testianalyysin aikana voi tuottaa tärkeää hyötyä, erityisesti silloin, jos ei käytetä muuta katselmointiprosessia ja/tai testausprosessi liittyy läheisesti katselmointiprosessiin. Testianalyysi ei ainoastaan todenna vaatimusten yhdenmukaisuutta, että ne ovat oikein ilmaistuja ja täydellisiä, vaan sen avulla voidaan myös varmistaa, että vaatimukset kuvaavat oikein asiakkaan, käyttäjien ja muiden sidosryhmien tarpeet. Esimerkiksi käyttäytymiseen pohjautuva kehitys (behavior driven development, BDD) ja hyväksymistestaukseen pohjautuva kehitys (acceptance test driven development, ATDD), joihin kuuluu testattavien tilanteiden ja testitapausten luominen käyttäjätarinoista ja hyväksymiskriteereistä ennen koodin toteutusta, myös todentavat ja kelpuuttavat käyttäjätarinoita ja hyväksymiskriteereitä sekä löytävät vikoja niistä (ks. ISTQB Perustason sertifikaattisisällön laajennus, Ketterä testaaja).

Testien suunnittelu

Testien suunnittelun aikana testattavat tilanteet lavennetaan ylemmän tason testitapauksiksi, niistä muodostetuiksi testijoukoiksi, sekä muuksi testimateriaaliksi. Näin ollen testianalyysi vastaa kysymykseen ”mitä testata?” kun taas testien suunnittelu vastaa kysymykseen ”kuinka testata?”.

Testien suunnitteluun kuuluvat seuraavat päätehtävät:

- testitapausten ja testijoukkojen suunnittelu ja priorisointi
- testattavia tilanteita ja testitapauksia tukevan tarvittavan testiaineiston tunnistaminen
- testiympäristön kokoonpanon suunnittelu ja tarvittavan infrastruktuurin ja välineiden tunnistaminen
- jäljitettävyyden varmistaminen testauksen pohjamateriaalin, testattavien tilanteiden, testitapausten ja testiproseduurien välillä (ks. alaluku 1.4.4).

Testattavien tilanteiden laventamiseen testitapauksiksi ja testijoukoiksi testien suunnittelun aikana käytetään usein testaustekniikoita (ks. luku 4).

Aivan kuten testianalyysi, myös testien suunnittelu voi johtaa samantyyppisten vikojen löytymiseen testauksen pohjamateriaalissa. Myös vikojen tunnistaminen testien suunnittelun aikana on tärkeä potentiaalinen hyöty, samoin kuin testianalyysissä.

Testien valmistelu

Testauksen toteutusvaiheessa laaditaan ja viimeistellään testien suorituksessa tarvittava testimateriaali sekä järjestetään testitapaukset testiproseduureiksi. Siinä missä testien suunnittelu vastaa kysymykseen ”kuinka testata”, testien toteutus vastaa kysymykseen ”onko meillä kaikki valmiina testien suoritusta varten?”.

Testien valmisteluun kuuluvat seuraavat päätehtävät:

- testiproseduurien laatiminen ja priorisointi sekä tarvittaessa automatisoitujen testiskriptien luominen
- testijoukkojen laatiminen testiproseduureista ja automatisoiduista testiskripteistä (jos niitä on)
- testijoukkojen järjestäminen testien suoritusaikataulussa niin, että se tuottaa mahdollisimman tehokkaan testien suorituksen (ks. alaluku 5.2.4)
- testiympäristön laatiminen (mukaan luettuna tarvittaessa testikehykset, palvelujen virtualisointi, simulaattorit ja muut infrastruktuuriin liittyvät osat) sekä sen todentaminen, että kaikki tarpeellinen on koottu oikein
- testiaineiston valmistelu ja sen varmistaminen, että aineisto on ladattu oikein testiympäristöön
- kaksisuuntaisen jäljitettävyyden varmistaminen ja päivittäminen testauksen pohjamateriaalin, testattavien tilanteiden, testitapausten, testiproseduurien ja testijoukkojen välillä (ks. alaluku 1.4.4).

Testien suunnittelun ja valmistelun tehtävät yhdistetään usein.

Tutkivassa testauksessa ja muissa kokemuspohjaisen testauksen tyypeissä testien suunnittelu ja valmistelu voi tapahtua ja voidaan dokumentoida osana testien suoritusta. Tutkiva testaus voi pohjautua testausohjeisiin (jotka on tuotettu osana testianalyysiä) ja tutkivat testit suoritetaan välittömästi, kun ne on suunniteltu ja toteutettu (ks. alaluku 4.4.2).

Testien suoritus

Testien suorituksen aikana testijoukot suoritetaan testien suoritusaikataulun mukaisesti.

Testien suoritukseen kuuluvat seuraavat päätehtävät:

- testattavien kohteiden, testityökalujen ja testimateriaalin yksilöivien tietojen ja versioiden kirjaus
- testien suoritus joko manuaalisesti tai testien suoritustyökaluja käyttämällä
- todellisten tulosten vertaaminen odotettuihin tuloksiin
- poikkeamien analysointi niiden todennäköisten syiden määrittämiseksi (esim. häiriöt voivat johtua koodissa olevista vioista, mutta myös vääriä positiivisia voi tapahtua (ks. alaluku 1.2.3)).
- vikojen raportointi havaittujen häiriöiden pohjalta (ks. alaluku 5.6)
- testien suorituksen tulosten kirjaaminen (esim. hyväksytty, hylätty, estynyt)
- testaustehtävien toistaminen joko poikkeamasta seuranneiden toimenpiteiden vuoksi tai osana suunniteltua testausta (esim. korjatun testin suoritus, uudelleentestaus ja/tai regressiotestaus)
- kaksisuuntaisen jäljitettävyyden todentaminen ja päivittäminen testauksen pohjamateriaalin, testattavien tilanteiden, testitapausten, testiproseduurien ja testitulosten välillä (ks. alaluku 1.4.4).

Testauksen päättäminen

Testauksen päätöstehtävien avulla kerätään tietoa tehdyistä testaustehtävistä kokemuksien sekä testimateriaaliin liittyvän ja muun oleellisen tiedon yhteenkokoamiseksi. Testauksen päätöstehtäviä tapahtuu projektin tiettyinä ajankohtina, kuten esimerkiksi silloin, kun ohjelmistojärjestelmä julkaistaan, testauspro-

jekti saadaan valmiiksi (tai peruutetaan), ketterän projektin iteraatio päättyy (esim. osana retrospektiiviä), testaustaso saadaan valmiiksi tai kun ylläpitojulkaisu on saatu valmiiksi.

Testauksen päättämiseen kuuluvat seuraavat päätehtävät:

- sen tarkistaminen, onko kaikki havaintoraportit suljettu, ja muutospyyntöjen laatiminen tai tehtävien lisääminen tuotteen kehitysjonoon niiden vikojen perusteella, jotka ovat selvittämättömiä testauksen suorituksen loppuessa.
- testauksen yhteenvetoraportin laatiminen sidosryhmille välitettäväksi
- testiympäristön, testiaineiston, testi-infrastruktuurin ja muun testimateriaalin viimeistely ja arkistointi myöhempää uusiokäyttöä varten
- testimateriaalin luovuttaminen ylläpitotiimeille, muille projektin tiimeille ja/tai muille sidosryhmille, joille voi olla hyötyä sen käytöstä
- valmistuneista testaustehtävistä kertyneiden kokemusten analysointi tulevissa iteraatioissa, julkaisuissa ja projekteissa tarvittavien muutosten määrittämiseksi
- kerätyn tiedon käyttäminen testausprosessin kypsyyden parantamiseksi.

1.4.3 Testauksen tuotokset

Testauksen tuotokset luodaan osana testausprosessia. Aivan kuten organisaatioiden tavoissa toteuttaa testausprosessia on merkittäviä eroja, merkittäviä eroja on myös siinä, mitä tuotoksia luodaan prosessin aikana, millä tavalla tuotokset organisoidaan ja miten niitä hallinnoidaan sekä miten ne nimetään. Tämä sertifikaattisisältö pohjautuu yllä kuvattuun testausprosessiin sekä tässä sertifikaattisisällössä muualla ja ISTQB:n sanastossa kuvattuihin tuotoksiin. ISO-standardi ISO/IEC/IEEE 29119-3 voi myös toimia pohjana testaustuotosten suhteen.

Monia tässä alaluvussa kuvattuja testauksen tuotoksia voidaan tallentaa ja hallita testauksen hallintavälineiden ja havaintojenhallintavälineiden avulla (ks. luku 6).

Testauksen suunnittelun tuotokset

Testauksen suunnittelun tuotoksiin kuuluu tyypillisesti yksi tai useampia testaussuunnitelmia. Testaussuunnitelma sisältää tietoa testauksen pohjamateriaalista, johon muut testauksen tuotokset liittyvät jäljitettävyyden kautta (ks. edempänä sekä alaluku 1.4.4), samoin kuin testauksen seurannan ja hallinnan aikana käytettävistä päätöskriteereistä (tai valmiin määritelmästä). Testaussuunnitelmat on kuvattu alaluvussa 5.2.

Testauksen seurannan ja hallinnan tuotokset

Testauksen seurannan ja hallinnan tuotoksiin kuuluu tyypillisesti erilaisia testauksen raportteja, kuten edistymisraportteja (tuotetaan jatkuvasti ja/tai määräajoin) ja yhteenvetoraportteja (tuotetaan edistymisen eri tarkistuspisteissä). Kaikkien testausraporttien pitäisi tarjota kohdeyleisölle oleellista tietoa testauksen edistymisestä raportin laatimishetkellä, mukaan luettuna testien suoritusten tulokset, kun ne ovat saatavilla.

Testauksen seurannan ja hallinnan tuotosten pitäisi myös käsitellä projektinhallinnallisia asioita, kuten tehtävien valmistuminen, resurssien allokointi ja käyttö sekä työmäärät.

Testauksen seurantaa ja hallintaa sekä näiden tehtävien aikana luotavia tuotoksia käsitellään tarkemmin tämän sertifikaattisisällön alaluvussa 5.3.

Testianalyysin tuotokset

Testianalyysin tuotoksiin kuuluvat määritellyt ja priorisoidut testattavat tilanteet, joista ihannetilanteessa jokainen on kaksisuuntaisesti jäljitettävissä asianomaisiin testauksen pohjamateriaalin osiin. Tutkivassa testauksessa testianalyysiin voi liittyä testausohjeiden laatiminen. Testianalyysi voi myös johtaa vikojen löytämiseen testauksen pohjamateriaalista sekä niiden raportointiin.

Testien suunnittelun tuotokset

Testien suunnittelun tuloksena syntyy testitapauksia ja testitapauksien joukkoja, joilla käydään läpi testianalyysissa määritellyt testattavat tilanteet. Hyvä käytäntö on usein suunnitella ylätason testitapauksia

ilman konkreettisia syötearvoja ja odotettuja tuloksia. Tällaiset ylätason testitapaukset ovat uudelleenkäytettäviä useilla eri testikierroksilla vaihtuvan konkreettisen aineiston kanssa, mutta ne dokumentoivat silti riittävän tarkasti testitapauksen tarkoituksen. Ihanteellisesti kaikista testitapauksista on kaksisuuntainen jäljitettävyys testattavaan tilanteeseen tai tilanteisiin, jotka se kattaa.

Testien suunnittelun yhteydessä suunnitellaan ja/tai tunnistetaan myös tarvittava testiaineisto, suunnitellaan testiympäristö ja tunnistetaan infrastruktuuri ja työkalut, vaikkakin näiden dokumentoinnin laajuus vaihtelee suuresti.

Testianalyysin aikana määriteltyjä testattavia tilanteita voidaan tarkentaa testien suunnittelun yhteydessä.

Testien toteutuksen tuotokset

Testien toteutuksen tuotoksiin kuuluvat

- testiproseduurit sekä niiden suoritusjärjestys
- testijoukot
- testien suoritusaikataulu.

Kun testien toteutus on valmis, testiproseduurien ja testauksen pohjamateriaalin välisen kaksisuuntaisen jäljitettävyyden avulla voidaan ihanteellisesti osoittaa testaussuunnitelmassa määriteltyjen kattavuuskriteerien täyttyminen testitapausten ja testattavien tilanteiden avulla.

Joissakin tilanteissa testien toteutukseen kuuluu työkaluilla tuotettavien tai työkalujen käyttämien tuotosten laatiminen, kuten palvelujen virtualisointi ja automatisoidut testiskriptit.

Testien toteutukseen voi myös kuulua testiaineiston ja testiympäristön luominen ja todentaminen. Aineiston ja/tai ympäristön todentamiseen liittyvän dokumentaation täydellisyys voi vaihdella merkittävästi.

Testiaineiston tehtävä on tarjota konkreettiset arvot testitapausten syötteiksi ja odotetuiksi tuloksiksi. Tällaiset konkreettiset arvot yhdessä niiden käyttöön liittyvien yksiselitteisten ohjeiden kanssa muuttavat ylätason testitapaukset suoritettaviksi alatason testitapauksiksi. Näitä samoja ylemmän tason testitapauksia voidaan käyttää eri testiaineiston kanssa, kun niitä suoritetaan testattavan kohteen eri versioilla. Konkreettiseen testiaineistoon liittyvät konkreettiset odotetut tulokset tunnistetaan käyttämällä testioraakkelia.

Tutkivassa testauksessa osa testien suunnittelun ja toteutuksen tuotoksista voi syntyä testien suorituksen aikana, vaikkakin tutkivan testauksen dokumentoinnin taso (samoin kuin testien jäljitettävyys testauksen pohjamateriaalin eri osiin) voi vaihdella huomattavasti.

Testianalyysin aikana määriteltyjä testattavia tilanteita voidaan tarkentaa testien valmistelun yhteydessä.

Testien suorituksen tuotokset

Testien suorituksen tuotoksiin kuuluvat

- yksittäisten testitapausten tai testiproseduurien tilaan liittyvä dokumentaatio (esim. valmis suoritettavaksi, hyväksytty, hylätty, estynyt, tarkoituksella ohitettu jne.)
- vikaraportit (ks. alaluku 5.6)
- testaukseen liittyvien testattavien nimikkeiden, testauksen kohteiden, testityökalujen ja testimateriaalin dokumentointi.

Kun testien suoritus on saatu valmiiksi, ihannetilanteessa voidaan kaksisuuntaisen jäljitettävyyden avulla määritellä jokaisen testimateriaalin osan tila ja siitä voidaan raportoida siihen liittyvien testiproseduurien pohjalta. Esimerkiksi pystymme löytämään vaatimukset, joiden pohjalta suunnitelluista testeistä kaikki on läpäisty, mihin vaatimuksiin liittyy hylkääntyneitä testejä ja/tai vikoja ja mihin vaatimuksiin liittyy testejä, jotka odottavat vielä suorittamista. Tämä mahdollistaa kattavuuskriteerien täyttymisen todentamisen sekä testituloksista raportoinnin sidosryhmien ymmärtämällä tavalla.

Testauksen päättämisen tuotokset

Testauksen päättämisen tuotoksiin kuuluvat testauksen yhteenvetoraportit, esim. ketterän projektin retrospektiivin jälkeen esiin tulleet seuraavien projektien tai iteraatioiden, muutosehdotuksien tai tuotteen kehitysjonon parantamiseen liittyvät toimenpiteet sekä viimeistelty testausmateriaali.

1.4.4 Jäljitettävyys testauksen pohjamateriaalin ja tuotosten välillä

Kuten alaluvussa 1.4.3 on mainittu, testauksen tuotokset ja tuotosten nimet vaihtelevat huomattavasti. Näistä vaihteluista huolimatta tehokkaan testauksen seurannan ja hallinnan kannalta on tärkeää suunnitella jäljitettävyys ja ylläpitää sitä koko testausprosessin läpi eri testauksen pohjamateriaalin osien ja erilaisten kyseisiin osiin liittyvien testauksen tuotosten välillä, kuten yllä on kuvattu. Testikattavuuden arvioinnin lisäksi hyvä jäljitettävyys auttaa

- muutosten vaikutusten analysoinnissa
- tekemään testauksesta auditoitavan
- täyttämään tietohallinnon kriteerit
- parantamaan testauksen edistymisraporttien ja yhteenvetoraporttien ymmärrettävyyttä tuomalla raportteihin mukaan myös tiedon testauksen pohjamateriaalin osien tilasta (esim. vaatimukset, joihin liittyvien testien tila on läpäisty tai epäonnistunut sekä vaatimukset, joihin liittyvät testit ovat vielä kesken)
- kuvaamaan testauksen tekniset piirteet sidosryhmille näiden ymmärtämien termien avulla
- tuottamaan tietoa tuotteen laadun ja testausprosessin kyvykkyyden arvioimiseksi sekä projektin etenemisen arvioimiseksi liiketoiminnan tavoitteita vastaan.

Jotkut testauksen hallintavälineet tarjoavat testauksen tuotoksia kuvaavia malleja, jotka ovat yhteneviä osan tai kaikkien tässä alaluvussa kuvattujen testauksen tuotoksien kanssa. Jotkut organisaatiot rakentavat omat hallintajärjestelmänsä tuotosten hallinnoimiseksi sekä tarvitsemansa jäljitettävyystiedon tuottamiseksi.

1.5 *Testauksen psykologia*

Ohjelmistokehityksessä, ohjelmistotestaus mukaan luettuna, ollaan tekemisissä ihmisten kanssa. Siksi ihmispsykologialla on merkittäviä vaikutuksia ohjelmistotestaukseen.

1.5.1 Ihmispsykologia ja testaus

Vikojen löytäminen staattisen testauksen aikana esimerkiksi vaatimuskatselmoinneissa tai käyttäjätarinoiden tarkentamisen aikana tai häiriöiden tunnistaminen dynaamisen testien suorituksen aikana voidaan käsittää tuotteen ja sen tekijän kritisoinniksi. Ihmispsykologiaan kuuluu osa, jota kutsutaan vahvistusharhaksi (eng. confirmation bias), ja se voi vaikeuttaa sellaisen tiedon hyväksymistä, joka poikkeaa yleisesti vallalla olevista uskomuksista. Koska esimerkiksi kehittäjät uskovat heidän koodinsa olevan oikein, heihin vaikuttaa vahvistusharha, jonka vuoksi heidän on vaikea hyväksyä, että koodi on väärin. Vahvistusharhan lisäksi muiden kognitiivisten ennakkokäsitysten vuoksi ihmisten voi olla vaikea ymmärtää tai hyväksyä testauksen tuottamaa tietoa. On myös tyypillinen inhimillinen piirre syyttää huonojen uutisten tuojaa, ja testauksen tuottama tieto sisältää usein huonoja uutisia.

Näiden psykologisten seikkojen vuoksi jotkut ihmiset voivat pitää testausta tuhoavana toimintana, vaikka se myötävaikuttaakin suuresti projektin etenemiseen ja tuotteen laatuun (ks. alaluvut 1.1 ja 1.2). Näiden ennakkokäsitysten vähentämiseksi vikoihin ja häiriöihin liittyvä tieto tulisi välittää rakentavalla tavalla. Näiden ennakkokäsitysten vähentämiseksi vikoihin ja häiriöihin liittyvä tieto tulisi välittää rakentavalla tavalla. Tämä pätee sekä staattiseen että dynaamiseen testaukseen.

Testaajilla ja testauspäälliköillä tulee olla hyvät vuorovaikutustaidot, jotta he pystyvät viestimään tehokkaasti vioista, häiriöistä, testituloksista, testauksen etenemisestä ja riskeistä sekä rakentamaan positiiviset suhteet työtovereiden kanssa. Seuraavassa on kuvattu hyvän viestinnän esimerkkejä:

- Aloita mieluummin yhteistyöllä kuin taistelulla. Muistuta kaikkia yhteisestä tavoitteesta, joka on parempilaatuinen järjestelmä.
- Korosta testauksen hyötyjä. Esimerkiksi materiaalien laatijoiden kannalta vioista saatava tieto voi auttaa heitä parantamaan tuotoksiaan ja osaamistaan. Organisaation kannalta testauksen aikana löydetyt ja korjatut viat säästävät aikaa ja rahaa ja pienentävät tuotteen laatuun liittyvää riskiä.

- Kerro testituloksista ja muista löydöksistä neutraalisti, tosiasioihin keskittyen ilman kritiikkiä viallisen tuotoksen laatijaa kohtaan. Kirjaa vikaraportit ja katselmointihavainnot objektiivisesti ja todenmukaisesti.
- Yritä ymmärtää, miltä toisesta henkilöstä tuntuu ja syyt siihen, miksi he voivat reagoida negatiivisesti saamaansa tietoon.
- Varmista, että toinen henkilö on ymmärtänyt, mitä on sanottu, ja päinvastoin.

Tyypillisiä testauksen tavoitteita käsiteltiin aikaisemmin (ks. alaluku 1.1). On selvää, että oikeiden testauksen tavoitteiden määrittelyllä on tärkeitä psykologisia vaikutuksia. Useimmat ihmiset pyrkivät suunnittelemaan toimintansa ja käyttäytymisensä tiimin, johdon ja muiden sidosryhmien asettamien tavoitteiden mukaisesti. On myös tärkeää, että testaajat sitoutuvat näihin tavoitteisiin niin, että heidän henkilökohtaiset ennakkonäkemyksensä vaikuttavat niihin mahdollisimman vähän.

1.5.2 Testaajan ja kehittäjän ajatusmallit

Kehittäjät ja testaajat ajattelevat usein eri tavalla. Toteutuksen päätavoite on suunnitella ja rakentaa tuote. Kuten aikaisemmin todettiin, testauksen tavoitteisiin kuuluvat tuotteen todentaminen ja kelpuutus, vikojen löytäminen ennen tuotteen julkaisua ja niin edelleen. Nämä ovat erilaisia tavoitteita, jotka vaativat erilaiset ajatusmallit. Näiden ajatusmallien käsitteleminen yhdessä auttaa saavuttamaan korkeamman tuotteen laatutason.

Ajatusmalli heijastaa yksilön oletuksia ja päätöksenteossa ja ongelmanratkaisussa suosimia menetelmiä. Testaajan ajatusmalliin pitäisi kuulua uteliaisuutta, ammatillista pessimismiä, kriittistä silmää, yksityiskohtien huomioimista sekä motivaatio hyvään ja positiiviseen viestintään sekä vuorovaikutukseen. Testaajan ajatusmalli kasvaa ja kypsyy sitä mukaa, kun testaaja saa lisää kokemusta.

Kehittäjän ajatusmalliin voi kuulua joitakin piirteitä testaajan ajatusmallista, mutta menestyvät kehittäjät ovat usein kiinnostuneempia ratkaisujen suunnittelusta ja toteutuksesta kuin sen pohtimisesta, mikä kyseisissä ratkaisuissa voisi olla väärin. Lisäksi vahvistusharha vaikeuttaa vikojen löytämistä heidän omasta työstään.

Oikeaa ajatusmallia käyttämällä kehittäjät pystyvät testaamaan omaa koodiaan. Erilaisissa ohjelmistokehityksen elinkaarimalleissa on usein eri tavat testaajien ja heidän tehtäviensä organisoimiseksi. Jos riippumattomat testaajat suorittavat osan testaustehtävistä, se tehostaa vikojen löytämistä, mikä on erityisen tärkeää, kun on kyse suurista, monimutkaisista tai turvallisuuskriittisistä järjestelmistä. Riippumattomien testaajien näkökulma testaukseen on erilainen kuin testausmateriaalin laatijoiden (eli liiketoiminta-analyytikoiden, tuoteomistajien, suunnittelijoiden ja ohjelmoijien), koska heidän kognitiiviset ennakkoasenteensa ovat erilaiset kuin materiaalin laatijoiden.

2 Testaus ohjelmistokehityksen elinkaaressa	100 minuuttia

Avainsanat

alfa-testaus, beta-testaus, ei-toiminnallinen testaus, hyväksymistestaus, integraatiotestaus, järjestelmäintegraatiotestaus, järjestelmätestaus, kaupallinen valmisohjelmisto, komponentti-integraatiotestaus, käyttöön soveltuvuuden hyväksymistestaus, käyttäjän hyväksymistestaus, lasilaatikkotestaus, peräkkäiselinkaarimalli, regressiotestaus, sopimukseen pohjautuva hyväksymistestaus, säädöksiin pohjautuva hyväksymistestaus, testauksen kohde, testauksen pohjamateriaali, testauksen tavoite, testaustaso, testitapaus, testityyppi, testiympäristö, toiminnallinen testaus, vaikutusanalyysi, varmistustestaus, yksikkötestaus, ylläpitotestaus,

Oppimistavoitteet: Testaus ohjelmistokehityksen elinkaaressa

2.1 Ohjelmistokehityksen elinkaarimallit

FL-2.1.1 (K2) Selittää ohjelmistokehityksen tehtävien ja testaustehtävien väliset suhteet ohjelmistokehityksen elinkaaressa

FL-2.1.2 (K1) Tunnistaa syyt, miksi ohjelmistokehityksen elinkaarimalleja täytyy muokata projektin ja tuotteen ominaisuuksien mukaan

2.2 Testaustasot

FL-2.2.1 (K2) Vertailla eri testaustasoja niiden tavoitteiden, pohjamateriaalin, kohteiden, tyypillisten vikojen ja häiriöiden sekä lähestymistapojen ja vastuiden perusteella

2.3 Testaustyypit

FL-2.3.1 (K2) Vertailla toiminnallista, ei-toiminnallista ja lasilaatikkotestausta

FL-2.3.2 (K1) Ymmärtää, että toiminnallista, ei-toiminnallista ja lasilaatikkotestausta tehdään kaikilla testaustasoilla

FL-2.3.3 (K2) Vertailla uudelleentestauksen ja regressiotestauksen tarkoituksia

2.4 Ylläpitotestaus

FL-2.4.1 (K2) Vetää yhteen ylläpitotestauksen syyt

FL-2.4.2 (K2) Kuvata vaikutusanalyysin rooli ylläpitotestauksessa

2.1 *Ohjelmistokehityksen elinkaarimallit*

Ohjelmistokehityksen elinkaarimalli kuvaa eri tyyppiset tehtävät, joita tehdään ohjelmistokehitysprojektin eri vaiheissa, sekä miten tehtävät liittyvät toisiinsa loogisesti ja ajallisesti. On olemassa monia erilaisia ohjelmistokehityksen elinkaarimalleja, joista jokainen vaatii erilaisia lähestymistapoja testaukseen.

2.1.1 Ohjelmistokehitys ja ohjelmistotestaus

Tärkeä osa testaajan roolia on tuntea tyypilliset ohjelmistokehitysmallit sopivien testaustoimenpiteiden valitsemiseksi.

Jokaiseen ohjelmistokehityksen elinkaarimalliin kuuluu useita hyvän testauksen ominaisuuksia:

- Jokaiselle kehitystehtävälle on olemassa vastaava testaustehtävä.
- Jokaisella testaustasolla on erityisesti sitä koskevia testaustavoitteita.
- Kunkin testaustason testien analysointi ja suunnittelu alkaa vastaavien toteutustehtävien aikana
- Testaajat osallistuvat vaatimusten ja suunnitelmien määrittelyn ja tarkennuksen yhteydessä käytäviin keskusteluihin ja ovat mukana katselmoimassa tuotoksia (esim. vaatimuksia, suunnitelmia, käyttäjätarinoita jne.) heti kun luonnoksia on käytettävissä.

Riippumatta siitä, mikä ohjelmistokehityksen elinkaarimalli on valittu, testaustehtävien pitäisi alkaa elinkaaren alkuvaiheissa aikaisen testauksen testausperiaatteen mukaisesti.

Tässä sertifikaattisisällössä luokitellaan tyypilliset ohjelmistokehityksen elinkaarimallit seuraavasti:

- peräkkäismallit
- iteratiiviset ja inkrementaaliset kehitysmallit.

Peräkkäismallissa ohjelmistokehityksen prosessi kuvataan tehtävien lineaarisena, perättäisenä sarjana. Tämä tarkoittaa, että kehitysprosessin kunkin vaiheen pitäisi alkaa, kun sitä edeltävä vaihe on valmis. Teoriassa vaiheet eivät mene päällekkäin, mutta käytännössä on hyödyllistä saada aikaista palautetta seuraavalta vaiheelta.

Vesiputousmallissa kehitystehtävät (esim. vaatimusten analysointi, suunnittelu, koodaus, testaus) tehdään valmiiksi yksi toisensa jälkeen. Tässä mallissa testaustehtävät tehdään vasta sitten, kun kaikki muut toteutustehtävät on tehty valmiiksi.

