Sertifioitu testaaja

Perustason sertifikaattisisältö

Versio CTFL 4.0-FI

Perustuu englanninkieliseen versioon v4.0/2023

International Software Testing Qualifications Board

Tekijänoikeusilmoitus

Versiohistoria

Englanninkielinen versio	Päivämäärä	Huomautuksia
CTFL v4.0	21.04.2023	CTFL v4.0 – General release version
CTFL v3.1.1	01.07.2021	CTFL v3.1.1 – Copyright and logo update
CTFL v3.1	11.11.2019	CTFL v3.1 – Maintenance release with minor updates
ISTQB 2018	27.04.2018	CTFL v3.0 – Candidate general release version
ISTQB 2011	1.04.2011	CTFL Syllabus Maintenance Release
ISTQB 2010	30.03.2010	CTFL Syllabus Maintenance Release
ISTQB 2007	01.05.2007	CTFL Syllabus Maintenance Release
ISTQB 2005	01.07.2005	Certified Tester Foundation Level Syllabus v1.0
ASQF V2.2	07.2003	ASQF Syllabus Foundation Level Version v2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25.02.1999	ISEB Software Testing Foundation Syllabus v2.0

Suomenkielinen versio	Päivämäärä	Huomautuksia
CTFL 4.0-FI	1.9.2023	Eng. version CTFL 4.0 käännös
1.0	10.10.2018	Ensimmäinen suomenkielinen versio

Sisällys

Kiitokset

Tämä asiakirja julkaistiin virallisesti ISTQB®:n yleiskokouksessa 21. huhtikuuta 2023.

Sen tuotti ISTQB:n yhdistetyn Perustason ja Ketterän työryhmän jäsenistä koottu tiimi: Laura Albert, Renzo Cerquozzi (varapuheenjohtaja), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (puheenjohtaja), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (puheenjohtaja), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie lrich (varapuheenjohtaja), Eshraka Zakaria.

Tiimi kiittää Stuart Reidia, Patricia McQuaidia ja Leanne Howardia heidän teknisestä katselmoinnistaan sekä katselmointitiimiä ja jäsenyhdistyksiä ehdotuksista ja kommenteista.

Seuraavat henkilöt osallistuivat tämän sertifikaattisisällön katselmointiin, kommentointiin ja äänestykseen: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Sæther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo ja Zsolt Hargitai.

ISTQB Perustason työryhmä (2018 versio): Klaus Olsen (puheenjohtaja), Tauhida Parveen (varapuheenjohtaja), Rex Black (projektipäällikkö), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery ja kaikkia jäsenyhdistyksiä niiden ehdotuksista.

ISTQB Perustason työryhmä (2011 versio): 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 jäsenyhdistyksiä ehdotuksista sertifikaattisisällön nykyiseksi versioksi.

ISTQB Perustason työryhmä (2010 versio): Thomas Müller (puheenjohtaja), Rahul Verma, Martin Klonk ja Armin Beer. Ydintiimi kiittää arviointitiimiä (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal) ja kaikkia jäsenyhdistyksiä ehdotuksista.

ISTQB Perustason työryhmä (2007 versio): Thomas Müller (puheenjohtaja), 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 ehdotuksista.

ISTQB Perustason työryhmä (2005 versio): Thomas Müller (puheenjohtaja), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson ja Erik van Veenendaal. Ydintiimi kiittää katselmointitiimiä ja kaikkia jäsenyhdistyksiä ehdotuksista.

0. Johdanto

0.1. Tämän sertifikaattisisällön tarkoitus

Tämä sertifikaattisisältö muodostaa perustan kansainväliselle ohjelmistotestauksen tutkinnolle perustasolla. ISTQB® on tuottanut tämän sertifikaattisisällön seuraavaa käyttöä 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 valmennuskursseja tai itsenäisesti).
5. Kansainväliselle ohjelmisto- ja järjestelmäkehittäjien yhteisölle ohjelmisto- ja järjestelmätestauksen ammatin edistämiseksi sekä kirjojen ja artikkeleiden pohjamateriaaliksi.

0.2. Sertifioitu testaaja -perustaso ohjelmistotestauksessa

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

0.3. Testaajien urapolku

ISTQB®-järjestelmä tarjoaa testausalan ammattilaisille tukea heidän uransa kaikissa vaiheissa tarjoamalla sekä laajaa että syvällistä tietoa. Henkilöt, jotka ovat saavuttaneet ISTQB® Perustason sertifikaatin, voivat myös olla kiinnostuneita Core-jatkotasoista (Test Analyst, Technical Test Analyst ja (Test Manager) ja sen jälkeen Expert-tasosta (Test Management tai Improving the Test Process). Jokainen, joka haluaa kehittää testauskäytäntöjen osaamistaan ketterässä ympäristössä, voi harkita Agile Technical Tester tai Agile Test Leadership in Scale -sertifiointeja. Specialist-polku tarjoaa sukelluksen syvälle alueisiin, joihin liittyy erityisiä testauksen lähestymistapoja ja testaustoimenpiteitä (esim. testiautomaatio, tekoälytestaus, mallipohjainen testaus, mobiilisovellusten testaus), jotka liittyvät tiettyihin testauksen alueisiin (esim. suorituskykytestaus, käytettävyystestaus, hyväksymistestaus, tietoturvatestaus) tai jotka kokoavat määrättyjen liike-toiminta-alueiden testaukseen liittyvää tietotaitoa (esim. autoteollisuus tai peliala). Käy osoitteessa www.istqb.org saadaksesi viimeisimmät tiedot ISTQB:n sertifioitu testaaja -järjestelmästä.

0.4. Liiketoiminnan tulokset

Tässä osassa luetellaan 14 liiketoiminnan tulosta, joita odotetaan henkilöltä, joka on saavuttanut perustason testaussertifikaatin.

Perustason sertifioitu testaaja pystyy...

FL-BO1	ymmärtämään, mitä testaus on ja miksi siitä on hyötyä
FL-BO2	ymmärtämään ohjelmistotestauksen peruskäsitteet
FL-BO3	tunnistamaan tilanteen mukaan käytettävät testauksen lähestymistavat ja toimenpiteet
FL-BO4	arvioimaan ja parantamaan dokumentaation laatua
FL-BO5	lisäämään testauksen tehokkuutta ja tuottavuutta
FL-BO6	sopeuttamaan testausprosessin ohjelmistokehityksen elinkaareen
FL-BO7	ymmärtämään testauksenhallinnan periaatteet
FL-BO8	kirjoittamaan ja kommunikoimaan selkeitä ja ymmärrettäviä vikaraportteja
FL-BO9	ymmärtää tekijät, jotka vaikuttavat testaukseen liittyviin prioriteetteihin ja työmäärään
FL-BO10	työskentelemään osana monitahoista tiimiä
FL-BO11	tiedostamaan testiautomaatioon liittyvät riskit ja hyödyt
FL-BO12	tunnistamaan testauksessa tarvittavat olennaiset taidot
FL-BO13	ymmärtämään riskin vaikutuksen testaukseen
FL-BO14	raportoimaan tehokkaasti testauksen edistymisestä ja laadusta

0.5. Tentittävät oppimistavoitteet ja kognitiivinen tietotaso

Oppimistavoitteet tukevat liiketoiminnan tuloksia, ja niitä käytetään Testauksen perustason sertifiointikokeiden luomiseen. Yleisesti ottaen, tämän sertifikaattisisällön lukujen 1 - 6 koko sisältö on tentittävissä K1-tasolla. Toisin sanoen, kokelasta voidaan pyytää tunnistamaan, muistamaan tai palauttamaan mieleen avainsana tai käsite, joka mainitaan missä tahansa näistä kuudesta luvusta. Erityiset oppimistavoitetasot on esitetty kunkin luvun alussa ja ne on luokiteltu seuraavasti:

- K1: Muistaa
- K2: Ymmärtää
- K3: Käyttää

Lisätietoja ja esimerkkejä oppimistavoitteista on liitteessä A. Kaikki avainsanoina heti lukujen otsikoiden alapuolella luetellut termit on muistettava (K1), vaikka niitä ei nimenomaisesti mainittaisikaan oppimistavoitteissa.

0.6. Perustason sertifiointikoe

Perustason sertifiointikoe perustuu tähän sertifikaattisisältöön. Tenttikysymyksiin vastaaminen voi edellyttää useampaan kuin yhteen sertifikaattisisällön lukuun perustuvan materiaalin käyttöä. Kaikki sertifikaattisisällön luvut ovat tentittävissä lukuun ottamatta johdantoa ja liitteitä. Standardeja ja kirjoja on sisällytetty mukaan viitteinä (luku 7), mutta niiden sisältöä ei voida tenttiä muuten kuin siltä osin, mitä niistä on esitetty tässä sertifikaattisisällössä. Lisätietoja: Perustason tentin rakenne ja säännöt.

0.7. Akkreditointi

ISTQB®:n jäsenyhdistys voi akkreditoida koulutustarjoajia, joiden kurssimateriaali noudattaa tätä sertifikaattisisältöä. Koulutustarjoajien on hankittava akkreditointiohjeet akkreditoinnin suorittavalta jäsenyhdistykseltä tai elimeltä. Akkreditoidun kurssin katsotaan olevan tämän sertifikaattisisällön mukainen, ja ISTQB-koe saadaan järjestää osana kurssia Tämän sertifikaattisisällön akkreditointiohjeet noudattavat Process Management and Compliance -työryhmän julkaisemia yleisiä akkreditointiohjeita.

0.8. Standardien käsittely

Perustason sertifiointisisällössä viitataan standardeihin (esim. IEEE- tai ISO-standardit). Nämä viitteet tarjoavat kehyksen (kuten laatuominaisuuksia koskevissa viittauksissa ISO 25010 -standardiin) tai lisätietoa, jos lukija niin haluaa. Standardeihin liittyviä dokumentteja ei ole tarkoitettu tentittäväksi. Katso luvusta 7 lisätietoja standardeista.

0.9. Ajan tasalla pysyminen

Ohjelmistoala muuttuu nopeasti. Näiden muutosten käsittelemiseksi ja oleellisten ja ajantasaisten tietojen tarjoamiseksi sidosryhmille ISTQB:n työryhmät ovat luoneet www.istqb.org verkkosivustolle linkkejä, joissa viitataan tukiasiakirjoihin ja standardien muutoksiin. Nämä tiedot eivät ole tentittäviä Perustason sertifikaattisisällön osalta.

0.10. Yksityiskohtaisuuden taso

Tämän sertifikaattisisällön yksityiskohtaisuuden taso mahdollistaa kansainvälisesti yhdenmukaiset kurssit ja tentit. Tämän tavoitteen saavuttamiseksi sertifikaattisisältö koostuu seuraavista osista:

- Yleiset opetustavoitteet, jotka kuvaavat perustason tarkoituksen
- Luettelo termeistä (avainsanoista), jotka opiskelijoiden on muistettava
- Kunkin osaamisalueen oppimistavoitteet, joissa kuvataan saavutettavat kognitiiviset oppimistulokset
- Avainkäsitteiden kuvaus, mukaan lukien viittaukset tunnustettuihin lähteisiin.

Sertifikaattisisällön sisältö ei ole kuvaus koko ohjelmistotestauksen osaamisalueesta; se vastaa perustason valmennuskursseilla käsiteltävää yksityiskohtaisuuden tasoa. Se keskittyy testauskäsitteisiin ja -tekniikoihin, joita voidaan soveltaa kaikkiin ohjelmistoprojekteihin käytetystä ohjelmistokehityksen elinkaarimallista riippumatta.

0.11. Kuinka tämä sertifikaattisisältö on järjestetty

Sertifikaattisisällössä on kuusi lukua, joiden sisältö on tentittävää. Kunkin luvun ylätason otsikko määrittää luvun lukuun käytettävän opetusajan. Ajoitusta ei anneta lukutason alapuolella. Akkreditoitujen koulutuskurssien osalta sertifikaattisisältö vaatii vähintään 1135 minuuttia (18 tuntia ja 55 minuuttia) opetusta, joka jakautuu kuuteen lukuun seuraavasti:

- Luku 1: Testauksen perusteet (180 minuuttia)
    - Opiskelija oppii testaukseen liittyvät perusperiaatteet, testauksen tarpeellisuuden syyt ja testauksen tavoitteet.
    - Opiskelija ymmärtää testausprosessin, tärkeimmät testaustoimenpiteet ja testimateriaalin.
    - Opiskelija ymmärtää testauksen kannalta olennaiset taidot.
- Luku 2: Testaus ohjelmistokehityksen elinkaaren ajan (130 minuuttia)
    - Opiskelija oppii, miten testaus nivoutuu osaksi eri ohjelmistokehityksen lähestymistapoja.
    - Opiskelija oppii testit ensin -lähestymistapojen käsitteet sekä DevOpsin.
    - Opiskelija perehtyy eri testaustasoihin, testaustyyppeihin ja ylläpitotestaukseen.
- Luku 3: Staattinen testaus (80 minuuttia)
    - Opiskelija perehtyy staattisen testauksen perusteisiin sekä palaute- ja arviointiprosessiin.
- Luku 4: Testianalyysi ja -suunnittelu (390 minuuttia)
    - Opiskelija oppii soveltamaan mustalaatikko-, lasilaatikko- ja kokemuspohjaisia testaustekniikoita testitapausten johtamiseen erilaisista ohjelmistotuotoksista.
    - Opiskelija oppii yhteistyöhön perustuvista testauksen lähestymistavoista.
- Luku 5: Testaustehtävien hallinta (335 minuuttia)
    - Opiskelija oppii suunnittelemaan testejä yleisellä tasolla sekä arvioimaan testauksen työmäärää.
    - Opiskelija oppii, miten riskit voivat vaikuttaa testauksen laajuuteen.
    - Opiskelija oppii seuraamaan ja valvomaan testauksen toimenpiteitä.
    - Opiskelija oppii, miten kokoonpanonhallinta tukee testausta.
    - Opiskelija oppii raportoimaan vikoja selkeästi ja ymmärrettävästi.
- Luku 6: Testaustyökalut (20 minuuttia)
    - Opiskelija oppii luokittelemaan työkaluja ja ymmärtämään testiautomaation riskit ja hyödyt.

1. Testauksen perusteet – 180 minuuttia

Avainsanat

häiriö, juurisyy, kattavuus, kelpuutus, laadunvarmistus, laatu, testattava kohde, testattava tilanne, testauksen hallinta, testauksen pohjamateriaali, testauksen päättäminen, testauksen seuranta, testauksen suunnittelu, testauksen tavoite, testaus, testiaineisto, testianalyysi, testien suoritus, testien suunnittelu, testien valmistelu, testimateriaali, testiproseduuri, testitapaus, testitulos, todentaminen, vika, vikojenjäljitys, virhe

Luvun 1 oppimistavoitteet:

1.1 Mitä testaus on?

FL-1.1.1 (K1) Tunnistaa tyypilliset testauksen tavoitteet

FL-1.1.2 (K2) Erottaa testauksen vikojenjäljityksestä

1.2 Miksi testaus on välttämätöntä?

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

FL-1.2.2 (K1) Muistaa testauksen ja laadunvarmistuksen välisen suhteen

FL-1.2.3 (K2) Erottaa juurisyyn, virheen, vian ja häiriön toisistaan

1.3 Testauksen periaatteet

FL-1.3.1 (K2) Selittää seitsemän testausperiaatetta

1.4 Testaustoimenpiteet, testimateriaali ja testaukseen liittyvät roolit

FL-1.4.1 (K2) Osaa kuvata eri testaustoimenpiteet ja -tehtävät

FL-1.4.2 (K2) Osaa selittää, miten tilanne vaikuttaa testausprosessiin

FL-1.4.3 (K2) Erottaa testaustoimenpiteitä tukevan testimateriaalin

FL-1.4.4 (K2) Osaa selittää jäljitettävyyden ylläpitämisen merkityksen

FL-1.4.5 (K2) Pystyy vertaamaan testauksen eri rooleja

1.5 Keskeiset taidot ja hyvät käytännöt testauksessa

FL-1.5.1 (K2) Osaa antaa esimerkkejä testauksessa tarvittavista yleisistä taidoista

FL-1.5.2 (K1) Muistaa tiimiperustaisen lähestymistavan edut

FL-1.5.3 (K2) Erottaa testauksen riippumattomuuden edut ja haitat

1.1. Mitä testaus on?

Ohjelmistojärjestelmät ovat olennainen osa jokapäiväistä elämäämme. Useimmilla ihmisillä on kokemusta ohjelmistoista, jotka eivät toimineet odotetulla tavalla. Ohjelmisto, joka ei toimi oikein, voi johtaa moniin ongelmiin, kuten rahan, ajan tai yrityksen maineen menetykseen, ja äärimmäisissä tapauksissa jopa loukkaantumiseen tai kuolemaan. Ohjelmistotestaus arvioi ohjelmiston laatua ja auttaa vähentämään ohjelmistovirheiden riskiä ohjelmistoa käytettäessä.

Ohjelmistotestaus on joukko toimenpiteitä, joilla löydetään vikoja ja arvioidaan ohjelmistotuotoksien laatua. Testattaessa näitä tuotoksia kutsutaan testauksen kohteiksi. Yleinen väärinkäsitys testauksesta on, että se tarkoittaa vain testien suorittamista (eli ohjelmiston käyttämistä ja testitulosten tarkistamista). Ohjelmistotestaus sisältää kuitenkin myös muita toimenpiteitä, ja se on sovitettava yhteen ohjelmistokehityksen elinkaaren kanssa (katso luku 2).

Toinen yleinen testaukseen liittyvä väärinkäsitys on, että testaus keskittyy vain testattavan kohteen todentamiseen. Vaikka verifiointi eli sen tarkistaminen, täyttääkö järjestelmä sille asetetut vaatimukset, kuuluu testaukseen, siihen kuuluu myös kelpuutus, jossa tarkistetaan, täyttääkö järjestelmä käyttäjien ja muiden sidosryhmien tarpeet toimintaympäristössään.

Testaus voi olla dynaamista tai staattista. Dynaamiseen testaukseen kuuluu ohjelmiston suorittaminen, kun taas staattiseen testaukseen ei. Staattiseen testaukseen kuuluvat katselmoinnit (katso luku 3) ja staattinen analyysi. Dynaamisessa testauksessa käytetään erityyppisiä testaustekniikoita ja testauksen lähestymistapoja testitapausten johtamiseen (ks. luku 4).

Testaus ei ole vain tekninen toimenpide. Se täytyy myös suunnitella, sitä on hallinnoitava, sen työmäärä on arvioitava, ja sitä on seurattava ja hallittava asianmukaisesti (katso luku 5).

Testaajat käyttävät työkaluja (katso luku 6), mutta on tärkeää muistaa, että testaus on suurelta osin älyllistä toimintaa, joka vaatii testaajilta erikoisosaamista, analyyttisten taitojen käyttöä sekä kriittistä ajattelua ja järjestelmäajattelun soveltamista (Myers 2011, Roman 2018).

ISO/IEC/IEEE 29119-1 -standardi tarjoaa lisätietoja ohjelmistotestauksen käsitteistä.

1.1.1. Testauksen tavoitteet

Tyypillisiä testauksen tavoitteita ovat:

- tuotosten, kuten vaatimusten, käyttäjätarinoiden, suunnitelmien ja koodin, arviointi
- häiriöiden aiheuttaminen ja vikojen löytäminen
- testauksen kohteen vaaditun kattavuuden varmistaminen
- ohjelmistojen riittämättömän laadun riskin vähentäminen
- määritettyjen vaatimusten täyttymisen todentaminen
- sen todentaminen, että testauksen kohde täyttää sopimuksiin, lakeihin ja tai muihin säädöksiin perustuvat vaatimukset
- tiedon tuottaminen sidosryhmille tietoon pohjautuvien päätösten tekemiseksi
- luottamuksen lisääminen testauksen kohteen laatuun
- sen todentaminen, onko testauksen kohde valmis ja toimiiko se sidosryhmien odotusten mukaisesti.

Testauksen tavoitteet voivat vaihdella riippuen tilanteesta, johon vaikuttavat testattava tuotos, testaustaso, riskit, käytettävä ohjelmistokehityksen elinkaari (Software Development LifeCycle, SDLC) sekä liiketoimintaan liittyvät tekijät, kuten yritysrakenne, kilpailutekijät tai markkinoillesaantiaika.

1.1.2. Testaus ja vikojenjäljitys

Testaus ja vikojenjäljitys ovat eri toimenpiteitä. Testauksella voidaan saada aikaan häiriöitä, jotka johtuvat ohjelmiston vioista (dynaaminen testaus) tai sillä voidaan löytää suoraan vikoja testauksen kohteesta (staattinen testaus).

Kun dynaamisella testauksella (katso luku 4) saadaan aikaan häiriö, vikojenjäljitykseen liittyy häiriön syiden (vikojen) löytäminen, analysointi ja poistaminen. Tässä tapauksessa tyypilliseen vikojenjäljitysprosessiin kuuluu

- häiriön toistaminen
- diagnosointi (juurisyyn löytäminen)
- syyn korjaaminen.

Tätä seuraavassa varmistustestauksessa tarkistetaan, ratkaisivatko korjaukset ongelman. Varmistustestauksen tekee mieluiten sama henkilö, joka suoritti alkuperäisen testin. Lisäksi voidaan tehdä myös regressiotestausta sen tarkistamiseksi, aiheuttivatko korjaukset vikoja testauksen kohteen muissa osissa (ks. kohta 2.2.3 lisätietoja varmistustestauksesta ja regressiotestauksesta).

Kun vika havaitaan staattisessa testauksessa, vikojenjäljitys liittyy vian poistamiseen. Vian toistamista tai diagnosointia ei tarvita, koska staattinen testaus löytää suoraan vikoja eikä voi aiheuttaa häiriöitä (katso luku 3).

1.2. Miksi testaus on tarpeen?

Testaus laadunvalvonnan muotona auttaa saavuttamaan sovitut tavoitteet sovitun testauksen laajuuden, ajan, laadun ja budjetin puitteissa. Testauksen myötävaikutus onnistumiseen ei saisi rajoittua testausryhmän toimintaan. Kuka tahansa sidosryhmän edustaja voi käyttää testaustaitojaan tuodakseen projektin lähemmäs onnistumista. Komponenttien, järjestelmien ja niihin liittyvän dokumentaation testaus auttaa tunnistamaan vikoja ohjelmistossa.

1.2.1. Testauksen myötävaikutus onnistumiseen

Testaus on kustannustehokas keino löytää vikoja. Löydetyt viat voidaan sitten poistaa (vikojenjäljityksellä - testauksen ulkopuolinen toimenpide), joten testaus myötävaikuttaa epäsuorasti korkealaatuisempien testauksen kohteiden aikaansaamiseen.

Testaus on keino arvioida suoraan testauksen kohteen laatua ohjelmistokehityksen elinkaaren eri vaiheissa. Näitä toimenpiteitä käytetään osana tukemassa laajempia projektinhallintatoimenpiteitä, kun tehdään päätöksiä siirtymisestä ohjelmistokehityksen elinkaaren seuraavaan vaiheeseen, kuten julkaisupäätös.

Testaus toimii käyttäjien epäsuorana edustajana kehitysprojektissa. Testaajat varmistavat, että heidän ymmärryksensä käyttäjien tarpeista otetaan huomioon koko kehityksen elinkaaren ajan. Vaihtoehtona on ottaa kehitysprojektiin mukaan edustava joukko käyttäjiä, mikä ei yleensä ole mahdollista korkeiden kustannusten ja sopivien käyttäjien saatavuuden puutteen vuoksi.

Testausta voidaan vaatia myös sopimusperusteisten tai lakisääteisten vaatimusten täyttämiseksi tai lakisääteisten standardien noudattamiseksi.

1.2.2. Testaus ja laadunvarmistus (Quality Assurance, QA)

Vaikka ihmiset käyttävät usein termejä "testaus" ja "laadunvarmistus" merkitykseltään samanlaisina, testaus ja laadunvarmistus eivät ole sama asia. Testaus on laadunvalvonnan (Quality Control, QC) muoto.