Toisin kuin vesiputousmallissa, V-malli ottaa testausprosessin mukaan läpi koko toteutusprosessin ja toteuttaa näin aikaisen testauksen periaatteen. Tämän lisäksi V-malli koostuu kutakin toteutusvaihetta vastaavista testaustasoista, mikä omalta osaltaan tukee aikaista testausta (ks. luku 2.2 koskien testaustasoja). Tässä mallissa kuhunkin testaustasoon liittyvien testien suoritus etenee järjestyksessä peräkkäin, mutta joissain tilanteissa tapahtuu päällekkäisyyksiä.

Peräkkäismallit tuottavat ohjelmiston, joka sisältää kaikki siihen suunnitellut ominaisuudet, mutta ohjelmiston toimitus sidosryhmille ja käyttäjille vaatii tyypillisesti kuukausia tai vuosia.

Inkrementaalisessa mallissa vaatimusten määrittely ja järjestelmän suunnittelu, toteutus ja testaus tehdään osissa, mikä tarkoittaa sitä, että ohjelmiston ominaisuudet laajenevat pala kerrallaan. Näiden palasten koko vaihtelee, ja jotkut ominaisuudet koostuvat isommista palasista ja jotkut pienemmistä. Palaset voivat olla jopa niin pieniä kuin käyttöliittymän ikkunan yksittäinen muutos tai uusi kyselyn vastausvaihtoehto.

Iteratiivisessa kehityksessä joukko ominaisuuksia määritellään, suunnitellaan, toteutetaan ja testataan yhdessä useamman sovitun mittaisen kierroksen aikana. Kierroksiin voi kuulua aikaisemmissa iteraatioissa toteutettujen ominaisuuksien muutoksia sekä muutoksia projektin tehtäväkokonaisuuteen. Jokainen iteraatio tuottaa toimivan ohjelmiston, joka on kasvava osa kaikkien ominaisuuksien muodostamaa kokonaisuutta, kunnes toimitetaan viimeinen ohjelmiston osa tai toteutus lopetetaan.

Esimerkkeihin kuuluvat:

- Rational Unified Process: Jokainen iteraatio on yleensä suhteellisen pitkä (esim. kaksi tai kolme kuukautta) ja ominaisuuksien kokonaisuudet (inkrementit) ovat vastaavasti suuria, kuten kaksi tai kolme toisiinsa liittyvien ominaisuuksien ryhmää.

- Scrum: Jokainen iteraatio on yleensä suhteellisen lyhyt (esim. tunteja, päiviä tai muutama viikko) ja ominaisuuksien inkrementit ovat vastaavasti pieniä, kuten muutama muokkaus ja/tai kaksi tai kolme uutta ominaisuutta.
- Kanban: Toteutus tehdään joko sovitun mittaisissa iteraatioissa tai ilman niitä, ja ne voivat tuottaa valmistuessaan joko yhden yksittäisen muutoksen tai ominaisuuden tai niihin voidaan koota ryhmä ominaisuuksia, jotka julkaistaan yhtä aikaa.
- Spiraali (tai prototyypit): Mallissa luodaan kokeellisia inkrementtejä, joista osa ehkä toteutetaan laajastikin uudelleen tai jopa hylätään myöhemmän kehitystyön aikana.

Näiden mallien avulla toteutettujen komponenttien tai järjestelmien kehittäminen sisältää usein päällekkäisyyksiä ja testitasojen iterointia läpi kehitystyön. Ihannetilanteessa jokainen ominaisuus testataan useilla testaustasoilla, kun se etenee kohti julkaisua. Joissain tapauksissa tiimit käyttävät jatkuvaa toimittamista tai jatkuvaa julkaisua, joista kumpaankin kuuluu useiden testaustasojen huomattava automatisointi osana julkaisuputkea. Moniin näitä kehitysmalleja käyttäviin menetelmiin kuuluu myös itseorganisoituvien tiimien käsite, joka voi muuttaa tapaa, jolla testaustyö organisoidaan, samoin kuin testaajien ja toteuttajien välisiä suhteita.

Nämä menetelmät muodostavat kasvavan järjestelmän, joka voidaan julkaista loppukäyttäjille joko ominaisuus kerrallaan, iteraatio kerrallaan tai perinteisemmällä pääjulkaisujen periaatteella. Riippumatta siitä, julkaistaanko ohjelmistoinkrementit loppukäyttäjille, regressiotestauksen tärkeys kasvaa järjestelmän kasvaessa.

Peräkkäismalleihin verrattuna iteratiiviset ja inkrementaaliset mallit voivat tuottaa käyttökelpoisia ohjelmistoja viikoissa tai jopa päivissä, mutta kaikkia vaatimuksia vastaavan tuotteen ne pystyvät toimittamaan vasta kuukausien tai jopa vuosien kuluttua.

Lisätietoa ohjelmistotestauksesta Ketterässä kehityksessä, ks. ISTQB-AT Perustason sertifikaattisisällön laajennus, Ketterä testaaja, Black 2017, Crispin 2008 ja Gregory 2015.

2.1.2 Ohjelmistokehityksen elinkaarimallit eri tilanteissa

Ohjelmistokehityksen elinkaarimallit pitää valita ja muokata projektin tilanteen ja tuotteen ominaisuuksien mukaan. Sopiva ohjelmistokehityksen elinkaarimalli pitäisi valita ja sitä pitäisi muokata projektin tavoitteiden, kehitettävänä olevan tuotteen tyypin, liiketoiminnan prioriteettien (esim. markkinoillesaantiaika) sekä tunnistettujen tuote- ja projektiriskien mukaan. Esimerkiksi pienen sisäiseen käyttöön tarkoitetun hallinnollisen järjestelmän testauksen pitäisi erota turvallisuuskriittisen järjestelmän, kuten auton jarrujen hallintajärjestelmän, kehittämisestä ja testaamisesta. Toisena esimerkkinä, joissain tapauksissa organisatoriset ja kulttuurilliset seikat voivat estää viestinnän tiimin jäsenten välillä, mikä voi haitata iteratiivista kehitystä.

Projektin tilanteesta riippuen voi olla tarpeen yhdistää tai järjestää uudelleen testaustasoja ja/tai testaustehtäviä. Esimerkiksi, kun ollaan integroimassa kaupallista ohjelmistotuotetta isompaan järjestelmään, ostaja saattaa suorittaa yhteentoimivuustestauksen järjestelmäintegraatiotestaustasolla (esim. integraatio infrastruktuuriin ja muihin järjestelmiin) sekä hyväksymistestaustasolla (toiminnallinen ja ei-toiminnallinen testaus sekä käyttäjien hyväksymistestaus ja käyttöön soveltuvuuden hyväksymistestaus). Lisää tietoa testaustasoista, ks. alaluku 2.2, ja testityypeistä, ks. alaluku 2.3.

Lisäksi ohjelmistokehityksen elinkaarimalleja voidaan yhdistää keskenään. Esimerkiksi V-mallia voidaan käyttää, kun kehitetään ja testataan taustajärjestelmiä ja niiden integrointia, kun taas Ketteriä kehitysmalleja voidaan käyttää front-endin kehittämisessä sekä sen käyttöliittymien ja toiminnallisuuden testaamisessa. Prototyyppejä voidaan laatia aikaisin projekteissa niin, että siirrytään inkrementaaliseen kehitysmalliin, kun kokeiluvaihe on valmis.

Esineiden Internetissä (eng. Internet of Things, IoT), joka koostuu monista erityyppisistä kohteista, kuten laitteista, tuotteista ja palveluista, käytetään tyypillisesti eri ohjelmistokehityksen elinkaarimalleja kullekin kohteelle. Tämä tuottaa erityisen haasteen, kun kehitetään järjestelmäversioita Esineiden Internettiin. Lisäksi tällaisten kohteiden ohjelmistokehityksen elinkaari korostaa enemmän ohjelmistokehityksen elinkaaren myöhäisempiä vaiheita (esim. käyttö, päivitykset ja käytöstä poistuminen) sen jälkeen, kun kohteet on otettu tuotantokäyttöön.

2.2 *Testaustasot*

Testaustasot ovat joukko testaustehtäviä, joita organisoidaan ja hallitaan yhdessä. Jokainen testaustaso on testausprosessin ilmentymä, joka koostuu alaluvussa 1.4 kuvatuista tehtävistä, jotka suoritetaan ohjelmiston tilan mukaan sen määrätyssä kehitysvaiheessa, lähtien yksittäisistä osista tai komponenteista kokonaiseen järjestelmään, tai järjestelmien järjestelmään, mikäli tarpeellista. Testaustasot liittyvät toisiin tehtäviin ohjelmistokehityksen elinkaaressa. Tässä sertifikaattisisällössä käsiteltävät testaustasot ovat

- yksikkötestaus
- integraatiotestaus
- järjestelmätestaus
- hyväksymistestaus.

Testaustasoja määrittävät seuraavat ominaisuudet:

- testauksen tavoitteet
- pohjamateriaali, jota käytetään testitapausten laatimiseksi
- testauksen kohde (mitä ollaan testaamassa)
- tyypilliset viat ja häiriöt
- erityiset lähestymistavat ja vastuut.

Jokainen testaustaso vaatii sopivan testiympäristön. Esimerkiksi hyväksymistestauksessa ihannetilanteessa käytetään tuotannonkaltaista testiympäristöä, kun taas yksikkötestauksessa toteuttajat tyypillisesti käyttävät omaa kehitysympäristöään.

2.2.1 Yksikkötestaus

Yksikkötestauksen tavoitteet

Yksikkötestauksessa (tunnetaan myös komponenttitestauksena) keskitytään komponentteihin, jotka ovat yksinään testattavia. Yksikkötestauksen tavoitteita ovat seuraavat:

- riskien pienentäminen
- sen todentaminen, että järjestelmän toiminnallinen ja ei-toiminnallinen käyttäytyminen vastaavat suunniteltua ja määriteltyä.
- luottamuksen kasvattaminen komponentin laatuun
- vikojen löytäminen komponentista
- estetään vikojen eteneminen ylemmille testaustasoille.

Joissakin tilanteissa, erityisesti inkrementaalisissa ja iteratiivisissa elinkaarimalleissa (esim. Ketterä kehitys), joissa koodi muuttuu jatkuvasti, automatisoiduilla komponenttien regressiotesteillä on avainrooli, kun halutaan varmistaa, että muutokset eivät ole rikkoneet olemassa olevia komponentteja.

Yksikkötestaus tehdään ohjelmistokehityksen elinkaarimallista ja järjestelmästä riippuen usein erossa muusta järjestelmästä, minkä vuoksi saatetaan tarvita tynkiä, palveluiden virtualisointia, testikehyksiä, tynkiä ja ajureita. Yksikkötestaus voi kattaa toiminnallisuuksia (esim. laskutoimitusten oikeellisuus), ei-toiminnallisia ominaisuuksia (esim. muistivuotojen etsiminen) ja rakenteellisia ominaisuuksia (esim. päätöstestaus).

Testauksen pohjamateriaali

Seuraavassa on lueteltu esimerkkejä tuotoksista, joita voidaan käyttää yksikkötestauksen pohjamateriaalina:

- yksityiskohtaiset suunnitelmat
- koodi

- tietomalli
- komponenttien määrittelyt.

Testauksen kohteet

Tyypillisiin yksikkötestauksen kohteisiin kuuluvat:

- komponentit, yksiköt tai moduulit
- koodi ja tietorakenteet
- luokat
- tietokantamoduulit.

Tyypilliset viat ja häiriöt

Yksikkötestauksessa löydettäviin tyypillisiin vikoihin ja häiriöihin kuuluvat:

- väärä toiminnallisuus (esim. ei vastaa suunnittelukuvauksia)
- tietovirtaongelmat
- virheellinen koodi ja logiikka.

Viat korjataan tavallisesti heti kun ne löydetään, usein ilman muodollista havaintojen hallintaa. Jos kehittäjät kuitenkin raportoivat vikoja, se tuottaa tärkeää tietoa perussyyanalyysiä ja prosessikehitystä varten.

Erityiset lähestymistavat ja vastuut

Komponenttitestauksen suorittaa tyypillisesti kehittäjä, joka kirjoitti koodin, mutta vähintäänkin se edellyttää, että testauksen tekijä pääsee käsiksi testattavana olevaan koodiin. Kehittäjät voivat vuorotella komponenttien toteutuksen ja vikojen etsimisen ja korjaamisen välillä. Kehittäjä laativat ja kirjoittavat usein testit sen jälkeen, kun he ovat kirjoittaneet komponentin koodin. Kuitenkin erityisesti ketterässä kehityksessä automatisoitavien yksikkötestien kirjoittaminen voi tapahtua ennen sovelluksen koodin kirjoittamista.

Ajatellaan esimerkiksi testiohjattua kehitystä (eng. Test Driven Development, TDD). Testiohjattu kehitys on erittäin iteratiivista ja perustuu kierroksiin, joissa ensin suunnitellaan automatisoidut testitapaukset, sen jälkeen toteutetaan ja integroidaan pieniä osia koodia, sitten suoritetaan yksikkötestit, korjataan mahdolliset viat ja lopulta refaktoroidaan koodi. Prosessi jatkuu, kunnes komponentti on rakennettu valmiiksi ja kaikki yksikkötestit menevät hyväksytysti läpi. Testiohjattu kehitys on esimerkki testit sin -lähestymistavasta. Vaikka testiohjattu kehitys on lähtöisin eXtreme Programmingista (XP), se on levinnyt myös muihin ketterän kehityksen muotoihin sekä ohjelmistokehityksen peräkkäismalleihin (ks. ISTQB-AT Perustason sertifikaattisisällön laajennus, Ketterä Testaaja).

2.2.2 Integraatiotestaus

Integraatiotestauksen tavoitteet

Integraatiotestaus keskittyy komponenttien tai järjestelmien väliseen vuorovaikutukseen. Integraatiotestauksen tavoitteita ovat seuraavat:

- riskien pienentäminen
- sen todentaminen, että järjestelmän toiminnallinen ja ei-toiminnallinen käyttäytyminen vastaavat suunniteltua ja määriteltyä
- luottamuksen kasvattaminen rajapintojen laatuun
- vikojen löytäminen (viat voivat olla itse rajapinnoissa tai komponenteissa tai järjestelmissä)
- estetään vikojen karkaaminen ylemmille testaustasoille.

Kuten yksikkötestauksessa, joissakin tilanteissa automatisoidut integraatioregressiotestit kasvattavat luottamusta siihen, että muutokset eivät ole rikkoneet olemassa olevia rajapintoja, komponentteja tai järjestelmiä.

Tässä sertifikaattisisällössä kuvataan kaksi eri integraatiotestauksen tasoa, joita voidaan suorittaa eri kokoisille testauskohteille seuraavasti:

- Komponentti-integraatiotestaus keskittyy yhteen liitettyjen komponenttien vuorovaikutukseen ja rajapintoihin. Komponentti-integraatiotestaus tehdään yksikkötestauksen jälkeen ja se on yleensä automatisoitua. Iteratiivisessa ja inkrementaalisessa kehityksessä komponentti-integraatiotestaus on yleensä osa jatkuvan integraation prosessia.
- Järjestelmäintegraatiotestaus keskittyy järjestelmien, pakettien ja mikropalveluiden välisiin vuorovaikutuksiin ja rajapintoihin. Järjestelmäintegraatiotestaus voi myös kattaa vuorovaikutuksen ulkoisten organisaatioiden (esim. web-palvelut) kanssa sekä niiden tarjoamat rajapinnat. Tässä tapauksessa kehittäjäorganisaatio ei hallinnoi ulkoisia rajapintoja, mikä voi aiheuttaa erilaisia haasteita testaukselle (esim. sen varmistaminen, että testausta estävät viat ulkoisen organisaation koodissa selvitetään, testiympäristöjen järjestäminen jne.) Järjestelmäintegraatiotestaus voidaan tehdä järjestelmätestauksen jälkeen tai samaan aikaan käynnissä olevien järjestelmätestauksen tehtävien kanssa (sekä peräkkäismallisessa kehityksessä että iteratiivisessa ja inkrementaalisessa kehityksessä).

Testauksen pohjamateriaali

Seuraavassa on lueteltu esimerkkejä tuotoksista, joita voidaan käyttää integraatiotestauksen pohjamateriaalina:

- ohjelmiston ja järjestelmän suunnittelukuvaukset
- sekvenssidiagrammit
- rajapintojen ja viestintäprotokollien määrittelyt
- käyttötapaukset
- yksikkö- tai järjestelmätason arkkitehtuurikuvaukset
- työnkulut
- ulkoisten rajapintojen määrittelyt.

Testauksen kohteet

Tyypillisiin integraatiotestauksen kohteisiin kuuluvat:

- alijärjestelmät
- tietokannat
- infrastruktuuri
- rajapinnat
- APIt
- mikropalvelut.

Tyypilliset viat ja häiriöt

Komponentti-integraatiotestauksessa löydettäviin tyypillisiin vikoihin ja häiriöihin kuuluvat:

- väärä aineisto, puuttuva aineisto tai virheellinen aineiston muuntaminen
- virheelliset sekvenssit tai rajapintakutsujen ajoitus
- rajapintojen epäyhteensopivuus
- häiriöt komponenttien välisessä viestinnässä
- käsittelemättömät tai väärin käsitellyt viestintähäiriöt komponenttien välillä
- väärät oletukset komponenttien välillä kulkevan tiedon merkityksestä, muodosta tai rajoista.

Järjestelmäintegraatiotestauksessa löydettäviin tyypillisiin vikoihin ja häiriöihin kuuluvat:

- epäyhdenmukaiset viestirakenteet järjestelmien välillä
- väärä aineisto, puuttuva aineisto tai virheellinen aineiston muuntaminen
- rajapintojen epäyhteensopivuus
- häiriöt järjestelmien välisessä viestinnässä
- käsittelemättömät tai väärin käsitellyt viestintähäiriöt järjestelmien välillä
- väärät oletukset komponenttien välillä kulkevan tiedon merkityksestä, muodosta tai rajoista
- pakollisten tietoturvasäädösten noudattamatta jättäminen.

Erityiset lähestymistavat ja vastuut

Komponentti-integraatiotestauksen ja järjestelmäintegraatiotestauksen tulisi keskittyä itse integraatioon. Jos esimerkiksi moduuli A integroidaan moduulin B kanssa, testien pitäisi keskittyä näiden moduulien väliseen viestintään, ei yksittäisten moduulien toiminnallisuuteen, sillä se olisi pitänyt kattaa yksikkötestauksen aikana. Jos integroidaan järjestelmää X järjestelmän Y kanssa, testien pitäisi keskittyä järjestelmien väliseen viestintään, ei yksittäisten järjestelmien toiminnallisuuteen, sillä se olisi pitänyt kattaa järjestelmätestauksessa. Testauksessa voidaan käyttää toiminnallisia, ei-toiminnallisia ja rakenteellisia testityyppejä.

Komponentti-integraatiotestaus on usein kehittäjien vastuulla. Järjestelmäintegraatiotestaus on yleensä testaajien vastuulla. Ihannetilanteessa järjestelmäintegraatiotestausta suorittavien testaajien pitäisi ymmärtää järjestelmän arkkitehtuuri ja heidän olisi pitänyt vaikuttaa integraation suunnitteluun.

Jos integrointitestit ja integraatiostrategia suunnitellaan ennen kuin komponentteja tai järjestelmiä rakennetaan, ne voidaan rakentaa järjestyksessä, joka tarvitaan testauksen suorittamiseksi kaikkein tehokkaimmin. Järjestelmälliset integrointistrategiat voivat perustua järjestelmäarkkitehtuuriin (kuten ylhäältä-alas tai alhaalta-ylös), toiminnallisiin tehtäviin, tapahtumien prosessointijärjestykseen tai johonkin muuhun järjestelmän tai komponentin piirteeseen. Vikojen eristämisen ja aikaisen löytämisen yksinkertaistamiseksi integraation pitäisi yleensä tapahtua askelittain (eli pieni määrä uusia komponentteja tai järjestelmiä kerrallaan) ennemmin kuin ”kertarysäyksellä” (eli integroidaan kaikki komponentit tai järjestelmät yhdellä kertaa). Kaikkein monimutkaisimpien rajapintojen riskianalyysi voi auttaa integraatiotestauksen kohdentamisessa.

Mitä suurempi integraatiokokonaisuus on, sitä vaikeammaksi tulee vikojen eristäminen tiettyyn komponenttiin tai järjestelmään, mikä voi johtaa lisääntyneeseen riskiin ja ongelmanselvityksen ajantarpeeseen. Tämä on yksi syy siihen, että jatkuvasta integraatiosta, joka tarkoittaa ohjelmiston integrointia komponentti kerrallaan (esim. toiminnallinen integraatio), on tullut yleinen käytäntö. Jatkuvaan integraatioon kuuluu usein automatisoitu regressiotestaus, ihannetilanteessa useilla testaustasoilla.

2.2.3 Järjestelmätestaus

Järjestelmätestauksen tavoitteet

Järjestelmätestaus keskittyy koko järjestelmän tai tuotteen käyttäytymiseen ja ominaisuuksiin, ja siinä tutkitaan usein järjestelmän suorittamia kokonaisia toimintoketjuja päästä päähän sekä ei-toiminnallisia käyttäytymisiä, jota järjestelmä osoittaa, kun se suorittaa kyseisiä toimintoketjuja. Järjestelmätestauksen tavoitteita ovat seuraavat:

- riskien pienentäminen
- sen todentaminen, että järjestelmän toiminnallinen ja ei-toiminnallinen käyttäytyminen vastaavat suunniteltua ja määriteltyä
- sen varmistaminen, että järjestelmä on valmis ja toimii odotetusti
- luottamuksen kasvattaminen koko järjestelmän laatuun
- vikojen löytäminen
- vikojen ylemmille testaustasoille tai tuotantokäyttöön etenemisen estäminen.

Tavoitteena voi tiettyjen järjestelmien kohdalla olla myös aineiston laadun todentaminen. Kuten yksikkötestauksessa ja integraatiotestauksessa, joissain tapauksissa automatisoidut järjestelmäregressiotestit kasvattavat luottamusta siihen, että muutokset eivät ole rikkoneet olemassa olevia ominaisuuksia tai päästä-päähän kulkevia toimintoketjuja. Järjestelmätestaus tuottaa usein tietoa, jota sidosryhmät käyttävät julkaisupäätöksen tekemisessä. Järjestelmätestaus voi myös varmistaa oikeudellisten tai muihin sääntelyihin liittyvien vaatimusten tai standardien mukaisuuden täyttymisen.

Testiympäristön pitäisi ihannetilanteessa vastata lopullista kohde- tai tuotantoympäristöä.

Testauksen pohjamateriaali

Seuraavassa on lueteltu esimerkkejä tuotoksista, joita voidaan käyttää järjestelmätestauksen pohjamateriaalina:

- järjestelmän ja ohjelmiston vaatimusmäärittelyt (toiminnalliset ja ei-toiminnalliset)
- riskianalyysiraportit
- käyttötapaukset
- eepokset ja käyttäjätarinat
- järjestelmän käyttäytymismallit
- tilakaaviot
- järjestelmä- ja käyttöohjeet.

Testauksen kohteet

Tyypillisiin järjestelmätestauksen kohteisiin kuuluvat:

- sovellukset
- laitteisto-/ohjelmistojärjestelmät
- käyttöjärjestelmät
- testattava järjestelmä
- järjestelmän kokoonpano ja kokoonpanoon liittyvä aineisto.

Tyypilliset viat ja häiriöt

Järjestelmätestauksessa löydettäviin tyypillisiin vikoihin ja häiriöihin kuuluvat:

- virheelliset laskutoimitukset
- virheellinen tai odottamaton järjestelmän toiminnallinen tai ei-toiminnallinen käyttäytyminen

- järjestelmän virheelliset kontrolli- ja/tai tietovuot
- häiriöt päästä päähän kulkevien toiminnallisten tehtävien oikeanlaisessa tai täydellisessä suorittamisessa
- häiriöt järjestelmän oikeanlaisessa toiminnassa tuotantoympäristö(i)ssä
- häiriöt järjestelmän järjestelmä- ja käyttöohjeiden mukaisessa käytössä.

Erityiset lähestymistavat ja vastuut

Järjestelmätestauksen pitäisi keskittyä koko järjestelmän päästä päähän tapahtuvaan sekä toiminnalliseen että ei-toiminnalliseen käyttäytymiseen. Järjestelmätestauksessa pitäisi käyttää testattavan järjestelmän eri piirteiden testaukseen parhaiten soveltuvia tekniikoita (ks. luku 4). Esimerkiksi päätöstauluja voidaan käyttää sen todentamiseksi, että järjestelmän toiminnallinen käyttäytyminen vastaa liiketoimintasäännöissä kuvattua toimintaa.

Järjestelmätestauksen suorittavat tyypillisesti riippumattomat testaajat. Viat määrittelyissä (esim. puuttuvat käyttäjätarinat, väärin kuvatut liiketoimintavaatimukset jne.) voivat johtaa odotettuun järjestelmän käyttäytymiseen liittyviin väärinymmärryksiin tai erimielisyyksiin. Tällaiset tilanteet voivat johtaa vääriin positiivisiin, jotka tuhlaavat aikaa, ja vääriin negatiivisiin, jotka vähentävät vikojen löytämisen tehokkuutta. Testaajien aikainen osallistuminen käyttäjätarinoiden tarkentamiseen tai staattisen testauksen tehtäviin, kuten katselmointeihin, auttavat vähentämään tällaisten tilanteiden syntymistä.

2.2.4 Hyväksymistestaus

Hyväksymistestauksen tavoitteet

Kuten järjestelmätestaus, myös hyväksymistestaus keskittyy tyypillisesti koko järjestelmän tai tuotteen käyttäytymiseen ja ominaisuuksiin. Hyväksymistestauksen tavoitteita ovat seuraavat:

- luottamuksen varmistaminen koko järjestelmän laatuun
- järjestelmän kelpuutus eli varmistetaan, että järjestelmä on valmis ja toimii odotetusti
- sen todentaminen, että järjestelmän toiminnallinen ja ei-toiminnallinen käyttäytyminen vastaavat määriteltyä.

Hyväksymistestaus voi tuottaa tietoa, jonka avulla voidaan arvioida, onko järjestelmä valmis julkaistavaksi ja asiakkaan (loppukäyttäjän) käytettäväksi. Hyväksymistestauksen aikana voidaan löytää vikoja, mutta vikojen löytäminen ei useinkaan ole tavoitteena, ja jos hyväksymistestauksessa löydetään suuri määrä vikoja, sitä voidaan joissakin tapauksissa pitää isona projektiriskinä. Hyväksymistestaus voi myös varmistaa oikeudellisten tai muihin sääntelyihin liittyvien vaatimusten tai standardien mukaisuuden täyttymisen.

Yleisiä hyväksymistestauksen muotoja ovat seuraavat:

- käyttäjän hyväksymistestaus
- käyttöön soveltuvuuden hyväksymistestaus
- sopimusten ja sääntelyiden perusteella tehtävä hyväksymistestaus
- alfa- ja betatestaus.

Jokainen näistä kuvataan seuraavassa tarkemmin.

Käyttäjän hyväksymistestaus

Käyttäjien tekemä järjestelmän hyväksymistestaus keskittyy tyypillisesti sen todentamiseen, sopiiko järjestelmä tarkoitettujen käyttäjien käytettäväksi oikeassa tai simuloidussa käyttöympäristössä. Päätavoitteena on kasvattaa luottamusta siihen, että käyttäjät voivat käyttää järjestelmää ja se vastaa heidän tarpeitaan, täyttää heidän vaatimuksensa, ja liiketoimintaprosessit voidaan suorittaa pienimmällä mahdollisella vaivalla, kustannuksilla ja riskeillä.

Käyttöön soveltuvuuden hyväksymistestaus

Toiminnan- tai järjestelmävalvojien suorittama hyväksymistestaus tehdään yleensä (simuloidussa) tuotantoympäristössä. Testit keskittyvät toiminnallisiin näkökulmiin ja niihin voi kuulua seuraavia:

- varmuuskopioinnin ja palautuksen testaus
- asentaminen, asennuksen poistaminen ja päivitys
- virhetilanteesta toipuminen
- käyttäjien hallinta
- ylläpitotehtävät
- aineiston luomis- ja migraatiotehtävät
- tietoturvahaavoittuvuuksien tarkistukset
- suorituskykytestaus.

Käyttöön soveltuvuuden hyväksymistestauksen päätarkoitus on kasvattaa luottamusta siihen, että toiminnan- ja järjestelmävalvojat voivat pitää järjestelmän toiminnassa käyttäjiä varten tuotantoympäristössä jopa poikkeuksellisissa tai vaikeissa olosuhteissa.

Sopimusten ja sääntelyiden perusteella tehtävä hyväksymistestaus

Sopimuspohjainen hyväksymistestaus tehdään sopimuksen hyväksymiskriteereitä vastaan, kun tehdään tilaustyönä kehitettyä ohjelmistoa. Hyväksymiskriteerit pitäisi määritellä, kun sopimus tehdään. Käyttäjät tai riippumattomat testaajat suorittavat yleensä sopimusten perusteella tehtävän hyväksymistestauksen.