Laadunvalvonta on tuotekeskeinen, korjaava lähestymistapa, joka keskittyy niihin toimintoihin, jotka tukevat sovitun laatutason saavuttamista. Testaus on keskeinen tapa toteuttaa laadunvalvontaa, johon lisäksi kuuluu myös muita muodollisia menetelmiä (mallin tarkistus ja oikeellisuuden todistaminen), simulointeja ja prototyyppien luomista.

Laadunvarmistus on prosessisuuntautunut, ennaltaehkäisevä lähestymistapa, joka keskittyy prosessien noudattamiseen ja parantamiseen. Se perustuu ajatukseen, että jos hyvää prosessia noudatetaan oikein, tuloksena syntyy hyvä tuote. Laadunvarmistus koskee sekä kehitys- että testausprosesseja, ja se on kaikkien projektissa mukana olevien vastuulla.

Testituloksia käytetään sekä laadunvarmistuksessa että laadunvalvonnassa. Laadunvalvonnassa niitä käytetään vikojen korjaamiseen, kun taas laadunvarmistuksessa ne tuottavat palautetta siitä, kuinka hyvin kehitys- ja testausprosessit toimivat.

1.2.3. Virheet, viat, häiriöt ja juurisyyt

Ihmiset tekevät virheitä, jotka tuottavat vikoja (bugeja), jotka puolestaan voivat johtaa häiriöihin. Ihmiset tekevät virheitä eri syistä, kuten kiireessä tai tuotosten, prosessien, infrastruktuurin tai vuorovaikutusten monimutkaisuuden vuoksi, tai yksinkertaisesti siksi, että he ovat väsyneitä tai heiltä puuttuu riittävä koulutus.

Vikoja voi löytyä dokumentaatiosta, kuten vaatimusmäärittelyistä tai testiskripteistä, lähdekoodista tai työtä tukevista tuotoksista, kuten koontitiedostosta. Jos aikaisemmin ohjelmistokehityksen elinkaaressa syntyneiden tuotoksien vikoja ei huomata, ne johtavat usein viallisiin tuotoksiin myöhemmin elinkaaren aikana. Jos vian sisältämä koodi suoritetaan, järjestelmä ei ehkä tee sitä, mitä sen pitäisi tehdä, tai tekee jotain, mitä sen ei pitäisi, ja tuloksena on häiriö. Jotkut viat johtavat aina häiriöön, jos ne suoritetaan, kun taas toiset johtavat häiriöön vain tietyissä olosuhteissa ja jotkut eivät välttämättä johda häiriöön koskaan.

Virheet ja viat eivät ole ainoa syy häiriöihin. Häiriöt voivat johtua myös ympäristöolosuhteista, kuten säteilystä tai sähkömagneettisista kentistä, jotka voivat aiheuttaa laiteohjelmistovikoja.

Juurisyy on ongelman esiintymiseen vaikuttava perustavanlaatuinen syy (esim. virheeseen johtava tilanne). Juurisyy tunnistetaan juurisyyanalyysillä, joka tehdään tyypillisesti, kun häiriö ilmenee tai vika tunnistetaan. Uskotaan, että muita vastaavia häiriöitä tai vikoja voidaan estää tai niiden esiintymistiheyttä vähentää puuttumalla juurisyyhyn, esimerkiksi poistamalla se.

1.3. Testauksen periaatteet

Vuosien varrella on ehdotettu useita testaukseen liittyviä periaatteita, jotka tarjoavat kaikkeen testaukseen päteviä yleisiä ohjeita. Tässä sertifikaattisisällössä kuvataan seitsemän tällaista periaatetta.

1. Testaus osoittaa vikojen olemassaolon, ei niiden puuttumista. Testauksella voidaan osoittaa, että testauksen kohteessa on vikoja, mutta sillä ei voida todistaa, että vikoja ei ole (Buxton 1970). Testaus vähentää todennäköisyyttä, että vikoja jää havaitsematta testauksen kohteessa, mutta vaikka vikoja ei löydy, testaus ei voi todistaa testauksen kohteen oikeellisuutta.

2. Täysin kattava testaus on mahdotonta. Kaiken testaaminen ei ole mahdollista paitsi vähäpätöisissä tapauksissa (Manna 1978). Sen sijaan, että yritettäisiin tehdä täysin kattavaa testausta, testauksen kohdentamiseen pitäisi käyttää testaustekniikoita (ks. luku 4), testitapausten priorisointia (ks. kohta 5.1.5) sekä riskipohjaista testausta (ks. kohta 5.2).

3. Aikainen testaus säästää aikaa ja rahaa. Prosessin aikaisessa vaiheessa poistetut viat eivät aiheuta myöhempiä vikoja materiaalin perusteella laadituissa tuotoksissa. Laatukustannukset pienenevät, koska ohjelmistokehityksen elinkaaressa esiintyy myöhemmin vähemmän vikoja (Boehm 1981). Jotta viat löydettäisiin varhaisessa vaiheessa, sekä staattinen testaus (katso luku 3) että dynaaminen testaus (ks. luku 4) pitäisi aloittaa mahdollisimman aikaisin.

4. Viat kasaantuvat. Pieni määrä järjestelmäkomponentteja sisältää yleensä suurimman osan havaituista vioista tai on vastuussa suurimmasta osasta käytön aikaisista häiriöistä (Enders 1975). Tämä ilmiö on esimerkki Pareton periaatteesta. Ennustetut vikakasautumat ja testauksen tai käytön aikana havaitut todelliset vikakasautumat ovat tärkeitä tietoja riskipohjaisessa testauksessa (ks. kohta 5.2).

5. Testit menettävät tehonsa. Jos samat testit toistetaan monta kertaa, niistä tulee yhä tehottomampia uusien vikojen havaitsemisessa (Beizer 1990). Tämän ilmiön välttämiseksi olemassa olevia testejä ja testiaineistoa on ehkä muokattava ja on ehkä laadittava uusia testejä. Joissakin tapauksissa samojen testien toistaminen voi kuitenkin tuottaa hyödyllisiä tuloksia, esimerkiksi automatisoidussa regressiotestauksessa (ks. kohta 2.2.3).

6. Testaus on tilanneriippuvaista. Testaukseen ei ole olemassa yhtä yleisesti sovellettavaa lähestymistapaa. Testausta tehdään eri tavoin eri tilanteissa (Kaner 2011).

7. Vikojen puuttumisen harhaluulo. On harhaluulo (eli väärinkäsitys) kuvitella, että ohjelmiston todentaminen varmistaa järjestelmän onnistumisen. Kaikkien määriteltyjen vaatimusten perusteellinen testaaminen ja kaikkien havaittujen vikojen korjaaminen voi silti tuottaa järjestelmän, joka ei täytä käyttäjien tarpeita ja odotuksia, joka ei auta saavuttamaan asiakkaan liiketoimintatavoitteita ja joka on huonompi verrattuna muihin, kilpaileviin järjestelmiin. Todentamisen lisäksi pitäisi huolehtia myös kelpuutuksesta. (Boehm 1981).

1.4. Testaustoimenpiteet, testimateriaali ja testaukseen liittyvät roolit

Testaus on tilanneriippuvaista, mutta ylemmällä tasolla on olemassa yhteisiä testaukseen liittyviä toimenpiteiden joukkoja, joita ilman testaus ei todennäköisesti saavuta sille asetettuja tavoitteita. Nämä toimenpiteiden joukot muodostavat testausprosessin. Testausprosessi voidaan räätälöidä tiettyyn tilanteeseen sopivaksi eri tekijöiden perusteella. Se, mitkä toimenpiteet sisällytetään testausprosessiin, miten ne toteutetaan ja milloin ne tapahtuvat, päätetään yleensä osana kulloisenkin tilanteen testaussuunnittelua (ks. kohta 5.1).

Seuraavissa kohdissa kuvataan tämän testausprosessin yleisiä piirteitä testaustoimenpiteiden ja -tehtävien, tilanteen vaikutuksen, testimateriaalin, testauksen pohjamateriaalin ja testimateriaalin välisen jäljitettävyyden sekä testauksen roolien osalta.

ISO/IEC/IEEE 29119-2 -standardi tarjoaa lisätietoja testausprosesseista.

1.4.1. Testaustoimenpiteet ja -tehtävät

Testausprosessi koostuu yleensä jäljempänä kuvatuista päätoimenpideryhmistä. Vaikka monet näistä toimenpiteistä saattavat näyttää noudattavan loogista järjestystä, ne toteutetaan usein iteratiivisesti tai rinnakkain. Nämä testaustoimenpiteet on yleensä räätälöitävä järjestelmän ja projektin mukaan.

Testauksen suunnittelu sisältää testauksen tavoitteiden määrittämisen ja sellaisen lähestymistavan valitsemisen, jolla kyseiset tavoitteet parhaiten saavutetaan kokonaistilanteen asettamissa rajoissa. Testauksen suunnittelua selitetään tarkemmin kohdassa 5.1.

Testauksen seuranta ja hallinta. Testauksen seurantaan kuuluu kaikkien testaustoimenpiteiden jatkuva seuraaminen ja todellisen edistymisen vertaaminen suunniteltua vastaan. Testauksen hallintaan kuuluu testauksen tavoitteiden saavuttamiseksi tarvittavien toimenpiteiden toteuttaminen. Testauksen seurantaa ja hallintaa kuvataan tarkemmin kohdassa 5.3.

Testianalyysi sisältää testauksen pohjamateriaalin analysoinnin testattavien ominaisuuksien tunnistamiseksi sekä niihin liittyvien testattavien tilanteiden määrittämisen ja priorisoinnin niihin liittyvien riskien ja riskitasojen perusteella (ks. kohta 5.2). Testauksen pohjamateriaali ja testauksen kohteet arvioidaan myös niiden mahdollisesti sisältämien vikojen tunnistamiseksi sekä niiden testattavuuden arvioimiseksi. Testianalyysia voidaan usein tehostaa käyttämällä testaustekniikoita (ks. luku 4). Testianalyysi vastaa kysymykseen "mitä testata?" mitattavissa olevien kattavuuskriteerien näkökulmasta.

Testien suunnittelu sisältää testattavien tilanteiden laventamisen testitapauksiksi ja muuksi testimateriaaliksi (esim. testausohjeet). Tähän toimenpiteeseen kuuluu usein kattavuuskohteiden tunnistaminen, jotka sitten toimivat pohjana testitapausten syötearvoja määriteltäessä. Testaustekniikoita (ks. luku 4) voidaan käyttää tämän toimenpiteen tukena. Testien suunnittelu sisältää myös testiaineistoon liittyvien vaatimusten määrittelyn, testiympäristön suunnittelun sekä muiden tarvittavien infrastruktuurien ja työkalujen tunnistamisen. Testien suunnittelu vastaa kysymykseen "miten testata?".

Testien valmistelu sisältää testien suorittamiseksi tarvittavan testimateriaalin (esim. testiaineiston) laatimisen tai hankkimisen. Testitapaukset voidaan järjestää testiproseduureiksi ja ne ryhmitellään usein testijoukoiksi. Manuaaliset ja automatisoidut testiskriptit luodaan. Testiproseduurit priorisoidaan ja järjestetään testien suoritusaikataulun puitteissa mahdollisimman tehokkaaseen suoritusjärjestykseen (ks. kohta 5.1.5).Testiympäristö pystytetään ja sen oikea toteutus todennetaan

Testien suorittaminen sisältää testien suorittamisen testien suoritusaikataulun mukaisesti (testiajot). Testien suorittaminen voi olla manuaalista tai automaattista. Testien suorittaminen voidaan tehdä monella tavalla, mukaan lukien jatkuva testaus tai paritestaussessiot. Todellisia testituloksia verrataan odotettuihin tuloksiin. Testitulokset kirjataan. Poikkeavuudet analysoidaan niiden todennäköisten syiden tunnistamiseksi. Analyysin pohjalta voidaan poikkeamat raportoida havaittujen häiriöiden perusteella (ks. kohta 5.5).

Testauksen päättämistoimenpiteet suoritetaan yleensä projektin määräajankohdissa (esim. julkaisu, iteraation loppu, testaustason valmistuminen) ja ne kohdistuvat selvittämättömiin vikoihin, muutospyyntöihin tai luotuihin tuotteen kehitysjonon kohtiin. Jatkossa mahdollisesti hyödyllinen testimateriaali tunnistetaan ja arkistoidaan tai luovutetaan asianmukaisille tiimeille. Testiympäristö suljetaan sovittuun tilaan. Testaustoimenpiteet analysoidaan kokemusten ja kehitysehdotusten tunnistamiseksi tulevia iteraatioita, julkaisuja tai projekteja varten (katso kohta 2.1.6). Testauksen loppuraportti luodaan ja toimitetaan sidosryhmille.

1.4.2. Testausprosessi eri tilanteissa

Testausta ei tehdä eristyksissä muusta ympäristöstä. Testaustoiminta on olennainen osa organisaatiossa toteutettavia kehitysprosesseja. Testausta rahoittavat myös sidosryhmät ja sen lopullisena tavoitteena on auttaa täyttämään sidosryhmien liiketoiminnalliset tarpeet. Siksi tapa, jolla testaus suoritetaan, riippuu useista tilannesidonnaisista tekijöistä, mukaan lukien

- sidosryhmät (tarpeet, odotukset, vaatimukset, yhteistyöhalukkuus jne.)
- tiimin jäsenet (taidot, tiedot, kokemustaso, käytettävissä olo, koulutustarpeet jne.)

- liiketoiminta-alue (testauksen kohteen kriittisyys, tunnistetut riskit, markkinoiden tarpeet, erityiset lakeihin pohjautuvat määräykset jne.)
- tekniset tekijät (ohjelmiston tyyppi, tuotearkkitehtuuri, käytetty teknologia jne.)
- projektin rajoitteet (laajuus, aika, budjetti, resurssit jne.)
- organisatoriset tekijät (organisaatiorakenne, käytössä olevat politiikat, noudatettavat käytännöt jne.)
- ohjelmistokehityksen elinkaari (suunnittelukäytännöt, kehitysmenetelmät jne.)
- työkalut (saatavuus, käytettävyys, vaatimustenmukaisuus jne.).

Nämä tekijät vaikuttavat moniin testaukseen liittyviin asioihin, mukaan lukien testausstrategia, käytettävät testaustekniikat, testiautomaation aste, vaadittu kattavuustaso, testausdokumentaation yksityiskohtaisuuden taso, raportointi jne.

1.4.3. Testimateriaali

Testimateriaali syntyy kohdassa 1.4.1 kuvattujen testaustoimenpiteiden tuotoksena. Se, miten eri organisaatiot tuottavat, muokkaavat, nimeävät, järjestävät ja hallinnoivat tuotoksiaan, vaihtelee paljon. Kunnollinen kokoonpanonhallinta (katso kohta 5.4) varmistaa tuotoksien yhdenmukaisuuden ja eheyden. Seuraava luettelo tuotoksista ei ole tyhjentävä.

- **Testauksen suunnittelun tuotoksiin** kuuluvat testaussuunnitelma, testauksen aikataulu, riskirekisteri sekä aloitus- ja lopetuskriteerit (ks. kohta 5.1). Riskirekisteri sisältää luettelon riskeistä ja niiden todennäköisyydestä, vaikutuksesta sekä tietoa riskien pienentämisestä (ks. luku 5.2). Testauksen aikataulu, riskirekisteri sekä aloitus- ja lopetuskriteerit ovat usein osa testaussuunnitelmaa.
- **Testauksen seurannan ja hallinnan tuotoksiin** kuuluvat testauksen edistymisraportit (ks. kohta 5.3.2), hallintaan liittyvien toimenpiteiden dokumentaatio (ks. kohta 5.3) sekä riskitiedot (ks. kohta 5.2).
- **Testianalyysin tuotoksia** ovat (priorisoidut) testattavat tilanteet (esim. hyväksymiskriteerit, ks. kohta 4.5.2) ja vikaraportit liittyen testauksen pohjamateriaalissa oleviin vikoihin (jos niitä ei ole korjattu suoraan).
- **Testien suunnittelun tuotoksia** ovat (priorisoidut) testitapaukset, testausohjeet, kattavuuskohteet sekä testiaineistoon ja testiympäristöön liittyvät vaatimukset.
- **Testien valmistelun tuotoksia** ovat testiproseduurit, automatisoidut testiskriptit, testijoukot, testiaineisto, testien suoritusaikataulu ja testiympäristön elementit. Esimerkkejä testiympäristön elementeistä ovat tyngät, ajurit, simulaattorit ja palveluiden virtualisoinnit.
- **Testien suorituksen tuotoksia** ovat testilokit ja vikaraportit (katso kohta 5.5).
- **Testauksen päättämisen tuotoksia** ovat testauksen loppuraportti (ks. kohta 5.3.2), toimenpiteet myöhempien projektien tai iteraatioiden parantamiseksi, dokumentoidut opitut kokemukset ja muutospyynnöt (esim. tuotteen kehitysjonon kohtina).

1.4.4. Jäljitettävyys testauksen pohjamateriaalin ja testimateriaalin välillä

Tehokkaan seurannan ja hallinnan toteuttamiseksi on tärkeää varmistaa jäljitettävyys ja ylläpitää sitä koko testausprosessin ajan testauksen pohjamateriaalin osien, näihin osiin liittyvän testimateriaalin (esim. testattavat tilanteet, riskit, testitapaukset), testitulosten ja löydettyjen vikojen välillä.

Tarkka jäljitettävyys tukee kattavuuden arviointia, joten on erittäin hyödyllistä, jos testauksen pohjamateriaalissa on määritelty mitattavissa olevat kattavuuskriteerit. Kattavuuskriteerit voivat toimia keskeisinä suorituskykyindikaattoreina (Key Performance Indicator, KPI) ohjaamassa toimenpiteitä, jotka osoittavat, missä määrin testauksen tavoitteet on saavutettu (ks. kohta 1.1.1). Esimerkiksi:

- Testitapausten ja vaatimusten välisellä jäljitettävyydellä voidaan varmistaa, että vaatimukset on katettu testitapauksilla.
- Testitulosten ja riskien välistä jäljitettävyyttä voidaan käyttää arvioitaessa testauksen kohteen jäljellä olevien riskien tasoa.

Kattavuuden arvioinnin lisäksi hyvä jäljitettävyys mahdollistaa muutosten vaikutusten määrittämisen, helpottaa testien auditointia ja auttaa täyttämään IT-hallinnon kriteerit. Hyvä jäljitettävyys tekee myös testauksen edistymis- ja loppuraporteista helpommin ymmärrettäviä, jos ne sisältävät testauksen pohjamateriaalin osien tilan. Tämä voi myös auttaa viestimään testauksen teknisistä näkökohdista sidosryhmille ymmärrettävällä tavalla. Jäljitettävyys tarjoaa tietoa, jonka avulla voidaan arvioida tuotteen laatua, prosessien kyvykkyyttä ja projektin etenemistä liiketoiminnan tavoitteisiin nähden.

1.4.5. Testauksen roolit

Tässä sertifikaattisisällössä käsitellään kahta testauksen pääroolia: testauksen hallintaan liittyvää roolia ja testausroolia. Näille kahdelle roolille nimetyt toimenpiteet ja tehtävät riippuvat eri tekijöistä, kuten projektin ja tuotteen tilanteesta, rooleissa toimivien ihmisten taidoista ja organisaatiosta.

Testauksen hallinnan rooli ottaa kokonaisvastuun testausprosessista, testaustiimistä sekä testaustoimenpiteiden johtamisesta. Testauksen hallinnan rooli keskittyy pääasiassa testauksen suunnitteluun, seurantaan ja hallintaan sekä testauksen päättämistehtäviin. Tapa, jolla testauksen hallinnan rooli toteutetaan, vaihtelee tilanteen mukaan. Esimerkiksi ketterässä ohjelmistokehityksessä ketterä tiimi voi hoitaa osan testauksen hallinnan tehtävistä. Useita tiimejä tai koko organisaatiota käsittäviä tehtäviä voivat suorittaa kehitystiimin ulkopuoliset testauspäälliköt.

Testausrooli ottaa kokonaisvastuun testauksen teknisestä puolesta. Testausrooli keskittyy pääasiassa testianalyysiin, testien suunnitteluun, testien valmisteluun ja testien suorittamiseen.

Eri ihmiset voivat toimia näissä rooleissa eri aikoina. Esimerkiksi testauksen hallintaroolia voi hoitaa tiimipäällikkö, testauspäällikkö, kehityspäällikkö jne. On myös mahdollista, että yksi henkilö hoitaa samaan aikaan testauksen ja testauksen hallinnan rooleja.

1.5. Testauksen keskeiset taidot ja hyvät käytännöt

Taito on henkilön omista tiedoista, kokemuksesta ja kyvyistä syntyvä kyvykkyys tehdä jotain hyvin. Hyvillä testaajilla tulisi olla eräitä keskeisiä taitoja, jotta he voivat tehdä työnsä hyvin. Hyvien testaajien tulisi olla tehokkaita tiimipelaajia ja pystyä tekemään testausta testauksen riippumattomuuden eri tasoilla .

1.5.1. Testauksessa vaadittavat yleiset taidot

Vaikka seuraavat taidot ovat yleisiä, ne ovat erityisen tärkeitä testaajille:

- Testausosaaminen (testauksen tehokkuuden kasvattamiseksi esim. testaustekniikoita käyttämällä)
- Perusteellisuus, huolellisuus, uteliaisuus, yksityiskohtien huomioiminen, järjestelmällisyys (vikojen, erityisesti vaikeasti löydettävien, tunnistamiseksi)

- Hyvät viestintätaidot, aktiivinen kuuntelu, tiimipelaajana toimiminen (kaikkien sidosryhmien kanssa tehokkaasti työskentelemiseksi, tiedon välittämiseksi muille, asioiden ymmärretyksi saamiseksi sekä vikojen raportoimiseksi ja niistä keskustelemiseksi)
- Analyyttinen ajattelu, kriittinen ajattelu, luovuus (testauksen tuottavuuden kasvattamiseksi)
- Tekninen osaaminen (testauksen tehostamiseksi esimerkiksi käyttämällä asianmukaisia testausvälineitä)
- Toimialueen tuntemus (loppukäyttäjien/liiketoiminnan edustajien ymmärtämiseksi ja heidän kanssaan kommunikoimiseksi).

Testaajat ovat usein huonojen uutisten tuojia. On yleinen inhimillinen piirre syyttää huonojen uutisten tuojaa. Tämä tekee viestintätaidoista ratkaisevan tärkeitä testaajille. Testituloksista kertominen voidaan kokea kritiikkinä tuotetta ja sen tekijää kohtaan. Vahvistusharha voi vaikeuttaa sellaisen tiedon hyväksymistä, joka on ristiriidassa nykyisten uskomusten kanssa. Jotkut ihmiset saattavat pitää testausta tuhoavana toimenpiteenä, vaikka se vaikuttaa suuresti projektin onnistumiseen ja tuotteiden laatuun. Tämän näkemyksen parantamiseksi tiedot vioista ja häiriöistä pitäisi viestiä rakentavalla tavalla.

1.5.2. Tiimiperustainen lähestymistapa

Yksi testaajan tärkeistä taidoista on kyky työskennellä tehokkaasti tiimissä ja vaikuttaa positiivisesti tiimin tavoitteisiin. Tiimiperustainen lähestymistapa - käytäntö, joka on peräisin Extreme Programming -ohjelmistokehitysmallista (ks. kohta 2.1) - perustuu tähän taitoon.

Tiimiperustaisessa lähestymistavassa jokainen tiimin jäsen, jolla on tarvittavat tiedot ja taidot, voi suorittaa minkä tahansa tehtävän, ja jokainen on vastuussa laadusta. Tiimin jäsenet työskentelevät samassa työtilassa (fyysisessä tai virtuaalisessa), koska yhteinen sijainti helpottaa viestintää ja vuorovaikutusta. Tiimiperustainen lähestymistapa parantaa tiimidynamiikkaa, edistää viestintää ja yhteistyötä tiimin sisällä ja luo synergiaa mahdollistamalla tiimiläisten eri taitojen hyödyntämisen projektin hyödyksi.