Sääntelyiden perusteella tehtävää hyväksymistestausta tehdään kaikkia sellaisia sääntelyitä vastaan, joita pitää noudattaa, kuten valtiovallan säätämät tai lakeihin tai turvallisuussäännöksiin pohjautuvat määräykset. Sääntelyihin pohjautuvan hyväksymistestauksen suorittavat usein käyttäjät tai riippumattomat testaajat, ja joskus sääntelyitä valvovat viranomaiset seuraavat testejä tai auditoivat niiden tulokset.

Sopimuksiin ja sääntelyihin pohjautuvan hyväksymistestauksen päätavoitteena on kasvattaa luottamusta siihen, että toteutus on tehty sopimusten tai sääntelyiden kanssa yhdenmukaisesti.

Alfa- ja betatestaus

Kaupallisten ohjelmistojen kehittäjät suorittavat tyypillisesti alfa- ja betatestausta halutessaan palautetta potentiaalisilta tai nykyisiltä käyttäjiltä, asiakkailta ja/tai operaattoreilta ennen kuin ohjelmistotuote lanseerataan markkinoille. Alfatestaus tehdään toteuttavan organisaation tiloissa, mutta sitä ei tee toteutustiimi vaan potentiaaliset tai nykyiset asiakkaat ja/tai operaattorit tai riippumaton testaustiimi. Betatestauksen suorittavat potentiaaliset tai nykyiset asiakkaat ja/tai operaattorit omissa tiloissaan. Betatestaus voi seurata alfatestausta tai se voidaan tehdä ilman sitä edeltävää alfatestausta.

Yksi alfa- ja betatestauksen tavoite on kasvattaa potentiaalisten tai nykyisen asiakkaiden ja/tai operaattoreiden luottamusta siihen, että he voivat käyttää järjestelmää normaaleissa, jokapäiväisissä olosuhteissa ja käyttöympäristössä saavuttaakseen tavoitteensa niin, että siihen liittyy mahdollisimman vähän vaikeuksia, kustannuksia ja riskejä. Toinen tavoite voi olla järjestelmän käyttötilanteisiin ja -ympäristöön liittyvien vikojen löytäminen, erityisesti silloin, kun toteutustiimin on vaikea toistaa näitä tilanteita ja ympäristöjä.

Testauksen pohjamateriaali

Seuraavassa on lueteltu esimerkkejä tuotoksista, joita voidaan käyttää hyväksymistestauksen kaikkien muotojen pohjamateriaalina:

- liiketoimintaprosessit
- käyttäjän tai liiketoiminnan vaatimukset
- säädökset, oikeudelliset vaatimukset tai standardit
- käyttötapaukset
- järjestelmävaatimukset
- järjestelmä- tai käyttäjädokumentaatio
- asennusproseduurit
- riskianalyysiraportit.

Lisäksi käyttöön soveltuvuuden hyväksymistestauksen testitapausten pohjamateriaalina voidaan käyttää yhtä tai useampia seuraavista tuotoksista:

- varmuuskopiointi- ja palautusproseduurit
- virhetilanteesta toipumisproseduurit
- ei-toiminnalliset vaatimukset
- operointiin liittyvä dokumentaatio
- julkaisu- ja asennusohjeet
- suorituskykytavoitteet
- tietokantapaketit
- tietoturvastandardit tai säädökset.

Tyypilliset testauksen kohteet

Hyväksymistestauksen kaikkien eri muotojen tyypillisiin kohteisiin kuuluvat seuraavat:

- testattava järjestelmä
- järjestelmän kokoonpano ja kokoonpanoon liittyvä aineisto
- liiketoimintaprosessit kokonaan integroidussa järjestelmässä
- toipumisjärjestelmät ja ulkopuoliset varajärjestelmät (liiketoiminnan jatkumisen ja häiriöistä toipumisen testaamiseksi)
- toiminnalliset ja ylläpitoprosessit
- lomakkeet
- raportit
- olemassa oleva ja konvertoitu tuotantoaineisto.

Tyypilliset viat ja häiriöt

Hyväksymistestauksen kaikissa eri muodoissa tyypillisesti löydettäviin vikoihin kuuluvat seuraavat:

- Järjestelmän työnkulut eivät vastaa liiketoiminnan tai käyttäjien vaatimuksia.
- Liiketoimintasääntöjä ei ole toteutettu oikein.
- Järjestelmä ei täytä sopimuksellisia tai säädöksellisiä vaatimuksia.
- Ei-toiminnalliset häiriöt, kuten tietoturvahaavoittuvuudet, riittämätön suoritustehokkuus suuren kuormituksen aikana tai vääränlainen toiminta tuetulla alustalla.

Erityiset lähestymistavat ja vastuut

Hyväksymistestaus on usein asiakkaiden, liiketoiminnan käyttäjien, tuoteomistajien tai järjestelmän operaattoreiden vastuulla ja muitakin sidosryhmiä voi olla mukana.

Peräkkäiselinkaarimalleissa hyväksymistestausta pidetään usein viimeisenä testaustasona, mutta sitä voidaan tehdä myös muulloin, esimerkiksi:

- Kaupallisen järjestelmän hyväksymistestaus voidaan tehdä, kun järjestelmää ollaan asentamassa tai integroimassa.
- Uuden toiminnallisuuden lisäyksen hyväksymistestaus voi tapahtua ennen järjestelmätestausta.

Iteratiivisessa kehityksessä projektitiimi voi käyttää hyväksymistestauksen eri muotoja jokaisen iteraation aikana ja lopussa, ja niihin voivat kuulua esimerkiksi uuden ominaisuuden todentaminen sen hyväksymiskriteereitä vastaan tai sen varmistaminen, että uusi ominaisuus vastaa käyttäjän tarpeita. Lisäksi alfa- ja betatestausta voidaan tehdä joko jokaisen iteraation lopussa, jokaisen iteraation valmistumisen jälkeen tai useamman iteraation joukon jälkeen. Voidaan myös tehdä käyttäjän hyväksymistestejä, käyttöön so-

veltuvuuden hyväksymistestejä sekä säädöksiin ja sopimuksiin pohjautuvia hyväksymistestejä, joko joka iteraation lopussa, jokaisen iteraation valmistumisen jälkeen tai useamman iteraation joukon jälkeen.

2.3 *Testaustyypit*

Testaustyyppi tarkoittaa joukkoa testaustehtäviä, jotka on tarkoitettu tietyn ohjelmiston tai ohjelmiston osan määrättyjen ominaisuuksien testaamiseen määrättyjen testauksen tavoitteiden perusteella. Näihin tavoitteisiin voivat kuulua:

- toiminnallisten laatuominaisuuksien arviointi, kuten valmius, oikeellisuus ja soveltuvuus
- ei-toiminnallisten laatuominaisuuksien arviointi, kuten luotettavuus, suorituskyvyn tehokkuus, tietoturva, yhteensopivuus ja käytettävyys
- sen arvioiminen, onko komponentin tai järjestelmän rakenne tai arkkitehtuuri oikein, täydellinen ja vastaako se määriteltyä
- muutosten vaikutuksen arviointi, kuten sen varmistaminen, että viat on korjattu (varmistustestaus) ja ohjelmiston käyttäytymisessä esiintyvien ohjelmiston tai ympäristön muutoksista johtuvien ei-tarkoitettujen muutosten etsiminen (regressiotestaus).

2.3.1 Toiminnallinen testaus

Järjestelmän toiminnallinen testaus käsittää testejä, jotka arvioivat toimintoja, jotka järjestelmän pitäisi suorittaa. Toiminnalliset vaatimukset voidaan kuvat tuotoksissa, kuten liiketoimintavaatimusten määrittelyissä, eepoksissa, käyttäjätarinoissa, käyttötapauksissa tai toiminnallisissa määrityksissä, tai ne voivat olla dokumentoimattomia. Toiminnot ovat se, "mitä" järjestelmän pitäisi tehdä.

Toiminnallisia testejä pitäisi tehdä kaikilla testaustasoilla (esim. komponenttien testauksen pitäisi perustua komponenttien määrityksiin), vaikka painopiste on eri joka testaustasolla (ks. alaluku 2.2).

Toiminnallinen testaus tutkii järjestelmän käyttäytymistä, joten mustalaatikkotekniikoita voidaan käyttää testattavien tilanteiden ja testitapauksien johtamisessa komponentin tai järjestelmän toiminnallisuuden testaamiseksi (ks. alaluku 4.2).

Toiminnallisen testauksen perusteellisuutta voidaan mitata toiminnallisen kattavuuden avulla. Toiminnallinen kattavuus tarkoittaa sitä, kuinka perusteellisesti testit ovat käyneet läpi jonkin tyyppisen toiminnallisen osan, ja se ilmaistaan katettujen osien prosenttiosuutena. Esimerkiksi käyttämällä testien ja toiminnallisten vaatimusten välistä jäljitettävyyttä voidaan laskea testien kattamien vaatimusten prosenttiosuus ja näin tunnistaa mahdollisia kattavuusaukkoja.

Toiminnallisten testien suunnittelu ja suoritus voivat edellyttää erityisiä taitoja tai osaamista, kuten sellaisten liiketoimintaan liittyvien ongelmien tuntemusta, joita ohjelmiston on tarkoitus ratkaista (esim. öljy- ja kaasuteollisuuden geologiset mallinnusohjelmistot) tai tietoa rooleista, joita ohjelmiston on tarkoitus palvella (esim. peliohjelmistot, jotka tarjoavat interaktiivista viihdettä).

2.3.2 Ei-toiminnallinen testaus

Ei-toiminnallinen testaus arvioi järjestelmien ja ohjelmistojen ominaisuuksia kuten käytettävyyttä, suorituskyvyn tehokkuutta tai tietoturvaa. Ohjelmistotuotteiden laatuominaisuuksien luokittelu on esitelty ISO-standardissa ISO/IED 25010. Ei-toiminnallinen testaus testaa sitä, “kuinka hyvin” järjestelmä käyttäytyy.

Vastoin yleistä väärinkäsitystä ei-toiminnallista testausta voidaan ja usein pitäisi tehdä kaikilla testaustasoilla ja niin aikaisin kuin mahdollista. Ei-toiminnallisten vikojen myöhäinen löytäminen voi olla äärimmäisen vaarallista projektin onnistumisen kannalta.

Mustalaatikkotekniikoita (ks. alaluku 4.2) voidaan käyttää ei-toiminnallisen testauksen testattavien tilanteiden ja testitapauksen johtamisessa. Esimerkiksi raja-arvoanalyysiä voidaan käyttää suorituskykytestien kuormitusehtojen määrittämiseen.

Ei-toiminnallisen testauksen perusteellisuutta voidaan mitata ei-toiminnallisen kattavuuden avulla. Ei-toiminnallinen kattavuus tarkoittaa sitä, kuinka perusteellisesti testit ovat käyneet läpi jonkin tyyppisen ei-toiminnallisen osan, ja se ilmaistaan katettujen osien prosenttiosuutena. Esimerkiksi käyttämällä jäljitettä-

vyyttä testien ja mobiilisovellusta käyttävien tuettujen laitteiden välillä voidaan laskea yhteentoimivuustesteillä katettujen laitteiden prosenttiosuus ja tunnistaa näin mahdollisia kattavuusaukkoja.

Ei-toiminnallisten testien suunnittelu ja suoritus voivat vaatia erityisiä taitoja tai osaamista, kuten tietämystä suunnitelmien tai teknologian luontaisista heikkouksista (esim. tietoturvahaavoittuvuudet, jotka liittyvät tiettyihin ohjelmointikieliin) tai määrätyistä käyttäjäkunnista (esim. terveydenhoitolaitosten hallintajärjestelmien käyttäjäroolit).

Lisätietoja ei-toiminnallisten laatuominaisuuksien testaamisesta: ISTQB-ATA Jatkotason sertifikaattisisältö, Testausasiantuntija; ISTQB-ATTA jatkotason sertifikaattisisältö, Tekninen testausasiantuntija; ISTQB-SEC Jatkotason sertifikaattisisältö, Tietoturvatestaaja; sekä muut ISTQB:n asiantuntijamoduulit.

2.3.3 Lasilaatikkotestaus

Lasilaatikkotestauksessa testitapaukset johdetaan järjestelmän sisäisen rakenteen tai toteutuksen pohjalta. Sisäiseen rakenteeseen voivat kuulua koodi, työnkulut ja/tai tietovirrat järjestelmän sisällä (ks. alaluku 4.3).

Lasilaatikkotestauksen perusteellisuutta voidaan mitata rakenteellisen kattavuuden avulla. Rakenteellinen kattavuus tarkoittaa sitä, kuinka perusteellisesti testit ovat käyneet läpi jonkin tyyppisen rakenteellisen osan, ja se ilmaistaan katettujen osien prosenttiosuutena.

Yksikkötestaustasolla koodikattavuus perustuu testattujen komponenttien sisältämän koodin prosenttiosuuteen ja sitä voidaan mitata koodin eri osien perusteella (kattavuusosat) kuten komponentin testattujen suoritettavien lauseiden prosenttiosuus tai testattujen päätösvaihtoehtojen prosenttiosuus. Tämäntyyppisiä kattavuuksia kutsutaan yhteisellä nimellä koodikattavuus. Komponentti-integraatiotestaustasolla lasilaatikkotestaus voi perustua järjestelmän arkkitehtuuriin, kuten komponenttien välisiin rajapintoihin, ja kattavuutta voidaan mitata testeissä katettujen rajapintojen prosenttiosuutena.

Lasilaatikkotestien suunnittelu ja suoritus voivat vaatia erityisiä taitoja tai osaamista, kuten tietoa siitä, kuinka koodi on rakennettu (esim. koodikattavuustyökalujen käyttämiseksi), kuinka tietoaineisto on varastoitu (esim. mahdollisten tietokantakyselyiden arvioimiseksi) ja kuinka kattavuustyökaluja käytetään ja miten niiden tuloksia tulkitaan oikein.

2.3.4 Muutokseen pohjautuva testaus

Kun järjestelmään tehdään muutoksia, joko vian korjaamiseksi tai uuden tai muuttuneen toiminnallisuuden vuoksi, järjestelmä pitäisi testata sen varmistamiseksi, että muutokset ovat korjanneet vian tai toteuttaneet toiminnallisuuden oikein eivätkä muutokset ole aiheuttaneet ennakoimattomia haitallisia seurauksia.

- Varmistustestaus**:** Kun vika on korjattu, ohjelmiston testauksessa voidaan käyttää kaikkia niitä testitapauksia, jotka eivät menneet läpi vian vuoksi, ja nämä testitapaukset pitäisi suorittaa uudelleen uudella ohjelmistoversiolla. Ohjelmisto voidaan testata myös uusilla testeillä, jos esimerkiksi vian syy oli puuttuva toiminnallisuus. Vähintäänkin vian aiheuttaman häiriön toistamiseksi tarvittavat askeleet pitäisi suorittaa uudelleen uudella ohjelmistoversiolla. Varmistustestauksen tarkoitus on varmistaa, että alkuperäinen vika on korjattu onnistuneesti.
- Regressiotestaus: On mahdollista, että yhteen paikkaan koodissa tehty muutos, olipa se sitten korjaus tai muuntyyppinen muutos, voi vahingossa vaikuttaa muiden koodin osien käyttäytymiseen, olivatpa ne sitten samassa komponentissa, saman järjestelmän eri komponenteissa tai jopa eri järjestelmissä. Muutoksiin voivat kuulua myös muutokset ympäristössä, kuten käyttöjärjestelmän tai tietokannan hallintajärjestelmän uusi versio. Tällaisia vahingossa aiheutettuja sivuvaikutuksia kutsutaan taantumiseksi (regressioksi). Regressiotestauksessa testit suoritetaan tällaisten tahattomien sivuvaikutusten löytämiseksi.

Varmistustestausta ja regressiotestausta tehdään kaikilla testaustasoilla.

Erityisesti iteratiivisissa ja inkrementaalisissa ohjelmistokehityksen elinkaarimalleissa (esim. Ketterä kehitys) uudet ominaisuudet, olemassa olevien ominaisuuksien muutokset ja koodin refaktorointi johtavat koodin jatkuviin muutoksiin, minkä vuoksi tarvitaan myös muutokseen pohjautuvaa testausta. Järjestelmän kehittyvän luonteen vuoksi varmistustestaus ja regressiotestaus ovat hyvin tärkeitä. Tämä on erityi-

sen oleellista, kun puhutaan Esineiden Internettiin liittyvistä järjestelmistä, joissa yksittäisiä esineitä (esim. laitteet) päivitetään tai korvataan jatkuvasti toisilla.

Regressiotestijoukkoja ajetaan monta kertaa ja ne yleisesti kehittyvät hitaasti, joten regressiotestaus on vahva ehdokas automatisoitavaksi. Näiden testien automatisointi pitäisi aloittaa aikaisin projektissa (ks. luku 6).

2.3.5 Testaustyypit ja testaustasot

Edellä mainittuja testaustyyppejä voidaan käyttää kaikilla testaustasoilla. Tämän havainnollistamiseksi seuraavassa annetaan esimerkkejä pankkisovelluksen toiminnallisista ja ei-toiminnallisista testeistä, lasilaatikkotesteistä sekä muutokseen pohjautuvista testeistä, alkaen toiminnallisista testeistä:

- Yksikkötestauksessa testit suunnitellaan sen perusteella, kuinka komponentin pitäisi laskea korkoa korolle.
- Komponentti-integraatiotestauksessa testit suunnitellaan sen perusteella, kuinka käyttöliittymän kautta tallennetut tilitiedot välittyvät liiketoimintalogiikalle.
- Järjestelmätestauksessa testit suunnitellaan sen pohjalta, kuinka tilinhaltijat voivat hakea luottorajaa shekkitililleen.
- Järjestelmäintegraatiotestauksessa testit suunnitellaan sen pohjalta, kuinka järjestelmä käyttää ulkoista mikropalvelua tilinhaltijan luottokelpoisuuden tarkistamiseksi.
- Hyväksymistestauksessa testit suunnitellaan sen pohjalta, kuinka pankinjohtaja hoitaa luottohakemuksen hyväksymisen tai hylkäämisen.

Seuraavat ovat esimerkkejä ei-toiminnallisista testeistä:

- Yksikkötestauksessa suorituskykytestit suunnitellaan sen arvioimiseksi, montako CPU-kierrosta tarvitaan monimutkaisen kokonaiskoron laskentaan.
- Komponentti-integraatiotestauksessa tietoturvatestit suunnitellaan käyttöliittymästä liiketoimintalogiikalle siirtyvän tiedon aiheuttamiin puskurin ylivuotoihin liittyvien haavoittuvuuksien löytämiseksi.
- Järjestelmätestauksessa siirrettävyystestit suunnitellaan sen varmistamiseksi, että esityskerros toimii kaikilla tuetuilla selaimilla ja mobiililaitteilla.
- Järjestelmäintegraatiotestauksessa luotettavuustestit suunnitellaan järjestelmän sietokyvyn testaamiseksi tilanteessa, jossa luottokelpoisuusmikropalvelu ei vastaa.
- Hyväksymistestauksessa käytettävyystestit suunnitellaan sen testaamiseksi, kuinka hyvin pankinjohtajan luottojenkäsittelyliittymä on vammaisten henkilöiden saavutettavissa (käytettävissä).

Seuraavat ovat esimerkkejä lasilaatikkotesteistä:

- Yksikkötestauksessa testit suunnitellaan täydellisen lause- ja päätöskattavuuden saavuttamiseksi (ks. alaluku 4.3) sellaisten komponenttien osalta, jotka suorittavat rahaan liittyviä laskutoimituksia.
- Komponentti-integraatiotestauksessa testit suunnitellaan testaamaan, kuinka tieto siirtyy selainkäyttöliittymän jokaiselta näytöltä seuraavalle näytölle ja liiketoimintalogiikkaan.
- Järjestelmätestauksessa testit suunnitellaan kattamaan web-sivujen sarjoja, joita voi esiintyä luottorajan hakemuksen yhteydessä.
- Järjestelmäintegraatiotestauksessa testit suunnitellaan kattamaan kaikki mahdolliset kyselytyypit, joita järjestelmästä lähetetään luottokelpoisuusmikropalvelulle.
- Hyväksymistestauksessa testit suunnitellaan kattamaan kaikki pankkien välisissä siirroissa taloudellista tietoa sisältävien tiedostojen tuetut rakenteet ja arvoalueet.

Lopuksi seuraavassa on esimerkkejä muutokseen pohjautuvista testeistä:

- Yksikkötestauksessa jokaista komponenttia varten rakennetaan automatisoidut regressiotestit ja ne sisällytetään jatkuvan integraation kehykseen.
- Komponentti-integraatiotestauksessa testit suunnitellaan rajapintoihin liittyvien vikojen korjausten varmistamiseksi, kun korjaukset kirjataan sisään koodikirjastoon.
- Järjestelmätestauksessa kaikki tietyn työnkulun testit suoritetaan uudelleen, jos jokin työnkulkuun kuuluvista näytöistä muuttuu.
- Järjestelmäintegraatiotestauksessa luottokelpoisuusmikropalvelun kanssa vuorovaikutuksessa olevan sovelluksen testit suoritetaan uudelleen päivittäin osana kyseisen mikropalvelun jatkuvaa julkaisua.
- Hyväksymistestauksessa kaikki aikaisemmin hylkääntyneet testit suoritetaan uudelleen sen jälkeen, kun hyväksymistestauksessa löydetty vika on korjattu.

Vaikka tässä alaluvussa annetaan esimerkkejä jokaisesta testityypistä jokaisella testitasolla, ei kaikkien ohjelmistojen osalta ole välttämätöntä, että jokainen testityyppi on edustettuna joka testitasolla. On kuitenkin tärkeää suorittaa soveltuvat testit jokaisella testitasolla, erityisesti aikaisimmalla tasolla, jolla testityyppi esiintyy.

2.4 *Ylläpitotestaus*

Kun ohjelmisto ja järjestelmät on julkaistu tuotantoympäristöön, niitä täytyy ylläpitää. Erilaiset muutokset toimitetuissa ohjelmistoissa ja järjestelmissä ovat miltei väistämättömiä, joko tuotantokäytössä löytyneiden vikojen korjaamiseksi, uuden toiminnallisuuden lisäämiseksi tai jo toimitetun toiminnallisuuden poistamiseksi tai muuttamiseksi. Ylläpitoa tarvitaan myös komponentin tai järjestelmän ei-toiminnallisten laatuominaisuuksien säilyttämiseksi tai parantamiseksi sen elinaikana, erityisesti liittyen suorituskykyyn, yhteensopivuuteen, luotettavuuteen, tietoturvaan ja siirrettävyyteen.

Kun muutoksia tehdään osana ylläpitoa, ylläpitotestausta pitäisi tehdä sekä sen arvioimiseksi, kuinka onnistuneesti muutokset tehtiin, että mahdollisten sivuvaikutusten (taantumien) löytämiseksi sellaisista järjestelmän osista, joita ei ole muutettu (mikä yleensä tarkoittaa suurinta osaa järjestelmästä). Ylläpitotestaus keskittyy järjestelmän muutosten testaukseen samoin kuin sellaisten muuttamattomien alueiden testaukseen, joihin muutokset olisivat voineet vaikuttaa. Ylläpitoon voi kuulua suunniteltuja julkaisuja sekä ei-suunniteltuja julkaisuja (pikakorjauksia).

Ylläpitojulkaisu voi testauksen laajuudesta riippuen vaatia ylläpitotestausta useilla testaustasoilla sekä useiden testaustyyppien käyttöä. Ylläpitotestauksen laajuus riippuu

- muutokseen liittyvästä riskistä, esimerkiksi kuinka laajasti ohjelmiston muutettu alue on vuorovaikutuksessa muiden komponenttien tai järjestelmien kanssa
- olemassa olevan järjestelmän koosta
- muutoksen koosta.

2.4.1 Ylläpitotestauksen aiheuttajat

Ohjelmiston ylläpidolle, ja siksi myös ylläpitotestaukselle, on monia syitä, ja se koskee sekä suunniteltuja että suunnittelemattomia muutoksia.

Voimme luokitella ylläpidon syyt seuraavasti:

- muutokset, kuten suunnitellut parannukset (esim. julkaisupohjaiset), korjaavat muutokset ja hätäkorjaukset, käyttöympäristöön liittyvät muutokset (kuten suunnitellut käyttöjärjestelmän tai tietokannan päivitykset), kaupallisen ohjelmiston päivitykset, sekä vikojen ja haavoittuvuuksien korjaukset
- migraatio, kuten esimerkiksi vaihto alustalta toiselle, mikä voi vaatia sekä uuden ympäristön että muutetun ohjelmiston toiminnallisia testejä, tai tietokonversion testaus, kun toisesta sovelluksesta peräisin oleva tieto sulautetaan ylläpidettävään järjestelmään
- eläköityminen, kun järjestelmä tulee elinkaarensa päähän.

Kun sovellus tai järjestelmä poistuu käytöstä, voidaan tarvita tietojen migraation tai arkistoinnin testausta, mikäli tietojen osalta vaaditaan pitkää säilytysaikaa. Myös tietojen palautusproseduurien testausta pitkän tietojen säilytysajan jälkeen voidaan tarvita. Lisäksi regressiotestausta voidaan tarvita sen varmistamiseksi, että edelleen käytössä oleva toiminnallisuus toimii yhä oikein.

Esineiden Internet –järjestelmien osalta ylläpitotestauksen tarpeen voivat aiheuttaa järjestelmään liitetyt täysin uudet tai muokatut esineet, kuten laitteistot ja ohjelmistopalvelut. Tällaisten järjestelmien ylläpitotestaus korostaa erityisesti eri tasoilla (esim. verkkotaso, sovellustaso) tehtävää integraatiotestausta sekä tietoturvanäkökulmia, erityisesti silloin, kun käsitellään henkilötietoja.

2.4.2 Ylläpidon vaikutusanalyysi

Vaikutusanalyysi arvioi ylläpitojulkaisussa tehtyjä muutoksia muutoksen tarkoitettujen seuraamusten ja sen aiheuttamien odotettujen ja mahdollisten sivuvaikutusten tunnistamiseksi sekä sellaisten järjestelmän alueiden tunnistamiseksi, joihin muutos vaikuttaa. Vaikutusanalyysi voi auttaa myös tunnistamaan, miten muutos vaikuttaa olemassa oleviin testeihin. Sivuvaikutukset ja järjestelmän alueet, joihin muutos vaikuttaa, täytyy testata regression varalta, mahdollisesti sen jälkeen, kun testit, joihin muutos on voinut vaikuttaa, on päivitetty.

Vaikutusanalyysi voidaan tehdä ennen muutosta, jolloin tieto muutoksen aiheuttamista mahdollisista seuraamuksista järjestelmän muilla alueilla auttaa tekemään päätöksen siitä, pitäisikö muutosta tehdä.

Vaikutusanalyysi voi olla vaikeaa, jos:

- määrittelyt (esim. liiketoimintavaatimukset, käyttäjätarinat, arkkitehtuuri) ovat vanhentuneita tai niitä ei ole
- testitapauksia ei ole dokumentoitu tai ne ovat vanhentuneita
- kaksisuuntaista jäljitettävyyttä testien ja testauksen pohjamateriaalin välillä ei ole ylläpidetty
- työkalutuki on vähäistä tai sitä ei ole
- mukana olevilla ihmisillä ei ole liiketoiminta- ja/tai järjestelmätietämystä
- ohjelmiston ylläpidettävyyteen ei ole kiinnitetty riittävästi huomiota kehitysvaiheessa.

3 Staattinen testaus — 135 minuuttia

Avainsanat

ad hoc –katselmointi, dynaaminen testaus, epämuodollinen katselmointi, läpikäynti, muodollinen katselmointi, näkökulmiin perustuva katselmointi, roolipohjainen katselmointi, skenaariopohjainen katselmointi, staattinen analyysi, staattinen testaus, tarkastus, tarkistuslistoihin perustuva katselmointi, tekninen katselmointi

Oppimistavoitteet: Staattinen testaus

3.1 Staattisen testauksen perusteet

FL-3.1.1 (K1) Tunnistaa ohjelmiston vaihetuotteet, joita voidaan tutkia erilaisilla staattisilla tekniikoilla

FL-3.1.2 (K2) Käyttää esimerkkejä staattisen testauksen merkityksen kuvaamiseen

FL-3.1.3 (K2) Selittää staattisten ja dynaamisten tekniikoiden erot ottaen huomioon tavoitteet, tunnistettavien vikojen tyypit ja näiden tekniikoiden roolin ohjelmiston elinkaaressa

3.2 Katselmointiprosessi

FL-3.2.1 (K2) Vetää yhteen eri tuotosten katselmointiprosessin tehtävät

FL-3.2.2 (K1) Tunnistaa muodollisen katselmoinnin eri roolit ja vastuut

FL-3.2.3 (K2) Selittää erot eri katselmointityyppien välillä: vapaamuotoinen katselmointi, läpikäynti, tekninen katselmointi ja tarkastus

FL-3.2.4 (K3) Soveltaa katselmointitekniikkaa vikojen löytämiseksi tuotoksesta

FL-3.2.5 (K2) Selittää tekijät, jotka myötävaikuttavat katselmoinnin onnistumiseen

3.1 *Staattisen testauksen perusteet*

Toisin kuin dynaamisessa testauksessa, joka vaatii testattavan ohjelmiston käyttämistä, staattinen testaus nojaa tuotosten manuaaliseen tarkastukseen (katselmoinnit) tai koodin tai muiden tuotosten arviointiin työkalujen avulla (staattinen analyysi). Molemmat staattisen testauksen tyypit arvioivat testattavana olevaa koodia tai tuotosta ilman, että sitä suoritetaan tai käytetään.