Testaajat tekevät tiivistä yhteistyötä muiden tiimin jäsenten kanssa sen varmistamiseksi, että haluttu laatutaso saavutetaan. Tähän kuuluu yhteistyö liiketoiminnan edustajien kanssa ja heidän tukemisensa sopivien hyväksymistestien luomiseksi sekä työskentely toteuttajien kanssa testausstrategiasta sopimiseksi ja testausautomaation lähestymistavoista päättämiseksi. Testaajat voivat näin siirtää testausosaamista muille tiimin jäsenille ja vaikuttaa tuotteen kehitykseen.

Tilanteesta riippuen tiimiperustainen lähestymistapa ei välttämättä aina ole sopiva. Joissakin, esimerkiksi turvallisuuskriittisissä, tilanteissa voidaan tarvita korkeaa testauksen riippumattomuuden tasoa.

1.5.3. Testauksen riippumattomuus

Tietynasteinen riippumattomuus tekee testaajasta tehokkaamman löytämään vikoja, ja tämä johtuu tekijän ja testaajan kognitiivisten harhojen välisistä eroista (vrt. Salman 1995). Riippumattomuus ei kuitenkaan korvaa tuttuutta; esimerkiksi toteuttajat voivat tehokkaasti löytää monia vikoja omasta koodistaan.

Tuotoksia voi testata niiden tekijä (ei riippumattomuutta), tekijän vertaiset samasta tiimistä (jonkin verran riippumattomuutta), testaajat tekijän tiimin ulkopuolelta mutta saman organisaation sisältä (korkea riippumattomuus) tai testaajat organisaation ulkopuolelta (erittäin korkea riippumattomuus). Useimmissa projekteissa on yleensä parasta käyttää testauksessa useita riippumattomuuden tasoja (esim. kehittäjät suorittavat komponentti- ja komponentti-integraatiotestauksen, testausryhmä suorittaa järjestelmä- ja järjestelmäintegraatiotestauksen ja liiketoiminnan edustajat suorittavat hyväksymistestauksen).

Testauksen riippumattomuuden tärkein etu on, että riippumattomat testaajat tunnistavat todennäköisesti erilaisia vikoja ja häiriöitä toteuttajiin verrattuna erilaisten taustojen, teknisten näkökulmien ja ennakkoasenteiden vuoksi. Lisäksi riippumaton testaaja voi todentaa, kyseenalaistaa tai kumota oletuksia, joita sidosryhmät ovat tehneet järjestelmän määrittelyn ja toteutuksen aikana.

Riippumattomuuteen liittyy kuitenkin myös joitakin haittoja. Riippumattomat testaajat voidaan eristää toteutustiimistä, mikä voi johtaa yhteistyön puutteeseen, viestintäongelmiin tai vihamieliseen suhteeseen toteutustiimin kanssa. Toteuttajat voivat menettää vastuuntuntonsa laadun suhteen. Riippumattomia testaajia voidaan pitää pullonkaulana tai heitä voidaan syyttää julkaisun viivästymisestä.

2. Testaus ohjelmistokehityksen elinkaaren aikana - 130 minuuttia

Avainsanat

ei-toiminnallinen testaus, hyväksymistestaus, integraatiotestaus, järjestelmäintegraatiotestaus, järjestelmätestaus, komponentti-integraatiotestaus, komponenttitestaus, lasilaatikkotestaus, mustalaatikkotestaus, regressiotestaus, shift-left, testauksen kohde, testaustaso, testaustyyppi, toiminnallinen testaus, varmistustestaus, ylläpitotestaus

Luvun 2 oppimistavoitteet:

2.1 Testaus ohjelmistokehityksen elinkaaressa

FL-2.1.1 (K2) Osaa selittää valitun ohjelmistokehityksen elinkaaren vaikutuksen testaukseen

FL-2.1.2 (K1) Muistaa hyvät testauskäytännöt, jotka koskevat kaikkia ohjelmistokehityksen elinkaaria

FL-2.1.3 (K1) Muistaa esimerkit testit ensin -lähestymistavoista kehityksessä

FL-2.1.4 (K2) Osaa kuvata, miten DevOps voi vaikuttaa testaukseen

FL-2.1.5 (K2) Osaa selittää shift-left-lähestymistavan

FL-2.1.6 (K2) Osaa selittää, miten jälkipalavereita voidaan käyttää prosessin parantamisen mekanismina

2.2 Testaustasot ja testaustyypit

FL-2.2.1 (K2) Erottaa eri testaustasot toisistaan

FL-2.2.2 (K2) Erottaa eri testaustyypit toisistaan

FL-2.2.3 (K2) Erottaa varmistustestauksen regressiotestauksesta

2.3 Ylläpitotestaus

FL-2.3.1 (K2) Osaa kuvata ylläpitotestauksen ja sen käynnistävät tekijät

2.1. Testaus ohjelmistokehityksen elinkaaren yhteydessä

Ohjelmistokehityksen elinkaarimalli on abstrakti, korkean tason esitys ohjelmistokehitysprosessista. Elinkaarimalli määrittelee, miten tässä prosessissa suoritetut eri kehitysvaiheet ja toimenpiteet liittyvät toisiinsa sekä loogisesti että kronologisesti. Esimerkkejä elinkaarimalleista ovat peräkkäismalliset ohjelmistokehitysmallit (esim. vesiputousmalli, V-malli), iteratiiviset kehitysmallit (esim. spiraalimalli, prototyypit) ja inkrementaaliset kehitysmallit (esim. Unified Process).

Joitakin ohjelmistokehitysprosessien toimenpiteitä voidaan kuvata myös yksityiskohtaisempien ohjelmistokehitysmenetelmien ja ketterien käytäntöjen pohjalta. Esimerkkejä ovat hyväksymistestiohjattu ohjelmistokehitys (Acceptance Test Driven Development, ATDD), käyttäytymiseen perustuva ohjelmistokehitys (Behavior Driven Development, BDD), sovellusaluekeskeinen suunnittelu (domain-driven design, DDD), eXtreme Programming (XP), ominaisuuksiin perustuva kehitys (Feature-Driven Development, FDD), Kanban, Lean IT, Scrum ja testiohjattu kehitys (Test-Driven Development, TDD).

2.1.1. Ohjelmistokehityksen elinkaaren vaikutus testaukseen

Testauksen onnistumiseksi se täytyy mukauttaa ohjelmistokehityksen elinkaareen. Elinkaarimallin valinta vaikuttaa

- testaustoimenpiteiden laajuuteen ja ajoitukseen (esim. testaustasot ja testaustyypit)
- testidokumentaation yksityiskohtaisuuden tasoon
- testaustekniikoiden ja testauksen lähestymistapojen valintaan
- testiautomaation laajuuteen
- testaajan rooliin ja vastuisiin.

Peräkkäismallisessa ohjelmistokehityksessä testaajat osallistuvat alkuvaiheessa tyypillisesti vaatimusten katselmointeihin, testianalyysiin ja testien suunnitteluun. Suoritettava koodi luodaan yleensä myöhemmissä vaiheissa, joten tyypillisesti dynaamista testausta ei voida tehdä ohjelmistokehityksen varhaisessa vaiheessa.

Joissakin iteratiivisissa ja inkrementaalisissa kehitysmalleissa oletetaan, että jokainen iteraatio tuottaa toimivan prototyypin tai tuoteinkrementin. Tämä tarkoittaa, että kussakin iteraatiossa voidaan tehdä sekä staattista että dynaamista testausta kaikilla testaustasoilla. Inkrementtien tiivis toimitus edellyttää nopeaa palautetta ja laajaa regressiotestausta.

Ketterässä ohjelmistokehityksessä oletetaan, että muutoksia voi tapahtua koko projektin ajan. Siksi ketterissä projekteissa suositaan kevyttä tuotosten dokumentointia ja laajaa testiautomaatiota regressiotestauksen helpottamiseksi. Lisäksi suurin osa manuaalisesta testauksesta tehdään yleensä käyttämällä kokemuspohjaisia testaustekniikoita (ks. kohta 4.4), jotka eivät edellytä laajaa etukäteen tehtävää testianalyysiä ja testien suunnittelua.

2.1.2. Ohjelmistokehityksen elinkaari ja hyvät testauskäytännöt

Valitusta ohjelmistokehityksen elinkaarimallista riippumatta hyviin testauskäytäntöihin kuuluvat seuraavat asiat:

- Jokaista ohjelmistokehityksen toimenpidettä vastaa testaustoimenpide, jotta kaikki kehitystoimenpiteet ovat laadunvalvonnan alaisia.
- Eri testaustasoilla (ks. luku 2.2.1) on määrätyt ja toisistaan poikkeavat testaustavoitteet, mikä mahdollistaa testauksen riittävän kattavuuden ja samalla päällekkäisyyksien välttämisen.

- Tietyn testaustason testianalyysi ja testien suunnittelu alkaa ohjelmistokehityksen elinkaarimallin vastaavassa vaiheessa, jotta testauksessa voidaan noudattaa aikaisen testauksen periaatetta (ks. kohta 1.3).
- Testaajat osallistuvat tuotosten katselmointeihin heti, kun materiaalin luonnokset ovat saatavilla, jotta aikaisempi testaus ja vikojen havaitseminen voivat tukea shift-left-strategiaa (katso kohta 2.1.5).

2.1.3. Testaus ohjelmistokehityksen ohjaajana

TDD, ATDD ja BDD ovat samankaltaisia kehityslähestymistapoja, joissa testit määritellään ohjelmistokehityksen ohjausvälineeksi. Kaikissa näissä lähestymistavoissa noudatetaan aikaisen testauksen periaatetta (ks. kohta 1.3) ja sovelletaan shift-left-lähestymistapaa (ks. kohta 2.1.5), koska testit määritellään ennen koodin kirjoittamista. Ne tukevat iteratiivista kehitysmallia. Näitä lähestymistapoja luonnehditaan seuraavasti:

Testiohjattu ohjelmistokehitys (TDD):

- Ohjaa koodausta testitapausten kautta (laajan ohjelmistosuunnittelun sijaan) (Beck 2003).
- Testit kirjoitetaan ensin, sitten kirjoitetaan testien vaatima koodi, ja sen jälkeen testit ja koodi refaktoroidaan.

Hyväksymistestiohjattu ohjelmistokehitys (ATDD) (ks. kohta 4.5.3):

- Testit johdetaan hyväksymiskriteereistä osana järjestelmän suunnitteluprosessia (Gärtner 2011).
- Testit kirjoitetaan ennen kuin testien edellyttämä sovelluksen osa toteutetaan.

Käyttäytymiseen perustuva ohjelmistokehitys (BDD)

- Sovelluksen toivottu käyttäytyminen kuvataan testitapauksina, jotka on kirjoitettu yksinkertaisella luonnollisella kielellä, jota sidosryhmien on helppo ymmärtää - yleensä käyttämällä Kun/Jos/Sitten-muotoa (Tšelimski 2010).
- Testitapaukset muunnetaan sen jälkeen automaattisesti suoritettaviksi testeiksi.

Kaikissa edellä mainituissa lähestymistavoissa testit voidaan säilyttää automatisoituina testeinä koodin laadun varmistamiseksi tulevien muokkausten yhteydessä/refaktoroinnissa.

2.1.4. DevOps ja testaus

DevOps on organisatorinen lähestymistapa, jonka tavoitteena on luoda synergiaa saamalla kehitys (testaus mukaan luettuna) ja tuotanto toimimaan yhdessä yhteisten tavoitteiden saavuttamiseksi. DevOps vaatii organisaatiossa kulttuurimuutosta, jotta kehityksen (testaus mukaan luettuna) ja tuotannon väliset kuilut voidaan kuroa umpeen ja samalla kohdella niiden toimintoja samanarvoisina. DevOps edistää tiimin itsenäisyyttä, nopeaa palautetta, integroituja työkaluketjuja ja teknisiä käytäntöjä, kuten jatkuvaa integraatiota (Continuous Integration, CI) ja jatkuvaa toimitusta (Continuous Delivery, CD). Näiden avulla tiimit voivat rakentaa, testata ja julkaista korkealaatuista koodia nopeammin DevOps-toimitusputken läpi (Kim 2016).

Testauksen näkökulmasta joitain DevOpsin etuja ovat seuraavat:

- Nopea palaute koodin laadusta ja siitä, vaikuttavatko muutokset haitallisesti olemassa olevaan koodiin.

- CI edistää shift-left-lähestymistapaa testauksessa (ks. kohta 2.1.5) kannustamalla kehittäjiä toimittamaan korkealaatuista koodia, jolle on tehty yksikkötestit ja staattinen analyysi.
- DevOps edistää automatisoituja prosesseja, kuten CI/CD, jotka helpottavat vakaiden testiympäristöjen luomista.
- DevOps tuo ei-toiminnalliset laatuominaisuudet paremmin esiin (esim. suorituskyky, luotettavuus).
- Automaatio toimitusputken läpi vähentää toistuvan manuaalisen testauksen tarvetta.
- Regressioriski on minimoitu automatisoitujen regressiotestien laajuuden ja määrän ansiosta.

DevOpsiin liittyy kuitenkin myös riskejä ja haasteita:

- DevOps-toimitusputki on määriteltävä ja luotava.
- CI/CD-työkalut on otettava käyttöön ja niitä on ylläpidettävä.
- Testiautomaatio vaatii lisäresursseja, ja sen käyttöönotto ja ylläpito voi olla vaikeaa.

Vaikka DevOpsiin liittyy pitkälle viety automatisoitu testaus, manuaalista testausta - erityisesti käyttäjän näkökulmasta - tarvitaan edelleen.

2.1.5. Shift-left-lähestymistapa

Aikaisen testauksen periaatetta (ks. kohta 1.3) kutsutaan joskus shift-leftiksi, koska se on lähestymistapa, jossa testaus tehdään aikaisemmin ohjelmistokehityksen elinkaaressa. Shift-left lähtee yleensä ajatuksesta, että testaus tulisi tehdä aikaisemmin (esim. ei jäädä odottamaan koodin toteutusta tai komponenttien integrointia), mutta se ei tarkoita, että testauksen myöhemmin ohjelmistokehityksen elinkaaressa voisi jättää tekemättä.

On olemassa joitakin hyviä käytäntöjä, jotka havainnollistavat, miten testauksessa saavutetaan "shift-left", mukaan lukien

- määrittelyjen katselmointi testauksen näkökulmasta. Näissä määrittelyiden katselmointitoimenpiteet paljastavat usein mahdollisia vikoja, kuten tulkinnanvaraisuuksia, puutteita ja epäjohdonmukaisuuksia.
- testitapausten kirjoittaminen ennen koodin kirjoittamista ja koodin suorittaminen testikehyksessä koodin toteutuksen aikana
- CI:n ja vielä mieluummin CD:n käyttäminen, koska se tuottaa nopeaa palautetta ja sisältää automatisoidut yksikkötestit, jotka seuraavat lähdekoodin mukana, kun se kirjataan koodikirjastoon
- lähdekoodin staattisen analyysin suorittaminen ennen dynaamista testausta tai osana automatisoitua prosessia
- ei-toiminnallisen testauksen tekeminen yksikkötestaustasosta alkaen, mikäli mahdollista. Tämä on eräänlainen shift-leftin muoto, koska ei-toiminnalliset testaustyypit suoritetaan yleensä myöhemmin ohjelmistokehityksen elinkaaressa, kun koko järjestelmä ja edustava testiympäristö ovat käytettävissä.

Shift-left voi johtaa ylimääräisiin koulutuksiin, lisätyöhön ja/tai lisäkustannuksiin prosessin aikaisemmassa vaiheessa, mutta sen odotetaan säästävän aikaa ja/tai kustannuksia myöhemmin prosessissa.

Shift-left-lähestymistavan kannalta on tärkeää, että sidosryhmät ymmärtävät sen merkityksen ja ovat sitoutuneet sen noudattamiseen.

2.1.6. Jälkipalaverit ja prosessien parantaminen

Jälkipalaverit (tunnetaan myös nimellä retrospektiivit pidetään usein projektin tai iteraation lopussa, julkaisun eri tarkistuspisteiden kohdalla tai niitä voidaan pitää tarpeen mukaan. Jälkipalavereiden ajoitus ja organisointi riippuu noudatettavasta ohjelmistokehityksen elinkaarimallista. Näissä palavereissa osallistujat (testaajien lisäksi myös esim. toteuttajat, arkkitehdit, tuoteomistajat, liiketoiminta-analyytikot) keskustelevat siitä,

- mikä onnistui ja pitäisi säilyttää
- mikä ei onnistunut ja mitä voitaisiin parantaa
- kuinka sisällyttää parannukset toimintaan ja toistaa onnistumiset tulevaisuudessa.

Tulokset on kirjattava ylös ja ne ovat yleensä osa testauksen loppuraporttia (ks. kohta 5.3.2). Jälkipalaverit ovat kriittisiä jatkuvan parantamisen onnistuneen toteuttamisen kannalta, ja on tärkeää, että kaikkien suositeltujen parannuksien toteuttamista seurataan.

Tyypillisiä hyötyjä testaukselle ovat:

- parempi testauksen tehokkuus/tuottavuus (esim. toteuttamalla prosessin kehitysehdotuksia)
- testimateriaalin laadun paraneminen (esim. katselmoimalla testausprosessit yhdessä)
- tiimin sitoutuminen ja oppiminen (esim. tuloksena mahdollisuudesta nostaa esiin ongelmia ja ehdottaa parannuskohtia)
- testauksen pohjamateriaalin laadun paraneminen (esim. koska vaatimusten laajuuteen ja laatuun liittyvät viat voidaan tuoda esiin ja ratkaista)
- parempi yhteistyö toteutuksen ja testauksen välillä (esim. kun yhteistyötä arvioidaan ja optimoidaan säännöllisesti).

2.2. Testaustasot ja testityypit

Testaustaso on joukko testaustoimenpiteitä, joita organisoidaan ja hallitaan yhdessä. Kukin testaustaso on testausprosessin ilmentymä, joka suoritetaan tietyssä ohjelmiston kehitysvaiheessa, yksittäisistä komponenteista kokonaisiin järjestelmiin tai, tilanteen mukaan, jopa järjestelmien järjestelmiin.

Testaustasot liittyvät ohjelmistokehityksen elinkaarimallin muihin toimenpiteisiin. Peräkkäiselinkaarimalleissa testaustasot määritellään usein niin, että yhden testaustason lopetuskriteerit ovat osa seuraavan tason aloituskriteerejä. Joissakin iteratiivisissa malleissa tämä ei välttämättä päde. Kehitystoimenpiteet voivat jatkua useiden testaustasojen läpi. Testaustasot voivat mennä ajallisesti päällekkäin.

Testaustyypit ovat tiettyihin laatuominaisuuksiin liittyviä testaustoimenpiteiden joukkoja ja suurin osa näistä toimenpiteistä voidaan tehdä kaikilla testaustasoilla.

2.2.1. Testaustasot

Tässä sertifikaattisisällössä kuvataan seuraavat viisi testaustasoa:

- **Komponenttitestaus** (tunnetaan myös nimellä yksikkötestaus) keskittyy yksittäisten komponenttien testaamiseen erikseen. Se vaatii usein erityistä tukea, kuten testipetejä tai yksikkötestauskehyksiä. Yleensä komponenttitestauksen tekevät toteuttajat kehitysympäristöissään.

- **Komponentti-integraatiotestaus** (tunnetaan myös nimellä yksikköintegraatiotestaus) keskittyy komponenttien välisten rajapintojen ja vuorovaikutuksen testaamiseen. Komponentti-integraatiotestaus riippuu voimakkaasti integroinnin strategisista lähestymistavoista, kuten alhaalta ylös, ylhäältä alas tai big-bang.
- **Järjestelmätestaus** keskittyy koko järjestelmän tai tuotteen yleiseen käyttäytymiseen ja kykyihin, mukaan lukien usein päästä-päähän -tehtävien toiminnallinen testaus ja laatuominaisuuksien ei-toiminnallinen testaus. Joidenkin ei-toiminnallisten laatuominaisuuksien osalta on suositeltavaa testata ne käyttämällä koko järjestelmää sopivassa testiympäristössä (esim. käytettävyys). Myös osajärjestelmien simulointi on mahdollista. Järjestelmätestauksen voi suorittaa riippumaton testausryhmä, ja se liittyy järjestelmän määrittelyihin.
- **Järjestelmäintegraatiotestaus keskittyy** testattavan järjestelmän ja muiden järjestelmien sekä ulkoisten palveluiden välisten rajapintojen testaamiseen. Järjestelmäintegraatiotestaus edellyttää sopivia, mielellään tuotantoympäristöjä vastaavia testiympäristöjä.
- **Hyväksymistestaus** keskittyy kelpuutukseen ja käyttöönottovalmiuden osoittamiseen, mikä tarkoittaa, että järjestelmä täyttää käyttäjän liiketoimintatarpeet. Ihannetapauksessa hyväksymistestauksen suorittavat järjestelmän aiotut käyttäjät. Hyväksymistestauksen tärkeimmät muodot ovat: käyttäjän hyväksymistestaus (User Acceptance Testing, UAT), käyttöön soveltuvuuden hyväksymistestaus, sopimuksiin ja säädöksiin perustuva hyväksymistestaus, alfa-testaus ja beta-testaus.

Testaustasot erotetaan toisistaan seuraavassa ei-tyhjentävässä listassa kuvattujen ominaisuuksien avulla, jotta vältetään testaustoimenpiteiden päällekkäisyydet:

- testauksen kohde
- testauksen tavoitteet
- testauksen pohjamateriaali
- viat ja häiriöt
- lähestymistavat ja vastuut.

2.2.2. Testaustyypit

On olemassa paljon eri testaustyyppejä, joita voidaan soveltaa projekteissa. Tässä sertifikaattisisällössä käsitellään seuraavia neljää testaustyyppiä.

Toiminnallinen testaus arvioi toiminnot, jotka komponentin tai järjestelmän tulisi suorittaa. Toiminnot ovat se, "mitä" testauksen kohteen pitäisi tehdä. Toiminnallisen testauksen päätavoitteena on tarkistaa toiminnallinen valmius, toiminnallinen oikeellisuus ja toiminnallinen tarkoituksenmukaisuus.

Ei-toiminnallinen testaus arvioi komponentin tai järjestelmän muita kuin toiminnallisia ominaisuuksia. Ei-toiminnallisessa testauksessa testataan, "kuinka hyvin järjestelmä käyttäytyy". Ei-toiminnallisen testauksen päätavoitteena on tarkistaa ohjelmiston ei-toiminnalliset laatuominaisuudet. ISO/IEC 25010 -standardi tarjoaa seuraavan luokittelun ei-toiminnallisille ohjelmiston laatuominaisuuksille:

- suorituskyvyn tehokkuus
- yhteensopivuus
- käytettävyys
- luotettavuus

- tietoturva
- ylläpidettävyys
- siirrettävyys.

Joskus on tarkoituksenmukaista, että ei-toiminnallinen testaus aloitetaan elinkaaren aikaisessa vaiheessa (esim. osana katselmointeja ja komponenttitestausta tai järjestelmätestausta). Monet ei-toiminnalliset testit johdetaan toiminnallisista testeistä, koska niissä käytetään samoja toiminnallisia testejä, mutta toimintoa suoritettaessa tarkistetaan, että ei-toiminnallinen vaatimus täyttyy (esim. tarkistetaan, että toiminto suoritetaan tietyssä ajassa tai että toiminnallisuus voidaan siirtää uudelle alustalle). Ei-toiminnallisten vikojen myöhäinen havaitseminen voi aiheuttaa vakavan uhan projektin onnistumiselle. Ei-toiminnallisessa testauksessa tarvitaan joskus hyvin erityinen testiympäristö, kuten käytettävyyslaboratorio käytettävyystestausta varten.

Mustalaatikkotestaus (ks. kohta 4.2) perustuu määrittelyihin ja siinä johdetaan testitapaukset testauksen kohteen ulkopuolisesta dokumentaatiosta. Mustalaatikkotestauksen päätavoitteena on tarkistaa järjestelmän käyttäytyminen sen määrittelyitä vastaan.

Lasilaatikkotestaus (ks. kohta 4.3) on rakennepohjaista ja siinä testitapaukset johdetaan järjestelmän toteutuksen tai sisäisen rakenteen (esim. koodi, arkkitehtuuri, työnkulut ja tietovirrat) pohjalta. Lasilaatikkotestauksen päätavoitteena on käydä testitapauksilla läpi ohjelmiston rakenne riittävän kattavasti.

Kaikkia neljää edellä mainittua testaustyyppiä voidaan käyttää kaikilla testaustasoilla, vaikka painopiste on erilainen joka tasolla. Eri testitekniikoita voidaan käyttää testattavien tilanteiden ja testitapausten johtamiseen kaikissa mainituissa testaustyypeissä.

2.2.3. Varmistustestaus ja regressiotestaus

Komponenttiin tai järjestelmään tehdään yleensä muutoksia joko sen parantamiseksi lisäämällä uusia ominaisuuksia tai sen korjaamiseksi poistamalla vikoja. Tällöin testaukseen pitäisi kuulua myös varmistustestaus ja regressiotestaus.

Varmistustestauksessa varmistetaan, että alkuperäinen vika on korjattu onnistuneesti. Riskistä riippuen ohjelmiston korjattu versio voidaan testata useilla tavoilla, kuten

- suorittamalla kaikki testitapaukset, jotka ovat aiemmin epäonnistuneet vian vuoksi, tai myös
- lisäämällä uusia testejä kattamaan kaikki muutokset, jotka olivat tarpeen vian korjaamiseksi.

Jos kuitenkin aikaa tai rahaa vikojen korjaamiseen on vähän, varmistustestaus voidaan rajoittaa vain niiden vaiheiden suorittamiseen, joiden pitäisi toistaa vian aiheuttama häiriö, ja sen tarkistamiseen, ettei häiriötä ilmene.

Regressiotestaus varmistaa, että muutokset, mukaan luettuna jo varmistustestattu korjaus, eivät ole aiheuttaneet haitallisia seurauksia. Nämä seuraukset voivat vaikuttaa samaan komponenttiin, johon muutos tehtiin, saman järjestelmän muihin komponentteihin tai jopa muihin liittymäjärjestelmiin. Regressiotestaus ei välttämättä rajoitu itse testauksen kohteeseen, vaan se voi liittyä myös ympäristöön. Suositeltavaa on tehdä ensin vaikutusanalyysi regressiotestauksen laajuuden optimoimiseksi. Vaikutusanalyysi näyttää, mihin ohjelmiston osiin muutos voi vaikuttaa.

Regressiotestijoukot ajetaan monta kertaa ja yleensä regressiotestitapausten määrä kasvaa jokaisen iteraation tai julkaisun myötä, joten regressiotestaus on vahva ehdokas automatisoitavaksi. Näiden testien automatisointi tulisi aloittaa projektin aikaisessa vaiheessa. Jos käytetään jatkuvaa integraatiota, kuten esimerkiksi DevOpsissa (ks. kohta 2.1.4), on hyvä käytäntö sisällyttää myös automatisoidut regressiotestit siihen. Tilanteesta riippuen tämä voi käsittää regressiotestejä eri tasoilla.

Testauksen kohteen varmistustestausta ja/tai regressiotestausta tarvitaan kaikilla testaustasoilla, jos vikoja korjataan ja/tai muutoksia tehdään näillä testaustasoilla.

2.3. Ylläpitotestaus

Ylläpitoa on eri tyyppejä: se voi olla korjaavaa, sillä voidaan mukautua ympäristön muutoksiin tai parantaa suorituskykyä tai ylläpidettävyyttä (katso lisätietoja ISO/IEC 14764 -standardista). Näin ollen ylläpito voi sisältää suunniteltuja julkaisuja/käyttöönottoja ja suunnittelemattomia julkaisuja/käyttöönottoja (pikakorjauksia). Ennen muutoksen tekemistä voidaan tehdä vaikutusanalyysi, joka järjestelmän muissa osissa mahdollisesti ilmenevien seurausten perusteella auttaa päättämään, kannattaako muutosta tehdä. Tuotantokäytössä olevan järjestelmän muutosten testaaminen sisältää sekä muutoksen toteutuksen onnistumisen arvioinnin että mahdollisten sivuvaikutusten tarkistamisen järjestelmän niissä osissa, joita ei ole muutettu (mikä on yleensä suurin osa järjestelmästä).

Ylläpitotestauksen laajuus riippuu yleensä

- muutokseen liittyvän riskin suuruudesta
- nykyisen järjestelmän koosta
- muutoksen koosta.

Ylläpidon ja ylläpitotestauksen käynnistävät tekijät voidaan luokitella seuraavasti:

- muutokset, kuten suunnitellut parannukset (eli julkaisupohjaiset), korjaavat muutokset tai pikakorjaukset
- toimintaympäristön päivitykset tai migraatiot esimerkiksi alustalta toiselle, jotka voi edellyttää sekä uuteen ympäristöön että muuttuneeseen ohjelmistoon liittyviä testejä tai tietojen konvertoinnin testausta, kun aineistoa siirretään toisesta sovelluksesta ylläpidettävään järjestelmään
- eläköityminen, esimerkiksi kun sovellus saavuttaa käyttöikänsä lopun. Kun järjestelmä poistetaan käytöstä, se voi edellyttää tietojen arkistoinnin testaamista, jos vaaditaan pitkiä tietojen säilytysaikoja. Arkistoinnin jälkeisten palautus- ja hakumenettelyjen testaus voi olla tarpeen myös siinä tapauksessa, että arkistoinnin aikana tarvitaan tiettyjä tietoja.

3. Staattinen testaus - 80 minuuttia

Avainsanat

dynaaminen testaus, epämuodollinen katselmointi, katselmointi, läpikäynti, muodollinen katselmointi, poikkeamat, staattinen analyysi, staattinen testaus, tarkastus, tekninen katselmointi

Luvun 3 oppimistavoitteet:

3.1 Staattisen testauksen perusteet

FL-3.1.1 (K1) Tunnistaa tuotokset, joita voidaan tutkia erilaisilla staattisilla testaustekniikoilla

FL-3.1.2 (K2) Osaa selittää staattisen testauksen arvon

FL-3.1.3 (K2) Pystyy vertaamaan ja vertailemaan staattista ja dynaamista testausta

3.2 Palaute ja katselmointiprosessi

FL-3.2.1 (K1) Tunnistaa sidosryhmien varhaisen ja säännöllisen palautteen edut

FL-3.2.2 (K2) Osaa kuvata katselmointiprosessin toimenpiteet

FL-3.2.3 (K1) Muistaa, mitkä vastuut on osoitettu päärooleille katselmointeja tehtäessä

FL-3.2.4 (K2) Pystyy vertaamaan ja vertailemaan eri katselmointityyppejä

FL-3.2.5 (K1) Muistaa tekijät, jotka vaikuttavat onnistuneeseen katselmointiin

3.1. Staattisen testauksen perusteet

Toisin kuin dynaamisessa testauksessa, staattisessa testauksessa testattavaa ohjelmistoa ei tarvitse suorittaa. Koodia, prosessimäärittelyitä, järjestelmäarkkitehtuurimäärittelyitä tai muita tuotoksia arvioidaan tarkastelemalla niitä manuaalisesti (esim. katselmoinnit) tai työkalun avulla (esim. staattinen analyysi). Testauksen tavoitteisiin kuuluvat laadun parantaminen, vikojen löytäminen ja ominaisuuksien, kuten luettavuuden, valmiuden, oikeellisuuden, testattavuuden ja yhdenmukaisuuden, arviointi. Staattista testausta voidaan soveltaa sekä todentamiseen että kelpuutukseen.

Testaajat, liiketoiminnan edustajat ja toteuttajat työskentelevät yhdessä esimerkkikartoituksissa, käyttäjätarinoita kirjoitettaessa ja kehitysjonon jalostusistunnoissa sen varmistamiseksi, että käyttäjätarinat ja niihin liittyvät tuotokset täyttävät niille määritellyt kriteerit, kuten Valmiin määritelmän (katso kohta 5.1.3). Katselmointitekniikoita voidaan käyttää varmistamaan, että käyttäjätarinat ovat valmiita ja ymmärrettäviä ja sisältävät testattavat hyväksymiskriteerit. Kysymällä oikeita kysymyksiä testaajat tutkivat, haastavat ja auttavat parantamaan ehdotettuja käyttäjätarinoita.

Staattinen analyysi voi tunnistaa ongelmia ennen dynaamista testausta ja vaatii usein vähemmän vaivaa, koska testitapauksia ei tarvita ja tyypillisesti käytetään työkaluja (katso luku 6). Staattinen analyysi sisällytetään usein jatkuvan integraation kehyksiin (ks. kohta 2.1.4). Vaikka staattista analyysiä käytetään suurelta osin tiettyjen koodivikojen havaitsemiseen, sitä käytetään myös ylläpidettävyyden ja tietoturvan arviointiin. Oikeinkirjoituksen tarkistusohjelmat ja luettavuustyökalut ovat muita esimerkkejä staattisen analyysin työkaluista.

3.1.1. Staattisella testauksella testattavat tuotokset

Lähes mikä tahansa tuotos voidaan testata staattisella testauksella. Esimerkkejä ovat vaatimusten määrittelydokumentit, lähdekoodi, testaussuunnitelmat, testitapaukset, tuotteen kehitysjonon kohdat, testausohjeet, projektidokumentaatio, sopimukset ja mallit.

Kaikki tuotokset, jotka voidaan lukea ja ymmärtää, voidaan katselmoida. Staattista analyysiä varten tuotokset tarvitsevat kuitenkin rakenteen, jota vastaan niitä voidaan verrata (esim. mallit, koodi tai teksti, joilla on muodollinen syntaksi).

Staattiseen testaukseen eivät sovi tuotokset, joita ihmisten on vaikea tulkita ja joita ei pitäisi analysoida työkaluilla (esim. 3. osapuolen suoritettava koodi oikeudellisista syistä johtuen).

3.1.2. Staattisen testauksen arvo

Staattinen testaus voi havaita viat ohjelmistokehityksen elinkaaren aikaisimmissa vaiheissa ja täyttää näin aikaisen testauksen periaatteen (ks. kohta 1.3). Se voi myös tunnistaa vikoja, joita ei voida löytää dynaamisella testauksella (esim. saavuttamaton koodi, suunnittelumallit, joita ei ole toteutettu halutulla tavalla, viat ei-suoritettavissa tuotoksissa).

Staattinen testaus tarjoaa mahdollisuuden arvioida tuotosten laatua ja rakentaa luottamusta niihin. Todentamalla dokumentoidut vaatimukset sidosryhmät voivat myös varmistaa, että ne vastaavat heidän todellisia tarpeitaan. Koska staattinen testaus voidaan tehdä ohjelmistokehityksen elinkaaren aikaisessa vaiheessa, mukana olevien sidosryhmien kesken pystytään luomaan yhteisymmärrys. Mukana olevien sidosryhmien välinen kommunikointi myös paranee. Tästä syystä on suositeltavaa ottaa staattiseen testaukseen mukaan edustajia monenlaisista sidosryhmistä.

Vaikka katselmointien toteuttaminen voi olla kallista, projektin kokonaiskustannukset ovat yleensä paljon pienemmät kuin silloin, kun katselmointeja ei tehdä, koska vikojen korjaamiseen tarvitaan vähemmän aikaa ja vaivaa myöhemmin projektissa.

Viat koodissa voidaan havaita staattisella analyysillä tehokkaammin kuin dynaamisella testauksella, mikä yleensä johtaa sekä vähempiin vikoihin koodissa että pienempään kokonaistyömäärään.

3.1.3. Staattisen ja dynaamisen testauksen erot

Staattisen testauksen ja dynaamisen testauksen käytännöt täydentävät toisiaan. Niillä on samanlaisia tavoitteita, kuten tuotoksissa olevien vikojen löytämisen tukeminen (ks. kohta 1.1.1), mutta niissä on myös joitakin eroja, kuten:

- Staattinen ja dynaaminen testaus (häiriöiden analysoinnin pohjalta) voivat molemmat johtaa vikojen löytämiseen, mutta on olemassa joitain vikatyyppejä, jotka löytyvät vain joko staattisella tai dynaamisella testauksella.
- Staattinen testaus löytää suoraan vikoja, kun taas dynaaminen testaus aiheuttaa häiriöitä, joihin liittyvät viat määritetään häiriön tarkemman analyysin avulla.
- Staattinen testaus voi löytää helpommin vikoja, jotka ovat koodin läpi kulkevilla harvemmin käytetyillä poluilla tai joihin on vaikea päästä käsiksi dynaamisella testauksella.
- Staattista testausta voidaan soveltaa ei-suoritettaviin tuotoksiin, kun taas dynaamista testausta voidaan käyttää vain suoritettaviin tuotoksiin.
- Staattista testausta voidaan käyttää mittaamaan laatuominaisuuksia, jotka eivät ole riippuvaisia koodin suorittamisesta (esim. ylläpidettävyys), kun taas dynaamista testausta voidaan käyttää mittaamaan laatuominaisuuksia, jotka ovat riippuvaisia koodin suorittamisesta (esim. suorituskyvyn tehokkuus).

Tyypillisiä vikoja, jotka on helpompi ja/tai halvempi löytää staattisen testauksen avulla, ovat:

- viat vaatimuksissa (esim. epäjohdonmukaisuudet, epäselvyydet, ristiriidat, puutteet, epätarkkuudet, päällekkäisyydet)
- suunnitteluviat (esim. tehottomat tietokantarakenteet, huono modularisointi)
- tietyntyyppiset koodausvirheet (esim. määrittelemättömät muuttujat, muuttujat, joille ei ole annettu arvoja, saavuttamaton tai kopioitu koodi, liiallinen koodin monimutkaisuus)
- poikkeamat standardeista (esim. koodausstandardien mukaisten nimeämiskäytäntöjen noudattamatta jättäminen)
- väärät rajapintamääritykset (esim. parametrien yhteensopimaton määrä, tyyppi tai järjestys)
- tietyntyyppiset tietoturvahaavoittuvuudet (esim. puskurin ylivuodot)
- aukot tai epätarkkuudet testauksen pohjamateriaalin kattavuudessa (esim. hyväksymiskriteeriä koskevat puuttuvat testit).

3.2. Palaute ja arviointiprosessi

3.2.1. Sidosryhmien varhaisen ja säännöllisen palautteen edut

Varhainen ja säännöllinen palaute mahdollistaa aikaisen mahdollisista laatuongelmista viestimisen. Jos sidosryhmän edustajat ovat vain vähän mukana ohjelmistokehityksen elinkaaren aikana, kehitet-

tävä tuote ei ehkä vastaa sidosryhmän alkuperäistä tai nykyistä näkemystä. Jos sidosryhmälle ei pystytä toimittamaan sitä, mitä se haluaa, se voi johtaa kalliiseen uusintatyöhön, määräaikojen ylittymiseen, syyttelyyn ja jopa projektin täydelliseen epäonnistumiseen.

Säännöllinen palaute sidosryhmiltä koko ohjelmistokehityksen elinkaaren ajan voi estää vaatimuksiin liittyviä väärinymmärryksiä ja varmistaa, että vaatimusten muutokset ymmärretään ja toteutetaan aikaisemmin. Tämä auttaa kehitystiimiä parantamaan ymmärrystään siitä, mitä he rakentavat. Sen myötä he voivat keskittyä niihin ominaisuuksiin, jotka tuottavat eniten arvoa sidosryhmille ja joilla on myönteisin vaikutus tunnistettuihin riskeihin.

3.2.2. Katselmointiprosessin vaiheet

ISO/IEC 20246 -standardi määrittelee yleisen katselmointiprosessin, joka tarjoaa jäsennellyn mutta joustavan kehyksen, jonka pohjalta voidaan räätälöidä sopiva katselmointiprosessi tiettyyn tilanteeseen. Jos vaadittu katselmointi on muodollisempi, tarvitaan enemmän eri toimenpiteisiin liittyviä tehtäviä.

Monien tuotosten koko tekee niistä liian suuria, jotta ne voitaisiin kattaa yhdellä katselmoinnilla. Katselmointiprosessi voidaan käynnistää pari kertaa koko tuotoksen katselmoimiseksi.

Katselmointiprosessin vaiheet ovat:

- **Suunnittelu.** Suunnitteluvaiheessa on määriteltävä katselmoinnin laajuus, joka käsittää katselmoinnin tarkoituksen, katselmoitavan tuotoksen, arvioitavat laatuominaisuudet, painopistealueet, lopetuskriteerit, sekä muut lisätiedot, kuten katselmointiin liittyvät standardit, työmäärä ja aikataulu.
- **Katselmoinnin aloitus.** Katselmoinnin aloituksessa tavoitteena on varmistaa, että kaikki asianosaiset ja muut asiat ovat valmiina katselmoinnin aloittamiseksi. Tähän sisältyy sen varmistaminen, että jokaisella osallistujalla on pääsy katselmoitavaan tuotokseen, hän ymmärtää roolinsa ja vastuunsa ja saa kaiken tarvittavan katselmoinnin suorittamiseksi.
- **Yksilöllinen katselmointi.** Jokainen katselmoija suorittaa yksilöllisen katselmoinnin arvioidakseen tarkasteltavan tuotoksen laatua ja tunnistaakseen poikkeamia, suosituksia ja kysymyksiä käyttämällä yhtä tai useampaa katselmointitekniikkaa (esim. tarkistuslistaan perustuva katselmointi, skenaariopohjainen katselmointi). ISO/IEC 20246 -standardi tarjoaa eri katselmointitekniikoiden tarkempia kuvauksia. Katselmoijat kirjaavat ylös kaikki tunnistamansa poikkeamat, suositukset ja kysymykset.
- **Viestintä ja analysointi.** Koska katselmoinnin aikana havaitut poikkeamat eivät välttämättä ole vikoja, kaikki poikkeamat on analysoitava ja niistä on keskusteltava. Jokaisen poikkeaman osalta pitäisi tehdä päätös sen tilasta, omistajuudesta ja vaadituista toimista. Tämä tehdään tyypillisesti katselmointipalaverissa, jossa osallistujat päättävät myös, mikä arvioidun tuotoksen laatutaso on ja mitä jatkotoimenpiteitä tarvitaan. Toimenpiteiden loppuun saattaminen saattaa edellyttää seurantakatselmointia.
- **Korjaaminen ja raportointi.** Jokaisesta viasta pitäisi tehdä vikaraportti, jotta korjaavia toimenpiteitä voidaan seurata. Kun lopetuskriteerit on saavutettu, tuotos voidaan hyväksyä. Katselmoinnin tulokset raportoidaan.

3.2.3. Roolit ja vastuut katselmoinneissa

Katselmoinneissa on mukana useita sidosryhmiä, joilla voi olla useita rooleja. Pääroolit ja niiden vastuualueet ovat:

- johtaja – päättää, mitä katselmoidaan, ja tarjoaa resurssit, kuten henkilöstön ja katselmointiin tarvittavan ajan
- tekijä – luo ja korjaa katselmoitavan tuotoksen
- puheenjohtaja (tunnetaan myös nimellä fasilitaattori) – huolehtii katselmointipalaverin tehokkaasta läpiviennistä, mukaan lukien sovittelu, ajanhallinta ja turvallinen katselmointiympäristö, jossa kaikki voivat puhua vapaasti
- kirjuri (tunnetaan myös nimellä sihteeri) – kokoaa poikkeamat katselmoijilta ja tallentaa katselmoinnin tiedot, kuten päätökset ja katselmointipalaverin aikana löydetyt uudet poikkeamat
- katselmoija – suorittaa katselmointeja. Katselmoija voi olla projektin parissa työskentelevä henkilö, asiasisällön asiantuntija tai minkä tahansa sidosryhmän muu edustaja
- katselmoinnin vastuuhenkilö - ottaa kokonaisvastuun katselmoinnista, eli esimerkiksi päättää, kuka osallistuu katselmointiin ja milloin ja missä katselmointi tehdään.

Muut, yksityiskohtaisemmat roolit ovat mahdollisia, kuten on kuvattu ISO/IEC 20246 -standardissa.

3.2.4. Katselmointityypit

Katselmointityyppejä on monia epämuodollisista katselmoinneista muodollisiin. Vaadittu muodollisuuden taso riippuu eri tekijöistä, kuten noudatettavasta ohjelmistokehityksen elinkaarimallista, kehitysprosessin kypsyydestä, katselmoitavan tuotoksen kriittisyydestä ja monimutkaisuudesta, oikeudellisista tai muihin säädöksiin perustuvista vaatimuksista ja kirjausketjun tarpeesta. Sama tuotos voidaan katselmoida käyttämällä eri katselmointityyppejä, sille voidaan esimerkiksi tehdä ensin epämuodollinen katselmointi ja myöhemmin muodollisempi.

Oikean katselmointityypin valitseminen on avain vaadittujen katselmointitavoitteiden saavuttamiseen (ks. kohta 3.2.5). Valinta ei perustu pelkästään tavoitteisiin, vaan myös muihin tekijöihin kuten projektin tarpeisiin, käytettävissä oleviin resursseihin, tuotoksen tyyppiin ja riskeihin, liiketoiminta-alueeseen ja yrityskulttuuriin.

Joitakin yleisesti käytettyjä katselmointityyppejä ovat:

- **Epämuodollinen katselmointi.** Epämuodollisissa katselmoinneissa ei noudateta määriteltyä prosessia eikä edellytetä muodollisesti dokumentoitua lopputulosta. Päätavoitteena on poikkeamien löytäminen.
- **Läpikäynti.** Tekijän vetämä läpikäynti voi palvella monia tavoitteita, kuten tuotoksen laadun arviointia ja luottamuksen rakentamista siihen, katselmoijien kouluttamista, yhteisymmärryksen saavuttamista, uusien ideoiden luomista, tekijöiden motivointia ja heidän kehittymisensä tukemista sekä poikkeamien havaitsemista. Katselmoijat voivat tehdä yksilöllisen katselmoinnin ennen läpikäyntiä, mutta tämä ei ole pakollista.
- **Tekninen katselmointi.** Teknisen katselmoinnin suorittavat teknisesti pätevät katselmoijat ja sitä johtaa puheenjohtaja. Teknisen katselmoinnin tavoitteena on saavuttaa yksimielisyys ja tehdä päätöksiä teknisiin ongelmiin liittyen, mutta myös löytää poikkeavuuksia, arvioida tuotoksen laatua ja rakentaa luottamusta siihen, luoda uusia ideoita sekä motivoida ja auttaa tekijöitä kehittymään
- **Tarkastus.** Koska tarkastukset ovat muodollisin katselmointityyppi, niissä noudatetaan koko yleistä prosessia (ks. kohta 3.2.2). Päätavoitteena on löytää mahdollisimman paljon poikkeavuuksia. Muita tavoitteita ovat laadun arviointi, luottamuksen rakentaminen tuotokseen

sekä tekijöiden motivointi ja auttaminen kehittymään. Mittaritietoja kerätään ja käytetään ohjelmistokehityksen elinkaarimallin parantamiseen, katselmointiprosessi mukaan luettuna. Tarkastuksissa tekijä ei voi toimia katselmoinnin vastuuhenkilönä tai kirjurina

3.2.5. Katselmointien menestystekijät

Katselmointien onnistumiseen vaikuttavat useat tekijät, joihin kuuluvat:

- selkeiden tavoitteiden ja mitattavien lopetuskriteerien määrittäminen. Tavoitteena ei koskaan pitäisi olla osallistujien arvioiminen.
- sopivan katselmointityypin valitseminen asetettujen tavoitteiden saavuttamiseksi tuotoksen tyypin, katselmoinnin osallistujien, projektin tarpeiden ja tilanteen mukaan
- katselmointien tekeminen pienissä osissa, jotta katselmoijat eivät menetä keskittymiskykyään yksilöllisen katselmoinnin ja/tai katselmointipalaverin aikana (jos sellainen pidetään)
- palautteen antaminen katselmoinnista sidosryhmille ja tekijöille, jotta nämä voivat parantaa tuotetta ja toimintaansa (ks. kohta 3.2.1)
- riittävän katselmointiin valmistautumisajan antaminen osallistujille
- johdon tuki katselmointiprosessille
- katselmointien sisällyttäminen osaksi organisaation kulttuuria oppimisen ja prosessien parantamisen edistämiseksi
- riittävän koulutuksen tarjoaminen kaikille osallistujille, jotta he tietävät, miten he täyttävät tehtävänsä
- kokousten fasilitointi.