Staattinen analyysi on tärkeää, kun on kyse turvallisuuskriittisistä tietokonejärjestelmistä (esim. ilmailu, terveydenhuolto tai ydinvoimaan liittyvät ohjelmistot), mutta se on tullut tärkeäksi ja tavanomaiseksi myös muissa ympäristöissä. Staattinen analyysi on tärkeä osa esimerkiksi tietoturvatestausta. Staattinen analyysi on usein sisällytetty automatisoituihin koonti- ja julkaisujärjestelmiin esimerkiksi ketterässä kehityksessä, jatkuvassa toimituksessa ja jatkuvassa julkaisussa.

3.1.1 Tuotokset, joita voidaan tutkia staattisella testauksella

Miltei mitä tahansa tuotosta voidaan tutkia käyttämällä staattista testausta (katselmointeja ja/tai staattista analyysia), esimerkiksi:

- määritykset, mukaan luettuna liiketoimintavaatimukset, toiminnalliset vaatimukset ja tietoturvavaatimukset
- eepokset, käyttäjätarinat ja hyväksymiskriteerit
- arkkitehtuuri ja suunnittelukuvaukset
- koodi
- testimateriaali, mukaan luettuna testisuunnitelmat, testitapaukset, testiproseduurit ja automatisoidut testiskriptit
- käyttöoppaat
- verkkosivut
- sopimukset, projektisuunnitelmat, aikataulut ja budjetit
- mallit, kuten aktiviteettikaaviot, joita voidaan käyttää mallipohjaisessa testauksessa (ks. ISTQB-MBT perustason sertifikaattisisällön laajennus, mallipohjainen testaaja, ja Kramer 2016).

Katselmointeja voidaan käyttää mihin tahansa tuotokseen, jota osallistujat osaavat lukea ja pystyvät ymmärtämään. Staattista analyysiä voidaan käyttää tehokkaasti mihin tahansa tuotokseen, jolla on muodollinen rakenne (tyypillisesti koodi tai mallit), johon on olemassa soveltuvia staattisen analyysin työkaluja. Staattista analyysiä voidaan jopa tehdä käyttämällä työkaluja, jotka arvioivat luonnollisella kielellä kirjoitettuja tuotoksia, kuten vaatimuksia (esim. oikeinkirjoituksen, kieliopin ja luettavuuden tarkastus).

3.1.2 Staattisen testauksen hyödyt

Staattiset testaustekniikat tarjoavat monenlaisia hyötyjä. Kun staattista testausta käytetään ohjelmistokehityksen elinkaaren aikaisessa vaiheessa, se mahdollistaa vikojen löytämisen aikaisin ennen dynaamisen testin suoritusta (esim. vaatimusten tai suunnittelukuvausten katselmoinneissa, tuotteen kehitysjonon tarkennuksessa jne.). Aikaisin löydetyt viat on usein paljon halvempi poistaa kuin myöhemmin elinkaaren aikana löydetyt viat, erityisesti verrattuna vikoihin, jotka löydetään sen jälkeen kun ohjelmisto on julkaistu ja aktiivisessa käytössä. Staattisten tekniikoiden käyttäminen vikojen löytämiseksi ja korjaamiseksi nopeasti on miltei aina organisaatiolle halvempaa kuin dynaamisen testauksen käyttäminen vikojen löytämiseksi testattavasta kohteesta ja niiden korjaaminen vasta sen jälkeen, erityisesti kun otetaan huomioon kustannukset, jotka liittyvät muiden tuotosten päivittämiseen sekä uudelleen- ja regressiotestauksen tekemiseen.

Muihin staattisen testauksen hyötyihin voivat kuulua mm.

- vikojen löytäminen ja korjaaminen tehokkaammin ja ennen dynaamista testausta
- sellaisten vikojen tunnistaminen, joita ei helposti löydy dynaamisessa testauksessa

- vikojen estäminen suunnittelukuvauksissa tai koodissa paljastamalla epäjohdonmukaisuuksia, tulkinnanvaraisuuksia, ristiriitaisuuksia, puutteita, epätarkkuuksia ja päällekkäisyyksiä vaatimuksissa
- kehityksen tuottavuuden kasvattaminen (esim. parempien suunnittelukuvausten ja helpommin ylläpidettävän koodin myötä)
- toteutuksen kustannusten ja tarvittavan ajan väheneminen
- testauksen kustannusten ja tarvittavan ajan väheneminen
- laadun kokonaiskustannusten väheneminen ohjelmiston elinkaaren aikana, koska myöhemmissä elinkaaren vaiheissa sekä tuotantoon julkaisun jälkeen esiintyy vähemmän häiriöitä
- viestinnän paraneminen tiimin jäsenten välillä katselmointeihin osallistumisen myötä.

3.1.3 Staattisen ja dynaamisen testauksen erot

Staattisella ja dynaamisella testauksella voi olla samoja tavoitteita (ks. alaluku 1.1.1), kuten tuotoksen laadun arviointi ja vikojen tunnistaminen niin aikaisin kuin mahdollista. Staattinen ja dynaaminen testaus täydentävät toisiaan löytämällä erityyppisiä vikoja.

Yksi pääeroista on, että staattinen testaus löytää suoraan vikoja tuotoksista sen sijaan, että se tunnistaisi vikojen aiheuttamia häiriöitä, kun ohjelmistoa käytetään. Vika voi piillä tuotoksessa hyvinkin pitkään aiheuttamatta häiriötä. Polkua vian sijaintipaikkaan käytetään harvoin tai se voi olla vaikea saavuttaa, joten ei ole helppoa rakentaa ja suorittaa dynaamista testiä, joka kohtaa vian. Staattinen testaus voi pystyä löytämään vian paljon vähemmällä työllä.

Toinen ero on, että staattista testausta voidaan käyttää parantamaan tuotosten yhdenmukaisuutta ja sisäistä laatua, kun dynaaminen testaus taas tyypillisesti keskittyy ulkoisesti havaittavaan käytökseen.

Dynaamiseen testaukseen verrattuna staattisen testauksen avulla helpommin ja halvemmalla löydettäviin ja korjattaviin vikoihin kuuluvat mm.

- viat vaatimuksissa (esim. epäjohdonmukaisuudet, moniselitteisyydet, ristiriidat, puutteet, epätarkkuudet ja ylimääräisyydet)
- viat suunnittelussa (esim. tehottomat algoritmit tai tietokantarakenteet, korkea kytkentöjen määrä, matala koheesio)
- koodausvirheet (esim. määrittelemättömät muuttujat, muuttujat, jotka on määritelty mutta joita ei käytetä, kuollut koodi, sama koodi kahteen kertaan)
- poikkeamat standardeista (esim. koodausstandardien noudattamatta jättäminen)
- väärät rajapintamääritykset (esim. kutsuva järjestelmä käyttää eri mittayksiköitä kuin kutsuttava järjestelmä)
- tietoturvahaavoittuvuudet (esim. alttius puskurin ylivuodoille)
- puutteet tai epätarkkuudet testauksen pohjamateriaalin jäljitettävyydessä tai kattavuudessa (esim. hyväksymiskriteereitä vastaan puuttuvat testit).

Sen lisäksi useimmat ylläpidettävyysvikojen tyypeistä voidaan löytää vain staattisella testauksella (esim. vääränlainen modularisaatio, heikko komponenttien uusiokäyttö, koodi, jota on vaikea analysoida ja muokata aiheuttamatta uusia vikoja).

3.2 *Katselmointiprosessi*

Katselmoinnit vaihtelevat epämuodollisesta muodolliseen. Epämuodollisille (vapaamuotoisille) katselmoinneille on tyypillistä, että ne eivät seuraa määritettyä prosessia eikä niillä ole muodollisia dokumentoituja tuloksia. Muodollisille katselmoinneille on tyypillistä tiimin osallistuminen, katselmointien tulosten dokumentointi sekä dokumentoitu katselmoinnin läpivientiprosessi. Katselmointiprosessin muodollisuus liittyy mm. sellaisiin tekijöihin kuin ohjelmistokehityksen elinkaarimalli, kehitysprosessin kypsyys, katselmoitavan tuotoksen monimutkaisuus, lakeihin tai sääntelyihin pohjautuvat vaatimukset ja/tai jäljitettävyyden tarve.

Katselmoinnin painopiste riippuu katselmoinnille sovituista tavoitteista (esim. vikojen löytäminen, ymmärryksen aikaansaaminen, osallistujien, kuten testaajien ja uusien tiiminjäsenten kouluttaminen tai keskustelu ja yhteisten päätösten tekeminen).

ISO-standardi ISO/IEC 20246 sisältää tarkemman kuvauksen eri tuotosten katselmointiprosesseista roolit ja katselmointitekniikat mukaan luettuna.

3.2.1 Tuotosten katselmointiprosessi

Katselmointiprosessiin kuuluvat seuraavat päätehtävät:

Suunnittelu

- Katselmoinnin laajuuden määrittäminen, mukaan luettuna katselmoinnin tarkoitus, mitä osia dokumentista katselmoidaan sekä arvioitavat laatuominaisuudet
- Työmäärän ja aikataulun arviointi
- Katselmoinnin piirteiden tunnistaminen, kuten katselmointityyppi ja roolit, tehtävät ja tarkistuslistat
- Katselmointiin osallistuvien henkilöiden valinta sekä roolien jako
- Aloitus- ja lopetusehtojen määrittäminen muodollisemmille katselmointityypeille (esim. tarkastus)
- Aloituskriteerien täyttymisen tarkistus (muodollisemmissa katselmoinneissa)

Katselmoinnin käynnistys

- Katselmoitavan tuotoksen ja muun materiaalin jakelu joko fyysisesti tai elektronisessa muodossa (esim. havaintolomakkeet, tarkistuslista sekä katselmoinnin kohteeseen liittyvät muut tuotokset)
- Katselmoinnin laajuuden, tavoitteiden, prosessin, roolien ja tuotosten selittäminen osallistujille
- Osallistujien katselmointia koskeviin kysymyksiin vastaaminen

Katselmointi (yksilöllinen valmistautuminen)

- Tuotoksen osan tai koko tuotoksen katselmointi
- Mahdollisten vikojen, suositusten ja kysymysten kirjaaminen

Havaintojen raportointi ja analysointi

- Tunnistetuista mahdollisista vioista raportointi (esim. katselmointipalaverissa)
- Epäiltyjen vikojen analysointi sekä niiden omistajan ja tilan määrittäminen
- Laatuominaisuuksien arviointi ja dokumentointi
- Katselmointihavaintojen arviointi päätöskriteerejä vastaan katselmointipäätöksen tekemiseksi (hylkäys; isot muutokset tarpeellisia; hyväksytään, mahdollisesti pienin muutoksin)

Korjaus ja raportointi

- Vikaraporttien laatiminen niiden havaintojen osalta, jotka vaativat muutoksia
- Katselmoidusta tuotoksesta löydettyjen vikojen korjaus (tyypillisesti tuotoksen tekijä tekee)
- Vioista tiedottaminen asianomaiselle henkilölle tai tiimille (kun kyseessä on löydös katselmoituun tuotokseen liittyvässä tuotoksessa)
- Vikojen päivitetyn tilan kirjaaminen (muodollisissa katselmoinneissa), mahdollisesti havainnon löytäjän hyväksynnän kera
- Mittaritietojen keruu (muodollisemmissa katselmointityypeissä)
- Lopetusehtojen täyttymisen tarkistus (muodollisemmissa katselmoinneissa)
- Tuotoksen hyväksyntä, kun lopetusehdot ovat täyttyneet

Katselmointien lopputulokset vaihtelevat katselmointityypin ja muodollisuuden mukaan, kuten on kuvattu alaluvussa 3.2.3.

3.2.2 Muodollisen katselmoinnin roolit ja vastuut

Tyypilliseen muodolliseen katselmointiin kuuluvat alla mainitut roolit:

Tuotoksen tekijä

- luo katselmoitavan tuotoksen
- korjaa katselmoitavassa tuotoksessa olevat viat (mikäli tarpeen).

Johto

- vastaa katselmointien suunnittelusta
- päättää katselmointien toteutuksesta
- antaa henkilöstön ja budjetin sekä määrittää ajankäytön
- seuraa jatkuvaa kustannustehokkuutta
- tekee hallinnollisia päätöksiä, mikäli lopputulokset ovat puutteellisia.

Fasilitaattori (kutsutaan usein puheenjohtajaksi)

- varmistaa katselmointikokousten tehokkaan läpiviennin (kun niitä pidetään)
- toimii tarvittaessa näkemyserojen sovittelijana
- on usein henkilö, josta katselmointien onnistuminen riippuu.

Katselmoinnin vetäjä

- ottaa kokonaisvastuun katselmoinnista
- päättää, ketkä katselmointiin osallistuvat ja sopii katselmoinnin toteutusajasta ja –paikasta.

Katselmoijat

- voivat olla asiasisällön asiantuntijoita, projektissa työskenteleviä henkilöitä, sidosryhmien edustajia, joille tuotoksella on merkitystä ja/tai henkilöitä, joilla on erityinen tekninen tai liiketoiminnallinen tausta
- tunnistavat mahdollisia vikoja katselmoitavassa tuotoksessa
- voivat edustaa erilaisia näkökulmia (esim. testaaja, ohjelmoija, käyttäjä, operaattori, liiketoiminta-analyytikko, käytettävyysasiantuntija jne.).

Sihteeri (tai kirjuri)

- kokoaa yksittäisen katselmointitehtävän aikana löydetyt viat
- kirjaa ylös katselmointikokouksessa esiin tulleet uudet mahdolliset viat, avoimet kysymykset sekä tehdyt päätökset (jos kokous pidetään).

Joissain katselmointityypeissä yksi henkilö voi edustaa useampaa roolia ja eri rooleihin liittyvät tehtävät voivat myös vaihdella katselmointityypin mukaan. Lisäksi esimerkiksi vikojen, kysymysten ja päätösten kirjaamiseen tarkoitettujen katselmointiprosessia tukevien työkalujen käytön lisääntymisen myötä sihteeriä ei monesti tarvita.

Myös yksityiskohtaisemmat roolit ovat mahdollisia, kuten on kuvattu ISO-standardissa 20246.

3.2.3 Katselmointityypit

Vaikka katselmointeja voidaan käyttää eri tarkoituksiin, yksi päätavoitteista on vikojen paljastaminen. Kaikki katselmointityypit voivat auttaa vikojen löytämistä ja katselmointityypin valinta pitäisi perustua projektin tarpeisiin, käytettävissä oleviin resursseihin, tuotteen tyyppiin ja riskeihin, liiketoiminta-alueeseen sekä yrityskulttuuriin ja muihin valintakriteereihin.

Katselmoinnit voidaan luokitella eri tekijöiden perusteella. Seuraavassa on esitelty neljä tyypillisintä katselmointityyppiä sekä niihin liittyvät ominaisuudet.

Epämuodollinen katselmointi (esim. paritarkastus, parikatselmointi)

- Päätarkoitus: Mahdollisten vikojen löytäminen
- Muita mahdollisia tarkoituksia: Uusien ideoiden tai ratkaisujen luominen, pienten ongelmien ratkaiseminen nopeasti
- Ei pohjaudu muodolliseen (dokumentoituun) prosessiin
- Ei aina sisällä katselmointipalaveria
- Voi olla materiaalin laatijan työtovereiden (paritarkastus) tai useamman henkilön tekemä
- Tulokset voidaan dokumentoida
- Hyödyllisyys vaihtelee katselmoijan mukaan
- Tarkistuslistojen käyttö valinnaista
- Käytetään hyvin tyypillisesti Ketterässä kehityksessä.

Läpikäynti

- Päätarkoitus: Vikojen löytäminen, ohjelmistotuotteen parantaminen, vaihtoehtoisten toteutustapojen pohtiminen, standardien- ja määritystenmukaisuuden arviointi
- Mahdolliset muut tarkoitukset: Tekniikoihin tai eri tyyleihin liittyvien ideoiden vaihtaminen, osallistujien koulutus, yhteisymmärryksen saavuttaminen
- Yksilöllinen valmistautuminen ennen katselmointipalaveria on valinnaista
- Tuotoksen tekijä vetää tyypillisesti katselmointipalaverin
- Sihteerin käyttö on pakollista
- Tarkistuslistojen käyttö on valinnaista
- Voi sisältää skenaarioiden käyttöä, kuivaharjoittelua tai simulaatioita
- Voi tuottaa vikaraportteja ja katselmointiraportteja
- Voi vaihdella käytännössä melko epämuodollisesta hyvin muodolliseen.

Tekninen katselmointi

- Päätarkoitus: Yhteisymmärryksen aikaansaaminen, mahdollisten vikojen löytäminen.
- Muita mahdollisia tarkoituksia: Tuotoksen laadun arviointi ja luottamuksen kasvattaminen tuotosta kohtaan, uusien ideoiden luominen, materiaalin laatijoiden motivointi ja avustaminen parempien tuotosten laatimiseen jatkossa, vaihtoehtoisten toteutustapojen pohtiminen
- Katselmoijien pitäisi olla tuotoksen tekijän teknisesti osaavia työtovereita sekä saman tai muiden alojen teknisiä asiantuntijoita
- Edellyttää yksilöllistä valmistautumista ennen katselmointipalaveria
- Katselmointipalaveri on valinnainen ja sen johtaa yleensä koulutettu puheenjohtaja (tyypillisesti ei tuotoksen tekijä)
- Sihteerin käyttö on pakollista, ihannetilanteessa tuotoksen tekijä ei toimi sihteerinä
- Tarkistuslistojen käyttö on valinnaista
- Tuottaa tyypillisesti vikaraportteja ja katselmointiraportteja.

Tarkastus

- Päätarkoitus: Mahdollisten vikojen löytäminen, tuotoksen laadun arviointi ja luottamuksen kasvattaminen tuotosta kohtaan sekä samanlaisten vikojen syntymisen estäminen tulevaisuudessa materiaalin laatijan oppimisen sekä perussyyanalyysin kautta

- Mahdolliset muut tarkoitukset: Materiaalin laatijoiden motivointi ja avustaminen parempien tuotosten laatimiseksi jatkossa, yhteisymmärryksen aikaansaaminen.
- Seuraa määriteltyä sääntöihin ja tarkistuslistoihin perustuvaa prosessia, johon kuuluvat dokumentoidut tuotokset
- Käyttää selkeästi määriteltyjä rooleja, joihin kuuluvat pakolliset alaluvussa 3.2.2 kuvatut, ja voi lisäksi käyttää lukijaa (joka lukee tuotosta ääneen katselmointipalaverin aikana)
- Edellyttää yksilöllistä valmistautumista ennen katselmointipalaveria
- Katselmoijat ovat joko materiaalin laatijan työtovereita tai muiden tuotoksen kannalta oleellisten alojen asiantuntijoita
- Käytetään määritettyjä aloitus- ja lopetuskriteerejä
- Sihteerin käyttö on pakollista
- Katselmointipalaverin johtaa koulutettu puheenjohtaja (ei materiaalin laatija)
- Materiaalin laatija ei voi toimia katselmoinnin vetäjänä, lukijana eikä sihteerinä
- Tuottaa vikaraportteja ja katselmointiraportteja
- Mittaritietoja kerätään ja käytetään koko ohjelmistokehitysprosessin parantamiseen, tarkastusprosessi mukaan luettuna.

Yksittäinen tuotos voi käydä läpi useamman kuin yhden tyyppisen katselmoinnin. Jos useampia kuin yhtä katselmointityyppiä käytetään, niiden järjestys voi vaihdella. Esimerkiksi epämuodollinen katselmointi voidaan tehdä ennen teknistä katselmointia sen varmistamiseksi, että tuotos on valmis tekniseen katselmointiin.

Yllä kuvatut katselmointityypit voidaan tehdä vertaiskatselmointeina, eli niitä suorittavat työtoverit suunnilleen samalta organisatoriselta tasolta.

Katselmoinneissa löydettyjen vikojen tyypit vaihtelevat erityisesti katselmoitavan tuotoksen mukaan. Ks. alaluku 3.1.3 esimerkkejä vioista, joita erilaisista tuotoksista voi löytyä katselmoinneissa ja Gilb 1993 lisää tietoa muodollisista katselmoinneista.

3.2.4 Katselmointitekniikoiden soveltaminen

On olemassa joukko katselmointitekniikoita, joita voidaan käyttää vikojen paljastamiseksi katselmointityön (eli yksilöllisen valmistautumisen) aikana. Näitä tekniikoita voidaan käyttää läpi yllä kuvattujen katselmointityyppien. Tekniikoiden tehokkuus voi vaihdella käytettävän katselmointityypin perusteella. Alla on kuvattu erimerkkejä eri katselmointityyppien yhteydessä käytettävistä eri katselmointitekniikoista.

Ad hoc

Ad hoc –katselmoinnissa katselmoijilla on vain vähän tai ei lainkaan tietoa siitä, kuinka katselmointitehtävä pitäisi suorittaa. Katselmoijat lukevat usein tuotoksen edeten suoraviivaisesti eteenpäin ja tunnistavat ja dokumentoivat havaintoja sitä mukaa, kun he kohtaavat niitä. Ad hoc –katselmointi on yleisesti käytetty tekniikka, joka vaatii vähän valmisteluja. Tämä tekniikan tehokkuus riippuu suuresti katselmoijan osaamisesta ja saattaa johtaa siihen, että eri katselmoijat raportoivat moneen kertaan samoja havaintoja.

Tarkistuslistoihin pohjautuvat

Tarkistuslistoihin pohjautuva katselmointi on järjestelmällinen tekniikka, jossa katselmoijat löytävät havaintoja katselmoinnin käynnistyksen yhteydessä jaetun (esim. fasilitaattorin jakaman) tarkistuslistan perusteella. Katselmoinnin tarkistuslista koostuu joukosta mahdollisiin vikoihin liittyviä kysymyksiä, jotka on laadittu kokemuksen perusteella. Tarkistuslistojen pitäisi olla erityisesti katselmoitavaan tuotokseen kohdistettuja ja niitä pitäisi ylläpitää säännöllisesti aikaisemmissa katselmoinneissa löytämättä jääneiden ongelmien kattamiseksi. Tarkistuslistoihin pohjautuvan tekniikan päähyöty on tyypillisten vikatyyppien järjestelmällinen kattaminen. On myös tärkeää huolehtia siitä, että katselmointityön aikana ei pelkästään seurata tarkistuslistaa vaan etsitään vikoja myös tarkistuslistan ulkopuolelta.

Skenaariot ja kuivaharjoitukset

Skenaariopohjaisessa katselmoinnissa katselmoijille annetaan järjestelmälliset ohjeet siitä, kuinka edetä läpi tuotoksen. Skenaariopohjainen lähestymistapa auttaa katselmoijia tekemään tuotokselle “kuivaharjoituksia”, jotka perustuvat tuotoksen odotettuun käyttöön (jos tuotos on dokumentoitu sopivassa muodossa, kuten esimerkiksi käyttötapaukset). Nämä skenaariot antavat katselmoijille paremmat lähtökohdat määrätyn tyyppisten vikojen tunnistamiseksi kuin yksinkertaiset tarkistuslistan kohdat. Kuten tarkistuslistoihin pohjautuvissa katselmoinnissa, katselmoijien ei pitäisi rajoittaa työtään pelkästään dokumentoituihin skenaarioihin, jotta muun tyyppisiä vikoja (esim. puuttuvat ominaisuudet) ei jää huomaamatta.

Rooleihin pohjautuva

Rooleihin pohjautuva katselmointi on tekniikka, jossa katselmoijat arvioivat tuotosta yksittäisten sidosryhmien roolien näkökulmasta. Tyypillisiin rooleihin kuuluvat määrätyn tyyppiset loppukäyttäjät (kokeneet, kokemattomat, iäkkäät, lapset jne.) ja organisaatioon kuuluvat määrätyt roolit (käyttäjien valvoja, järjestelmänvalvoja, suorituskykytestaaja jne.).

Näkökulmiin perustuva

Näkökulmiin perustuvassa katselmoinnissa, samoin kuin rooleihin pohjautuvassa katselmoinnissa, katselmoijat asettuvat eri sidosryhmien asemaan katselmoidessaan materiaalia. Tyypillisiin sidosryhmien näkökulmiin kuuluvat loppukäyttäjä, markkinointi, suunnittelija, testaaja tai operointi. Eri sidosryhmien näkökulmien käyttäminen johtaa syvällisempään materiaalin läpikäyntiin ja vähentää tilanteita, joissa useampi katselmoija löytää saman vian (tuplahavainnot).

Lisäksi näkökulmiin perustuvaan katselmointiin kuuluu, että katselmoijat yrittävät käyttää katselmoinnin kohteena olevaa tuotosta saadakseen aikaan tuotteen, joka heidän pitäisi siitä sen perusteella tuottaa. Esimerkiksi testaaja voi yrittää luoda hyväksymistestien luonnoksia tehdessään näkökulmiin perustuvaa katselmointia vaatimusmäärittelyille nähdäkseen, onko kaikki tarvittava tieto saatavilla. Näkökulmiin perustuvassa katselmoinnissa odotetaan lisäksi käyttävän tarkistuslistoja.

Empiiriset tutkimukset ovat osoittaneet, että näkökulmiin perustuva katselmointi on tehokkain vaatimusten ja teknisten tuotoksen katselmoinnissa käytettävä yleistekniikka. Onnistumisen avaintekijä on ottaa riskien perusteella mukaan eri sidosryhmien näkökulmia ja painottaa niitä oikealla tavalla. Lisää näkökulmiin perustuvasta katselmoinnista, ks. Shul 2000, ja eri katselmointityyppien tehokkuudesta, ks. Sauer 2000.

3.2.5 Katselmointien menestystekijät

Katselmoinnin onnistumiseksi on valittava tilanteeseen soveltuva katselmointityyppi sekä –tekniikat. Lisäksi on monia muita tekijöitä, jotka vaikuttavat katselmoinnin lopputulokseen.

Organisaatioon liittyviin menestystekijöihin kuuluvat mm. seuraavat:

- Joka katselmoinnilla on selkeät tavoitteet, jotka määritellään katselmoinnin suunnitteluvaiheessa ja joita käytetään mitattavina lopetusehtoina.
- Käytettävät katselmointityypit mahdollistavat asetettujen tavoitteiden saavuttamisen ja ovat sopivia ohjelmistotuotosten tyypin ja tason sekä katselmoijien suhteen.
- Käytettävät katselmointitekniikat, kuten tarkistuslistapohjainen tai roolipohjainen katselmointi, soveltuvat vikojen tehokkaaseen tunnistamiseen katselmoinnin kohteena olevasta tuotoksesta.
- Käytettävät tarkistuslistat kohdistuvat tärkeimpiin riskeihin ja ovat ajan tasalla.
- Isot dokumentit kirjoitetaan ja katselmoidaan pieninä osina, jotta laadunvalvonnan toteuttamiseksi materiaalin laatijat saavat aikaista ja säännöllistä palautetta vioista.
- Osallistujilla on riittävästi aikaa valmistautumiseen.
- Katselmoinnit aikataulutetaan riittävän ajoissa etukäteen.
- Yrityksen johto tukee katselmointiprosessia (esim. järjestämällä riittävästi aikaa katselmointitehtäville projektien aikatauluissa)

Ihmisiin liittyviin menestystekijöihin kuuluvat mm. seuraavat:

- Oikeat ihmiset ovat mukana katselmoinnin tavoitteiden saavuttamiseksi, esimerkiksi ihmiset, joilla on erilaista osaamista tai eri näkökulmia, ja jotka voivat käyttää dokumenttia työnsä perustana.
- Testaajia pidetään arvostettuina katselmoijina, jotka myötävaikuttavat katselmointiin ja oppivat lisää tuotoksesta, jolloin he voivat laatia tehokkaampia testejä aikaisemmin.
- Osallistujat panostavat riittävästi aikaa ja huomiota yksityiskohtiin.
- Katselmoinnit tehdään pienissä osissa, jotta katselmoijat eivät menetä keskittymistään katselmointityön ja/tai katselmointipalaverin aikana (jos sellainen pidetään).
- Löydetyt viat hyväksytään, niitä arvostetaan ja ne käsitellään objektiivisesti.
- Palaveria hallinnoidaan hyvin, jotta osallistujat pitävät sitä hyödyllisenä aikansa käyttämisenä.
- Katselmoinnissa vallitsee luottamuksen ilmapiiri; tulosta ei käytetä osallistujien arviointiin.
- Osallistujat välttävät elekieltä ja käytöstä, jotka voisivat ilmaista tylsistymistä, ärsyyntymistä tai vihamielisyyttä toisia osallistujia kohtaan.
- Riittävää koulutusta on tarjolla, erityisesti kun kyse on muodollisemmista katselmointityypeistä kuten tarkastuksista.
- Oppimisen ja prosessinparantamisen kulttuuria edistetään.