4. Testianalyysi ja testien suunnittelu - 390 minuuttia

Avainsanat

ekvivalenssiositus, haarakattavuus, hyväksymiskriteerit, hyväksymistestiohjattu kehitys, kattavuus, kattavuuskohde, kokemuspohjainen testaustekniikka, lasilaatikkotestaustekniikka, lausekattavuus, mustalaatikkotestaustekniikka, päätöstaulutestaus, raja-arvoanalyysi, tarkistuslistapohjainen testaus, testaustekniikka, tilasiirtymätestaus, tutkiva testaus, virheenarvaus, yhteistyöhön pohjautuva testauksen lähestymistapa

Luvun 4 oppimistavoitteet:

4.1 Yleistä testaustekniikoista

FL-4.1.1 (K2) Erottaa toisistaan mustalaatikko-, lasilaatikko- ja kokemuspohjaiset testaustekniikat

4.2 Mustalaatikkotekniikat

FL-4.2.1 (K3) Osaa käyttää ekvivalenssiositusta testitapausten johtamiseen

FL-4.2.2 (K3) Osaa käyttää raja-arvoanalyysiä testitapausten johtamiseen

FL-4.2.3 (K3) Osaa käyttää päätöstaulutestausta testitapausten johtamiseen

FL-4.2.4 (K3) Osaa käyttää tilasiirtymätestausta testitapausten johtamiseen

4.3 Lasilaatikkotekniikat

FL-4.3.1 (K2) Osaa selittää lausetestauksen

FL-4.3.2 (K2) Osaa selittää haaratestauksen

FL-4.3.3 (K2) Osaa selittää lasilaatikkotestauksen arvon

4.4 Kokemukseen pohjautuvat testaustekniikat

FL-4.4.1 (K2) Osaa selittää virheenarvauksen

FL-4.4.2 (K2) Osaa selittää tutkivan testauksen

FL-4.4.3 (K2) Osaa selittää tarkistuslistoihin pohjautuvan testauksen

4.5. Yhteistyöhön perustuvat testauksen lähestymistavat

FL-4.5.1 (K2) Osaa selittää, miten käyttäjätarinoita kirjoitetaan yhteistyössä kehittäjien ja liiketoiminnan edustajien kanssa

FL-4.5.2 (K2) Osaa luokitella hyväksymiskriteerien kirjoittamisen eri vaihtoehdot

FL-4.5.3 (K3) Osaa käyttää hyväksymistestiohjattua kehitystä (ATDD) testitapausten johtamiseen

4.1. Yleistä testaustekniikoista

Testaustekniikat tukevat testaajaa testianalyysissä (mitä testataan) ja testien suunnittelussa (miten testataan). Testaustekniikat auttavat laatimaan järjestelmällisesti suhteellisen pienen mutta riittävän joukon testitapauksia. Testaustekniikat auttavat myös testaajaa määrittämään testattavia tilanteita, tunnistamaan kattavuuskohteet ja tunnistamaan testiaineiston testianalyysin ja testien suunnittelun aikana. Lisätietoja testaustekniikoista ja näistä niihin liittyvistä toimenpiteistä löytyy ISO/IEC/IEEE 29119-4 -standardista ja (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jørgensen 2014, Ammann 2016, Forgács 2019).

Tässä sertifikaattisisällössä testaustekniikat luokitellaan mustalaatikko-, lasilaatikko- ja kokemuspohjaisiin tekniikoihin.

Mustalaatikkotekniikat (tunnetaan myös määrittelypohjaisina tekniikoina) perustuvat testauksen kohteen määritetyn käyttäytymisen analyysiin ilman viittauksia kohteen sisäiseen rakenteeseen. Siksi testitapaukset ovat riippumattomia siitä, miten ohjelmisto on toteutettu. Näin ollen jos toteutus muuttuu, mutta vaadittu käyttäytyminen pysyy samana, testitapaukset ovat edelleen hyödyllisiä.

Lasilaatikkotekniikat (tunnetaan myös rakennepohjaisina tekniikoina) perustuvat testauksen kohteen sisäisen rakenteen ja prosessoinnin analyysiin. Koska testitapaukset ovat riippuvaisia siitä, miten ohjelmisto on suunniteltu, ne voidaan luoda vasta testauksen kohteen suunnittelun tai toteutuksen jälkeen.

Kokemuspohjaiset testaustekniikat käyttävät tehokkaasti testaajien tietämystä ja kokemusta testitapausten suunnittelussa ja toteutuksessa. Näiden tekniikoiden tehokkuus riippuu suuresti testaajan taidoista. Kokemuspohjaisilla tekniikoilla voidaan löytää vikoja, jotka saattavat jäädä huomaamatta mustalaatikko- ja lasilaatikkotekniikoita käytettäessä. Siksi kokemuspohjaiset tekniikat täydentävät mustalaatikko- ja lasilaatikkotekniikoita.

4.2. Mustalaatikkotekniikat

Seuraavissa osissa käsitellään seuraavia yleisesti käytettyjä mustalaatikkotekniikoita:

- ekvivalenssiositus
- raja-arvoanalyysi
- päätöstaulutestaus
- tilasiirtymätestaus.

4.2.1. Ekvivalenssiositus

Ekvivalenssiositus (Equivalence Partitioning, EP) jakaa tiedot osioihin (tunnetaan myös ekvivalenssijoukkoina tai ekvivalenssiluokkina) sen oletuksen pohjalta, että testauksen kohde käsittelee tietyn osion kaikki elementit samalla tavalla. Teoria tämän tekniikan taustalla on, että jos yhtä ekvivalenssiluokan arvoa testattaessa testitapaus löytää vian, tämä vika pitäisi löytyä myös testitapauksilla, jotka testaavat minkä tahansa muun arvon samasta luokasta. Siksi yksi testi kutakin osiota kohden riittää.

Ekvivalenssiluokkia voidaan tunnistaa mihin tahansa testauksen kohteeseen liittyvään tietoelementtiin liittyen, mukaan luettuna syötteet, tulokset, kokoonpanon osat, sisäiset arvot, aikariippuvaiset arvot ja rajapintaparametrit. Luokat voivat olla jatkuvia tai erillisiä, järjestettyjä tai järjestämättömiä, äärellisiä tai äärettömiä. Luokat eivät saa olla päällekkäisiä eivätkä ne saa olla tyhjiä.

Yksinkertaisten testauksen kohteiden osalta ekvivalenssiositus voi olla helppoa, mutta käytännössä sen ymmärtäminen, miten testauksen kohde käsittelee eri arvoja, on usein monimutkaista. Siksi ekvivalenssiositus tulee tehdä huolellisesti.

Kelvollisia arvoja sisältävää osiota kutsutaan kelvolliseksi osioksi. Epäkelpoja arvoja sisältävää osiota kutsutaan epäkelvoksi osioksi. Kelvollisten ja epäkelpojen arvojen määritelmät voivat vaihdella tiimien ja organisaatioiden välillä. Esimerkiksi kelvollisiksi arvoiksi voidaan tulkita sellaiset, jotka testauksen kohteen tulisi käsitellä, tai joiden prosessointi kuvataan määrittelyissä. Epäkelvot arvot voidaan tulkita sellaisiksi, jotka testauksen kohteen tulisi jättää huomiotta tai hylätä, tai joiden prosessointia ei ole kuvattu testauksen kohteen määrittelyissä.

Ekvivalenssiosituksessa kattavuuskohteita ovat ekvivalenssiluokat. Jotta tällä tekniikalla saavutetaan 100 % kattavuus, testitapausten on testattava kaikki tunnistetut osiot (mukaan luettuna epäkelvot osiot) vähintään kerran. Kattavuus mitataan laskemalla vähintään yhdellä testitapauksella testattujen osioiden lukumäärä ja jakamalla se tunnistettujen osioiden kokonaismäärällä, ja ilmaistaan prosentteina.

Monet testauksen kohteet sisältävät useita luokkien tai osioiden joukkoja (esim. testauksen kohteet, joilla on useampi kuin yksi syöteparametri), mikä tarkoittaa, että testitapaus kattaa luokkia eri luokkien joukoista. Yksinkertaisin kattavuuskriteeri useiden osiojoukkojen tilanteessa on nimeltään jokavalintakattavuus (Ammann 2016). Jokavalintakattavuus edellyttää, että testitapaukset kattavat jokaisen osiojoukon jokaisen osion vähintään kerran. Jokavalintakattavuus ei ota huomioon osioiden yhdistelmiä.

4.2.2. Raja-arvoanalyysi

Raja-arvoanalyysi (Boundary Value Analysis, BVA) on tekniikka, joka perustuu ekvivalenssiluokkien rajojen testaamiseen. Siksi raja-arvoanalyysiä voidaan käyttää vain järjestettyihin luokkiin. Luokan pienin ja suurin arvo ovat sen raja-arvoja. Jos raja-arvoanalyysissä kaksi elementtiä kuuluu samaan luokkaan, kaikkien niiden välisten elementtien on myös kuuluttava kyseiseen luokkaan.

Raja-arvoanalyysi keskittyy luokkien raja-arvoihin, koska toteuttajat tekevät todennäköisemmin virheitä näiden raja-arvojen kanssa. Tyypilliset raja-arvoanalyysin havaitsemat viat sijaitsevat paikoissa, joissa toteutetut rajat on sijoitettu väärin tarkoitettujen sijaintien ylä- tai alapuolelle tai ne jätetty kokonaan pois.

Tämä sertifikaattisisältö kattaa kaksi versiota raja-arvoanalyysistä: 2-arvo- ja 3-arvoraja-arvoanalyysin. Ne eroavat toisistaan rajakohtaisten kattavuuskohteiden suhteen, jotka on testattava 100% kattavuuden saavuttamiseksi.

2-arvoanalyysissä (Craig 2002, Myers 2011) jokaiselle raja-arvolle on kaksi kattavuuskohdetta: kyseinen raja-arvo ja sen lähin naapuri, joka kuuluu viereiseen luokkaan. 100% kattavuuden saavuttamiseksi 2-arvotekniikalla testitapausten on testattava kaikki kattavuuskohteet eli kaikki tunnistetut raja-arvot. Kattavuus mitataan laskemalla testattujen raja-arvojen lukumäärä ja jakamalla se tunnistettujen raja-arvojen kokonaismäärällä, ja ilmaistaan prosentteina.

3-arvoanalyysissä (Koomen 2006, O'Regan 2019) jokaiselle raja-arvolle on kolme kattavuuskohdetta: kyseinen raja-arvo ja sen molemmat naapurit. Siksi 3-arvotekniikassa jotkin kattavuuskohteet eivät välttämättä ole raja-arvoja. 100% kattavuuden saavuttamiseksi 3-arvotekniikalla testitapausten on testattava kaikki kattavuuskohteet eli tunnistetut raja-arvot ja niiden naapurit. Kattavuus mitataan laskemalla testattujen raja-arvojen ja niiden naapureiden lukumäärä ja jakamalla se tunnistettujen raja-arvojen ja niiden naapureiden kokonaismäärällä, ja se ilmaistaan prosentteina.

3-arvoanalyysi on tiukempi kuin 2-arvoanalyyysi, koska se voi havaita 2-arvoanalyysissä huomaamatta jääviä vikoja. Esimerkiksi, jos päätös "jos ($x \leq 10$) ... " on toteutettu virheellisesti muodossa "jos ($x = 10$) ... ", mikään 2-arvoanalyysin pohjalta johdettu testiaineisto ($x = 10$, $x = 11$) ei pysty paljastamaan vikaa. Kuitenkin $x = 9$, johdettuna 3-arvoanalyysin pohjalta, todennäköisesti havaitsee sen.

4.2.3. Päätöstaulutestaus

Päätöstaulujen avulla testataan sellaisten järjestelmävaatimusten toteutusta, jotka määrittävät, miten erilaiset ehtojen yhdistelmät johtavat erilaisiin tuloksiin. Päätöstaulut ovat tehokas keino monimutkaisen logiikan, kuten liiketoimintasääntöjen, tallentamiseen.

Päätöstaulua luotaessa määritellään järjestelmän ehdot ja niiden tuloksena tapahtuvat toimenpiteet. Nämä muodostavat taulun rivit. Jokainen sarake vastaa päätössääntöä, joka kuvaa ehtojen yksilöllisen yhdistelmän ja siihen liittyvät toimenpiteet. Rajatussa päätöstaulussa kaikki ehtojen ja toimenpiteiden arvot (paitsi merkityksettömät tai mahdottomat; katso alla) kuvataan Boolen arvoina (tosi tai epätosi). Vaihtoehtoisesti laajennetussa päätöstaulussa joillakin tai kaikilla ehdoilla ja toimenpiteillä voi olla myös useita arvoja (esim. numeroalueet, ekvivalenssiluokat, diskreetit arvot).

Ehtojen merkintätapa on seuraava: "T" (tosi) tarkoittaa, että ehto täyttyy. "E" (epätosi) tarkoittaa, että ehto ei täyty. "–" tarkoittaa, että ehdon arvolla ei ole merkitystä toimenpiteen lopputuloksen kannalta. "N/A" tarkoittaa, että ehto on käytännössä mahdoton kyseisen säännön kannalta. Toimenpiteissä: "X" tarkoittaa, että toimenpiteen pitäisi tapahtua. Tyhjä tarkoittaa, että toimenpiteen ei pitäisi tapahtua. Myös muita merkintöjä voidaan käyttää.

Täydellisessä päätöstaulussa on tarpeeksi sarakkeita kattamaan kaikki ehtojen yhdistelmät. Taulua voidaan yksinkertaistaa poistamalla sarakkeet, jotka sisältävät mahdottomia ehtojen yhdistelmiä. Taulua voidaan pienentää myös yhdistämällä yhdeksi sarakkeeksi sarakkeet, joissa tietyt ehdot eivät vaikuta tulokseen. Päätöstaulun pienennysalgoritmit eivät kuulu tämän sertifikaattisisällön piiriin.

Päätöstaulutestauksessa kattavuuskohteita ovat sarakkeet, jotka sisältävät ehtojen mahdollisia yhdistelmiä. 100% kattavuuden saavuttamiseksi tällä tekniikalla testitapausten on testattava kaikki nämä sarakkeet. Kattavuus mitataan laskemalla testattujen sarakkeiden lukumäärä ja jakamalla se mahdollisia ehtoja sisältävien sarakkeiden kokonaismäärällä, ja ilmaistaan prosentteina.

Päätöstaulutestauksen vahvuus on, että se tarjoaa järjestelmällisen lähestymistavan kaikkien ehtojen yhdistelmien tunnistamiseen, joista osa saattaisi muuten jäädä huomaamatta. Se auttaa myös löytämään aukkoja tai ristiriitoja vaatimuksissa. Jos ehtoja on monia, kaikkien päätössääntöjen testaaminen voi olla aikaa vievää, koska sääntöjen määrä kasvaa eksponentiaalisesti ehtojen lukumäärän myötä. Tällaisessa tapauksessa testattavien sääntöjen määrän vähentämiseksi voidaan käyttää pienennettyä päätöstaulua tai riskipohjaista lähestymistapaa.

4.2.4. Tilasiirtymätestaus

Tilasiirtymäkaavio mallintaa järjestelmän käyttäytymistä kuvaamalla sen mahdolliset tilat ja kelvolliset tilasiirtymät. Siirtymän käynnistää tapahtuma, jota voidaan lisäksi ohjata porttiehdolla. Siirtymien oletetaan olevan välittömiä ja ne voivat joskus johtaa ohjelmiston suorittamiin toimenpiteisiin. Yleinen siirtymätapahtumaa kuvaavan merkinnän syntaksi on seuraava: "tapahtuma [porttiehto]/toimenpide". Porttiehdot ja toimenpiteet voidaan jättää pois, jos niitä ei ole tai ne ovat testaajan kannalta merkityksettömiä.

Tilataulukko on tilasiirtymäkaaviota vastaava malli. Sen rivit edustavat tiloja ja sen sarakkeet tapahtumia (yhdessä porttiehtojen kanssa, jos niitä on). Taulukon sisältö (solut) edustaa siirtymiä ja sisältää tulostilan sekä tuloksena tapahtuvat toimenpiteet, jos ne on määritetty. Toisin kuin tilasiirtymäkaaviosta, tilataulukosta näkyvät yksiselitteisesti myös epäkelvot siirtymät, joita edustavat tyhjät solut.

Tilasiirtymäkaavioon tai tilataulukkoon perustuva testitapaus esitetään yleensä tapahtumaketjuna, joka johtaa tilamuutosten (ja tarvittaessa toimenpiteiden) sarjaan. Yksi testitapaus voi kattaa ja yleensä kattaakin useita siirtymiä tilojen välillä.

Tilasiirtymätestaukselle on olemassa monia kattavuuskriteereitä. Tässä sertifikaattisisällössä käsitellään niistä kolmea.

Kaikkien tilojen kattavuudessa kattavuuskohteita ovat tilat. 100 % kaikkien tilojen kattavuuden saavuttamiseksi testitapausten on käytävä kaikissa tiloissa. Kattavuus mitataan laskemalla läpikäytyjen tilojen lukumäärä jaettuna tilojen kokonaismäärällä, ja se ilmaistaan prosentteina.

Kelvollisten siirtymien kattavuudessa (jota kutsutaan myös 0-switch-kattavuudeksi) kattavuuskohteita ovat yksittäiset kelvolliset siirtymät. Jotta saavutetaan 100 % kelvollisten siirtymien kattavuus, testitapausten on testattava kaikki kelvolliset siirtymät. Kattavuus mitataan laskemalla testattujen kelvollisten siirtymien lukumäärä ja jakamalla se kelvollisten siirtymien kokonaismäärällä, ja se ilmaistaan prosentteina.

Kaikkien siirtymien kattavuudessa kattavuuskohteita ovat kaikki tilataulukossa näkyvät siirtymät. 100% kaikkien siirtymien kattavuuden saavuttamiseksi testitapausten on testattava kaikki kelvolliset siirtymät ja yritettävä suorittaa epäkelpoja siirtymiä. Vain yhden epäkelvon siirtymän testaaminen yhdessä testitapauksessa auttaa välttämään vikojen peittymistä eli tilannetta, jossa yksi vika estää toisen havaitsemisen. Kattavuus mitataan laskemalla testitapauksilla testattujen tai yritettyjen kelvollisten ja epäkelpojen siirtymien lukumäärä ja jakamalla se kelvollisten ja epäkelpojen siirtymien kokonaismäärällä, ja se ilmaistaan prosentteina.

Kaikkien tilojen kattavuus on heikompi kuin kelvollisten siirtymien kattavuus, koska se voidaan tyypillisesti saavuttaa ilman kaikkien siirtymien testausta. Kelvollisten siirtymien kattavuus on yleisimmin käytetty kattavuuskriteeri. Täyden kelvollisten siirtymien kattavuuden saavuttaminen takaa täyden kaikkien tilojen kattavuuden. Kaikkien siirtymien täyden kattavuuden saavuttaminen takaa sekä täyden kaikkien tilojen kattavuuden että täyden kelvollisten siirtymien kattavuuden, ja sen pitäisi olla vähimmäisvaatimus tehtävä- ja turvallisuuskriittisille ohjelmistoille.

4.3. Lasilaatikkotekniikat

Tässä osiossa keskitytään kahteen koodiin liittyvään lasilaatikkotekniikkaan niiden suosion ja yksinkertaisuuden vuoksi:

- lausetestaus
- haaratestaus.

Joissakin turvallisuuskriittisissä, tehtäväkriittisissä tai tiukkaa säännönmukaisuutta vaativissa ympäristöissä käytetään tehokkaampia tekniikoita perusteellisemman koodikattavuuden saavuttamiseksi. On myös olemassa lasilaatikkotekniikoita, joita käytetään korkeammilla testaustasoilla (esim. API-testaus) tai jotka käyttävät kattavuuksia, jotka eivät liity koodiin (esim. neuronien kattavuus neuroverkkotestauksessa). Näitä tekniikoita ei käsitellä tässä sertifikaattisisällössä.

4.3.1. Lausetestaus ja lausekattavuus

Lausetestauksessa kattavuuskohteita ovat suoritettavat lauseet. Tavoitteena on suunnitella testitapauksia, jotka käyvät läpi koodin lauseita, kunnes hyväksyttävä kattavuustaso saavutetaan. Kattavuus mitataan laskemalla testitapausten läpikäymien lauseiden lukumäärä, joka jaetaan koodissa olevien suoritettavien lauseiden kokonaismäärällä, ja se ilmaistaan prosentteina.

Kun 100% lausekattavuus saavutetaan, se varmistaa, että kaikki koodin suoritettavat lauseet on suoritettu vähintään kerran. Tämä tarkoittaa erityisesti sitä, että myös jokainen vian sisältävä lause suoritetaan, mikä voi aiheuttaa häiriön, joka osoittaa vian olemassaolon. Lauseen suorittaminen testitapauksella ei kuitenkaan kaikissa tapauksissa löydä vikoja. Se ei esimerkiksi välttämättä paljasta vikoja, jotka ovat aineistoriippuvaisia (esim. jako nollalla, mikä epäonnistuu vain, kun nimittäjän arvoksi on asetettu nolla). 100% lausekattavuus ei myöskään takaa, että koko päätöksentekologiikka on testattu, koska esimerkiksi kaikkia koodin haaroja ei välttämättä käydä läpi (ks. luku 4.3.2).

4.3.2. Haaratestaus ja haarakattavuus

Haara tarkoittaa hallinnan siirtoa kontrollivuokaavion kahden solmun välillä. Kontrollivuokaavio kuvaa mahdolliset sarjat, joissa testauksen kohteen lähdekoodin lauseet suoritetaan. Jokainen hallinnan siirto voi olla joko ehdoton (eli suoraan etenevä koodi) tai ehdollinen (eli päätösvaihtoehto).

Haaratestauksessa kattavuuskohteita ovat koodin haarat ja tavoitteena on suunnitella testitapauksia niiden läpikäymiseksi, kunnes hyväksyttävä kattavuustaso saavutetaan. Kattavuus mitataan laskemalla testitapausten läpikäymien haarojen lukumäärä ja jakamalla se haarojen kokonaismäärällä, ja se ilmaistaan prosentteina.

Kun saavutetaan 100% haarakattavuus, kaikki koodin haarat, ehdottomat ja ehdolliset, on käyty läpi testitapauksilla. Ehdolliset haarat vastaavat yleensä jos-niin-päätöskohdan tosi- tai epätosi-päätösvaihtoehtoja, switch/case-lausekkeen tulosta tai päätöstä poistua silmukasta tai jatkaa siinä. Haaran testaaminen testitapauksella ei kuitenkaan kaikissa tapauksissa löydä vikoja. Se ei esimerkiksi välttämättä paljasta vikoja, jotka edellyttävät tietyn polun suorittamista koodissa.

Haarakattavuus sisältää lausekattavuuden. Tämä tarkoittaa, että mikä tahansa testitapausten joukko, joka saavuttaa 100% haarakattavuuden, saavuttaa myös 100% lausekattavuuden (mutta ei päinvastoin).