Lisää tietoa onnistuneista katselmoinneista, ks. Gilb 1993, Wiegers 2002 ja van Veenendaal 2004.

4 Testaustekniikat — 330 minuuttia

Avainsanat

ekvivalenssiositus, kattavuus, kokemusperustainen testaustekniikka, käyttötapaustestaus, lasilaatikkotestaustekniikka, lausekattavuus, mustalaatikkotestaustekniikka, päätöskattavuus, päätöstaulutestaus, raja-arvoanalyysi, tarkistuslistoihin pohjautuva testaus, testaustekniikka, tilasiirtymätestaus, tutkiva testaus, virheenarvaus

Oppimistavoitteet: Testaustekniikat

4.1 Testaustekniikoiden luokittelu

FL-4.1.1 (K2) Selittää mustalaatikko-, lasilaatikko- ja kokemuspohjaisten testaustekniikoiden piirteet sekä niiden väliset samankaltaisuudet ja erot

4.2 Mustalaatikkotekniikat

FL-4.2.1 (K3) Käyttää ekvivalenssiositusta testitapausten johtamiseksi annetuista vaatimuksista

FL-4.2.2 (K3) Käyttää raja-arvoanalyysiä testitapausten johtamiseksi annetuista vaatimuksista

FL-4.2.3 (K3) Käyttää päätöstaulutestausta testitapausten johtamiseksi annetuista vaatimuksista

FL-4.2.4 (K3) Käyttää tilasiirtymätestausta testitapausten johtamiseksi annetuista vaatimuksista

FL-4.2.5 (K2) Selittää, kuinka testitapauksia johdetaan käyttötapauksesta

4.3 Lasilaatikkotekniikat

FL-4.3.1 (K2) Selittää lausekattavuus

FL-4.3.2 (K2) Selittää päätöskattavuus

FL-4.3.3 (K2) Selittää lause- ja päätöskattavuuden merkitys

4.4 Kokemuspohjaiset tekniikat

FL-4.4.1 (K2) Selittää virheenarvaus

FL-4.4.2 (K2) Selittää tutkiva testaus

FL-4.4.3 (K2) Selittää tarkistuslistapohjainen testaus

4.1 *Testaustekniikoiden luokittelu*

Testaustekniikoiden tarkoitus, tässä alaluvussa esitellyt tekniikat mukaan luettuna, on auttaa testattavien tilanteiden, testitapausten ja testiaineiston tunnistamisessa.

4.1.1 Testaustekniikoiden valinta

Käytettävien testaustekniikoiden valinta riippuu monista tekijöistä, mukaan luettuna seuraavat:

- komponentin tai järjestelmän tyyppi
- komponentin tai järjestelmän monimutkaisuus
- järjestelmää tai testausta säätelevät standardit
- asiakkaalta saadut tai sopimukselliset vaatimukset
- riskitasot
- riskityypit
- testauksen tavoitteet
- saatavilla oleva dokumentaatio
- testaajien tietämys ja osaaminen
- saatavilla olevat työkalut
- aika ja budjetti
- ohjelmistokehityksen elinkaarimalli
- ohjelmiston odotettu käyttötapa
- aikaisempi kokemus testattavan komponentin tai järjestelmän testauksessa käytetyistä tekniikoista
- vikatyypit, joita komponentin tai järjestelmän ennakoidaan sisältävän.

Jotkut tekniikat soveltuvat paremmin tiettyihin tilanteisiin ja tietyille testitasoille, toiset soveltuvat kaikille testitasoille. Kun testaajat luovat testitapauksia, he käyttävät yleensä testaustekniikoiden joukkoa parhaiden tulosten saavuttamiseksi testauksesta.

Testaustekniikoiden käyttö testianalyysissä sekä testien suunnittelussa ja valmistelussa voi vaihdella hyvin epämuodollisesta (vähän tai ei lainkaan dokumentaatiota) hyvin muodolliseen. Muodollisuuden sopiva aste riippuu testauksen kokonaistilanteesta, johon liittyvät testaus- ja kehitysprosessien kypsyys, aikarajoitteet, turvallisuuteen tai sääntelyihin liittyvät vaatimukset, mukana olevien henkilöiden tietämys ja osaaminen sekä käytettävä ohjelmistokehityksen elinkaarimalli.

4.1.2 Testaustekniikoiden luokittelu ja ominaisuudet

Tässä sertifikaattisisällössä testaustekniikat jaetaan mustalaatikkotekniikoihin, lasilaatikkotekniikoihin ja kokemuspohjaisiin tekniikoihin.

Mustalaatikkotekniikat (kutsutaan myös käyttäytymiseen perustuviksi tekniikoiksi) pohjautuvat tilanteeseen soveltuvan testauksen pohjamateriaalin analyysiin (esim. muodollinen vaatimusdokumentaatio, määrittelyt, käyttötapaukset, käyttäjätarinat tai liiketoimintaprosessit). Näitä tekniikoita voidaan käyttää sekä toiminnallisessa että ei-toiminnallisessa testauksessa. Mustalaatikkotestaus keskittyy testattavan kohteen syötteisiin ja tuloksiin kiinnittämättä huomiota sen sisäiseen rakenteeseen.

Lasilaatikkotekniikat (joita kutsutaan myös rakenteellisiksi tai rakenteeseen perustuviksi tekniikoiksi) perustuvat testattavan kohteen arkkitehtuurin, yksityiskohtaisten suunnitelmien, sisäisen rakenteen tai koodin analysointiin. Toisin kuin mustalaatikkotekniikat, lasilaatikkotekniikat keskittyvät testattavan kohteen rakenteeseen ja sen suorittamaan tietojen käsittelyyn.

Kokemuspohjaiset tekniikat hyödyntävät testien suunnittelussa, valmistelussa ja suorituksessa kehittäjien, testaajien ja käyttäjien kokemusta. Näitä tekniikoita yhdistetään usein mustalaatikko- ja lasilaatikkotekniikoiden kanssa.

Mustalaatikkotekniikoiden yleisiin piirteisiin kuuluvat seuraavat:

- Testattavat tilanteet, testitapaukset ja testiaineisto johdetaan testauksen pohjamateriaalista, johon saattaa kuulua ohjelmistovaatimuksia, määrittelyitä, käyttötapauksia ja käyttäjätarinoita.
- Testitapauksia voidaan käyttää paljastamaan vaatimusten ja niiden toteutuksen välisiä aukkoja samoin kuin poikkeamia vaatimuksista.
- Kattavuuden mittaus perustuu testauksen pohjamateriaalin testattaviin kohtiin sekä testauksen pohjamateriaaliin sovellettuihin tekniikoihin.

Lasilaatikkotekniikoiden yleisiin piirteisiin kuuluvat seuraavat:

- Testattavat tilanteet, testitapaukset ja testiaineisto perustuvat testauksen pohjamateriaaliin, johon voi kuulua koodi, ohjelmistoarkkitehtuuri, yksityiskohtaiset suunnitelmat tai mikä tahansa muu materiaali, joka sisältää tietoa ohjelmiston rakenteesta.
- Kattavuuden mittaus perustuu valitun rakenteen testattaviin osiin (esim. koodi tai rajapinnat).
- Määrittelyitä käytetään usein lisätietolähteenä testitapausten odotettujen lopputulosten määrittämiseksi.

Kokemuspohjaisten tekniikoiden yleisiin piirteisiin kuuluvat seuraavat:

- Testattavat tilanteet, testitapaukset ja testiaineisto johdetaan testauksen pohjamateriaalista, johon saattaa kuulua testaajien, kehittäjien, käyttäjien ja muiden sidosryhmien tietämys ja kokemus.

Tähän tietämykseen ja kokemukseen kuuluvat ohjelmiston odotettu käyttö, sen ympäristö, todennäköiset viat sekä niiden jakauma.

Kansainvälinen standardi ISO/IEC/IEE 29119-4 sisältää kuvauksen testaustekniikoista ja niiden vastaavista kattavuusmittareista (lisää tietoa tekniikoista, ks. Craig 2002 ja Copeland 2004).

4.2 *Mustalaatikkotekniikat*

4.2.1 Ekvivalenssiositus

Ekvivalenssiosituksessa tieto jaetaan ryhmiin (tunnetaan myös ekvivalenssiluokkina) niin, että kaikki tietyn ryhmän jäsenet odotetaan käsiteltävän samalla tavalla (ks. Kaner 2013 ja Jorgensen 2014). Ekvivalenssiryhmiä/luokkia on sekä kelvollisille että epäkelvoille arvoille.

- Kelvolliset arvot ovat arvoja, jotka komponentin tai järjestelmän pitäisi hyväksyä. Kelvolliset arvot ovat arvoja, jotka komponentin tai järjestelmän pitäisi hyväksyä.
- Epäkelvot arvot ovat arvoja, jotka komponentin tai järjestelmän pitäisi hylätä. Epäkelpoja arvoja sisältävää ekvivalenssiluokkaa kutsutaan “epäkelvoksi ekvivalenssiluokaksi”.
- Mitkä tahansa järjestelmään liittyvät tietoelementit, kuten syötteet, tulokset, sisäiset arvot, aikariippuvaiset arvot (esim. ennen tai jälkeen tapahtuman) ja rajapintaparametrit (esim. integrointitestauksen aikana testattaviin integroitaviin komponentteihin liittyvät) voidaan jakaa ekvivalenssiluokkiin.
- Jokainen luokka voidaan jakaa tarvittaessa alaluokkiin.
- Jokaisen arvon tulee kuulua yhteen ja vain yhteen ekvivalenssiluokkaan.
- Kun epäkelpoja luokkia käytetään testitapauksissa, ne pitäisi testata erikseen, ts. ei yhdistettynä muiden epäkelpojen ekvivalenssiluokkien kanssa, sen varmistamiseksi, että häiriöt eivät jää peittoon. Häiriöt voivat jäädä peittoon silloin, kun useita häiriöitä tapahtuu samaan aikaan, mutta vain yksi niistä on näkyvä ja aiheuttaa muiden jäämisen huomaamatta.

Jotta tätä tekniikkaa käyttämällä saavutetaan 100 % kattavuus, testitapausten pitää kattaa kaikki tunnistetut luokat (epäkelvot luokat mukaan luettuna) niin, että joka luokasta käytetään vähintään yhtä arvoa. Kattavuus mitataan laskemalla vähintään yhdellä testillä testattujen ekvivalenssiluokkien määrä, jakamalla se kaikkien tunnistettujen ekvivalenssiluokkien määrällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna. Ekvivalenssiositusta voidaan käyttää kaikilla testaustasoilla.

4.2.2 Raja-arvoanalyysi

Raja-arvoanalyysi (eng. Boundary Value Analysis, BVA) on ekvivalenssiosituksen laajennus, mutta sitä voidaan käyttää vain silloin, kun luokka on järjestetty eli koostuu numeerisista tai peräkkäisistä tiedoista. Luokan pienin ja suurin arvo (tai ensimmäinen ja viimeinen arvo) ovat sen raja-arvot (Beizer 1990).

Ajatellaan esimerkiksi syötekenttää, joka hyväksyy syötteenä yksinumeroisen kokonaisluvun, ja syötteen lähteenä käytetään näppäimistöä, jolla vain kokonaislukujen syöttäminen on mahdollista. Kelvollinen alue on 1 – 5, nämä arvot mukaan luettuna. Tässä tilanteessa voimme tunnistaa kolme ekvivalenssiluokkaa: epäkelpo (liian pieni); kelvollinen; epäkelpo (liian suuri). Kelvollisen ekvivalenssiluokan raja-arvot ovat 1 ja 5. Epäkelvon (liian suuren) ekvivalenssiluokan raja-arvot ovat 6 ja 9. Epäkelvolla (liian pienellä) ekvivalenssiluokalla on vain yksi raja-arvo, 0, koska tässä luokassa on vain yksi arvo.

Yllä kuvatussa esimerkissä tunnistimme kaksi raja-arvoa yhtä rajaa kohden. Epäkelvon (liian pienen) ja kelvollisen luokan välinen raja tuottaa testattavat arvot 0 ja 1. Kelvollisen ja epäkelvon (liian suuren) luokan välinen raja tuottaa testattavat arvot 5 ja 6. Joissain tämän tekniikan variaatioissa tunnistetaan kolme arvoa rajaa kohti: arvo juuri ennen rajaa, rajalla sekä juuri rajan yli. Kolmiarvotekniikkaa käyttämällä edellisessä esimerkissä alemman rajan testattavat arvot ovat 0, 1 ja 2, ja ylemmän rajan testattavat arvot ovat 4, 5, ja 6 (Jorgensen 2014).

Ohjelmat käyttäytyvät todennäköisemmin väärin ekvivalenssiluokkien rajoilla kuin luokkien sisällä. On tärkeää muistaa, että sekä määritettyjen että toteutettujen rajojen sijainti voi olla väärin joko niiden tarkoitetun sijainnin ylä- tai alapuolella, rajoja voi puuttua kokonaan tai mukaan on voitu lisätä ei-toivottuja vääriä rajoja. Raja-arvoanalyysi ja -testaus paljastavat miltei kaikki tällaiset viat pakottamalla ohjelmiston käyttäytymään toisen luokan mukaisesti kuin sen, johon raja-arvon pitäisi kuulua.

Raja-arvoanalyysiä voidaan käyttää kaikilla testitasoilla. Tätä tekniikkaa käytetään yleensä, kun testataan vaatimuksia, joihin liittyy numerojoukkojen käsittelyä (päivämäärät ja kellonajat mukaan luettuna). Luokan rajakattavuus mitataan laskemalla testattujen rajojen määrä, jakamalla luku tunnistettujen rajojen arvojen määrällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna.

4.2.3 Päätöstaulutestaus

Yhdistelmiä käsittelevät testaustekniikat ovat hyödyllisiä testattaessa järjestelmävaatimuksia, jotka kuvaavat, miten ehtojen erilaisilla yhdistelmillä päädytään erilaisiin lopputuloksiin. Yksi lähestymistapa tällaiseen testaukseen on päätöstaulutestaus.

Päätöstaulut ovat hyvä tapa kirjata monimutkaisia liiketoimintasääntöjä, joita järjestelmän pitää toteuttaa. Kun päätöstauluja luodaan, testaaja tunnistaa ehdot (usein syötteet) ja niistä seuraavat järjestelmän toimenpiteet (usein tulokset). Nämä muodostavat taulukon rivit, tyypillisesti ehdot taulukon yläosassa ja toimenpiteet taulukon alaosassa. Jokainen taulukon sarake vastaa päätössääntöä, joka määrittelee yksilöllisen ehtoyhdistelmän, joka johtaa sääntöön liittyvien toimenpiteiden suorittamiseen. Ehtojen ja toimenpiteiden arvot kuvataan tavallisesti Boolen arvoina (tosi tai epätosi) tai erillisinä arvoina (esim. punainen, vihreä, sininen), mutta ne voivat myös olla numeroita tai numeroalueita. Näitä eri tyyppisiä ehtoja ja toimenpiteitä voidaan kuvata samassa taulukossa.

Tyypillinen päätöstaulun notaatio on seuraava:

Ehdot:

- K tarkoittaa, että ehto on tosi (voidaan kuvata myös arvona T tai 1)
- E tarkoittaa, että ehto on epätosi (voidaan kuvata myös arvona E tai 0)
- --- tarkoittaa, että ehdon arvolla ei ole merkitystä (voidaan merkitä myös N/A)

Toimenpiteet:

- X tarkoittaa, että toimenpiteen pitäisi tapahtua (voidaan kuvata myös arvona K, T tai 1)

- Tyhjä tarkoittaa, että toimenpiteen ei pitäisi tapahtua (voidaan kuvata myös arvona E tai 0)

Täydessä päätöstaulussa on riittävästi sarakkeita kattamaan kaikki ehtojen yhdistelmät. Taulua voidaan pienentää poistamalla mahdottomia yhdistelmiä sisältävät sarakkeet, teoriassa mahdollisia mutta käytännössä mahdottomia yhdistelmiä sisältävät sarakkeet sekä sarakkeet, jotka testaavat sellaisia ehtoyhdistelmiä, joilla ei ole vaikutusta lopputulokseen. Lisää tietoa päätöstaulujen pienentämisestä, ks. ISTQB-ATA Jatkotason sertifikaattisisältö, Testausasiantuntija.

Tyypillinen vähimmäiskattavuus päätöstaulutestauksessa on, että jokaista taulukon päätössääntöä kohden on ainakin yksi testitapaus. Tämä tarkoittaa yleensä kaikkien ehtoyhdistelmien kattamista. Kattavuus mitataan laskemalla vähintään yhdellä testitapauksella testattujen päätössääntöjen määrä, jakamalla se kaikkien päätössääntöjen määrällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna.

Päätöstaulutestauksen vahvuus on siinä, että se auttaa tunnistamaan kaikki tärkeät ehtoyhdistelmät, joista jotkut voisivat muuten jäädä huomaamatta. Se auttaa myös löytämään mahdollisia aukkoja vaatimuksista. Sitä voidaan käyttää kaikilla testaustasoilla kaikissa tilanteissa, joissa ohjelmiston käyttäytyminen riippuu ehtojen yhdistelmistä.

4.2.4 Tilasiirtymätestaus

Komponentit tai järjestelmät voivat vastata tapahtumaan eri tavoin sen hetkisen tilanteen tai aikaisemman tapahtumahistorian perusteella (esim. tapahtumat, jotka on suoritettu sen jälkeen, kun järjestelmä alustettiin). Aikaisempi historia voidaan vetää yhteen käyttämällä tiloja. Tilasiirtymäkaavio kuvaa mahdolliset ohjelmiston tilat samoin kuin sen, kuinka ohjelmisto saapuu tilaan, poistuu siitä sekä siirtyy eri tilojen välillä. Siirtymän saa aikaan tapahtuma (esim. käyttäjä syöttää arvon kenttään). Tapahtuma johtaa siirtymään. Jos sama tapahtuma voi johtaa kahteen tai useampaan siirtymään samasta tilasta, kyseistä tapahtumaa voidaan ohjata porttiehdolla. Tilan muutos voi johtaa ohjelmiston suorittamaan toimenpiteeseen (esim. tulostaa laskutoimituksen tuloksen tai virheilmoituksen).

Tilasiirtymätaulu kuvaa kaikki tilojen väliset sallitut siirtymät sekä mahdollisesti epäkelvot siirtymät sekä kelvolliseen siirtymään liittyvät tapahtumat, porttiehdot ja toimenpiteet. Tilasiirtymäkaaviot kuvaavat yleensä vain kelvolliset siirtymät ja epäkelvot siirtymät jätetään pois.

Testit voidaan suunnitella kattamaan tyypillinen tilasiirtymäketju, käymään läpi jokainen tila, käymään läpi jokainen siirtymä tai tietyt siirtymien sarjat, tai testaamaan epäkelpoja tilasiirtymiä.

Tilasiirtymätestausta käytetään valikkopohjaisten sovellusten testauksessa ja sitä käytetään laajasti sulautetussa ohjelmistoteollisuudessa. Tekniikka sopii myös eri tiloja sisältävän liiketoimintaskenaarion mallintamiseen tai näytön navigoinnin testaamiseen. Tilan käsite on abstrakti – se voi tarkoittaa muutamaa koodiriviä tai koko liiketoimintaprosessia.

Kattavuus mitataan yleensä laskemalla testattujen tunnistettujen tilojen tai siirtymien määrä, jakamalla se testattavan kohteen sisältämien tunnistettujen tilojen tai siirtymien kokonaismäärällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna. Lisää tietoa tilasiirtymätestauksen kattavuuskriteereistä, ks. ISTQB-ATA Jatkotason sertifikaattisisältö, Testausasiantuntija.

4.2.5 Käyttötapaustestaus

Testejä voidaan johtaa käyttötapauksista, jotka ovat erityinen tapa suunnitella ohjelmiston osien välisiä vuorovaikutuksia, käyttämällä käyttötapauksessa kuvattuja ohjelmiston toimintoihin liittyviä vaatimuksia. Käyttötapauksiin liittyvät toimijat (järjestelmää käyttävät ihmiset, ulkoiset laitteet tai muut komponentit tai järjestelmät) sekä kohteet (komponentti tai järjestelmä, johon käyttötapausta sovelletaan).

Jokainen käyttötapaus kuvaa jonkin käyttäytymisen, jonka kohde voi suorittaa yhteistyössä yhden tai useamman toimijan kanssa (UML 2.3.1 2017). Käyttötapaus voidaan määritellä vuorovaikutuksina ja toimenpiteinä, esiehdot ja jälkiehdot mukaan luettuna, ja siinä voidaan käyttää luonnollista kieltä, mikäli se sopii tilanteeseen. Toimijoiden ja kohteen väliset vuorovaikutukset voivat johtaa muutoksiin kohteen tilassa. Vuorovaikutukset voidaan kuvata graafisesti työkaavioiden, aktiviteettikaavioiden tai liiketoimintaprosessimallien avulla.

Käyttötapaus voi sisältää sen peruskäyttäytymiseen liittyviä mahdollisia vaihtoehtoja, kuten poikkeuksellinen käyttäytyminen ja virheenkäsittely (järjestelmän vaste ja toipuminen ohjelmointi-, sovellus- ja tietoliikennevirheistä, joka johtaa esimerkiksi virheilmoitukseen). Testit suunnitellaan testaamaan näitä määriteltyjä käyttäytymistapoja (peruskäyttäytyminen, poikkeustilanteet tai vaihtoehdot, sekä virheenkäsittely).

Kattavuutta voidaan mitata laskemalla testeillä testattujen käyttötapauksissa kuvattujen käyttäytymismallien määrä, jakamalla se käyttäytymismallien kokonaismäärällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna.

Lisää tietoa käyttötapaustestauksen kattavuuskriteereistä, ks. ISTQB-ATA Jatkotason sertifikaattisisältö, Testausasiantuntija.

4.3 *Lasilaatikkotekniikat*

Lasilaatikkotestaus perustuu testattavan kohteen sisäiseen rakenteeseen. Lasilaatikkotekniikoita voidaan käyttää kaikilla testaustasoilla, mutta tässä alaluvussa käsiteltävää kahta koodiin liittyvää tekniikkaa käytetään tyypillisesti yksikkötestaustasolla. On olemassa edistyneempiä tekniikoita, joita käytetään joissakin turvallisuuskriittisissä, tehtävien kannalta oleellisissa tai suurta eheyttä vaativissa ympäristöissä perusteellisemman kattavuuden aikaansaamiseksi, mutta niitä ei käsitellä tässä yhteydessä. Lisää tietoa näistä tekniikoista, ks. ISTQB Jatkotason sertifikaattisisältö, Tekninen Testausasiantuntija.

4.3.1 Lausetestaus ja -kattavuus

Lausetestauksessa testataan ohjelmakoodin suoritettavat lauseet. Kattavuus mitataan laskemalla testien suorittamien lauseiden lukumäärä, jakamalla se kaikkien testauksen kohteen sisältämien suoritettavien lauseiden lukumäärällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna.

4.3.2 Päätöstestaus ja -kattavuus

Päätöstestauksessa suoritetaan koodin sisältämiä päätöksiä ja testataan koodi, joka suoritetaan päätösten tulosten perusteella. Tämän tekemiseksi, testitapaukset seuraavat kontrollivirtoja, jotka lähtevät päätöskohdasta (esim. IF-lauseessa tarvitaan yksi testi tosi-vaihtoehtoa ja yksi epätosi-vaihtoehtoa varten, CASE-lauseessa jokainen mahdollinen tulos vaatii oman testitapauksensa, oletuslopputulos mukaan luettuna).

Kattavuus mitataan laskemalla testien suorittamien päätösvaihtoehtojen lukumäärä, jakamalla se kaikkien testauksen kohteen sisältämien päätösvaihtoehtojen lukumäärällä ja kertomalla sadalla, mikä kertoo kattavuuden prosenttilukuna.

4.3.3 Lause- ja päätöstestauksen merkitys

Kun saavutetaan 100 % lausekattavuus, se varmistaa, että koodin kaikki suoritettavat lauseet on testattu ainakin kerran, mutta se ei takaa, että koko päätöksentekologiikka on testattu. Näistä kahdesta tässä sertifikaattisisällössä esitellystä lasilaatikkotekniikasta lausetestaus voi tuottaa pienemmän kattavuuden kuin päätöstestaus.

Kun saavutetaan 100 % päätöskattavuus, on suoritettu kaikki päätösvaihtoehdot, mikä tarkoittaa sekä tosi-vaihtoehtoa että epätosi-vaihtoehtoa, vaikka epätosi-ehtoa ei seuraisikaan eksplisiittinen epätosi-lause (esim. tilanne, jossa IF-lausetta ei koodissa seuraa ELSE-haara). Lausekattavuus auttaa löytämään vikoja koodista, jota muut testit eivät käyneet läpi. Päätöskattavuus auttaa löytämään koodista vikoja tilanteessa, jossa muut testit eivät ole suorittaneet sekä tosi- että epätosi-vaihtoehtoja.

100 % päätöskattavuuden saavuttaminen takaa 100 % lausekattavuuden (mutta ei toisinpäin).

4.4 *Kokemuspohjaiset tekniikat*

Kokemuspohjaisessa testauksessa testit luodaan testaajien osaamisen ja intuition sekä samankaltaisista sovelluksista ja teknologioista kertyneen osaamisen perusteella. Nämä tekniikat voivat auttaa tunnistamaan testejä, joita ei ole helppo tunnistaa muilla järjestelmällisemmillä tekniikoilla. Testaajan lähestymistavoista ja kokemuksesta riippuen nämä tekniikat voivat tuottaa hyvinkin erilaisia kattavuuden ja tehokkuuden asteita. Kattavuuden arviointi voi olla vaikeaa ja mittaaminen jopa mahdotonta näitä tekniikoita käytettäessä.

Seuraavissa alaluvuissa esitellään yleisesti käytettyjä kokemuspohjaisia tekniikoita.

4.4.1 Virheenarvaus

Virheenarvaus on tekniikka, jolla pyritään ennakoimaan virheiden, vikojen ja häiriöiden esiintymisiä käyttämällä testaajan tietämystä esimerkiksi

- sovelluksen aikaisemmasta toiminnasta
- kehittäjien tyypillisesti tekemistä virheistä
- häiriöistä, joita on esiintynyt muissa sovelluksissa.

Järjestelmällinen lähestymistapa virheenarvaukseen on luoda lista mahdollisista virheistä, vioista ja häiriöistä, ja suunnitella testejä, jotka pyrkivät paljastamaan kyseiset häiriöt sekä ne aiheuttaneet viat. Näitä virhe-, vika- ja häiriölistoja voidaan laatia kokemuksen, vika- ja häiriötietojen pohjalta sekä perustuen yleiseen tietämykseen siitä, miksi ohjelmistot eivät toimi.

4.4.2 Tutkiva testaus

Tutkivassa testauksessa vapaamuotoisia (ei ennalta määriteltyjä) testejä suunnitellaan, suoritetaan, kirjataan ja arvioidaan dynaamisesti testauksen aikana. Testauksen tuloksia käytetään lisätiedon saamiseksi komponentista tai järjestelmästä sekä testien laatimiseksi sellaisille alueille, jotka voivat tarvita lisää testausta.

Tutkivassa testauksessa käytetään joskus istuntopohjaista testausta rakenteellisuuden tuomiseksi testaukseen. Istuntopohjaisessa testauksessa tutkiva testaus tehdään rajatuissa aikaikkunoissa ja testaaja käyttää testausohjetta, joka sisältää testauksen tavoitteet testauksen ohjaamiseksi. Testaaja voi käyttää istuntolomakkeita suoritettujen askelien sekä esiin tulleiden havaintojen dokumentointiin.

Tutkiva testaus on hyödyllisintä silloin, kun määrittelykuvauksia on vähän tai ne ovat riittämättömiä, tai testauksella on merkittäviä aikapaineita. Tutkiva testaus on myös hyödyllistä täydentämässä muita, muodollisempia testaustekniikoita.