4.3.3. Lasilaatikkotestauksen arvo

Kaikkiin lasilaatikkotekniikoihin liittyvä perusvahvuus on se, että koko ohjelmiston toteutus otetaan huomioon testauksen aikana, mikä helpottaa vikojen havaitsemista myös silloin, kun ohjelmiston määrittelyt ovat epämääräisiä, vanhentuneita tai puutteellisia. Vastaavasti heikkous on, että jos ohjelmistoon ei ole toteutettu yhtä tai useampaa vaatimusta, lasilaatikkotestaus ei välttämättä havaitse puutteesta johtuvia vikoja (Watson 1996).

Lasilaatikkotekniikoita voidaan käyttää staattisessa testauksessa (esim. koodin pöytätestauksen aikana). Ne soveltuvat hyvin katselmointeihin, joissa kohteena on koodi, joka ei ole vielä valmis suoritettavaksi (Hetzel 1988), samoin kuin pseudokoodi tai muu korkean tason tai ylhäältä alas -logiikka, joka voidaan mallintaa kontrollivuokaaviolla.

Pelkän mustalaatikkotestauksen tekeminen ei tuota tietoja koodin todellisesta kattavuudesta. Lasilaatikkokattavuuden mittaaminen tarjoaa objektiivista mittaritietoa kattavuudesta ja tuottaa tarvittavaa tietoa lisätestien luomista varten kattavuuden kasvattamiseksi ja kasvattaa näin luottamusta jatkossa koodiin.

4.4. Kokemuspohjaiset testaustekniikat

Tässä kohdassa käsitellään seuraavia yleisesti käytettyjä kokemuspohjaisia testaustekniikoita:

- virheenarvaus
- tutkiva testaus
- tarkistuslistoihin perustuva testaus.

4.4.1. Virheenarvaus

Virheenarvaus on tekniikka, jota käytetään virheiden, vikojen ja häiriöiden esiintymisen ennakointiin testaajan tietämyksen perusteella, mukaan luettuna:

- miten sovellus on toiminut aiemmin

- minkä tyyppisiä virheitä kehittäjät yleensä tekevät, ja minkä tyyppisiä vikoja syntyy näiden virheiden seurauksena
- minkälaisia häiriöitä on esiintynyt muissa vastaavissa sovelluksissa.

Yleensä virheet, viat ja häiriöt voivat liittyä syötteisiin (esim. oikeaa syötettä ei hyväksytty, väärät tai puuttuvat parametrit), tulokset (esim. väärä muoto, väärä tulos), logiikkaan (esim. puuttuvat tapaukset, väärä operaattori), laskentaan (esim. väärä muuttuja, väärä laskutoimitus), rajapintoihin (esim. parametrien epäyhteensopivuus, yhteensopimattomat tyypit) tai tietoihin (esim. virheellinen alustus, väärä tyyppi).

Vikahyökkäykset ovat järjestelmällinen lähestymistapa virheenarvauksen toteuttamiseen. Tämä tekniikka edellyttää, että testaaja luo tai hankkii listan mahdollisista virheistä, vioista ja häiriöistä, ja suunnittelee testit, jotka tunnistavat virheisiin liittyvät viat, paljastavat nämä viat tai aiheuttavat häiriöitä. Nämä listat voidaan laatia kokemuksen, vika- ja häiriötietojen tai ohjelmistojen toimimattomuuden syihin liittyvän yleisen tietämyksen pohjalta.

Katso (Whittaker 2002, Whittaker 2003, Andrews 2006) lisätietoja virheenarvauksesta ja vikahyökkäyksistä.

4.4.2. Tutkiva testaus

Tutkivassa testauksessa testit suunnitellaan, suoritetaan ja arvioidaan samanaikaisesti, kun testaaja oppii testauksen kohteesta. Testauksen avulla opitaan lisää testauksen kohteesta, tutkitaan sitä syvällisemmin kohdennetuilla testeillä ja luodaan testejä testaamattomille alueille.

Tutkivaa testausta tehdään joskus käyttämällä istuntopohjaista testausta testauksen jäsentämiseksi. Istunto- tai sessiopohjaisessa lähestymistavassa tutkiva testaus suoritetaan määriteltyjen aikarajojen sisällä. Testaaja käyttää testauksen ohjaamiseen testausohjetta, joka sisältää testauksen tavoitteet. Testisessiota seuraa yleensä jälkipalaveri, jossa testaaja keskustelee testisession tuloksista kiinnostuneiden sidosryhmien kanssa. Tässä lähestymistavassa testauksen tavoitteita voidaan pitää korkean tason testattavina tilanteina. Kattavuuskohteet tunnistetaan ja niitä testataan testisession aikana. Testaaja voi käyttää testi-istuntolomakkeita suoritettujen toimenpiteiden ja tehtyjen havaintojen dokumentointiin.

Tutkiva testaus on hyödyllistä, kun määrittelyitä on vähän tai ne ovat riittämättömiä tai testaukseen kohdistuu merkittävä aikapaine. Tutkiva testaus on myös hyödyllistä täydentämään muita muodollisempia testaustekniikoita. Tutkiva testaus on tehokkaampaa, jos testaaja on kokenut, hänellä on toimialaosaamista sekä paljon olennaisia taitoja, kuten analyyttisiä taitoja, uteliaisuutta ja luovuutta (ks. kohta 1.5.1).

Tutkivassa testauksessa voidaan käyttää muita testitekniikoita (esim. ekvivalenssiositusta). Lisätietoja tutkivasta testauksesta löytyy (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3. Tarkistuslistoihin pohjautuva testaus

Tarkistuslistoihin pohjautuvassa testauksessa testaaja suunnittelee, toteuttaa ja suorittaa testejä tarkistuslistan sisältämien testattavien tilanteiden kattamiseksi. Tarkistuslistoja laadittaessa pohjana voidaan käyttää kokemusta, tietoa siitä, mikä on käyttäjälle tärkeää, tai ymmärrystä siitä, miksi ja miten ohjelmisto epäonnistuu. Tarkistuslistojen ei tulisi sisältää asioita, jotka voidaan tarkistaa automaattisesti, jotka soveltuvat paremmin aloitus-/lopetuskriteereiksi tai jotka ovat liian yleisiä (Brykczynski 1999).

Tarkistuslistan kohdat laaditaan usein kysymyksen muodossa. Jokainen kohta pitäisi voida tarkistaa erikseen ja suoraan. Kohdat voivat liittyä vaatimuksiin, graafisen käyttöliittymän ominaisuuksiin, laa-

tuominaisuuksiin tai muunlaisiin testattaviin tilanteisiin. Tarkistuslistoja voidaan luoda tukemaan erilaisia testaustyyppejä, mukaan luettuna toiminnallinen ja ei-toiminnallinen testaus (esim. käytettävyystestauksen 10 heuristiikkaa (Nielsen 1994)).

Jotkut tarkistuslistan kohdat voivat ajan myötä vähitellen muuttua tehottomammiksi, koska kehittäjät oppivat välttämään samojen virheiden tekemistä. Uusia kohtia on ehkä myös lisättävä, jotta voidaan ottaa huomioon äskettäin löydetyt vakavat viat. Siksi tarkistuslistoja tulisi päivittää säännöllisesti vika-analyysin perusteella. On kuitenkin varottava tekemästä tarkistuslistasta liian pitkää (Gawande 2009).

Jos yksityiskohtaisia testitapauksia ei ole, tarkistuslistoihin pohjautuva testaus voi tarjota ohjeita ja jonkinasteista johdonmukaisuutta testaukselle. Jos tarkistuslistat ovat korkean tason listoja, varsinaisessa testauksessa esiintyy todennäköisesti jonkin verran vaihtelua, mikä johtaa mahdollisesti suurempaan kattavuuteen mutta huonompaan toistettavuuteen.

4.5. Yhteistyöhön perustuvat testausmenetelmät

Jokaisella edellä mainitulla tekniikalla (ks. kohdat 4.2, 4.3 ja 4.4) on vikojen löytämiseen liittyvä erityistavoite. Yhteistyöhön perustuvat lähestymistavat puolestaan keskittyvät myös vikojen välttämiseen yhteistyön ja viestinnän avulla.

4.5.1. Käyttäjätarinoiden kirjoittaminen yhteistyönä

Käyttäjätarina edustaa ominaisuutta, joka on arvokas joko järjestelmän tai ohjelmiston käyttäjälle tai ostajalle. Käyttäjätarinoissa on kolme kriittistä näkökohtaa (Jeffries 2000), joita kutsutaan yhdessä "3 C:ksi”i:

- Kortti (Card)– käyttäjätarinan kuvaava ilmaisuväline (esim. hakemistokortti, merkintä sähköisessä taulussa)
- Keskustelu (Conversation) – selittää, miten ohjelmistoa käytetään (voi olla dokumentoitu tai suullinen)
- Vahvistus (Confirmation) – hyväksymiskriteerit (ks. kohta 4.5.2).

Yleisin muoto käyttäjätarinalle on "Kun olen [rooli], haluan [saavutettava tavoite], jotta voin [tuloksena oleva liiketoiminnallinen arvo roolin edustajalle]", ja sitä täydentävät hyväksymiskriteerit.

Kun käyttäjätarinoita kirjoitetaan yhteistyönä, voidaan käyttää erilaisia tekniikoita, kuten aivoriihiä ja miellekarttoja. Yhteistyön avulla tiimi saa yhteisen näkemyksen siitä, mitä pitäisi toimittaa, ottamalla huomioon kolme näkökulmaa: liiketoiminnan, toteutuksen ja testauksen.

Hyvän käyttäjätarinan tulisi olla riippumaton Independent), neuvoteltavissa oleva (Negotiable), arvokas (Valuable), arvioitavissa (Estimable), pieni (Small) ja testattava (Testable) (INVEST). Jos sidosryhmän edustaja ei osaa testata käyttäjätarinaa, tämä voi viitata siihen, että käyttäjätarina ei ole riittävän selkeä tai että se ei kuvaa jotain heille arvokasta asiaa tai että sidosryhmä tarvitsee vain apua testauksessa (Wake 2003).

4.5.2. Hyväksymiskriteerit

Käyttäjätarinan hyväksymiskriteerit ovat ehdot, jotka käyttäjätarinan toteutuksen on täytettävä, jotta sidosryhmät hyväksyvät sen. Tästä näkökulmasta hyväksymiskriteerejä voidaan pitää testattavina tilanteina, joita testien tulisi testata. Hyväksymiskriteerit ovat yleensä keskustelun tulos (katso kohta 4.5.1).

Hyväksymiskriteerejä käytetään

- käyttäjätarinan laajuuden määrittämiseen
- yhteisymmärryksen saavuttamiseen sidosryhmien kesken
- sekä positiivisten että negatiivisten skenaarioiden kuvaamiseen
- pohjana käyttäjätarinan hyväksymistestaukselle (ks. kohta 4.5.3)
- tarkan suunnittelun ja työmäärien arvioinnin mahdollistamiseen.

On olemassa useita tapoja kirjoittaa käyttäjätarinan hyväksymiskriteerit. Kaksi yleisintä muotoa ovat

- skenaariosuuntautunut (esim. BDD:ssä käytetty Kun/Jos/Sitten (Given/When/Then) -muoto, ks. kohta 2.1.3)
- sääntösuuntautunut (esim. luettelomainen tarkistuslista tai syöte-tulos-yhdistelmien taulukko).

Useimmat hyväksymiskriteerit voidaan dokumentoida käyttämällä jompaa kumpaa näistä kahdesta muodosta. Tiimi voi kuitenkin käyttää muuta, mukautettua muotoa, kunhan hyväksymiskriteerit ovat hyvin määritellyt ja yksiselitteiset.

4.5.3. Hyväksymistestiohjattu kehitys (Acceptance Test-Driven Development, ATDD)

ATDD on testit ensin -lähestymistapa (ks. kohta 2.1.3). Testitapaukset luodaan ennen käyttäjätarinan toteutusta. Testitapauksia luovat tiimin jäsenet, joilla on erilaisia näkökulmia, esim. asiakkaat, toteuttajat ja testaajat (Adzic 2009). Testitapaukset voidaan suorittaa manuaalisesti tai automatisoidusti.

Ensimmäinen vaihe on määrittelytyöpaja, jossa tiimin jäsenet analysoivat ja keskustelevat käyttäjätarinasta ja sen hyväksymiskriteereistä (jos niitä ei ole vielä määritelty) ja kirjoittavat ne. Käyttäjätarinan epätäydellisyys, tulkinnanvaraisuudet tai viat ratkaistaan tämän prosessin aikana. Seuraava vaihe on testitapausten laatiminen. Sen voi tehdä koko tiimi tai testaaja itsenäisesti. Testitapaukset perustuvat hyväksymiskriteereihin ja niitä voidaan pitää esimerkkeinä siitä, kuinka ohjelmisto toimii. Tämä auttaa tiimiä toteuttamaan käyttäjätarinan oikein.

Koska esimerkit ja testit tarkoittavat samaa, näitä termejä käytetään usein rinnakkain. Testien suunnittelun aikana voidaan käyttää kohdissa 4.2, 4.3 ja 4.4 kuvattuja testaustekniikoita.

Tyypillisesti ensimmäiset testitapaukset ovat positiivisia; niillä varmistetaan oikea käyttäytyminen ilman poikkeuksia tai virhetilanteita ja ne sisältävät suoritettavan toimintoketjun tilanteessa, jossa kaikki sujuu odotetusti. Kun positiiviset testitapaukset on suoritettu, tiimin pitäisi suorittaa negatiivista testausta. Lopuksi tiimin tulisi kattaa myös ei-toiminnalliset laatuominaisuudet (esim. suorituskyvyn tehokkuus, käytettävyys). Testitapaukset pitäisi kirjoittaa sidosryhmien kannalta ymmärrettävällä tavalla. Tyypillisesti testitapaukset sisältävät luonnollisella kielellä kirjoitettuja lauseita, joilla kuvataan tarvittavat esiehdot (jos sellaisia on), syötteet ja jälkiehdot.

Testitapausten on katettava kaikki käyttäjätarinan ominaisuudet, eivätkä ne saa mennä tarinan ulkopuolelle. Hyväksymiskriteerit voivat kuitenkin kuvata joitain käyttäjätarinassa kuvattuja ongelmia. Lisäksi kahden testitapauksen ei pitäisi testata käyttäjätarinan samaa ominaisuutta.

Kun testitapaukset kuvataan testiautomaatioympäristön tukemassa muodossa, toteuttajat voivat automatisoida ne kirjoittamalla tarvittavan koodin samalla, kun he toteuttavat käyttäjätarinan kuvaaman ominaisuuden. Hyväksymistesteistä tulee sitten suoritettavia vaatimuksia.

5. Testaustehtävien hallinta - 335 minuuttia

Avainsanat

aloituskriteerit, lopetuskriteerit, projektiriski, riski, riskianalyysi, riskiarviointi, riskien pienentäminen, riskien seuranta, riskien tunnistaminen, riskienhallinta, riskienvalvonta, riskipohjainen testaus, riskitaso, testauksen edistymisraportti, testauksen hallinta, testauksen loppuraportti, testauksen lähestymistapa, testauksen suunnittelu, testausneljännekset, testaussuunnitelma, testien seuranta, testipyramidi, tuoteriski, vikaraportti, vikojenhallinta

Luvun 5 oppimistavoitteet:

5.1 Testauksen suunnittelu

FL-5.1.1 (K2) Osaa kuvata testaussuunnitelman tarkoituksen ja sisällön

FL-5.1.2 (K1) Tunnistaa, miten testaaja tuo lisäarvoa iteraation ja julkaisun suunnitteluun

FL-5.1.3 (K2) Osaa verrata aloitus- ja lopetuskriteerejä ja erottaa ne toisistaan

FL-5.1.4 (K3) Osaa käyttää arviointitekniikoita testauksessa tarvittavan työmäärän laskemiseen

FL-5.1.5 (K3) Osaa priorisoida testitapaukset

FL-5.1.6 (K1) Muistaa testipyramidin käsitteet

FL-5.1.7 (K2) Osaa selittää testausneljännekset ja niiden suhteet testaustasoihin ja testaustyyppeihin

5.2 Riskienhallinta

FL-5.2.1 (K1) Osaa määrittää riskitason käyttämällä riskin todennäköisyyttä ja riskivaikutusta

FL-5.2.2 (K2) Erottaa projektiriskit ja tuoteriskit toisistaan

FL-5.2.3 (K2) Osaa selittää, miten tuoteriskianalyysi voi vaikuttaa testauksen perusteellisuuteen ja laajuuteen

FL-5.2.4 (K2) Osaa selittää, mihin toimenpiteisiin voidaan ryhtyä analysoitujen tuoteriskien perusteella

5.3 Testauksen seuranta, hallinta ja päättäminen

FL-5.3.1 (K1) Tunnistaa testauksessa käytettävät mittarit

FL-5.3.2 (K2) Osaa kuvata testiraporttien tarkoitukset, sisällöt ja kohderyhmät

FL-5.3.3 (K2) Osaa antaa esimerkkejä testauksen tilasta tiedottamisesta

5.4 Kokoonpanonhallinta

FL-5.4.1 (K2) Osaa kuvata, miten konfiguraation hallinta tukee testausta

5.5 Vikojenhallinta

FL-5.5.1 (K3) Osaa laatia vikaraportin

5.1. Testauksen suunnittelu

5.1.1. Testaussuunnitelman tarkoitus ja sisältö

Testaussuunnitelmassa kuvataan testausprojektin tavoitteet, resurssit ja prosessit. Testaussuunnitelma

- dokumentoi keinot ja aikataulun testitavoitteiden saavuttamiseksi
- auttaa varmistamaan, että suoritetut testaustoimenpiteet täyttävät määritetyt kriteerit
- toimii viestintävälineenä tiimin jäsenten ja muiden sidosryhmien kanssa
- osoittaa, että testaus noudattaa olemassa olevaa testauspolitiikkaa ja testausstrategiaa (tai selittää, miksi testaus poikkeaa niistä.

Testauksen suunnittelu ohjaa testaajien ajattelua ja pakottaa testaajat kohtaamaan tulevat haasteet, jotka liittyvät riskeihin, aikatauluihin, ihmisiin, työkaluihin, kustannuksiin, työmäärään jne. Testaussuunnitelman laatimisprosessi on hyödyllinen tapa käydä läpi testausprojektin tavoitteiden saavuttamiseksi tarvittavat tehtävät.

Testaussuunnitelman sisällössä kuvataan tyypillisesti

- testauksen konteksti (esim. laajuus, testauksen tavoitteet, rajoitteet, testauksen pohjamateriaali)
- testausprojektin oletukset ja rajoitteet
- sidosryhmät (esim. roolit, vastuut, merkitys testaukselle, resurssi- ja koulutustarpeet)
- viestintä (esim. viestinnän muodot ja tiheys, dokumenttien mallipohjat)
- riskirekisteri (esim. tuoteriskit, projektiriskit)
- testauksen lähestymistavat (esim. testaustasot, testaustyypit, testaustekniikat, testauksen tuotokset, aloitus- ja lopetuskriteerit, testauksen riippumattomuus, kerättävät metriikat, testiaineiston vaatimukset, testiympäristön vaatimukset, poikkeamat organisaation testauspolitiikasta ja testausstrategiasta)
- budjetti ja aikataulu.

Lisätietoja testaussuunnitelmasta ja sen sisällöstä löytyy ISO/IEC/IEEE 29119-3 -standardista.

5.1.2. Testaajan panos iteraation ja julkaisun suunnittelussa

Iteratiivisissa ohjelmistokehityksen elinkaarimalleissa tehdään tyypillisesti kahdenlaista suunnittelua: julkaisun suunnittelua ja iteraation suunnittelua.

Julkaisun suunnittelu ennakoi tuotteen julkaisua, määrittelee ja uudelleenmäärittelee tuotteen kehitysjonon, ja siihen voi sisältyä suurempien käyttäjätarinoiden jakaminen pienempien käyttäjätarinoiden joukoksi. Se on myös testauksen lähestymistavan ja testaussuunnitelman perusta läpi kaikkien iteraatioiden. Julkaisun suunnittelussa mukana olevat testaajat osallistuvat testattavien käyttäjätarinoiden ja hyväksymiskriteerien kirjoittamiseen (ks. kohta 4.5), projekti- ja laaturiskianalyyseihin (ks. kohta 5.2), arvioivat käyttäjätarinoihin liittyvän työmäärän (ks. kohta 5.1.4), määrittävät testauksen lähestymistavan ja suunnittelevat julkaisun testauksen.

Iteraation suunnittelu katsoo eteenpäin yksittäisen iteraation loppuun ja siinä käsitellään iteraation kehitysjonoa. Iteraation suunnittelussa mukana olevat testaajat osallistuvat käyttäjätarinoiden yksityis-

kohtaiseen riskianalyysiin, määrittävät käyttäjätarinoiden testattavuuden, pilkkovat käyttäjätarinat tehtäviksi (erityisesti testaustehtäviksi), arvioivat kaikkien testaustehtävien työmäärän sekä tunnistavat ja tarkentavat testauksen kohteen toiminnalliset ja ei-toiminnalliset ominaisuudet.

5.1.3. Aloitus- ja lopetuskriteerit

Aloituskriteerit määrittävät tietyn toimenpiteen aloittamisen edellytykset. Jos aloituskriteerit eivät täyty, on todennäköistä, että toimenpide osoittautuu vaikeammaksi, aikaa vievämmäksi, kalliimmaksi ja riskialttiimmaksi. Lopetuskriteerit määrittävät, mitä on saavutettava, jotta toimenpide voidaan todeta suoritetuksi. Aloitus- ja lopetuskriteerit pitäisi määrittää kullekin testaustasolle, ja ne vaihtelevat testauksen tavoitteiden mukaan.

Tyypillisiä aloituskriteerejä ovat resurssien saatavuus (esim. ihmiset, työkalut, ympäristöt, testiaineisto, budjetti, aika), testimateriaalin saatavuus (esim. testauksen pohjamateriaali, testattavat vaatimukset, käyttäjätarinat, testitapaukset) ja testauksen kohteen alkuperäinen laatutaso (esim. kaikki savutestit on läpäisty).

Tyypillisiä lopetuskriteerejä ovat testauksen perusteellisuuteen liittyvät mittaritiedot (esim. saavutettu kattavuustaso, ratkaisemattomien vikojen määrä, vikatiheys, hylättyjen testitapausten määrä) ja valmistumiskriteerit (esim. suunnitellut testit on suoritettu, staattinen testaus on suoritettu, kaikki löydetyt viat on raportoitu, kaikki regressiotestit on automatisoitu).

Ajan tai budjetin loppumista voidaan pitää myös kelvollisina lopetuskriteerinä. Vaikka muut lopetuskriteerit eivät täyty, testauksen lopettaminen tällaisissa olosuhteissa voi olla hyväksyttävää, jos sidosryhmät ovat katselmoineet ja hyväksyneet ilman lisätestausta tehtävään käyttöönottoon liittyvän riskin.

Ketterässä ohjelmistokehityksessä lopetuskriteerejä kutsutaan usein Valmiin määritelmäksi (DoD, Definition of Done), joka määrittelee tiimin objektiiviset mittarit julkaisukelpoiselle kohteelle. Aloituskriteerejä, jotka käyttäjätarinan on täytettävä kehitys- ja/tai testaustoimenpiteiden aloittamiseksi, kutsutaan Valmistellun määritelmäksi (DoR, Definition of Ready).

5.1.4. Työmäärän arviointitekniikat

Testauksen työmäärän arviointiin kuuluu testausprojektin tavoitteiden saavuttamiseksi tarvittavan testaukseen liittyvän työn arvioiminen. On tärkeää tehdä selväksi sidosryhmille, että arvio perustuu useisiin oletuksiin ja että siihen liittyy aina arviointivirheen mahdollisuus. Pienten tehtävien arviointi on yleensä tarkempaa kuin isojen. Siksi isoa tehtävää arvioitaessa se voidaan purkaa joukoksi pienempiä tehtäviä, joiden työmäärä voidaan sitten arvioida.

Tässä sertifikaattisisällössä kuvataan seuraavat neljä arviointitekniikkaa.