Tutkiva testaus liittyy vahvasti reaktiivisiin testauksen lähestymistapoihin (ks. alaluku 5.2.2). Tutkivassa testauksessa voidaan käyttää mustalaatikko-, lasilaatikko- ja kokemuspohjaisia tekniikoita.

4.4.3 Tarkistuslistoihin pohjautuva testaus

Tarkistuslistoihin pohjautuvassa testauksessa testaajat suunnittelevat, toteuttavat ja suorittavat testit tarkistuslistassa kuvattujen testattavien tilanteiden kattamiseksi. Testaajat luovat uuden tarkistuslistan tai laajentavat olemassa olevaa tarkistuslistaa osana testianalyysiä, mutta he voivat myös käyttää olemassa olevaa tarkistuslistaa muokkaamatta sitä. Tällaiset tarkistuslistat voivat pohjautua kokemukseen, tietämykseen siitä, mikä on käyttäjän kannalta tärkeää, tai ymmärrykseen siitä, miksi ja miten ohjelmisto toimii väärin.

Tarkistuslistoja voidaan luoda tukemaan eri testaustyyppejä, mukaan luettuna toiminnallinen ja ei-toiminnallinen testaus. Yksityiskohtaisten testitapausten puuttuessa tarkistuslistoihin perustuva testaus voi tarjota suuntalinjat ja edes jonkinasteista yhdenmukaisuutta. Koska tarkistuslistat ovat ylätason listoja, varsinaisessa testauksessa tapahtuu todennäköisesti jonkinlaista vaihtelua, mikä johtaa mahdollisesti suurempaan kattavuuteen mutta pienempään toistettavuuteen.

5 Testauksen hallinta — 225 minuuttia

Avainsanat

aloitusehdot, kokoonpanonhallinta, lopetusehdot, projektiriski, riski, riskipohjainen testaus, riskitaso, testaaja, testauksen edistymisraportti, testauksen hallinta, testauksen lähestymistapa, testauksen seuranta, testauksen suunnittelu, testauksen työmäärän arviointi, testauksen yhteenvetoraportti, testauspäällikkö, testausstrategia, testaussuunnitelma, tuoteriski, vikojenhallinta

Oppimistavoitteet: Testauksenhallinta

5.1. Testausorganisaatio

FL-5.1.1 (K2) Selittää riippumattoman testauksen hyödyt ja haitat

FL-5.1.2 (K1) Tunnistaa testauspäällikön ja testaajan tehtävät

5.2 Testaussuunnittelu ja työmääräarviointi

FL-5.2.1 (K2) Kuvata testaussuunnitelman tarkoitus ja sisältö

FL-5.2.2 (K2) Kuvata eri testauksen lähestymistapojen väliset erot

FL-5.2.3 (K2) Antaa esimerkkejä mahdollisista aloitus- ja lopetusehdoista

FL-5.2.4 (K3) Käyttää priorisointiin sekä teknisiin ja loogisiin riippuvuuksiin liittyvää tietoa hyödyksi annetun testijoukon suorituksen aikatauluttamisessa

FL-5.2.5 (K1) Tunnistaa tekijät, jotka vaikuttavat testaukseen liittyvään työmäärään

FL-5.2.6 (K2) Selittää kahden eri työmäärien arviointitekniikan, metriikkalähtöisen ja asiantuntijalähtöisen tekniikan, väliset erot

5.3 Testauksen seuranta ja hallinta

FL-5.3.1 (K1) Tunnistaa testauksessa käytettävät metriikat

FL-5.3.2 (K2) Vetää yhteen testausraporttien tarkoitukset, sisällöt ja kohdeyleisöt

5.4 Kokoonpanonhallinta

FL-5.4.1 (K2) Kuvata, miten kokoonpanonhallinta tukee testausta

5.5 Riskit ja testaus

FL-5.5.1 (K1) Määritellä riskitaso todennäköisyyttä ja vaikutusta käyttämällä

FL-5.5.2 (K2) Erottaa projektiriskit ja tuoteriskit toisistaan

FL-5.5.3 (K2) Kuvata esimerkkien avulla, kuinka tuoteriskien analysointi voi vaikuttaa testauksen perusteellisuuteen ja laajuuteen

5.6 Vikojenhallinta

FL-5.6.1 (K3) Kirjoittaa vikaraportti, jossa kuvataan testauksen aikana löydetyt viat

5.1 *Testausorganisaatio*

5.1.1 Riippumaton testaus

Testaustehtäviä voivat suorittaa henkilöt, jotka toimivat erityisissä testausrooleissa, tai muissa rooleissa toimivat ihmiset (esim. asiakkaat). Tietty riippumattomuuden määrä tehostaa usein testaajan virheenlöytökykyä materiaalin laatijan ja testaajan kognitiivisten harhojen erojen vuoksi (ks. alaluku 1.5). Riippumattomuus ei kuitenkaan ole tuttuuden korvike, ja kehittäjät voivat tehokkaasti löytää monia vikoja omasta koodistaan.

Testauksen riippumattomuuden asteet sisältävät seuraavia vaihtoehtoja (matalimmasta riippumattomuuden asteesta korkeimpaan):

- ei riippumattomia testaajia: ainoa käytettävissä oleva testauksen muoto on, että kehittäjät testaavat oman koodinsa
- riippumattomat kehittäjät tai testaajat kehitystiimissä tai projektitiimissä: esimerkkinä kehittäjät, jotka testaavat tiimin toisten kehittäjien tuotoksia
- organisaation sisäinen riippumaton testaustiimi tai ryhmä, joka raportoi projektin johdolle tai liikkeenjohdolle
- liiketoiminnan tai käyttäjäyhteisön edustajista kootut riippumattomat testaajat tai testaajat, joilla on erityisosaamista tietyistä testaustyypeistä, kuten käytettävyys, tietoturva, suorituskyky, säännelyt ja niiden noudattaminen, tai siirrettävyys
- ulkoisesta organisaatiosta tulevat testaajat, jotka työskentelevät joko toimeksiantajan tiloissa (sisäisesti ulkoistettu) tai muualla (ulkoistettu).

Useimpien projektien kannalta on yleensä parasta käyttää useita testaustasoja, joista osassa testauksen hoitavat riippumattomat testaajat. Kehittäjien pitäisi osallistua testaukseen erityisesti alemmilla tasoilla, jotta he voisivat vaikuttaa oman työnsä laatuun.

Tavat, joilla riippumaton testaus toteutetaan, vaihtelevat käytettävän ohjelmistokehityksen elinkaarimallin mukaan. Esimerkiksi ketterässä kehityksessä testaajat voivat olla osa kehitystiimiä. Joissakin ketteriä menetelmiä käyttävissä organisaatioissa näiden testaajien voidaan myös katsoa olevan osa laajempaa riippumatonta testaustiimiä. Lisäksi tällaisissa organisaatioissa tuoteomistajat voivat suorittaa joka iteraation lopussa hyväksymistestauksen käyttäjätarinoiden kelpuuttamiseksi.

Riippumattoman testauksen mahdollisiin hyötyihin kuuluvat seuraavat seikat:

- Riippumattomat testaajat tunnistavat todennäköisesti erilaisia häiriöitä kuin kehittäjät heidän erilaisten taustojensa, teknisten näkökulmien ja ennakkonäkemysten vuoksi.
- Riippumaton testaaja voi todentaa, haastaa tai todistaa vääräksi olettamuksia, joita sidosryhmät ovat tehneet järjestelmän määrittelyn ja toteutuksen aikana.

Riippumattoman testauksen mahdollisiin haittoihin kuuluvat seuraavat seikat:

- Testaus on erossa toteutustiimistä, mikä johtaa yhteistyön puutteeseen, toteutustiimille annettavan palautteen viivästymisiä tai jopa vihamieliseen suhteeseen toteuttajien kanssa.
- Kehittäjät voivat menettää vastuuntuntonsa laadunvarmistuksen suhteen.
- Riippumattomat testaajat voidaan nähdä pullonkaulana tai heitä voidaan syyttää julkaisun viivästymisestä.
- Riippumattomilta testaajilta saattaa puuttua tärkeää tietoa (esim. liittyen testattavaan kohteeseen).

Monet organisaatiot pystyvät onnistuneesti saavuttamaan riippumattoman testauksen edut ja välttämään siihen liittyvät haitat.

5.1.2 Testauspäällikön ja testaajan tehtävät

Tässä sertifikaattisisällössä käsitellään kahta testaukseen liittyvää roolia, testauspäälliköitä ja testaajia. Tehtävät ja toimenpiteet, joita näissä kahdessa roolissa toimivat ihmiset tekevät, riippuvat projektin ja tuotteen tilanteesta, rooleissa toimivien ihmisten osaamisesta sekä organisaatiosta.

Testauspäällikön tehtäviin kuuluu yleisvastuu testausprosessista ja testaustehtävien menestyksekkäästä johtamisesta. Testauksen hallinnan roolia voi hoitaa ammattimainen testauspäällikkö tai projektipäällikkö, kehityspäällikkö tai laatupäällikkö. Isommissa projekteissa tai organisaatioissa useat testaustiimit voivat raportoida testauspäällikölle, testauksen valmentajalle tai testauskoordinaattorille, ja jokaista tiimiä johtaa testauksen johtaja tai päätestaaja.

Tyypillisiin testauspäällikön tehtäviin voivat kuulua seuraavat:

- organisaation testauspolitiikan ja testausstrategian laatiminen tai katselmointi
- testaustoimenpiteiden suunnitteleminen tilanteen mukaan testauksen tavoitteet ja riskit huomioon ottaen. Tähän saattaa kuulua testauksen lähestymistapojen valinta, testaukseen tarvittavan ajan, työmäärän ja kustannusten arviointi, resurssien hankinta, käytettävien testaustasojen ja testikierrosten määrittäminen sekä virheidenhallinnan suunnittelu.
- testaussuunnitelmien kirjoittaminen ja päivittäminen
- testaussuunnitelmien koordinointi projektipäälliköiden, tuoteomistajien ja muiden kanssa
- testausnäkökulman tuominen muihin projektin tehtäviin, kuten integraation suunnitteluun
- testien analysoinnin, suunnittelun, toteutuksen ja suorituksen käynnistäminen, testauksen edistymisen ja tulosten seuranta sekä päätöskriteerien (tai valmiin määritelmän) tilanteen tarkistaminen
- testauksen edistymisraporttien ja yhteenvetoraporttien laatiminen kerätyn tiedon perusteella sekä raporttien jakelu
- suunnittelun mukauttaminen testien tulosten ja edistymisen perusteella (nämä on joskus dokumentoitu testauksen edistymisraporteissa ja/tai testauksen yhteenvetoraportissa, kun kyse on muusta projektissa jo valmiiksi saadusta testauksesta) sekä tarvittaviin toimenpiteisiin ryhtyminen testauksen hallinnoimiseksi
- virheenhallintajärjestelmän sekä riittävän kokoonpanonhallintajärjestelmän pystyttämisen tukeminen
- sopivien metriikoiden käyttöönotto testauksen edistymisen mittaamiseksi sekä testauksen ja tuotteen laadun arvioimiseksi
- testausprosessin tukemiseksi tarvittavien työkalujen valinnan ja käyttöönoton tukeminen, mukaan lukien työkalun valintaan (sekä mahdolliseen hankintaan ja/tai tukeen) käytettävän budjetin ehdottaminen, ajan ja työpanoksen varaaminen pilottiprojektille sekä jatkuvan tuen tarjoaminen työkalun käyttöä varten
- testausympäristö(je)n toteutuksesta päättäminen
- testaajien, testaustiimin ja testausammatin puolestapuhuminen ja asian ajaminen organisaatiossa
- testaajien osaamisen ja urapolkujen kehittäminen (koulutussuunnitelmien, suoritusarviointien, valmennuksen yms. avulla).

Testauspäällikön roolin toteutustapa vaihtelee ohjelmistokehityksen elinkaarimallin mukaan. Esimerkiksi Ketterässä kehityksessä Ketterä tiimi ja siinä työskentelevä testaaja hoitavat usein osan edellä mainituista tehtävistä, erityisesti ne, jotka liittyvät tiimin tekemään jokapäiväiseen testaustyöhön. Jotkut tehtävät saattavat koskea useita tiimejä tai koko organisaatiota tai ne voivat liittyä henkilöstöhallinnollisiin asioihin, ja niitä voivat hoitaa kehitystiimin ulkopuoliset testauspäälliköt, joita joskus kutsutaan testauksen valmentajiksi. Lisää tietoa testausprosessin hallinnasta, ks. Black 2009.

Tyypillisiin testaajan tehtäviin voivat kuulua seuraavat:

- testaussuunnitelmien katselmointi ja niihin myötävaikuttaminen
- vaatimusten, käyttäjätarinoiden ja hyväksymiskriteerien, määritysten sekä mallien (eli testauksen pohjamateriaalin) analysointi, katselmointi ja arviointi testattavuuden näkökulmasta
- testattavien tilanteiden tunnistaminen ja dokumentointi sekä jäljitettävyyden varmistaminen testitapausten, testattavien tilanteiden ja testauksen pohjamateriaalin välille
- testiympäristön suunnittelu, pystyttäminen ja todentaminen, usein yhteistyössä järjestelmähallinnan ja verkonhallinnan kanssa
- testitapausten ja testiproseduurien suunnittelu ja toteutus
- testiaineiston valmistelu ja hankinta
- yksityiskohtaisen testien suoritusaikataulun laatiminen
- testien suorittaminen, tulosten arviointi sekä odotetuista tuloksista poikkeavien havaintojen kirjaaminen
- sopivien työkalujen käyttäminen testausprosessin helpottamiseksi
- testien automatisointi tarpeen mukaan (tarvittaessa kehittäjän tai testiautomaatioasiantuntijan tukemana)
- ei-toiminnallisten ominaisuuksien arviointi, kuten suorituskyvyn tehokkuus, luotettavuus, käytettävyys, tietoturva, yhteensopivuus ja siirrettävyys
- toisten laatimien testien katselmointi.

Ihmiset, jotka työskentelevät testianalyysin, testisuunnittelun, tiettyjen testityyppien tai testausautomaation parissa, voivat olla näiden roolien asiantuntijoita. Tuotteeseen ja projektiin liittyvien riskien sekä valitun ohjelmistokehityksen elinkaarimallin mukaan eri ihmiset voivat toimia testaajina eri testaustasoilla. Esimerkiksi yksikkötestaus- ja komponentti-integraatiotestaustasolla testaajan roolin hoitaa usein kehittäjä. Hyväksymistestaustasolla testaajan roolin hoitavat usein liiketoiminta-analyytikot, toimialan asiantuntijat sekä käyttäjät. Järjestelmätestaustasolla sekä järjestelmäintegraatiotestaustasolla testaajan roolista vastaa usein riippumaton testaustiimi. Käyttöön soveltuvuuden hyväksymistestaustasolla testaajan roolin hoitaa usein operaattori ja/tai järjestelmänvalvoja.

5.2 *Testaussuunnittelu ja työmääräarviointi*

5.2.1 Testaussuunnitelman tarkoitus ja sisältö

Testaussuunnitelma linjaa kehitys- ja ylläpitoprojekteissa tehtävät testaustoimenpiteet. Suunnitteluun vaikuttavat organisaation testauspolitiikka ja testausstrategia, käytettävät ohjelmistokehityksen elinkaari ja menetelmät (ks. alaluku 2.1), testauksen laajuus, tavoitteet, riskit, rajoitteet, kriittisyys, testattavuus sekä resurssien saatavuus.

Mitä pidemmälle projekti ja testaussuunnittelu edistyvät, sitä enemmän tietoa on saatavilla ja sitä tarkempia yksityiskohtia voidaan sisällyttää testaussuunnitelmaan. Testauksen suunnittelu on jatkuva tehtävä ja sitä tehdään läpi tuotteen elinkaaren. (Huomaa, että tuotteen elinkaari voi jatkua yli projektin rajojen niin, että ylläpitovaihe sisältyy siihen.) Palautetta testaustehtävistä pitäisi käyttää muuttuvien riskien tunnistamisessa, jotta suunnittelua voidaan muokata vastaavasti. Suunnittelu voidaan dokumentoida kokonaistestaussuunnitelmaan ja erillisiin tasokohtaisiin testaussuunnitelmiin, kuten järjestelmätestaus- ja hyväksymistestaussuunnitelmat, tai erillisiin testaustyyppikohtaisiin suunnitelmiin, kuten käytettävyystestaus- ja suorituskykytestaussuunnitelmat. Testauksen suunnittelutehtäviin voivat kuulua seuraavat ja jotkut näistä voidaan dokumentoida testaussuunnitelmaan:

- testauksen laajuuden, tavoitteiden ja riskien määrittely
- testauksen yleisten lähestymistapojen määrittely
- testaustehtävien integrointi ja koordinointi ohjelmiston elinkaaren tehtävien kanssa
- päätösten tekeminen sen suhteen, mitä testataan, ketä ihmisiä ja mitä muita resursseja tarvitaan eri testaustehtävien suorittamiseen, sekä kuinka testaustehtävät tullaan suorittamaan

- testianalyysin, testien suunnittelun, valmistelun ja suorituksen sekä arviointitehtävien aikataulutaminen, joko määräpäiviksi (esim. peräkkäismallisessa kehityksessä) tai joka iteraation tilanteen perusteella (esim. iteratiivisessa kehityksessä)
- testauksen seurannassa ja hallinnassa tarvittavien mittareiden valinta
- budjetin laatiminen testaustehtävien osalta
- testausdokumentaation yksityiskohtaisuuden tason ja rakenteen määrittely (esim. mallipohjia tai mallidokumentteja käyttämällä).

Testaussuunnitelmien sisältö vaihtelee ja se voi kattaa yllä kuvattua enemmän asioita. Esimerkkitestaussuunnitelmia löytyy ISO-standardista ISO/IEC/IEE 29119-3.

5.2.2 Testausstrategiat ja lähestymistavat

Testausstrategia tarjoaa yleistason kuvauksen kuvauksen testausprosessista yleensä tuote- tai organisaatiotasolla. Tyypillisiin testauksen lähestymistapoihin kuuluvat seuraavat:

- **Analyyttinen:** Tämä testausstrategiatyyppi pohjautuu jonkin tekijän (esim. vaatimus tai riski) analysointiin. Riskipohjainen testaus on esimerkki analyyttisestä lähestymistavasta, jossa testit suunnitellaan ja priorisoidaan riskitason perusteella.
- **Mallipohjainen:** Tässä testauksen lähestymistavassa testit suunnitellaan jonkin tuotteen mallin tai vaaditun ominaisuuden, kuten toiminnon, liiketoimintaprosessin, sisäisen rakenteen tai ei-toiminnallisen ominaisuuden (esim. luotettavuus) perusteella. Esimerkkeihin tällaisista malleista kuuluvat liiketoimintamallit, tilamallit ja luotettavuuden kasvumallit.
- **Menetelmällinen:** Tässä lähestymistavassa käytetään järjestelmällisesti jotain ennalta määriteltyjä testijoukkoja tai testattavia tilanteita, kuten esimerkiksi tyypillisistä tai todennäköisistä häiriötyypeistä laadittuja vikaluokittelumalleja, tärkeiden laatuominaisuuksien listoja tai yrityksen omia mobiilisovelluksiin tai web-sivuihin liittyviä ulkoasustandardeja.
- **Prosesseihin tai standardeihin perustuva:** Tähän lähestymistapaan kuuluu testien analysointi, suunnittelu ja valmistelu ulkoisten sääntöjen ja standardien perusteella, kuten esimerkiksi teollisuusalaan liittyvät standardit, prosessidokumentaatio, testauksen pohjamateriaalin perusteellinen tunnistaminen ja käyttö tai muut organisaatioon kohdentuvat tai sen määrittämät prosessit tai standardit.
- **Ohjaava (tai konsultatiivinen):** Tämä testauksen lähestymistapa nojautuu pääasiassa neuvoihin, opastukseen tai ohjeisiin, joita antavat sidosryhmät, liiketoiminta-asiantuntijat tai teknologia-asiantuntijat, jotka voivat olla testaustiimin tai koko organisaation ulkopuolisia.
- **Regressiota ehkäisevä:** Tätä testauksen lähestymistapaa ohjaa halu välttää olemassa olevien kyvykkyyksien heikkenemistä. Tähän lähestymistapaan kuuluvat olemassa olevan testausmateriaalin (erityisesti testitapausten ja testiaineiston) uudelleenkäyttö, regressiotestien mittava automatisointi sekä vakiintuneet testijoukot.
- **Reaktiivinen:** Tässä lähestymistavassa testaus reagoi testattavana olevaan komponenttiin tai järjestelmään ja testien suorituksen aikana tapahtuviin tilanteisiin eikä niinkään ole ennalta suunniteltua (kuten edellä kuvatuissa strategioissa). Testit suunnitellaan, toteutetaan ja voidaan suorittaa välittömästi aikaisempien testitulosten pohjalta saadun tiedon perusteella. Tutkiva testaus on yleinen reaktiivisissa lähestymistavoissa käytettävä tekniikka.

Sopiva testauksen lähestymistapa luodaan usein yhdistämällä useita näistä lähestymistavoista. Esimerkiksi riskipohjainen testaus (analyyttinen lähestymistapa) voidaan yhdistää tutkivan testauksen kanssa (reaktiivinen lähestymistapa); ne täydentävät toisiaan ja voivat johtaa tehokkaampaan testaukseen, kun niitä käytetään yhdessä.

Siinä missä testausstrategia tarjoaa testausprosessista yleistetyn kuvauksen, testauksen lähestymistapa räätälöi testausstrategian tiettyä projektia tai julkaisua varten. Testauksen lähestymistapa on aloituskohta, kun lähdetään valitsemaan testaustekniikoita, testaustasoja ja testityyppejä, ja kun määritetään aloitus- ja lopetusehtoja (tai työstövalmiin (Definition of Ready, DoR) tai valmiin (Definition of Done, DoD) määritelmää). Strategian räätälöinti perustuu päätöksiin, joita tehdään projektin monimutkaisuuden ja tavoitteiden, kehitettävän tuotteen tyypin sekä tuoteriskien analyysin perusteella. Valittu lähestymistapa

riippuu tilanteesta ja valinnassa voidaan ottaa huomioon esimerkiksi riskit, turvallisuus, käytettävissä olevat resurssit ja osaaminen, teknologia, järjestelmän luonne (esim. räätälöity järjestelmä vs. valmisohjelmisto), testauksen tavoitteet sekä lait ja säädökset.

5.2.3 Aloitus- ja lopetuskriteerit (Työstövalmiin ja Valmiin määritelmät)

Jotta ohjelmiston ja testauksen laatua voitaisiin hallita tehokkaasti, on järkevää määritellä kriteerit sille, milloin tietyn testaustehtävän pitäisi alkaa ja milloin tehtävä on saatu valmiiksi. Aloituskriteerit (kutsutaan tyypillisesti Ketterässä kehityksessä Työstövalmiin määritelmäksi) määrittelevät tiettyyn testaustehtävään liittyvät ennakkoehdot. Jos aloituskriteerit eivät täyty, on todennäköistä, että tehtävä osoittautuu odotettua vaikeammaksi, aikaavievämmäksi, kalliimmaksi ja riskialttiimmaksi. Päätöskriteerit (kutsutaan tyypillisesti Ketterässä kehityksessä Valmiin määritelmäksi) määrittelevät, mitkä ehdot pitää täyttää, jotta testaustaso tai testijoukko voidaan todeta valmiiksi. Aloitus- ja lopetuskriteerit pitäisi määritellä jokaiselle testaustasolle ja testaustyypille, ja ne eroavat toisistaan testauksen tavoitteiden mukaan.

Tyypillisiä aloituskriteerejä ovat:

- testattavien vaatimusten, käyttäjätarinoiden ja/tai mallien (esim. kun käytetään mallipohjaista testauksen lähestymistapaa) saatavuus
- sellaisten testattavien kohteiden saatavuus, jotka ovat täyttäneet aiempien testaustasojen lopetuskriteerit
- testausympäristön saatavuus
- tarvittavien työkalujen saatavuus
- testiaineiston ja muiden tarvittavien resurssien saatavuus.

Tyypillisiä lopetuskriteerejä ovat:

- Suunnitellut testit on suoritettu.
- Määritelty kattavuus (koskien esim. vaatimuksia, käyttäjätarinoita, hyväksymiskriteereitä, riskejä, koodia) on saavutettu.
- Selvittämättömien vikojen määrä on sallituissa rajoissa.
- Arvioitu jäljellä olevien vikojen määrä on riittävän matala.
- Luotettavuuden, suorituskyvyn tehokkuuden, käytettävyyden, tietoturvan ja muiden oleellisten laatuominaisuuksien arvioidut tasot ovat riittävät.

Vaikka päätöskriteerit eivät täyttyisikään, on myös tyypillistä, että testaustehtäviä rajoitetaan budjettiylitysten, varatun ajan täyttymisen ja/tai tuotteen markkinoille saamiseen liittyvien paineiden vuoksi. Näissä olosuhteissa voi testauksen päättäminen olla hyväksyttävää, jos projektin sidosryhmät ja liiketoiminnan edustajat ovat katselmoineet riskit, jotka liittyvät tuotantoon siirtymiseen ilman lisätestausta, ja hyväksyneet ne.

5.2.4 Testien suoritusaikataulu

Sen jälkeen, kun erilaiset testitapaukset ja testiproseduurit on tehty valmiiksi (ja osa testiproseduureista mahdollisesti automatisoitu) ja järjestetty testijoukoiksi, testijoukoille voidaan laatia aikataulu, joka määrittelee niiden suoritusjärjestyksen. Testien suoritusaikataulussa pitäisi ottaa huomioon sellaisia tekijöitä kuin priorisointi, riippuvuudet, varmistustestaus, regressiotestaus sekä testien suorituksen kannalta kaikkein tehokkain järjestys.

Ihannetilanteessa testitapaukset järjestettäisiin suoritettavaksi niiden prioriteettitason mukaisesti niin, että korkeimman prioriteetin testit suoritettaisiin tyypillisesti ensin. Tämä käytäntö ei kuitenkaan ehkä toimi, jos testitapauksilla tai testauksen kohteena olevilla ominaisuuksilla on riippuvuuksia. Jos korkean prioriteetin testitapaus on riippuvainen alemman prioriteetin testitapauksesta, alemman prioriteetin testitapaus täytyy suorittaa ensin. Samoin jos testitapausten kesken on riippuvuuksia, ne täytyy järjestää riippuvuuksien perusteella huolimatta niiden välisistä prioriteeteista. Varmistustestauksen ja regressiotestauksen testit on myös priorisoitava muutoksista saatavan nopean palautteen tärkeyden perusteella, mutta jälleen on otettava huomioon myös riippuvuudet.

Joissain tapauksissa on mahdollista laatia erilaisia testisarjoja, joiden tehokkuuden taso vaihtelee. Näissä tilanteissa on tehtävä kompromisseja testien suorituksen tehokkuuden ja priorisoinnin noudattamisen välillä.

5.2.5 Testauksen työmäärään vaikuttavat tekijät

Testauksen työmäärän arviointiin kuuluu tietyn projektin, julkaisun tai iteraation testauksen tavoitteiden saavuttamiseksi tarvittavan testaukseen liittyvän työn määrän ennustaminen. Työmäärään vaikuttaviin seikkoihin voivat kuulua tuotteen, kehitysprosessin ja ihmisten ominaisuudet sekä testauksen tulokset, kuten on kuvattu alla.

Tuotteen ominaisuudet

- Tuotteeseen liittyvät riskit
- Testauksen pohjamateriaalin laatu
- Tuotteen koko
- Tuotteeseen liittyvän liiketoiminta-alueen monimutkaisuus
- Laatuominaisuuksiin liittyvät vaatimukset (esim. tietoturva, luotettavuus)
- Testausdokumentaation yksityiskohtaisuuden taso
- Lain- ja sääntelynmukaisuuden noudattamisen vaatimukset.

Kehitysprosessin ominaisuudet

- Organisaation vakaus ja kypsyys
- Käytettävä ohjelmistokehitysmalli
- Testauksen lähestymistavat
- Käytettävät työkalut
- Testausprosessi
- Aikapaine.

Ihmisten ominaisuudet

- Mukana olevien ihmisten osaaminen ja kokemus erityisesti samantyyppisistä projekteista ja tuotteista (esim. liiketoimintaosaaminen)
- Tiimin yhteenkuuluvuus ja johtaminen.

Testitulokset

- Löydettyjen vikojen määrä ja vakavuus
- Tarvittavan uusintatyön määrä.

5.2.6 Työmääräarvioinnin tekniikat

On olemassa joukko arviointitekniikoita, joita voidaan käyttää riittävän testauksen työmäärän arvioimiseksi. Kaksi yleisimmin käytettyä tekniikkaa ovat

- metriikkalähtöinen tekniikka: testaustyömäärän arviointi aiempien samanlaisten projektien metriikoiden pohjalta tai tyypillisten arvojen perusteella
- asiantuntijalähtöinen tekniikka: testaustyömäärän arviointi testaustehtävistä vastaavien kokemuksen perusteella tai asiantuntijoiden avulla.

Esimerkiksi Ketterässä kehityksessä edistymiskäyrät ovat esimerkki metriikkapohjaisesta lähestymistavasta, jossa työmäärän arvioinnin ja raportoinnin jälkeen sitä käytetään tiimin vauhdin määrittämiseksi, kun ollaan päättämässä työmäärää, jonka tiimi voi tehdä seuraavassa iteraatiossa; suunnittelupokeri on taas esimerkki asiantuntijalähtöisestä lähestymistavasta, jossa tiimin jäsenet arvioivat kokemuksensa

perusteella tietyn ominaisuuden tuottamiseksi tarvittavaa työmäärää (ISTQB-AT Perustason sertifikaattisisällön laajennus, Ketterä testaaja).

Peräkkäismallisissa projekteissa vikojenpoistamismallit ovat esimerkkejä metriikkalähtöisestä lähestymistavasta, jossa vikojen määrä ja niiden poistamiseksi tarvittu aika kirjataan ja raportoidaan, ja sitä käytetään sen jälkeen perustana tulevien samankaltaisten projektien työmäärän arvioinnissa; Wideband Delphi –arviointitekniikka on esimerkki asiantuntijalähtöisestä lähestymistavasta, jossa asiantuntijoista muodostetut ryhmät laativat työmääräarvioita kokemuksensa perusteella (ISTQB-ATM Jatkotason sertifikaattisisältö, Testauspäällikkö).

5.3 *Testauksen seuranta ja hallinta*

Testauksen seurannan tarkoituksena on kerätä tietoa sekä antaa palautetta ja näkyvyyttä testaustehtäviin. Seurattava tieto voidaan kerätä manuaalisesti tai automaattisesti ja sitä pitäisi käyttää testauksen edistymisen arvioinnissa sekä kun mitataan, täyttyvätkö testauksen päätöskriteerit, joita voivat olla esimerkiksi tuoteriskien, vaatimusten tai hyväksymiskriteerien kattavuustavoitteet, tai onko Ketterässä projektissa Valmiin määritelmään liittyvät testaustehtävät tehty valmiiksi.

Testauksen kontrollointi kuvaa kaikki ohjaavat ja korjaavat toimenpiteet, jotka suoritetaan informaation ja kerättyjen ja (mahdollisesti) raportoitujen metriikoiden seurauksena. Toimenpiteet voivat kattaa mitkä tahansa testaustehtävät ja vaikuttaa mihin tahansa muihin ohjelmiston elinkaaren aktiviteettiin tai tehtävään.

Esimerkkejä testauksen kontrolloinnin toimenpiteistä ovat

- testien uudelleenpriorisointi, kun tunnistettu riski toteutuu (esim. ohjelmisto toimitetaan myöhässä)
- testausaikataulun muuttaminen sen perusteella, milloin testiympäristö tai muut resurssit ovat käytettävissä
- uudelleenarviointi, täyttääkö testattava kohde aloitus- tai lopetuskriteerit tehdyn korjaustyön jälkeen.

5.3.1 Testauksessa käytetyt mittarit

Mittaritietoa pitäisi kerätä sekä testitehtävien aikana että niiden lopussa, jotta voidaan arvioida

- eteneminen suunniteltua aikataulua ja budjettia vastaan
- senhetkinen testattavan kohteen laatu
- testauksen lähestymistapojen riittävyys
- testaustehtävien tehokkuus suhteessa tavoitteisiin.

Tyypillisiin testausmetriikoihin kuuluvat

- testitapausten valmisteluun käytetyn työmäärän prosenttiosuus suunnitellusta (tai toteutettujen testitapausten prosenttiosuus suunnitelluista)
- testiympäristön valmisteluun käytetyn työmäärän prosenttiosuus suunnitellusta
- testitapausten suorittaminen (esim. ajettujen/ajamattomien testitapausten määrä, läpäistyt ja epäonnistuneet testitapaukset sekä läpäistyt ja epäonnistuneet testattavat tilanteet)
- vikatiedot (esim. vikatiheys, löydetyt ja korjatut viat, häiriötiheys ja uudelleentestauksen tulokset)
- vaatimusten, käyttäjätarinoiden, hyväksymiskriteerien, riskien tai koodin testikattavuus
- tehtävien valmiusaste, resurssien allokointi ja käyttö sekä työmäärä
- testauksen kustannukset, mukaan luettuna kustannukset verrattuna seuraavan vian löytämisen tuottamiin hyötyihin sekä kustannukset verrattuna seuraavan testin suorittamisen tuottamiin hyötyihin.

5.3.2 Testiraporttien tarkoitus, sisältö ja kohderyhmä

Testauksen raportoinnin tarkoituksena on koota yhteen ja viestiä testaustehtäviin liittyvää tietoa sekä testaustoimenpiteiden (esimerkiksi testaustason) aikana että niiden päättyessä. Testaustoimenpiteiden aikana tehtyä testiraporttia voidaan kutsua testauksen edistymisraportiksi, kun taas testiraporttia, joka tehdään testaustoimenpiteen päättyessä, voidaan kutsua testauksen yhteenvetoraportiksi.

Testauspäällikkö toimittaa testauksen seurannan ja hallinnan aikana säännöllisesti edistymisraportteja sidosryhmille. Testauksen edistymisraporttien ja yhteenvetoraporttien tavanomaisen sisällön lisäksi tyypillinen testauksen edistymisraportti voi sisältää myös seuraavia asioita:

- testaustehtävien tila ja edistyminen verrattuna testaussuunnitelmaan
- edistymistä haittaavat seikat
- seuraavalle raportointijaksolle suunniteltu testaus
- testattavan kohteen laatu.

Kun päätöskriteerit on saavutettu, testauspäällikkö laatii testauksen yhteenvetoraportin. Tämä raportti tarjoaa yhteenvedon tehdystä testauksesta viimeisimmän testauksen edistymisraportin sekä muun asiaan liittyvän tiedon perusteella.

Tyypilliset testauksen edistymisraportit ja yhteenvetoraportit voivat sisältää seuraavaa:

- yhteenveto suoritetusta testauksesta
- testausjakson aikaisten tapahtumien kuvaus
- poikkeamat suunnitelmasta, mukaan luettuna poikkeamat liittyen testaustehtävien aikatauluun, kestoon tai työmääriin
- testauksen tila ja tuotteen laatu suhteessa päätöskriteereihin tai Valmiin määritelmään
- seikat, jotka estivät tai estävät edelleen etenemisen
- vikoihin, testitapauksiin, testikattavuuteen, tehtävien etenemiseen ja resurssien käyttöön liittyvät mittaritiedot (esim. kuten kuvattu alaluvussa 5.3.1)
- jäljellä olevat riskit (ks. alaluku 5.5)
- toteutetut uudelleenkäytettävät testauksen tuotokset.

Testausraportin sisältö vaihtelee projektin, organisaation vaatimusten sekä ohjelmistokehityksen elinkaarimallin mukaan. Esimerkiksi monimutkaisessa projektissa, jolla on monia sidosryhmiä tai tarkasti säännellyssä projektissa voidaan tarvita yksityiskohtaisempaa ja perusteellisempaa raportointia kuin nopeassa ohjelmistopäivityksessä. Toinen esimerkki: Ketterässä kehityksessä testauksen edistymisraportointi voidaan toteuttaa tehtävätaulujen, vioista laadittujen yhteenvetojen ja edistymiskäyrien avulla, joista voidaan keskustella päiväpalaverissa (ks. ISTQB-AT Perustason sertifikaattisisällön laajennus, Ketterä testaaja).

Sen lisäksi, että testausraportit pitää räätälöidä projektin tilanteen mukaan, ne pitäisi räätälöidä myös niiden kohderyhmän perusteella. Tekniselle yleisölle tai testaustiimille toimitettavan tiedon laatu ja määrä voi erota siitä, mitä kuvataan johdon yhteenvetoraportissa. Ensimmäisessä tapauksessa yksityiskohtainen tieto vikatyypeistä ja trendeistä voi olla tärkeää. Jälkimmäisessä tapauksessa karkeamman tason raportti (esim. yhteenveto vioista niiden prioriteetin mukaan, budjetista, aikataulusta ja hyväksytyistä/hylätyistä/ei testatuista tilanteista) voi olla sopivampi.

ISO-standardissa ISO/IEC/IEE 29119-3 kuvataan kaksi testausraporttia, testauksen edistymisraportti sekä testauksen valmistumisraportti (tässä sertifikaattisisällössä testauksen yhteenvetoraportti), ja se sisältää kuvauksen kummankin raporttityypin rakenteesta ja esimerkkejä.

5.4 *Kokoonpanonhallinta*

Kokoonpanonhallinnan tarkoitus on luoda ja säilyttää komponentin tai järjestelmän ja testimateriaalin sekä niiden välisten yhteyksien eheys läpi projektin ja tuotteen elinkaaren.

Jotta kokoonpanonhallinta tukisi testausta oikealla tavalla, siihen voi sisältyä sen varmistamista, että

- kaikki testausmateriaalin osat ovat yksilöityjä ja versiohallinnan alaisia, niitä seurataan muutosten varalta ja ne liittyvät toisiinsa
- kaikki testausmateriaalin osat ovat yksilöityjä ja versiohallinnan alaisia, niitä seurataan muutosten varalta, ja ne liittyvät toisiinsa sekä testauksen kohteisiin, jotta jäljitettävyys voidaan säilyttää läpi testausprosessin
- kaikkiin yksilöityihin dokumentteihin ja ohjelmiston osiin viitataan yksiselitteisesti testausdokumentaatiossa.

Kokoonpanon hallinnan toimenpiteet ja infrastruktuuri (työkalut) pitäisi tunnistaa ja ja niiden toteutus aloittaa testaussuunnittelun aikana.

5.5 *Riskit ja testaus*

5.5.1 Riskin määritelmä

Riski tarkoittaa tulevaisuudessa mahdollisesti tapahtuvaa asiaa tai tapahtumaa, jolla on negatiivisia seurauksia. Riskitaso määritellään tapahtuman todennäköisyyden ja vaikutuksen (tapahtumaa seuraavan vahingon) perusteella.

5.5.2 Tuote- ja projektiriskit

Tuoteriskiin liittyy mahdollisuus, että tuotos (esim. määrittely, komponentti, järjestelmä tai testi) voi epäonnistua sen käyttäjien ja/tai sidosryhmien todellisten tarpeiden täyttämisessä. Kun tuoteriskit liittyvät tuotteen tiettyyn laatuominaisuuteen (esim. toiminnalliseen sopivuuteen, luotettavuuteen, suorituskyvyn tehokkuuteen, käytettävyyteen, tietoturvaan, yhteensopivuuteen, ylläpidettävyyteen ja siirrettävyyteen), tuoteriskejä kutsutaan myös laaturiskeiksi. Esimerkkeihin tuoteriskeistä kuuluvat seuraavat:

- Ohjelmisto ei tee sen tarkoitettuja toimintoja määrittelyiden mukaisesti.
- Ohjelmisto ei suorita sen aiottuja toimintoja käyttäjän, asiakkaiden ja/tai sidosryhmien tarpeiden mukaisesti.
- Ohjelmistoarkkitehtuuri ei tue riittävästi jotakin tai useampaa ei-toiminnallista vaatimusta.
- Määrätty laskutoimitus suoritetaan joissakin olosuhteissa väärin.
- Silmukkarakenne on toteutettu väärin.
- Korkeata suorituskykyä edellyttävän tapahtumienkäsittelyjärjestelmän vasteajat eivät ole riittäviä.
- Käyttäjien kokemuksista kerätty palaute ei täytä tuotteelle asetettuja odotuksia.

Projektiriskit liittyvät tilanteisiin, joissa niillä toteutuessaan voi olla negatiivinen vaikutus projektin kykyyn saavuttaa sen tavoitteet. Esimerkkeihin projektiriskeistä kuuluvat seuraavat:

- Projektiin liittyvät seikat:
    - Toimituksessa, tehtävien valmistumisessa tai päätöskriteerien tai valmiin määritelmän täyttämisessä voi tapahtua viivästyksiä.
    - Epätarkat työmääräarviot, varojen siirtäminen korkeamman prioriteetin projekteihin tai yleiset organisaatiossa tapahtuvat kustannusten leikkaukset voivat johtaa riittämättömään rahoitukseen.
    - Myöhäiset muutokset voivat johtaa merkittäviin uudelleentoteutuksiin.
- Organisaatioon liittyvät seikat:
    - Osaamista, koulutusta ja henkilöstöä ei ole riittävästi.
    - Henkilöstöasiat voivat aiheuttaa ristiriitoja ja ongelmia.
    - Käyttäjät, liiketoiminnan edustajat tai asiasisällön asiantuntijat eivät ole käytettävissä liiketoiminnan prioriteettien aiheuttamien ristiriitojen vuoksi.

- Poliittiset seikat:
  - Testaajat eivät kerro riittävästi tarpeistaan ja/tai testituloksista.
  - Kehittäjät ja/tai testaajat eivät hyödynnä testauksesta ja katselmoinneista saatua tietoa (esim. ei paranneta kehityksen tai testauksen käytäntöjä).
  - Vääränlaiset asenteet tai odotukset testausta kohtaan (ei esimerkiksi ymmärretä testauksen aikana löytyneiden vikojen arvoa).
- Tekniset seikat:
  - Vaatimuksia ei ole määritelty riittävän hyvin.
  - Vaatimuksia ei saavuteta vallitsevien rajoitteiden vuoksi.
  - Testiympäristö ei ole ajoissa valmiina.
  - Tietojen konversio, migraation suunnittelu ja niiden työkalutuki ovat myöhässä.
  - Toteutusprosessin heikkoudet voivat vaikuttaa tuotosten, kuten suunnitelmien, koodin, kokoonpanon, testiaineiston ja testitapausten, yhdenmukaisuuteen tai laatuun.
  - Huono vikojenhallinta ja vastaavat ongelmat voivat johtaa kasaantuneisiin vikoihin ja muuhun tekniseen velkaan.
- Toimittajaan liittyvät seikat:
  - Kolmas osapuoli ei pysty toimittamaan tarvittavaa tuotetta tai palvelua tai menee konkurssiin.
  - Sopimukselliset seikat aiheuttavat ongelmia projektille.

Projektiriskit voivat vaikuttaa sekä toteutus- että testaustehtäviin. Joissain tilanteissa projektipäälliköt ovat vastuussa kaikkien projektiriskien käsittelystä, mutta ei ole epätavallista, että testauspäälliköillä on vastuu testaukseen liittyvistä projektiriskeistä.

5.5.3 Riskipohjainen testaus ja tuotteen laatu

Riskejä käytetään testauksen aikana tarvittavan työpanoksen kohdentamiseen. Niitä käytetään, kun päätetään, mistä ja milloin aloittaa testaus, ja kun tunnistetaan alueita, joihin on kiinnitettävä enemmän huomiota. Testausta käytetään epäsuotuisan tapahtuman todennäköisyyden tai tapahtuman vaikutuksen pienentämiseen. Testausta käytetään riskejä pienentävänä toimenpiteenä ja tuottamaan palautetta tunnistetuista riskeistä sekä jäljellä olevista (selvittämättömistä) riskeistä.

Testauksen riskipohjainen lähestymistapa tarjoaa ennakoivia mahdollisuuksia tuoteriskien riskitason pienentämiseksi. Siihen liittyy tuoteriskien analysointi, johon kuuluu tuoteriskien tunnistaminen ja jokaisen riskin todennäköisyyden ja vaikutuksen arviointi. Tuloksena saatavaa tietoa tuoteriskeistä käytetään ohjaamaan testauksen suunnittelua, testitapausten määrittelyä, valmistelua ja suoritusta sekä testauksen seurantaa ja hallintaa. Tuoteriskien varhainen analysointi myötävaikuttaa projektin onnistumiseen.

Riskipohjaisessa lähestymistavassa tuoteriskien analyysin tuloksia käytetään, kun määritetään

- käytettävät testaustekniikat
- käytettävät testaustasot ja testaustyypit (esim. tietoturvatestaus, saavutettavuustestaus)
- testauksen laajuus
- testauksen priorisointi, jotta kriittiset viat löydetään niin aikaisin kuin mahdollista
- voidaanko muita kuin testaustehtäviä käyttää riskien vähentämiseen (esim. koulutuksen tarjoaminen kokemattomille suunnittelijoille).

Riskipohjainen testaus hyödyntää tuoteriskien analysoinnissa projektin sidosryhmien yhteistä tietämystä ja näkemystä. Tuotehäiriön mahdollisuuden minimoimiseksi riskienhallinnan toimenpiteet tarjoavat ohjatun lähestymistavan, jolla voi

- analysoida (ja uudelleenarvioida säännöllisesti), mikä voi mennä pieleen (riskit)

- määritellä, mitkä riskit on tärkeää käsitellä
- toteuttaa toimenpiteet näiden riskien pienentämiseksi
- laatia varasuunnitelmia riskien käsittelemiseksi, jos ne muuttuvat todellisiksi tilanteiksi.

Tämän lisäksi testaus voi tunnistaa uusia riskejä, auttaa määrittämään, mitä riskejä pitäisi pienentää, ja pienentää riskeihin liittyvää epävarmuutta.

5.6 *Vikojenhallinta*

Koska yksi testauksen tavoitteista on löytää vikoja, testauksen aikana löydetyt viat pitäisi kirjata ylös. Vikojen kirjaamistapa voi vaihdella testattavana olevan komponentin tai järjestelmän tilanteen, testaustason sekä käytetyn ohjelmistokehitysmallin perusteella. Kaikki tunnistetut viat pitäisi tutkia ja niitä pitäisi seurata niiden löytämisestä ja luokittelusta ratkaisuun asti (esim. vian korjaus ja ratkaisun onnistunut varmistustestaus, vian siirtäminen myöhempään julkaisuun, hyväksyminen pysyvänä tuotteen rajoitteena jne.). Jotta kaikkia vikoja voidaan hallita ratkaisuun asti, organisaation pitää luoda vikojenhallintaprosessi, johon sisältyy käsittelyn työnkulku ja säännöt vikojen luokittelulle. Kaikkien vikojenhallintaan osallistuvien tulee hyväksyä tämä prosessi, mukaan luettuna suunnittelijat, kehittäjät, testaajat ja tuoteomistajat. Joissain organisaatioissa vikojen kirjaus ja seuranta voi olla hyvin epämuodollista.

Vikojenhallintaprosessin aikana joidenkin raporttien voidaan todeta kuvaavan vääriä positiivisia, ei todellisia vioista johtuneita häiriöitä. Testi voi esimerkiksi epäonnistua, jos verkkoyhteys katkeaa tai tapahtuu aikakatkaisu. Tämä tilanne ei johdu testattavassa kohteessa olevasta viasta mutta se on poikkeama, joka täytyy tutkia. Testaajien pitäisi yrittää minimoida väärien positiivisten raportointi vikoina.

Vikoja voidaan raportoida toteutuksen, staattisen analyysin, katselmointien, dynaamisen testauksen tai ohjelmistotuotteen käytön aikana. Vikaraportin kohteena voivat olla koodissa tai järjestelmässä olevat viat, tai viat kaikenlaisessa dokumentaatiossa, mukaan luettuna vaatimukset, käyttäjätarinat ja hyväksymiskriteerit, toteutuksen aikaiset dokumentit, testausdokumentit, käyttäjän ohjeet tai asennusohjeet. Tehokkaan ja tuottavan vikojenhallintaprosessin aikaansaamiseksi organisaatioiden täytyy määritellä standardit vikojen käsittelyssä käytettäville attribuuteille, luokitteluille ja työnkuluille.

Tyypillisillä vikaraporteilla on seuraavat tavoitteet:

- tarjota kehittäjille ja muille osapuolille tietoa liittyen tapahtuneeseen ei-toivottuun tilanteeseen, jotta nämä voivat tunnistaa sen vaikutukset, eristää ongelman mahdollisimman vähäisellä uusintatestauksella ja korjata mahdollisen vian tarvittavalla tavalla tai muuten ratkaista ongelman
- tarjota testauspäälliköille keino seurata tuotoksen laatua ja testauksen seurauksia (esim. jos vikoja raportoidaan paljon, testaajat ovat käyttäneet paljon aikaa raportointiin testien suorittamisen sijaan ja tarvitaan myös enemmän varmistustestausta)
- tarjota ideoita kehitys- ja testausprosessien kehittämiseksi.

Dynaamisen testauksen aikana laadittu vikaraportti sisältää tyypillisesti seuraavia asioita:

- tunniste
- otsikko ja lyhyt yhteenveto raportoitavasta viasta
- vikaraportin päivämäärä, tekijäorganisaatio ja tekijä
- testauksen kohteen tunniste (testattavana ollut kokoonpanon osa) ja ympäristö
- ohjelmistokehityksen elinkaaren vaihe, jossa vika havaittiin
- vian kuvaus sen toistamiseksi ja ratkaisemiseksi, mukaan luettuna lokit, tietokannan kopiot, näytönkuvat tai nauhoitukset (jos vika löydettiin testin suorituksen aikana)
- odotetut ja todelliset tulokset
- vian vaikutuksen laajuus tai vakavuus sidosryhmien kannalta
- korjauksen kiireellisyys tai prioriteetti

- vikaraportin tila (esim. avoin, lykätty, kaksoiskappale, odottaa korjausta, odottaa varmistustestausta, avattu uudelleen, suljettu)
- johtopäätökset, suositukset ja hyväksynnät
- yleiset huomiot, kuten muut alueet, joihin viasta aiheutuva muutos voi vaikuttaa
- muutoshistoria, kuten projektiryhmän jäsenten suorittamien toimenpiteiden sarja vian eristämiseksi, korjaamiseksi ja korjauksen varmistamiseksi
- viitteet, mukaan luettuna testitapaus, joka paljasti ongelman.

Jotkut näistä tiedoista voivat tulla mukaan raporttiin ja/tai niitä voidaan hallita automaattisesti, mikäli käytetään vianhallintavälineitä, esim. automaattinen tunnisteen määrittäminen, vikaraportin tilan määrittäminen ja päivittäminen työnkulun aikana jne. Staattisen testauksen, erityisesti katselmointien, aikana löydetyt viat dokumentoidaan yleensä eri tavalla, esimerkiksi katselmointipalaverin muistiinpanoihin.

Esimerkki vikaraportin sisällöstä löytyy ISO-standardista ISO/IEC/IEEE 29119-3 (jossa vikaraportteja kutsutaan havaintoraporteiksi).

6 Testausta avustavat työkalut — 40 minuuttia

Avainsanat

aineisto-ohjattu testaus, avainsanaohjattu testaus, suorituskykytestaustyökalu, testauksenhallintatyökalu, testiautomaatio, testien suoritustyökalu

Oppimistavoitteet: Testaustyökalut

6.1. Testaustyökaluihin liittyviä asioita

FL-6.1.1 (K2) Luokitella testaustyökalut niiden käyttötarkoituksen sekä niiden tukemien testaustehtävien mukaan

FL-6.1.2 (K1) Tunnistaa testausautomaation hyödyt ja riskit

FL-6.1.3 (K1) Muistaa testien suoritustyökaluihin sekä testauksenhallintavälineisiin liittyvät erityisseikat

6.2. Työkalujen tehokas käyttö

FL-6.2.1 (K1) Tunnistaa työkalujen valinnan pääperiaatteet

FL-6.2.2 (K1) Muistaa tavoitteet, jotka liittyvät pilottiprojektin käyttöön työkalujen käyttöönoton yhteydessä

FL-6.2.3 (K1) Tunnistaa testaustyökalujen arviointiin, hankintaan, käyttöönottoon ja jatkuvaan tukeen liittyvät menestystekijät

6.1 *Testaustyökaluihin liittyviä asioita*

Testaustyökaluja voidaan käyttää tukemaan yhtä tai useampaa testaustehtävää. Näihin työkaluihin kuuluvat

- työkalut, joita käytetään suoraan testauksessa, kuten testien suoritustyökalut ja testiaineiston valmisteluvälineet.
- työkalut, jotka auttavat vaatimusten, testitapausten, testiproseduurien, automatisoitujen testiskriptien, testitulosten, testiaineistojen ja vikojen hallinnassa sekä testien suorituksen raportoinnissa ja seurannassa
- työkalut, joita käytetään tutkimiseen ja arviointiin
- mitkä tahansa työkalut, jotka auttavat testausta (tässä merkityksessä myös taulukkolaskentaohjelmisto on myös testaustyökalu).

6.1.1 Testaustyökalujen luokittelu

Testaustyökaluja voidaan käyttää tilanteesta riippuen yhteen tai useampaan tarkoitukseen:

- testaustehtävien tehostaminen automatisoimalla toistuvia tehtäviä tai tehtäviä, jotka manuaalisesti tehtynä vaativat merkittävästi resursseja (esim. testien suoritus, regressiotestaus)
- testaustehtävien tehokkuuden parantaminen tukemalla manuaalisia testaustehtäviä läpi koko testausprosessin (ks. alaluku 1.4)
- testaustehtävien laadun parantaminen mahdollistamalla yhdenmukaisempi testaus ja parempi vikojen toistettavuus
- sellaisten tehtävien automatisointi, joita ei voida suorittaa manuaalisesti (esim. laajamittainen suorituskykytestaus)
- testauksen luotettavuuden lisääminen (esim. automatisoimalla isojen aineistojen vertailu tai simuloimalla käyttäytymistä).

Työkalut voidaan luokitella monen kriteerin perusteella, kuten niiden käyttötarkoitus, hinnoittelu, lisensointimalli (esim. kaupallinen tai avoimen lähdekoodin työkalu) ja niiden käyttämä teknologia. Tässä sertifikaattisisällössä työkalut on luokiteltu niiden tukemien testaustehtävien mukaan.

Jotkut työkalut tukevat selkeästi yhtä tai pääasiassa vain yhtä tehtävää, toiset tukevat useampaa kuin yhtä, mutta ne on luokiteltu sen tehtävän alle, johon ne liittyvät läheisimmin. Yksittäisen välinetoimittajan välineet, esimerkiksi sellaiset, jotka on suunniteltu toimimaan yhdessä, voivat olla tarjolla integroituna välineperheenä.

Jotkut testityökalutyypit voivat olla tunkeutuvia, jolloin työkalu itse voi vaikuttaa testin lopputulokseen. Esimerkiksi sovelluksen todellinen vasteaika voi olla erilainen suorituskykytestaustyökalujen suorittaminen ylimääräisten tehtävien vuoksi tai kattavuustyökalut voivat vääristää saavutettua koodikattavuuden määrää. Tunkeutuvien työkalujen käytön aiheuttamia seurauksia kutsutaan mittausjärjestelyjen vaikutukseksi.

Jotkut työkalut tarjoavat enemmän tyypillisesti ohjelmistokehittäjille sopivaa tukea (esim. työkalut, joita käytetään yksikkö- ja komponentti-integrointitestauksessa). Tällaiset työkalut on merkitty kirjaimella ("K") alla olevissa luokitteluissa.

Testausta ja testimateriaalin hallintaa avustavat työkalut

Hallintatyökalut voivat olla käyttökelpoisia kaikissa testaustoimenpiteissä koko ohjelmistokehityksen elinkaaren aikana. Esimerkkeihin testauksen ja testimateriaalin hallintaa tukevista työkaluista kuuluvat:

- testauksenhallinnan ja sovelluksen elinkaarenhallinnan työkalut (Application Lifecycle Management, ALM)
- vaatimustenhallintatyökalut (esim. jäljitettävyys testattaviin kohteisiin)
- vianhallintatyökalut

- kokoonpanonhallintatyökalut
- jatkuvan integraation työkalut (K).

Staattista testausta avustavat työkalut

Staattisen testauksen työkalut liittyvät luvussa 3 kuvattuihin tehtäviin ja hyötyihin. Esimerkkeihin näistä työkaluista kuuluvat:

- katselmointia tukevat työkalut
- staattisen analyysin työkalut (K).

Testien suunnittelua ja toteutusta avustavat työkalut

Testisuunnittelutyökalut auttavat ylläpidettävien tuotosten luomista testien suunnittelun ja toteutuksen aikana, mukaan luettuna testitapaukset, testiproseduurit ja testiaineisto. Esimerkkeihin näistä työkaluista kuuluvat:

- testisuunnittelutyökalut
- mallipohjaisen testauksen työkalut
- testiaineiston valmisteluvälineet
- hyväksymistestiohjatussa kehityksessä (Acceptance Test Driven Development, ATDD) ja käyttäytymisohjatussa kehityksessä (Behavior Driven Development, BDD) käytettävät työkalut
- testiohjatun kehityksen (Test Driven Development) työkalut (K).