Suhdelukuihin perustuva arvio. Tässä metriikoihin perustuvassa tekniikassa organisaation aiemmista projekteista kerätään mittaritietoja, jotka mahdollistavat vastaavien projektien "vakiosuhteiden" johtamisen. Organisaation omien projektien suhdeluvut (esim. historiatietojen perusteella laskettuna) ovat yleensä paras lähde arviointiprosessissa. Näitä vakiosuhteita voidaan sitten käyttää uuden projektin testaustyömäärän arvioimiseen. Jos esimerkiksi edellisessä projektissa toteutus- ja testaustyön välisen työmäärän suhde oli 3:2 ja nykyisessä hankkeessa toteutustyömäärän odotetaan olevan 600 henkilötyöpäivää, voidaan testauksen työmäärän arvioida olevan 400 henkilötyöpäivää.

Ekstrapolointi. Tässä metriikoihin perustuvassa tekniikassa mittaaminen aloitetaan projektissa mahdollisimman varhaisessa vaiheessa. Kun havaintoja on tarpeeksi, projektissa jäljellä olevan työn vaatima työmäärä voidaan arvioida ekstrapoloimalla nämä tiedot (yleensä soveltamalla matemaattista mallia). Tämä menetelmä soveltuu hyvin iteratiivisiin ohjelmistokehityksen elinkaarimalleihin. Tiimi voi esimerkiksi ekstrapoloida tulevan iteraation testauksen työmäärän kolmen viimeisimmän iteraation keskimääräisenä työmääränä.

Wideband Delphi. Tässä iteratiivisessa, asiantuntijapohjaisessa tekniikassa asiantuntijat tekevät kokemukseen perustuvia työmääräarvioita. Jokainen asiantuntija arvioi itsekseen tehtävän työmäärän. Tulokset kerätään ja jos niissä on poikkeamia, jotka ylittävät sovitut rajat, asiantuntijat keskustelevat senhetkisistä arvioistaan. Jokaista asiantuntijaa pyydetään sitten tekemään jälleen itsenäisesti uusi arvio palautteen perusteella. Tätä prosessia toistetaan, kunnes saavutetaan yksimielisyys. Suunnittelupokeri (Planning Poker) on muunnelma Wideband Delphistä, jota käytetään yleisesti ketterässä ohjelmistokehityksessä. Suunnittelupokerissa arviot tehdään yleensä käyttämällä kortteja, joiden numerot edustavat työmäärän kokoa.

Kolmipistearvio. Tässä asiantuntijapohjaisessa tekniikassa asiantuntijat tekevät kolme työmääräarviota: optimistisimman arvion (a), todennäköisimmän arvion (m) ja pessimistisimmän arvion (b). Lopullinen työmääräarvio (E) on näiden painotettu aritmeettinen keskiarvo. Tämän tekniikan suosituimmassa versiossa arvio lasketaan seuraavasti: E = (a + 4*m + b) / 6. Tämän tekniikan etuna on, että sen avulla asiantuntijat voivat laskea mittausvirheen: SD = (b - a) / 6. Jos esimerkiksi arviot (henkilötyötunteina) ovat a = 6, m = 9 ja b = 18, lopullinen arvio on 10 ± 2 henkilötyötuntia (eli 8 - 12 henkilötyötuntia), koska E = (6 + 4*9 + 18) / 6 = 10 ja SD = (18 – 6) / 6 = 2.

Katso (Kan 2003, Koomen 2006, Westfall 2009) lisää näistä ja monista muista testiarviointitekniikoista.

5.1.5. Testitapausten priorisointi

Kun testitapaukset ja testiproseduurit on määritetty ja järjestetty testijoukoiksi, testijoukoille voidaan laatia testien suoritusaikataulu, joka määrittää järjestyksen, jossa ne suoritetaan. Testitapauksia priorisoitaessa voidaan ottaa huomioon eri tekijöitä. Yleisimmin käytetyt testitapausten priorisointistrategiat ovat seuraavat:

- Riskipohjainen priorisointi, jossa testien suoritusjärjestys perustuu riskianalyysin tuloksiin (ks. kohta 5.2.3). Testitapaukset, jotka kattavat tärkeimmät riskit, suoritetaan ensin.
- Kattavuuteen perustuva priorisointi, jossa testien suoritusjärjestys perustuu kattavuuteen (esim. lausekattavuus). Testitapaukset, jotka tuottavat suurimman kattavuuden, suoritetaan ensin. Toisessa vaihtoehdossa, jota kutsutaan lisäkattavuuden priorisoinniksi, suurimman kattavuuden saavuttava testitapaus suoritetaan ensin; seuraava testitapaus on se, joka tuottaa aina seuraavaksi suurimman lisäkattavuuden.
- Vaatimuksiin perustuva priorisointi, jossa testin suoritusjärjestys perustuu vaatimusten prioriteetteihin, jotka on jäljitetty vastaaviin testitapauksiin. Vaatimusten priorisoinnin tekevät sidosryhmät. Tärkeimpiin vaatimuksiin liittyvät testitapaukset suoritetaan ensin.

Ihannetilanteessa testitapaukset järjestettäisiin suoritettavaksi niiden prioriteettitasojen perusteella käyttämällä esimerkiksi yhtä edellä mainituista priorisointistrategioista. Tämä käytäntö ei kuitenkaan välttämättä toimi, jos testitapauksilla tai testattavilla ominaisuuksilla on riippuvuuksia. Jos korkeamman prioriteetin testitapaus on riippuvainen testitapauksesta, jolla on alempi prioriteetti, alemman prioriteetin testitapaus on suoritettava ensin.

Testien suoritusjärjestyksessä on otettava huomioon myös resurssien saatavuus. Esimerkiksi tarvittavat testaustyökalut, testiympäristöt tai henkilöt voivat olla käytettävissä vain tiettynä, rajattuna aikana.

5.1.6. Testipyramidi

Testipyramidi on malli, joka kuvaa, kuinka eri testit voivat olla yksityiskohtaisuudeltaan eri tasoisia. Testipyramidimalli tukee tiimiä testiautomaatiossa ja testauksen työmäärän kohdentamisessa osoittamalla, että testiautomaation eri tasot tukevat erilaisia tavoitteita. Pyramidikerrokset edustavat testiryhmiä. Mitä korkeampi kerros, sitä matalampi testin yksityiskohtaisuuden taso, testin itsenäisyys ja testin

suoritusaika. Alimman kerroksen testit ovat pieniä, itsenäisiä, nopeita ja testaavat toiminnallisuuden pienen osan, joten yleensä niitä tarvitaan paljon kohtuullisen kattavuuden saavuttamiseksi. Ylin kerros edustaa monimutkaisia, korkean tason päästä päähän -testejä. Nämä korkean tason testit ovat yleensä hitaampia kuin alempien kerrosten testit, ja ne testaavat tyypillisesti ison toiminnallisuuden, joten yleensä niitä tarvitaan vain muutama kohtuullisen kattavuuden saavuttamiseksi. Kerrosten lukumäärä ja nimeäminen voivat vaihdella. Esimerkiksi alkuperäinen testipyramidimalli (Cohn 2009) määrittelee kolme tasoa: "yksikkötestit", "palvelutestit" ja "käyttöliittymätestit". Toinen suosittu malli määrittelee yksikkö- (komponentti-) testit, integraatio- (komponentti-integraatio-) testit ja päästä päähän -testit. Myös muita testaustasoja (ks. kohta 2.2.1) voidaan käyttää.

5.1.7. Testausneljännekset

Brian Marickin (Marick 2003, Crispin 2008) määrittelemät testausneljännekset yhdistävät ketterässä ohjelmistokehityksessä testaustasot sopivien testaustyyppien, toimenpiteiden, testaustekniikoiden ja tuotosten kanssa. Malli tukee testauksen hallintaa näiden visualisoinnissa sen varmistamiseksi, että kaikki asianmukaiset testaustyypit ja testaustasot sisältyvät ohjelmistokehityksen elinkaareen, ja sen ymmärtämiseksi, että jotkin testaustyypit ovat oleellisempia tietyillä testaustasoilla kuin toiset. Tämä malli tarjoaa myös tavan erottaa ja kuvata eri tyyppiset testit kaikille sidosryhmille, mukaan lukien toteuttajat, testaajat ja liiketoiminnan edustajat.

Mallissa testit voivat olla liiketoimintasuuntautuneita tai teknologiasuuntautuneita. Testit voivat myös tukea tiimiä (eli ohjata toteutusta) tai arvioida tuotetta (eli mitata sen käyttäytymistä odotuksia vastaan). Näiden kahden näkökulman yhdistelmä määrittää neljä neljännestä:

- 1. neljännes, Q1 (teknologiasuuntautunut, tiimiä tukeva). Tämä neljännes sisältää komponentti- ja komponentti-integraatiotestit. Nämä testit pitäisi automatisoida ja sisällyttää CI-prosessiin.
- 2. neljännes, Q2 (liiketoimintasuuntautunut, tiimiä tukeva). Tämä neljännes sisältää toiminnallisia testejä, esimerkkejä, käyttäjätarinatestejä, prototyyppejä käyttäjäkokemuksen arviointia varten, API-testausta ja simulaatioita. Nämä testit testaavat hyväksymiskriteerejä vastaan ja voivat olla manuaalisia tai automatisoituja.
- 3. neljännes, Q3 (liiketoimintasuuntautunut, tuotetta arvioiva). Tämä neljännes sisältää tutkivan testauksen, käytettävyystestauksen, käyttäjän hyväksymistestauksen. Nämä testit ovat käyttäjäsuuntautuneita ja usein manuaalisia.
- 4. neljännes, Q4 (teknologiasuuntautunut, tuotetta arvioiva). Tämä neljännes sisältää savutestit ja ei-toiminnalliset testit (paitsi käytettävyystestit). Usein nämä testit automatisoidaan.

5.2. Riskienhallinta

Organisaatiot kohtaavat monia sisäisiä ja ulkoisia tekijöitä, jotka aiheuttavat epävarmuutta sen suhteen, saavuttaako organisaatio tavoitteensa ja jos, niin milloin (ISO 31000). Riskienhallinnan avulla organisaatiot voivat lisätä tavoitteiden saavuttamisen todennäköisyyttä, parantaa tuotteidensa laatua ja lisätä sidosryhmien luottamusta ja uskoa.

Tärkeimmät riskienhallintatoimet ovat:

- riskianalyysi (sisältää riskien tunnistamisen ja arvioinnin; ks. kohta 5.2.3)
- riskienhallinta (sisältää riskien pienentämisen ja seurannan; ks. kohta 5.2.4).

Testauksen lähestymistapaa, jossa testauksen toimenpiteet valitaan, priorisoidaan ja niitä hallitaan riskianalyysin ja riskienhallinnan perusteella, kutsutaan riskipohjaiseksi testaukseksi.

5.2.1. Riskin määritelmä ja ominaisuudet

Riski on mahdollinen tapahtuma, vaara, uhka tai tilanne, jonka toteutuminen aiheuttaa haitallisen vaikutuksen. Riskiä voidaan kuvata kahdella tekijällä:

- riskin todennäköisyys – riskin toteutumisen mahdollisuus (suurempi kuin nolla ja pienempi kuin yksi)
- riskin vaikutus (haitta) – kyseisen tapahtuman seuraukset.

Nämä kaksi tekijää kuvaavat riskitason, joka on riskin mittari. Mitä korkeampi riskitaso, sitä tärkeämpää on sen käsittely.

5.2.2. Projektiriskit ja tuoteriskit

Ohjelmistotestauksessa kiinnitetään yleensä huomiota kahdentyyppisiin riskeihin: projektiriskeihin ja tuoteriskeihin.

Projektiriskit liittyvät projektin hallintaan ja seurantaan. Projektiriskejä ovat

- organisatoriset ongelmat (esim. viivästykset tuotosten toimituksissa, epätarkat työnmääräarviot, kustannusten leikkaaminen)
- henkilöstöongelmat (esim. riittämättömät taidot, ristiriidat, viestintäongelmat, henkilöstöpula)
- tekniset ongelmat (esim. toteutuksen laajuuden muuttuminen hallitsemattomasti, heikko työkalutuki)
- toimittajaan liittyvät ongelmat (esim. kolmannen osapuolen toimituksen epäonnistuminen, tukiyrityksen konkurssi).

Projektiriskit voivat toteutuessaan vaikuttaa projektin aikatauluun, budjettiin tai laajuuteen, mikä vaikuttaa projektin kykyyn saavuttaa sen tavoitteet.

Tuoteriskit liittyvät tuotteen laatuominaisuuksiin (esim. kuvattu ISO 25010 -laatumallissa). Esimerkkejä tuoteriskeistä ovat puuttuva tai väärä toiminnallisuus, virheelliset laskutoimitukset, ajonaikaiset virheet, huono arkkitehtuuri, tehottomat algoritmit, riittämätön vasteaika, huono käyttökokemus, tietoturvahaavoittuvuudet. Toteutuessaan tuoteriskit voivat johtaa erilaisiin negatiivisiin seurauksiin, mukaan lukien

- käyttäjien tyytymättömyys
- tuottojen, luottamuksen, maineen menetys
- kolmansille osapuolille aiheutuvat vahingot
- korkeat ylläpitokustannukset, helpdeskin ylikuormitus
- rikosoikeudelliset seuraamukset
- äärimmäisissä tapauksissa fyysiset vahingot, vammat tai jopa kuolema.

5.2.3. Tuoteriskianalyysi

Testauksen näkökulmasta tuoteriskianalyysin tavoitteena on lisätä tietoisuutta tuoteriskeistä, jotta testaustyö voidaan kohdentaa tavalla, joka minimoi tuoteriskien jäljelle jäävän riskitason. Ihannetilanteessa tuoteriskianalyysi alkaa ohjelmistokehityksen elinkaaren aikaisessa vaiheessa.

Tuoteriskianalyysi koostuu riskien tunnistamisesta ja riskien arvioinnista. Riskien tunnistamisessa on kyse kattavan riskiluettelon laatimisesta. Sidosryhmät voivat tunnistaa riskejä käyttämällä erilaisia tekniikoita ja työkaluja, kuten aivoriihiä, työpajoja, haastatteluja tai syy-seurauskaavioita. Riskien arviointiin kuuluu tunnistettujen riskien luokittelu, riskien todennäköisyyden, vaikutuksen ja riskitason määrittäminen, priorisointi ja tapojen ehdottaminen riskien käsittelemiseksi. Luokittelu auttaa pienentämistoimenpiteiden kohdentamisessa, koska yleensä samaan luokkaan kuuluvia riskejä voidaan pienentää samalla tavalla.

Riskien arvioinnissa voidaan käyttää määrällistä tai laadullista lähestymistapaa tai niiden yhdistelmää. Määrällisessä lähestymistavassa riskitaso lasketaan kertomalla riskin todennäköisyys ja vaikutus keskenään. Laadullisessa lähestymistavassa riskitaso voidaan määrittää käyttämällä riskimatriisia.

Tuoteriskianalyysi voi vaikuttaa testauksen perusteellisuuteen ja laajuuteen. Sen tuloksia käytetään

- suoritettavan testauksen laajuuden määrittämisessä
- erityisten testaustasojen määrittämisessä ja ehdottamaan käytettäviä testaustyyppejä
- käytettävien testaustekniikoiden ja saavutettavan kattavuuden määrittämisessä
- kunkin tehtävän vaatiman työmäärän arvioinnissa
- testauksen priorisoinnissa, jotta kriittiset viat löydetään mahdollisimman aikaisin
- selvitettäessä, voitaisiinko testauksen lisäksi käyttää muita toimenpiteitä riskien vähentämiseksi.

5.2.4. Tuoteriskien hallinta

Tuoteriskien hallinta käsittää kaikki toimenpiteet, joihin ryhdytään tunnistettujen ja arvioitujen tuoteriskien perusteella. Tuoteriskien hallinta koostuu riskien pienentämisestä ja riskien seurannasta. Riskien pienentämiseen kuuluu riskien arvioinnissa ehdotettujen toimenpiteiden toteuttaminen riskitason alentamiseksi. Riskien seurannan tavoitteena on varmistaa, että pienentämistoimenpiteet ovat tehokkaita, hankkia lisätietoja riskien arvioinnin parantamiseksi ja tunnistaa uusia esiin nousevia riskejä.

Tuoteriskien hallinnan kannalta, kun riski on analysoitu, siihen voidaan reagoida monilla tavoin esimerkiksi pienentämällä riskejä testaamalla, hyväksymällä riski, siirtämällä riski toiselle osapuolelle tai laatimalla valmiussuunnitelma (Veenendaal 2012). Tuoteriskien pienentämiseksi testaamalla voidaan tehdä seuraavat toimenpiteet:

- Valitse testaajat, joilla on kyseiseen riskityyppiin nähden oikea määrä kokemusta ja taitoja.
- Käytä sopivaa testauksen riippumattomuuden tasoa.
- Tee katselmointeja ja käytä staattista analyysiä.
- Käytä sopivia testaustekniikoita ja kattavuustasoja.
- Käytä tilanteeseen liittyvien laatuominaisuuksien perusteella sopivia testaustyyppejä.
- Tee dynaamista testausta, regressiotestaus mukaan luettuna.

5.3. Testauksen seuranta, hallinta ja päättäminen

Testien seurannassa on kyse testausta koskevan tiedon keräämisestä. Näitä tietoja käytetään testauksen edistymisen arviointiin ja sen mittaamiseen, täyttyvätkö testauksen lopetuskriteerit tai onko niihin liittyvät testaustehtävät, kuten tuoteriskien, vaatimusten tai hyväksymiskriteerien kattavuustavoitteiden saavuttaminen, tehty hyväksytysti.

Testauksen hallinta käyttää testauksen seurannasta saatuja tietoja tuottaakseen toimintaohjeiden muodossa opastusta ja tarvittavia korjaavia toimenpiteitä tehokkaimman ja tuottavimman testauksen aikaansaamiseksi. Toimintaohjeita ovat muun muassa seuraavat:

- testien uudelleenpriorisointi, kun tunnistetusta riskistä tulee ongelma
- aloitus- tai lopetuskriteerien täyttymisen uudelleenarviointi testauksen kohteen muutosten vuoksi
- testausaikataulun muuttaminen testiympäristön toimituksen viivästymisen huomioon ottamiseksi
- uusien resurssien lisääminen tarpeen mukaan.

Testauksen päättämisessä kerätään tietoa suoritetuista testaustoimenpiteistä kokemuksen, testimateriaalin ja muiden asiaankuuluvien tietojen kokoamiseksi. Testauksen päättämistoimenpiteet tapahtuvat projektin määräajankohdissa, kuten esimerkiksi kun testaustaso on valmis, ketterä iteraatio päättyy, testausprojekti valmistuu (tai peruutetaan), ohjelmistojärjestelmä julkaistaan tai ylläpitoversio valmistuu.

5.3.1. Testauksessa käytettävät mittarit

Testauksen mittaritietoja kerätään osoittamaan edistymistä suunniteltuun aikatauluun ja budjettiin nähden, testauksen kohteen nykyistä laatua ja testaustoimenpiteiden tehokkuutta suhteessa tavoitteisiin tai iteraation päämäärään. Testauksen seuranta kerää erilaisia mittaritietoja testauksen hallinnan ja päättämisen tueksi.

Yleisiä testauksen mittareita ovat:

- projektin edistymisen mittarit (esim. tehtävien valmistuminen, resurssien käyttö, testauksen työmäärä)
- testauksen edistymisen mittarit (esim. testitapausten valmistelun edistyminen, testiympäristön valmistelun edistyminen, suoritettujen/ei suoritettujen ja hyväksyttyjen/hylättyjen testitapausten määrä, testien suoritusaika)
- tuotteen laatumittarit (esim. saatavuus, vasteaika, keskimääräinen häiriöaika)
- vikamittarit (esim. löydettyjen/korjattujen vikojen lukumäärä ja prioriteetit, vikatiheys, vikojen löytöprosentti)
- riskimittarit (esim. jäännösriskitaso)
- kattavuusmittarit (esim. vaatimuskattavuus, koodikattavuus)
- kustannusmittarit (esim. testauksen kustannukset, organisaation laatukustannukset).

5.3.2. Testauksen raporttien tarkoitus, sisältö ja yleisö

Testauksen raportointi tiivistää ja välittää testaukseen liittyvää tietoa testauksen aikana ja sen jälkeen. Testauksen edistymisraportit tukevat testauksen jatkuvaa hallintaa ja niiden on tarjottava riittävästi tietoa, jotta testauksen aikatauluun, resursseihin tai testaussuunnitelmaan voidaan tehdä muutoksia, kun niitä tarvitaan suunnitelmasta poikkeamisen tai muuttuneiden olosuhteiden vuoksi. Testauksen loppuraporteissa tehdään yhteenveto testauksen tietystä vaiheesta (esim. testaustaso, testauskierros, iteraatio) ja niissä voidaan antaa tietoja myöhempää testausta varten.

Testauksen seurannan ja hallinnan aikana testaustiimi luo testauksen edistymisraportteja sidosryhmille pitääkseen heidät ajan tasalla. Testauksen edistymisraportteja luodaan yleensä säännöllisesti (esim. päivittäin, viikoittain jne.), ja ne sisältävät tietoa

- testijaksosta (ajankohta)
- testauksen edistymisestä (esim. edellä tai jäljessä aikataulusta), mukaan lukien merkittävät poikkeamat
- testauksen esteistä ja niiden tilapäisistä ratkaisuista
- testauksen etenemisestä mittareilla kuvattuna (katso esimerkkejä kohdasta 5.3.1)
- uusista ja muuttuneista riskeistä testijakson aikana
- seuraavalle jaksolle suunnitellusta testauksesta.

Testauksen loppuraportti laaditaan testauksen päättämisen aikana, kun projekti, testaustaso tai testaustyyppi valmistuu ja kun, ihannetilanteessa, sen lopetuskriteerit ovat täyttyneet. Raportissa hyödynnetään testauksen edistymisraportteja ja muita tietoja. Tyypillinen testauksen loppuraportti sisältää

- yhteenvedon testauksesta
- testauksen ja tuotteen laadun arvioinnin alkuperäisen testaussuunnitelman perusteella (ts. testauksen tavoitteet ja lopetuskriteerit)
- poikkeukset testaussuunnitelmasta (esim. poikkeamat suunnitellusta aikataulusta, kestosta ja työmäärästä)
- testauksen esteet ja niiden tilapäiset ratkaisut
- mittaritietoja testauksen edistymisraporttien pohjalta
- jäljellä olevat riskit, korjaamattomat viat
- testauksen kannalta oleelliset oppimiskokemukset.

Eri kohderyhmät tarvitsevat raporteissa erilaisia tietoja ja vaikuttavat raportoinnin muodollisuuden asteeseen ja tiheyteen. Testauksen edistymisestä raportointi muille saman tiimin jäsenille on usein säännöllistä ja epämuodollista, kun taas raportointi valmistuneen projektin testauksesta noudattaa tiettyä mallipohjaa ja tapahtuu vain kerran.

ISO/IEC/IEEE 29119-3 -standardi sisältää mallipohjia ja esimerkkejä testauksen edistymisraporteista (kutsutaan testauksen tilanneraporteiksi) ja testauksen loppuraporteista.

5.3.3. Testauksen tilanteesta tiedottaminen

Paras tapa tiedottaa testauksen tilanteesta vaihtelee riippuen testauksen hallintaan liittyvistä huolenaiheista, organisaation testausstrategioista, sääntelystandardeista tai itseorganisoituvien tiimien kohdalla (ks. kohta 1.5.2) itse tiimistä. Vaihtoehtoihin kuuluvat

- suullinen viestintä tiimin jäsenten ja muiden sidosryhmien kanssa
- koontinäytöt (esim. CI/CD-koontinäytöt, tehtävätaulut ja edistymiskaaviot)
- sähköiset viestintäkanavat (esim. sähköposti, chat)
- verkkodokumentaatio
- muodolliset testausraportit (ks. 5.3.2 kohta).