Joissakin tapauksissa testien suunnittelua ja toteutusta tukevat työkalut voivat myös tukea testien suoritusta ja tulosten kirjaamista tai niiden tuottamat tulokset siirtyvät suoraan toisiin työkaluihin, jotka tukevat testien suoritusta ja kirjaamista.

Testien suoritusta ja kirjaamista avustavat työkalut

Testien suorituksen ja kirjaamisen tehtävien tukemiseen on olemassa monia työkaluja. Esimerkkeihin näistä työkaluista kuuluvat:

- testien suoritustyökalut (esim. regressiotestien suorittamiseksi)
- kattavuuden mittaustyökalut (esim. vaatimuskattavuus, koodikattavuus (K))
- testikehykset (K).

Suorituskyvyn mittaamista ja dynaamista analyysia avustavat työkalut

Suorituskyvyn mittaamisen ja dynaamisen analyysin työkalut ovat keskeisiä suorituskyky- ja kuormitustestaustehtävien tukemisessa, sillä näitä tehtäviä ei voi tehdä tehokkaasti manuaalisesti. Esimerkkeihin näistä työkaluista kuuluvat:

- suorituskykytestaustyökalut
- monitorointityökalut
- dynaamisen analyysin työkalut (K).

Erityisiä testaustarpeita tukevat työkalut

Yleistä testausprosessia tukevien työkalujen lisäksi on monia muita työkaluja, jotka tukevat testaukseen liittyviä erityisempiä osa-alueita. Näihin kuuluu työkaluja, jotka keskittyvät esimerkiksi

- aineiston laadun arviointiin
- tietojen konversioon ja migraatioon
- käytettävyystestaukseen
- saavutettavuustestaukseen
- lokalisaatiotestaukseen

- tietoturvatestaukseen
- siirrettävyystestaukseen (esim. testataan ohjelmistoa useilla tuetuilla alustoilla).

6.1.2 Testiautomaation hyödyt ja riskit

Pelkästään työkalun hankkiminen ei takaa onnistumista. Jokainen uuden työkalun käyttöönotto organisaatiossa vaatii panostusta todellisten ja kestävien hyötyjen saavuttamiseksi. Työkalujen käyttämisessä testauksessa on potentiaalisia hyötyjä ja mahdollisuuksia, mutta myös riskejä. Tämä pätee erityisesti testien suoritustyökaluihin (joihin usein viitataan testiautomaationa).

Testien suoritusta tukevien työkalujen mahdollisiin hyötyihin kuuluvat seuraavat:

- toistuvan manuaalisen työn väheneminen (esim. regressiotestien suoritus, ympäristön pystytykseen ja purkamiseen liittyvät tehtävät, saman testiaineiston uudelleensyöttäminen sekä ohjelmointistandardien noudattamisen tarkistaminen)
- parempi yhdenmukaisuus ja toistettavuus (esim. testiaineisto luodaan johdonmukaisesti, testit suoritetaan työkalulla samassa järjestyksessä ja samaa suoritustiheyttä noudattaen ja testit johdetaan yhdenmukaisesti vaatimuksista)
- puolueettomampi arviointi (esim. staattiset mittarit, kattavuus)
- helpompi tiedonsaanti testauksesta (esim. tilastot ja kaaviot testauksen edistymisestä, vikamääristä ja suorituskyvystä).

Testien suoritusta tukevien työkalujen mahdollisiin riskeihin kuuluvat seuraavat:

- epärealistiset odotukset työkalua kohtaan (mukaan luettuna toiminnallisuus ja helppokäyttöisyys)
- työkalun käyttöönottoon tarvittavan ajan, kulujen ja työmäärän (mukaan luettuna koulutus ja ulkopuolinen asiantuntemus) aliarviointi
- työkalusta saatavien merkittävien ja jatkuvien hyötyjen saamiseksi tarvittavan ajan ja panostuksen aliarviointi (esim. testausprosessin muutostarpeet ja jatkuvat parannukset työkalun käyttötapoihin)
- työkalun luoman testimateriaalin ylläpitoon tarvittavan työmäärän aliarviointi
- liika tukeutuminen työkaluun (työkalua pidetään testien suunnittelun tai suorituksen korvaajana tai käytetään automatisoituja testejä, vaikka testaus olisi parempi tehdä manuaalisesti)
- testimateriaalin versionhallinnan laiminlyönti
- kriittisten työkalujen, kuten vaatimustenhallintatyökalujen, kokoonpanonhallintatyökalujen, vianhallintatyökalujen sekä useiden työkalutoimittajien toimittamien työkalujen välisiin suhteisiin ja yhteentoimivuuteen liittyvien ongelmien laiminlyönti
- työkalutoimittajan toiminnan lakkaaminen, välineen poistuminen markkinoilta tai myynti toiselle toimittajalle
- työkalutoimittajan huonot tukipyyntöjen vastaukset, päivitykset ja vikakorjaukset
- avoimen lähdekoodin projektin keskeytyminen
- työkalu yhteensopimattomuus uuden alustan tai teknologian kanssa
- epäselvät työkalun omistussuhteet (esim. liittyen koulutukseen, päivityksiin jne.).

6.1.3 Testien suoritustyökaluihin ja testauksenhallintatyökaluihin liittyviä erityishuomioita

Kun ollaan valitsemassa ja integroimassa testauksen suoritusvälinettä ja testauksen hallintavälinettä organisaatioon, on joukko asioita, joita on syytä harkita välineiden sulavan ja onnistuneen hankinnan aikaansaamiseksi.

Testien suoritustyökalut

Testien suoritustyökalut suorittavat testejä käyttämällä automatisoituja testiskriptejä. Tällaiset työkalut vaativat usein huomattavan panostuksen, jotta niillä saadaan merkittäviä hyötyjä.

Testien tallentaminen nauhoittamalla testaajan manuaalisia toimenpiteitä voi vaikuttaa houkuttelevalta, mutta tämä menetelmä ei skaalaudu suureen määrään testejä. Nauhoitettu skripti on lineaarinen esitys suoritetusta testistä aineistoineen ja toimenpiteineen. Tällainen skripti voi olla epävakaa odottamattomien tapahtumien sattuessa. Näiden työkalujen viimeisin sukupolvi, joka hyödyntää “älykästä” kuvantallennusteknologiaa, on lisännyt tämän tyyppisten työkalujen hyödyllisyyttä, vaikka generoidut skriptit yhtä vaativatkin jatkuvaa ylläpitoa, kun järjestelmän käyttöliittymä muuttuu ajan myötä.

Aineisto-ohjatussa lähestymistavassa testisyötteet ja odotetut tulokset talletetaan erikseen, tavallisesti taulukkolaskentaohjelmistoon, ja siinä käytetään geneerisempää skriptiä, joka pystyy lukemaan syöteaineiston ja suorittamaan saman testin eri aineistoilla. Testaajat, jotka eivät tunne skriptauskieltä, voivat sitten laatia uutta testiaineistoa näille ennalta tehdyille skripteille.

Avainsanaohjatussa lähestymistavassa geneerinen skripti käsittelee avainsanoja (kutsutaan myös toimisanoiksi), jotka kuvaavat suoritettavia tehtäviä, ja avainsanat kutsuvat sitten niihin liittyvää skiptiä, joka prosessoi tilanteeseen liittyvän testiaineiston. Testaajat (vaikka eivät tuntisikaan skriptauskieltä) voivat luoda testit käyttämällä avainsanoja ja niihin liittyvää aineistoa, joita voidaan räätälöidä testattavan sovelluksen mukaisesti. Lisää tietoa ja esimerkkejä aineisto-ohjatusta ja avainsanaohjatusta lähestymistavoista, ks. ISTQB-TAE Jatkotason sertifikaattisisältö, Testiautomaatioasiantuntija, Fewster 1999 ja Buwalda 2001.

Yllä kuvatut lähestymistavat edellyttävät, että mukana on joku, jolla on skriptikielen osaamista (testaaja, kehittäjä tai testiautomaatioon erikoistunut henkilö). Käytetystä skriptaustekniikasta riippumatta jokaisen testin odotettua tulosta pitää verrata testin todelliseen tulokseen, joko dynaamisesti (kun testiä suoritetaan) tai myöhemmin (suorituksen jälkeen), mikä edellyttää testituloksen tallentamista vertailua varten.

Mallipohjaisen testauksen (Model-Based Testing, MBT) työkalut mahdollistavat toiminnallisen määrittelyn tallentamisen mallin muodossa, esimerkiksi aktiviteettikaaviona. Tämän tehtävän suorittaa yleensä järjestelmäsuunnittelija. Mallipohjaisen testauksen työkalut tulkitsevat mallin ja luovat niistä testitapaussuunnitelmia, jotka voidaan sitten tallentaa testauksenhallintavälineeseen ja/tai suorittaa testien suoritusvälineellä (ks. ISTQB-MBT Foundation Level Model-Based Testing Syllabus.

Testauksenhallintatyökalut

Testauksenhallintatyökalujen tarvitsee usein olla yhteydessä toisiin työkaluihin monista eri syistä, kuten:

- hyödyllisen tiedon tuottaminen muodossa, joka täyttää organisaation tarpeet
- jatkuvan jäljitettävyyden säilyttäminen vaatimuksenhallintavälineeseen tallennettuihin vaatimuksiin
- testattavan kohteen versiotietojen linkittäminen kokoonpanonhallintatyökaluihin.

Tämä on erityisen tärkeää ottaa huomioon, kun käytetään integroitua työkalua (esim. ALM), johon sisältyy testauksenhallintamoduuli (ja mahdollisesti vikojenhallintajärjestelmä) sekä muita moduuleita (esim. projektin aikataulu ja budjettitiedot), joita käyttävät eri ryhmät organisaatiossa.

6.2 *Työkalujen tehokas käyttö*

6.2.1 Työkalun valinnan pääperiaatteet

Seuraavassa on lueteltu tärkeimpiä asioita, joita on harkittava, kun organisaatioon ollaan valitsemassa testaustyökalua:

- organisaation kypsyyden, vahvuuksien ja heikkouksien arviointi
- työkalun tukemien parannettuun testausprosessiin liittyvien mahdollisuuksien tunnistaminen
- testattavissa kohteissa käytettävien teknologioiden ymmärtäminen, jotta voidaan valita kyseisen teknologian kanssa yhteensopiva työkalu

- organisaatiossa jo käytössä olevat koonti- ja jatkuvan integraation työkalut, jotta voidaan varmistaa työkalujen yhteensopivuus ja integraatio
- työkalun arviointi selkeitä vaatimuksia ja objektiivisia kriteereitä vastaan
- työkalun saatavuus ilmaiseen koekäyttöön (ja kuinka pitkäksi aikaa)
- työkalutoimittajan (mukaan luettuna toimittajan tarjoaman koulutuksen ja tuen sekä muiden kaupallisten näkökulmien) sekä ei-kaupallisten (esimerkiksi avoin lähdekooin) työkalujen saatavilla olevan tuen arviointi
- työkalun käytön tuomien sisäisten valmennus- ja ohjausvaatimusten tunnistaminen
- koulutustarpeiden arviointi, ottaen huomioon myös työkalu(je)n kanssa suoraan työskentelevien henkilöiden testaus- ja testiautomaatiotaidot
- eri lisensointimallien (esim. kaupalliset tai avoimen lähdekoodin työkalut) hyvät ja huonot puolet
- panos-tuotos-suhteen arviointi todellisen liiketoimintacasen perusteella (jos tätä on vaadittu).

Viimeisenä askeleena pitäisi käyttää proof-of-concept–arviointia sen toteamiseksi, toimiiko työkalu tehokkaasti testattavan ohjelmiston kanssa ja senhetkisessä ympäristössä tai, mikäli tarpeen, ympäristöön tarvittavien muutosten tunnistamiseksi, jotta työkalua voidaan käyttää tehokkaasti.

6.2.2 Työkalun tuominen organisaatioon pilottiprojektien avulla

Työkalun valinnan ja onnistuneen proof-of-concept –menettelyn jälkeen valitun työkalun käyttöönotto organisaatiossa alkaa yleensä pilottiprojektilla, jolla on seuraavat tavoitteet:

- hankkia syvällisempää osaamista työkalusta sekä sen vahvuuksien ja heikkouksien ymmärtäminen
- arvioida, kuinka työkalu soveltuu olemassa oleviin prosesseihin ja käytäntöihin, ja päättää, mitä pitäisi muuttaa
- päättää työkalun ja testimateriaalin käytössä, hallinnassa, tallentamisessa ja ylläpidossa noudatettavista vakiokäytännöistä (esim. tiedostojen ja testien nimeämiskäytännöt, ohjelmointistandardien valinta, kirjastojen luominen ja testijoukkojen modulaarisuuden määrittäminen)
- arvioida, saadaanko työkalusta hyötyä kohtuullisilla kustannuksilla.
- ymmärtää, mitä mittaritietoja työkalun avulla halutaan kerätä ja raportoida, ja muokata työkalua sen varmistamiseksi, että näitä mittaritietoja voidaan tallentaa ja raportoida.

6.2.3 Työkalujen menestystekijät

Työkalujen arviointiin, hankintaan, käyttöönottoon ja jatkuvaan tukeen organisaatiossa liittyy seuraavia menestystekijöitä:

- vaiheittainen työkalun käyttöönotto läpi organisaation
- prosessien sovittaminen ja parantaminen yhteensopivaksi työkalun käytön kanssa
- koulutuksen, valmennuksen ja mentoroinnin tarjoaminen työkalun käyttäjille
- työkalun käyttöön liittyvien perussääntöjen määrittäminen (esim. sisäiset automaatiostandardit)
- toimintatapojen määrittäminen tiedon keräämiseksi välineen todellisesta käytöstä
- työkalun käytön ja hyötyjen seuranta
- tuen tarjoaminen työkalujen käyttäjille
- kokemusten kerääminen kaikilta käyttäjiltä.

On myös tärkeää varmistaa, että työkalu integroidaan teknisesti ja organisatorisesti ohjelmistokehityksen elinkaareen, ja tähän saattaa liittyä erillisiä toiminnoista vastaavia organisaatioita ja/tai kolmansia osapuolia. Kokemuksia ja neuvoja testien suoritustyökalujen käytöstä, ks. Graham 2012.

7 Viitteet

Standardit

ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering - Software testing - Part 1: Concepts and definitions

ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering - Software testing - Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering - Software testing - Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering - Software testing - Part 4: Test techniques

ISO/IEC 25010, (2011) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

ISO/IEC 20246: (2017) Software and systems engineering — Work product reviews

UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

ISTQB asiakirjat

ISTQB Sanasto

ISTQB Perustason yleiskatsaus 2018

ISTQB-MBT Perustason sertifikaattisisällön laajennus Mallipohjainen testaaja

ISTQB-AT Perustason sertifikaattisisällön laajennus Ketterä testaaja

ISTQB-ATA Jatkotason sertifikaattisisältö Testausasiantuntija

ISTQB-ATM Jatkotason sertifikaattisisältö Testauspäällikkö

ISTQB-SEC Jatkotason sertifikaattisisältö Tietoturvatestaaja

ISTQB-TAE Jatkotason sertifikaattisisältö Testiautomaatioasiantuntija

ISTQB-ETM Asiantuntijatason sertifikaattisisältö Testauksen hallinta

ISTQB-EITP Asiantuntijatason sertifikaattisisältö Testausprosessin kehittäminen

Kirjat ja artikkelit

Beizer, B. (1990*) Software Testing Techniques (2e)*, Van Nostrand Reinhold: Boston MA

Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon UK

Black, R. (2009) *Managing the Testing Process (3e)*, John Wiley & Sons: New York NY

Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading MA

Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood MA

Craig, R. and Jaskiel, S. (2002) *Systematic Software Testing*, Artech House: Norwood MA

Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston MA

Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Harlow UK

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading MA

Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston MA

Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston MA

Jorgensen, P. (2014) *Software Testing, A Craftsman's Approach (4e)*, CRC Press: Boca Raton FL

Kaner, C., Bach, J. and Pettichord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York NY

Kaner, C., Padmanabhan, S. and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-Driven Press: New York NY

Kramer, A., Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York NY

Myers, G. (2011) *The Art of Software Testing*, (3e), John Wiley & Sons: New York NY

Sauer, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," *IEEE Transactions on Software Engineering*, *Volume 26, Issue 1, pp 1-*

Shull, F., Rus, I., Basili, V. July 2000. "How Perspective-Based Reading can Improve Requirement Inspections." *IEEE Computer, Volume 33, Issue 7, pp 73-79*

van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8 - 10), UTN Publishers: The Netherlands

Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston MA

Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York NY

Muut lähteet (ei suoraan viitattu tässä Sertifikaattisisällössä)

Black, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: London UK

Hetzel, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: Wellesley MA

Spillner, A., Linz, T., and Schaefer, H. (2014) *Software Testing Foundations (4e)*, Rocky Nook: San Rafael CA

8 Liite A – Sertifikaattisisällön tausta

Tämän dokumentin historia

Tämä dokumentti on ISTQB Sertifioitu Testaaja, Perustason sertifikaattisisältö, ISTQB:n (www.istqb.org) hyväksymän sertifiointijärjestelmän ensimmäinen taso.

Tämä dokumentti valmisteltiin vuosien 2014 ja 2018 välillä työryhmässä, joka koostui International Software Testing Qualifications Boardin (ISTQB) nimeämistä jäsenistä. Vuoden 2018 version katselmoivat ensin kaikkien ISTQB:n jäsenyhdistysten edustajat ja sen jälkeen kansainvälisestä ohjelmistotestauksen yhteisöstä poimitut edustajat.

Perussertifikaatin tavoitteet

- Saada tunnustusta testaukselle oleellisena ja ammattimaisena ohjelmistosuunnittelun alueena.
- Tarjota vakioitu rakenne testaajien urapolulle.
- Luoda ammatillisesti sertifioiduille testaajille tilaisuus tulla työnantajien, asiakkaiden ja kollegoiden tunnustamiksi ja nostaa testaajien profiilia.
- Kannustaa yhdenmukaisia ja hyviä testauskäytäntöjä kaikilla ohjelmistokehityksen tieteenaloilla.
- Tunnistaa testausaiheet, jotka ovat merkityksellisiä ja arvokkaita toimialalle.
- Luoda ohjelmistotoimittajille mahdollisuus palkata sertifioituja testaajia ja siten saada kaupallista hyötyä kilpailijoihin nähden tuomalla esiin testaajien rekrytointikäytäntöjään.
- Antaa testaajille ja testauksesta kiinnostuneille mahdollisuus hankkia kansainvälisesti tunnustettu sertifiointi aiheessa.

Kansainvälisen sertifioinnin tavoitteet

- Pystyä vertailemaan testausosaamista eri maissa.
- Mahdollistaa testaajien siirtyminen helpommin maasta toiseen.
- Mahdollistaa monikansallisten/kansainvälisten projektien yhteinen ymmärrys testauskysymyksistä.
- Lisätä sertifioitujen testaajien määrää maailmanlaajuisesti.
- Saada enemmän vaikutusta/arvoa kansainvälisenä aloitteena kuin maakohtaisen lähestymisen pohjalta
- Kehittää yhteinen kansainvälisen testausymmärryksen ja osaamisen ryhmä sertifikaattisisällön ja sanaston kautta, sekä kasvattaa kaikkien osallistujien testausosaamisen tasoa.
- Edistää testausta ammattina useammissa maissa.
- Luoda testaajille mahdollisuus saada tunnustettu sertifiointi omalla äidinkielellään
- Mahdollistaa osaamisen ja resurssien jakaminen maiden välillä.
- Tuoda useiden maiden osallistumisen myötä testaajille ja tälle sertifioinnille kansainvälistä tunnustusta.

Aloitusvaatimukset tälle sertifikaatille

Aloitusvaatimuksena ISTQB Ohjelmistotestauksen Perustason sertifikaattikokeen suorittamiselle on, että kokelailla on mielenkiintoa ohjelmistotestaukseen. Kuitenkin vahvasti suositellaan, että

- kokelailla on vähintään minimitausta joko ohjelmistokehityksestä tai ohjelmistotestauksesta, kuten kuuden kuukauden kokemus joko järjestelmä- tai hyväksymistestaajana tai ohjelmistokehittäjänä

- kokelaat käyvät ISTQB:n standardien mukaan akkreditoidun kurssin (jonkin ISTQB:n tunnustaman jäsenyhdistyksen akkreditoima).

Ohjelmistotestauksen perussertifikaatin tausta ja historia

Ohjelmistotestaajien riippumaton sertifiointi alkoi Yhdistyneessä Kuningaskunnassa (UK) British Computer Society:n Information Systems Examination Board (ISEB) -organisaatiossa, kun Software Testing Board perustettiin 1998 (www.bcs.org.uk/iseb). Vuonna 2002, saksalainen ASQF alkoi tukea saksalaisten testaajien sertifiointisuunnitelmaa (www.asqf.de). Tämä sertifikaattisisältö pohjautuu sekä ISEB:n että ASQF:n sertifikaattisisältöihin. Se sisältää uudelleenjärjesteltyä, päivitettyä ja uutta aineistoa, ja sen painopiste on suunnattu aiheisiin, joista on eniten käytännön apua testaajille.

Olemassa oleva Ohjelmistotestauksen Perustason sertifikaatti (esim. ISEB:n, ASQF:n tai ISTQB:n tunnustaman kansallisen hallituksen myöntämä), joka on myönnetty ennen tämän Kansainvälisen Sertifikaatin julkaisua, vastaa Kansainvälistä Sertifikaattia. Perustason sertifikaatti ei vanhene eikä sitä tarvitse uusia. Myöntämispäivämäärä löytyy sertifikaatista.

Jokaisessa jäsenmaassa paikallisia näkökohtia hallinnoidaan kansallisessa tai alueellisessa ISTQB:n tunnustamassa Ohjelmistotestauksen jäsenyhdistyksessä. ISTQB määrittelee jäsenyhdistysten vastuut, mutta ne toimeenpannaan kussakin maassa. Maiden johtokuntien tehtäviin tulevat kuulumaan koulutuksen tarjoajien akkreditointi ja kokeiden järjestäminen.

9 Liite B - Oppimistavoitteet/osaamistaso

Seuraavia oppimistavoitteita sovelletaan tähän sertifikaattisisältöön. Jokainen sertifikaattisisällön aihe tentitään kokeessa sen oppimistavoitteen mukaisesti.

Taso 1: Muistaa (K1)

Kokelas tunnistaa, muistaa ja pystyy palauttamaan mieleensä termin tai käsitteen.

Avainsanat**:** Muistaa, palauttaa mieleen, tunnistaa, tietää

Esimerkkejä

Tunnistaa "häiriön" määritelmän:

- "palvelun toimimattomuus loppukäyttäjän tai muun sidosryhmän jäsen näkökulmasta" tai
- "ohjelmiston poikkeama odotetusta toimituksesta, palvelusta tai tuloksesta".

Taso 2: Ymmärtää (K2)

Kokelas osaa valita syyt tai perustelut aiheeseen liittyville väitteille ja osaa vetää yhteen, vertailla, luokitella, ryhmitellä ja antaa esimerkkejä asiasta.

Avainsanat: Vetää yhteen, yleistää, tiivistää, luokitella, vertailla, sijoittaa, asettaa vastakkain, valaista esimerkeillä, tulkita, selittää, kuvata, päätellä, tehdä johtopäätös, ryhmitellä, laatia malleja

Esimerkkejä

Osaa selittää syyt, miksi testien analysointi ja suunnittelu pitäisi tehdä niin aikaisin kuin mahdollista:

- Jotta löydetään viat silloin, kun niiden poistaminen on halvempaa
- Jotta löydetään tärkeimmät viat ensin.

Osaa selittää yhtäläisyydet ja erot integrointi- ja järjestelmätestauksen välillä:

- Yhtäläisyydet: Sekä integraatio- että järjestelmätestauksen kohteisiin kuuluu useampi kuin yksi komponentti, ja sekä integraatio- että järjestelmätestaukseen voi kuulua ei-toiminnallisia testaustyyppejä.
- Erot: integrointitestaus keskittyy rajapintoihin ja vuorovaikutuksiin, kun taas järjestelmätestaus keskittyy koko järjestelmän piirteisiin, kuten päästä-päähän prosessointiin.

Taso 3: Soveltaa (K3)

Kokelas osaa valita oikean toimintamallin tai tekniikan soveltamistavan ja käyttää sitä annetussa yhteydessä.

Avainsanat: soveltaa, suorittaa, käyttää, noudattaa toimintatapaa, soveltaa toimintatapaa

Esimerkkejä

- Tunnistaa raja-arvot kelvollisille ja epäkelvollisille osioille.
- Osaa valita testitapaukset annetusta tilasiirtymäkaaviosta niin, että kaikki siirtymät saadaan katettua.

Viitteet (Oppimistavoitteiden kognitiivisia tasoja varten)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and As-sessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon: Boston MA

10 Liite C - Julkaisuseloste

Vuoden 2018 ISTQB Perustason sertifikaattisisältöä laadittaessa on tehty vuoden 2011 version mittava päivitys ja uudelleenkirjoitus. Tästä syystä tarjolla ei ole yksityiskohtaisia luku- ja alalukukohtaisia julkaisuselosteita. Tässä dokumentissa on kuitenkin kuvattu yhteenveto päämuutoksista. Lisäksi ISTQB tarjoaa kuvauksen vuoden 2011 ja 2018 Perustason sertifikaattisisältöjen oppimistavoitteiden välisestä jäljitettävyydestä erillisessä julkaisuselostedokumentissa, josta ilmenevät oppimistavoitteiden lisäykset, päivitykset sekä poistot.

Vuoden 2017 alussa yli 550 000 ihmistä yli sadassa maassa oli osallistunut perustason kokeeseen ja sertifioituja testaajia on maailmanlaajuisesti yli 500 000. Jos oletetaan heidän kaikkien lukeneen Perustason sertifikaattisisällön läpäistäkseen kokeen, on Perustason sertifikaattisisältö todennäköisesti kaikkein luetuin ohjelmistotestauksen dokumentti ikinä!

Tämä laaja päivitys on tehty tämä tausta huomioon ottaen ja tarkoituksena on kasvattaa maailmanlaajuisen testausyhteisön seuraavien 500 000 ihmisen saamaa lisäarvoa, jota ISTQB tarjoaa.

Tässä versiossa kaikkia oppimistavoitteita (LO) on muokattu niin, että niistä on tehty yksilöiviä ja on voitu luoda selkeä jäljitettävyys jokaisesta oppimistavoitteesta sitä vastaavaan sisältöön (sekä koekysymyksiin) ja sisällön eri osista (ja koekysymyksistä) takaisin niihin liittyvään oppimistavoitteeseen. Lisäksi eri luvuille määriteltyjä opetusaikoja on muutettu realistisemmiksi verrattuna vuoden 2011 versioon käyttämällä muiden ISTQB:n sertifikaattisisältöjen yhteydessä käytettyjä toimiviksi todettuja heuristiikkoja ja kaavoja, jotka perustuvat kussakin luvussa käsiteltäviksi tarkoitettujen oppimistavoitteiden analyysiin.

Vaikka tämä on Perustason sertifikaattisisältö, jossa esitellään aikojen saatossa kestäväksi todettuja hyviä käytäntöjä ja tekniikoita, olemme tehneet muutoksia materiaalin uudenaikaistamiseksi, erityisesti kun kyse on ohjelmistokehityksen menetelmistä (esim. Scrum ja jatkuva julkaisu) ja teknologioista (esim. Esineiden Internet). Olemme päivittäneet viitteenä käytetyt standardit niiden uudenaikaistamiseksi seuraavasti:

1. ISO/IEC/IEEE 29119 korvaa IEE-standardin 829.
2. ISO/IEC 25010 korvaa ISO 9126:n.
3. ISO/IEC 20246 korvaa ISO 1028:n.

Koska ISTQB:n tarjoama sertifikaattivalikoima on kasvanut dramaattisesti viime vuosikymmenen aikana, olemme tämän lisäksi lisänneet kattavasti ristiviitteitä muiden ISTQB:n sertifikaattisisältöjen sisältämään asiaan liittyvään oleelliseen materiaaliin, ja olemme myös katselmoineet materiaalin huolellisesti yhdenmukaisuuden varmistamiseksi sekä kaikkien sertifikaattisisältöjen että ISTQB Sanaston välillä. Tavoitteena on tehdä tästä versiosta helpommin luettava, ymmärrettävä, opittava ja käännettävä keskittymällä käytännön hyödyllisyyden sekä tietojen ja taitojen välisen tasapainon kasvattamiseen.

Yksityiskohtaisempi analyysi tässä julkaisussa tehdyistä muutoksista: ks. ISTQB Sertifioitu Testaaja, Perustason Sertifikaattisisältö, Yleiskatsaus 2018.