Näistä vaihtoehdoista voidaan käyttää yhtä tai useampaa. Muodollisempi viestintä voi olla tarkoituksenmukaisempaa hajautetuissa tiimeissä, joissa suora kasvokkain tapahtuva viestintä ei aina ole mahdollista maantieteellisen etäisyyden tai aikaerojen vuoksi. Eri sidosryhmät ovat tyypillisesti kiinnostuneita erityyppisistä tiedoista, joten viestintä pitäisi räätälöidä sen mukaisesti.

5.4. Kokoonpanonhallinta

Testauksessa kokoonpanonhallinta (Configuration Management, CM) tarjoaa menetelmän esimerkiksi testaussuunnitelmien, testausstrategioiden, testattavien tilanteiden, testitapausten, testiskriptien, testitulosten, testilokien ja testiraporttien eli testauksen tuotosten tunnistamiseen, hallintaan ja seurantaan kokoonpanon osina.

Monimutkaiseen kokoonpanon osaan (esim. testiympäristö) liittyen kokoonpanonhallinta kirjaa osat, joista se koostuu, niiden väliset suhteet ja versiot. Jos kokoonpanon osa hyväksytään testattavaksi, siitä tulee pohjakokoonpano ja sitä voidaan muuttaa vain muodollisen muutostenhallintaprosessin kautta.

Kokoonpanonhallinta pitää kirjaa muuttuneista kokoonpanon osista, kun uusi pohjakokoonpano luodaan. Aiempaan pohjakokoonpanoon on mahdollista palata aiempien testitulosten toistamiseksi.

Testauksen tukemiseksi kokoonpanonhallinta varmistaa seuraavat asiat:

- Kaikki kokoonpanon osat, mukaan lukien testattavat kohteet (testauksen kohteen yksittäiset osat), nimetään yksilöivästi, versioidaan, niitä seurataan muutosten varalta ja niillä on yhteys muihin kokoonpanon osiin, jotta jäljitettävyys voidaan säilyttää koko testausprosessin ajan.
- Kaikkiin tunnistettuihin dokumentteihin ja ohjelmiston osiin viitataan yksiselitteisesti testausdokumentaatiossa.

Jatkuva integraatio, jatkuva toimitus, jatkuva julkaisu ja niihin liittyvä testaus toteutetaan tyypillisesti osana automatisoitua DevOps-putkea (ks. kohta 2.1.4), johon automatisoitu kokoonpanonhallinta yleensä sisältyy.

5.5. Vikojenhallinta

Koska yksi tärkeimmistä testauksen tavoitteista on vikojen löytäminen, vakiinnutettu vikojenhallintaprosessi on välttämätön. Vaikka käytämme tässä termiä "vika", raportoidut poikkeamat voivat osoittautua oikeiksi vioiksi tai joksikin muuksi (esim. väärä positiivinen, muutospyyntö) - tämä selvitetään vikaraporttien käsittelyn aikana. Poikkeamia voidaan raportoida missä tahansa ohjelmistokehityksen elinkaaren vaiheessa, ja raportointitapa riippuu elinkaarimallista. Vikojenhallintaprosessi sisältää vähintään työnkulun yksittäisten poikkeamien käsittelemiseksi niiden havaitsemisesta niiden sulkemiseen sekä säännöt niiden luokittelua varten. Työnkulku sisältää tyypillisesti toimenpiteet ilmoitettujen poikkeamien kirjaamiseksi, analysoimiseksi ja luokittelemiseksi, sopivasta vasteesta päättämiseksi, kuten vian korjaaminen tai jättäminen sikseen, sekä lopulta vikaraportin sulkemiseksi. Kaikkien asianomaisten sidosryhmien on noudatettava prosessia. Staattisen testauksen (erityisesti staattisen analyysin) viat on suositeltavaa käsitellä samalla tavalla.

Tyypillisillä vikaraporteilla on seuraavat tavoitteet:

- antaa raportoitujen vikojen käsittelystä ja ratkaisemisesta vastaaville riittävästi tietoa ongelman ratkaisemiseksi
- tarjota keino tuotoksen laadun seuraamiseksi

- tarjota ideoita kehitys- ja testausprosessin parantamiseksi.

Dynaamisen testauksen aikana kirjattava vikaraportti sisältää yleensä seuraavat tiedot:

- yksilöivä tunniste
- otsikko, jossa on lyhyt yhteenveto raportoidusta poikkeamasta
- päivämäärä, jolloin poikkeama havaittiin, raportoiva organisaatio ja raportin laatija, mukaan lukien hänen roolinsa
- testauksen kohteen ja testiympäristön tunnistetiedot
- vian asiayhteys (esim. suoritettu testitapaus, suoritettu testaustoimenpide, ohjelmistokehityksen elinkaaren vaihe ja muut oleelliset tiedot, kuten käytetty testaustekniikka, tarkistuslista tai testiaineisto)
- häiriön kuvaus sen toistamiseksi ja selvittämiseksi, mukaan luettuna askeleet, joilla poikkeama havaittiin, sekä kaikki oleelliset testilokit, tietokantavedokset, näytönkuvat tai tallenteet
- odotetut tulokset ja todelliset tulokset
- vian vakavuus (vaikutuksen aste) suhteessa sidosryhmien etuihin tai vaatimuksiin
- korjauksen kiireellisyys
- vian tila (esim. avoinna, lykätty, kaksoiskappale, odottaa korjausta, odottaa varmistustestausta, avattu uudelleen, suljettu, hylätty)
- viittaukset (esim. testitapaukseen).

Osa näistä tiedoista voi kirjautua raporttiin automaattisesti vikojenhallintatyökaluja käytettäessä (esim. tunniste, päivämäärä, tekijä ja alkutila). Vikaraporttien mallipohjia ja esimerkkejä vikaraporteista löytyy ISO/IEC/IEEE 29119-3 -standardista, jossa kutsutaan vikaraporttia havaintoraportiksi.

6. Testaustyökalut - 20 minuuttia

Avainsanat

Testiautomaatio

Luvun 6 oppimistavoitteet:

6.1 Testauksen työkalutuki

FL-6.1.1 (K2) Selittää, miten erityyppiset testaustyökalut tukevat testausta

6.2 Testiautomaation hyödyt ja riskit

FL-6.2.1 (K1) Muistaa testiautomaation hyödyt ja riskit

6.1. Testauksen työkalutuki

Testaustyökalut tukevat ja helpottavat monia testauksen toimenpiteitä. Esimerkkejä ovat muiden muassa seuraavat:

- hallintatyökalut - lisäävät testausprosessin tehokkuutta helpottamalla ohjelmistokehityksen elinkaaren, vaatimusten, testien, vikojen ja kokoonpanon hallintaa
- staattisen testauksen työkalut – tukevat testaajaa katselmointien ja staattisten analyysien tekemisessä
- testien suunnittelu- ja valmistelutyökalut – helpottavat testitapausten, testiaineiston ja testiproseduurien laatimista
- testien suoritus- ja kattavuustyökalut – helpottavat testien automaattista suorittamista ja kattavuuden mittaamista
- ei-toiminnallisen testauksen työkalut – mahdollistavat sellaisen ei-toiminnallisen testauksen, jota testaajan on vaikea tai mahdoton suorittaa manuaalisesti
- DevOps-työkalut - tukevat DevOps-toimitusputkea, työnkulun seurantaa, automatisoituja koontiprosesseja, CI / CD:tä
- yhteistyötyökalut – helpottavat viestintää
- skaalautuvuutta ja käyttöönoton standardointia tukevat työkalut (esim. virtuaalikoneet, konttityökalut)
- mitkä tahansa muut työkalut, jotka auttavat testauksessa (esim. laskentataulukko on testaustyökalu testauksen yhteydessä).

6.2. Testiautomaation hyödyt ja riskit

Pelkkä työkalun hankkiminen ei takaa menestystä. Jokainen uusi työkalu vaatii panostusta todellisten ja kestävien hyötyjen saavuttamiseksi (esim. työkalun käyttöönotto, ylläpito ja koulutus). On myös joitakin riskejä, joita on analysoitava ja lievennettävä.

Testiautomaation käytön mahdollisia hyötyjä ovat seuraavat:

- ajan säästäminen vähentämällä toistuvaa manuaalista työtä (esim. regressiotestien suorittaminen, saman testiaineiston syöttäminen uudelleen, odotettujen tulosten vertaaminen todellisiin tuloksiin ja tarkastus koodausstandardeja vastaan)
- yksinkertaisten inhimillisten virheiden estäminen paremman yhdenmukaisuuden ja toistettavuuden myötä (esim. testit johdetaan yhdenmukaisesti vaatimuksista, testiaineistot luodaan järjestelmällisesti ja testit suoritetaan työkalulla samassa järjestyksessä samoin väliajoin)
- objektiivisempi arviointi (esim. kattavuus) ja sellaisten mittaritietojen tuottaminen, jotka ovat liian monimutkaisia ihmisten tuotettavaksi
- helpompi pääsy testausta koskeviin tietoihin testauksenhallinnan ja raportoinnin tueksi (esim. tilastot, kaaviot ja kootut tiedot testauksen edistymisestä, vikojen määrästä ja testin suorittamisen kestosta)
- lyhyemmät testien suoritusajat mahdollistavat vikojen varhaisemman havaitsemisen, nopeamman palautteen ja nopeamman markkinoillesaantiajan

- enemmän aikaa testaajille uusien, syvällisempien ja tehokkaampien testien suunnitteluun.

Testiautomaation käytön mahdollisia riskejä ovat seuraavat:

- epärealistiset odotukset työkalun tuottamien etujen suhteen (mukaan lukien toiminnallisuus ja helppokäyttöisyys)
- epätarkat arviot työkalun käyttöönottoon, testiskriptien ylläpitoon ja olemassa olevan manuaalisen testausprosessin muuttamiseen tarvittavasta ajasta, kustannuksista ja työmäärästä
- testaustyökalun käyttäminen, kun manuaalinen testaus on tarkoituksenmukaisempaa
- liika luottamus työkaluun, esimerkiksi ihmisen kriittisen ajattelun tarpeen huomiotta jättäminen
- riippuvuus työkalun toimittajasta, joka voi lopettaa toimintansa, poistaa työkalun käytöstä, myydä sen toiselle toimittajalle tai tarjota huonoa tukea (esim. vastaukset kyselyihin, päivitykset ja vikojen korjaukset)
- käytetään avoimen lähdekoodin ohjelmistoa, joka voidaan hylätä, mikä tarkoittaa, että uusia päivityksiä ei ole enää saatavilla, tai jonka sisäiset komponentit voivat vaatia melko usein jatkokehityksenä tehtäviä päivityksiä
- automaatiotyökalu ei ole yhteensopiva kehitysalustan kanssa
- valitaan epäsopiva työkalu, joka ei täytä lakisääteisiä vaatimuksia ja/tai turvallisuusstandardeja.

7. Viittaukset

Standardit

ISO/IEC/IEEE 29119-1 (2022) Software and systems engineering – Software testing – Part 1: General Concepts

ISO/IEC/IEEE 29119-2 (2021) Software and systems engineering – Software testing – Part 2: Test processes

ISO/IEC/IEEE 29119-3 (2021) Software and systems engineering – Software testing – Part 3: Test documentation

ISO/IEC/IEEE 29119-4 (2021) 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

ISO/IEC/IEEE 14764:2022 – Software engineering – Software life cycle processes – Maintenance

ISO 31000 (2018) Risk management – Principles and guidelines

Kirjat

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

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: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

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

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

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

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Artikkelit ja verkkosivut

Brykczynski, B. (1999) "A survey of software inspection checklists," *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) "The logic of computer programming," *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/old-blog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158*

Salman. I. (2016) “Cognitive biases in software quality and testing,” *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) “INVEST in Good Stories, and SMART Tasks,” https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

8. Liite A – Oppimistavoitteet/kognitiivinen tietotaso

Seuraavat oppimistavoitteet on määritelty soveltuviksi tähän sertifikaattisisältöön. Jokainen sertifikaattisisällön aihe tentitään sen oppimistavoitteen mukaisesti. Oppimistavoitteet alkavat toimintaverbillä, joka vastaa sen kognitiivista tietotasoa alla luetellulla tavalla.

Taso 1: Muistaa (K1) – kokelas muistaa, tunnistaa ja pystyy palauttamaan mieleen termin tai käsitteen.

Toimintaverbit: identifioi, palauttaa mieleen, muistaa, tunnistaa.

Esimerkkejä:

- "Tunnistaa tyypilliset testauksen tavoitteet."
- "Pystyy palauttamaan mieleen testipyramidin käsitteet. "
- "Tunnistaa, miten testaaja tuo lisäarvoa iteraation ja julkaisun suunnitteluun."

Taso 2: Ymmärtää (K2) – kokelas osaa valita syyt tai selitykset aiheeseen liittyville väitteille ja tehdä yhteenvedon, vertailla, luokitella ja antaa esimerkkejä testauksen käsitteestä.

Toimintaverbit: luokitella, vertailla, vastakkainasetella, erottaa, havainnollistaa, selittää, antaa esimerkkejä, tulkita, vetää yhteen.

Esimerkkejä:

- "Luokittelee eri vaihtoehdot hyväksymiskriteerien kirjoittamiseen."
- "Vertaa testauksen eri rooleja" (etsii yhtäläisyyksiä, eroja tai molempia).
- "Erottaa projektiriskit ja tuoteriskit" (mahdollistaa käsitteiden erottamisen toisistaan).
- "Antaa esimerkkejä testaussuunnitelman tarkoituksesta ja sisällöstä."
- "Selittää tilanteen vaikutuksen testausprosessiin."
- "Tekee yhteenvedon katselmointiprosessin toimenpiteistä."

Taso 3: Käyttää (K3) – kokelas osaa suorittaa toimenpiteen kohdatessaan tutun tehtävän tai valita oikean menettelyn ja käyttää sitä tietyssä tilanteessa.

Toimintaverbit: hyödyntää, toteuttaa, valmistella, käyttää.

Esimerkkejä:

- "Käyttää testitapausten priorisointia" (pitäisi viitata menettelyyn, tekniikkaan, prosessiin, algoritmiin jne.).
- "Laatii vikaraportin."
- "Käyttää raja-arvoanalyysiä testitapausten johtamiseen."

Viitteet oppimistavoitteiden kognitiivisille tasoille:

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching

Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

9. Liite B – Liiketoiminnan tulosten jäljitettävyysmatriisi ja oppimistavoitteet

Tässä osassa esitellään liiketoiminnan tuloksiin liittyvien Perustason oppimistavoitteiden määrä sekä jäljitettävyys Perustason liiketoimintatulosten ja Perustason oppimistavoitteiden välillä.

Liiketoiminnan tulokset: Perustaso		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Ymmärtää, mitä testaus on ja miksi siitä on hyötyä	6													
BO2	Ymmärtää ohjelmistotestauksen peruskäsitteet		22												
BO3	Tunnistaa tilanteen mukaan käytettävät testauksen lähestymistavat toimenpiteet			6											
BO4	Arvioi ja parantaa dokumentaation laatua				9										
BO5	Lisää testauksen tehokkuutta ja tuottavuutta					20									
BO6	Sopeuttaa testausprosessin ohjelmistokehityksen elinkaareen						6								
BO7	Ymmärtää testauksenhallinnan periaatteet							6							
BO8	Kirjoittaa ja kommunikoi selkeitä ja ymmärrettäviä vikaraportteja								1						
BO9	Ymmärtää tekijät, jotka vaikuttavat testaukseen liittyviin prioriteet-teihin ja ponnisteluihin									7					
BO10	Työskentelee osana monitahoista tiimiä										8				
BO11	Tiedostaa testiautomaatioon liittyvät riskit ja hyödyt.											1			
BO12	Tunnistaa testauksessa tarvittavat olennaiset taidot												5		
BO13	Ymmärtää riskien vaikutuksen testaukseen													4	
BO14	Raportoi tehokkaasti testauksen edistymisestä ja laadusta														4

Luku/kohta	Oppimistavoite	K-taso	LIIKETOIMINNAN TULOKSET														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
1 luku	**Testauksen perusteet**															
1.1	**Mitä testaus on?**															
1.1.1	Tunnistaa tyypilliset testauksen tavoitteet	K1	X													
1.1.2	Erottaa testauksen vikojenjäljityksestä	K2		X												
1.2	**Miksi testaus on välttämätöntä?**															
1.2.1	Osaa kertoa esimerkein, miksi testaus on tarpeellista	K2	X													
1.2.2	Muistaa testauksen ja laadunvarmistuksen välisen suhteen	K1		X												
1.2.3	Erottaa juurisyyn, virheen, vian ja häiriön toisistaan	K2		X												
1.3	**Testauksen periaatteet**															
1.3.1	Selittää seitsemän testausperiaatetta	K2		X												
1.4	**Testaustoimenpiteet, testimateriaali ja testaukseen liittyvät roolit**															
1.4.1	Osaa kuvata eri testaustoimenpiteet ja -tehtävät	K2			X											
1.4.2	Osaa selittää, miten tilanne vaikuttaa testausprosessiin	K2			X			X								
1.4.3	Erottaa testaustoimenpiteitä tukevan testimateriaalin	K2			X											
1.4.4	Osaa selittää jäljitettävyyden ylläpitämisen merkityksen	K2				X	X									
1.4.5	Pystyy vertaamaan testauksen eri rooleja	K2										X				

1.5	**Keskeiset taidot ja hyvät käytännöt testauksessa**															
1.5.1	Osaa antaa esimerkkejä testauksessa tarvittavista yleisistä taidoista	K2												X		
1.5.2	Muistaa tiimiperustaisen lähestymistavan edut	K1										X				
1.5.3	Erottaa testauksen riippumattomuuden edut ja haitat	K2			X											
2 luku	**Testaus ohjelmistokehityksen elinkaaren aikana**															
2.1	**Testaus ohjelmistokehityksen elinkaaressa**															
2.1.1	Osaa selittää valitun ohjelmistokehityksen elinkaaren vaikutuksen testaukseen	K2						X								
2.1.2	Muistaa hyvät testauskäytännöt, jotka koskevat kaikkia ohjelmistokehityksen elinkaaria	K1						X								
2.1.3	Muistaa esimerkit testit ensin -lähestymistavoista kehityksessä	K1					X									
2.1.4	Osaa kuvata, miten DevOps voi vaikuttaa testaukseen	K2					X	X			X	X				
2.1.5	Osaa selittää shift-left-lähestymistavan	K2					X	X								
2.1.6	Osaa selittää, miten jälkipalavereita voidaan käyttää prosessin parantamisen mekanismina	K2					X					X				
2.2	**Testaustasot ja testityypit**															
2.2.1	Erottaa eri testaustasot toisistaan	K2		X	X											
2.2.2	Erottaa eri testaustyypit toisistaan	K2		X												
2.2.3	Erottaa varmistustestauksen regressiotestauksesta	K2		X												
2.3	**Ylläpitotestaus**															
2.3.1	Osaa kuvata ylläpitotestauksen ja sen käynnistävät tekijät	K2		X					X							
3 luku	**Staattinen testaus**															

3.1	**Staattisen testauksen perusteet**															
3.1.1	Tunnistaa tuotokset, joita voidaan tutkia erilaisilla staattisilla testaustekniikoilla	K1				X	X									
3.1.2	Osaa selittää staattisen testauksen arvon	K2	X			X	X									
3.1.3	Pystyy vertaamaan ja vertailemaan staattista ja dynaamista testausta	K2				X	X									
3.2	**Palaute ja katselmointiprosessi**															
3.2.1	Tunnistaa sidosryhmien varhaisen ja säännöllisen palautteen edut	K1	X			X						X				
3.2.2	Osaa kuvata katselmointiprosessin toimenpiteet	K2			X	X										
3.2.3	Muistaa, mitkä vastuut on osoitettu päärooleille katselmointeja tehtäessä	K1				X								X		
3.2.4	Pystyy vertaamaan ja vertailemaan eri katselmointityyppejä	K2		X												
3.2.5	Muistaa tekijät, jotka vaikuttavat onnistuneeseen katselmointiin	K1					X							X		
4 luku	**Testianalyysi ja suunnittelu**															
4.1	**Yleistä testaustekniikoista**															
4.1.1	Erottaa toisistaan mustalaatikko-, lasilaatikko- ja kokemuspohjaiset testaustekniikat	K2		X												
4.2	**Mustalaatikkotekniikat**															
4.2.1	Osaa käyttää ekvivalenssiositusta testitapausten johtamiseen	K3					X									
4.2.2	Osaa käyttää raja-arvoanalyysiä testitapausten johtamiseen	K3					X									
4.2.3	Osaa käyttää päätöstaulutestausta testitapausten johtamiseen	K3					X									
4.2.4	Osaa käyttää tilasiirtymätestausta testitapausten johtamiseen	K3					X									
4.3	**Lasilaatikkotekniikat**															

4.3.1	Osaa selittää lausetestauksen	K2		X												
4.3.2	Osaa selittää haaratestauksen	K2		X												
4.3.3	Osaa selittää lasilaatikkotestauksen arvon	K2	X	X												
4.4	**Kokemukseen pohjautuvat testaustekniikat**															
4.4.1	Osaa selittää virheenarvauksen	K2		X												
4.4.2	Osaa selittää tutkivan testauksen	K2		X												
4.4.3	Osaa selittää tarkistuslistoihin pohjautuvan testauksen	K2		X												
4.5	**Yhteistyöhön perustuvat testauksen lähestymistavat**															
4.5.1	Osaa selittää, miten käyttäjätarinoita kirjoitetaan yhteistyössä kehittäjien ja liiketoiminnan edustajien kanssa	K2				X						X				
4.5.2	Osaa luokitella hyväksymiskriteerien kirjoittamisen eri vaihtoehdot	K2										X				
4.5.3	Osaa käyttää hyväksymistestiohjattua kehitystä testitapausten johtamiseen	K3					X									
5 luku	**Testaustehtävien hallinta**															
5.1	**Testauksen suunnittelu**															
5.1.1	Osaa kuvata testaussuunnitelman tarkoituksen ja sisällön	K2		X					X							
5.1.2	Tunnistaa, miten testaaja tuo lisäarvoa iteraation ja julkaisun suunnitteluun	K1	X									X		X		
5.1.3	Osaa verrata aloitus- ja lopetuskriteerejä ja erottaa ne toisistaan	K2				X		X								X
5.1.4	Osaa käyttää arviointitekniikoita testauksessa tarvittavan työmäärän laskemiseen	K3							X		X					
5.1.5	Osaa priorisoida testitapaukset	K3							X		X					
5.1.6	Muistaa testipyramidin käsitteet	K1		X												

5.1.7	Osaa selittää testausneljännekset ja niiden suhteet testaustasoihin ja testaus-tyyppeihin	K2		X							X					
5.2	**Riskienhallinta**															
5.2.1	Osaa määrittää riskitason käyttämällä riskin todennäköisyyttä ja riskivaikutusta	K1							X						X	
5.2.2	Erottaa projektiriskit ja tuoteriskit toisistaan	K2		X											X	
5.2.3	Osaa selittää, miten tuoteriskianalyysi voi vaikuttaa testauksen perusteellisuu-teen ja laajuuteen	K2					X				X				X	
5.2.4	Osaa selittää, mihin toimenpiteisiin voidaan ryhtyä analysoitujen tuoteriskien perusteella	K2		X			X								X	
5.3	**Testauksen seuranta, hallinta ja päättäminen**															
5.3.1	Tunnistaa testauksessa käytettävät mittarit	K1									X					X
5.3.2	Osaa kuvata testiraporttien tarkoitukset, sisällöt ja kohderyhmät	K2					X				X					X
5.3.3	Osaa antaa esimerkkejä testauksen tilasta tiedottamisesta	K2												X		X
5.4	**Kokoonpanonhallinta**															
5.4.1	Osaa kuvata, miten konfiguraation hallinta tukee testausta	K2					X		X							
5.5	**Vikojenhallinta**															
5.5.1	Osaa laatia vikaraportin	K3		X						X						
6 luku	**Testaustyökalut**															
6.1	**Testauksen työkalutuki**															
6.1.1	Selittää, miten erityyppiset testaustyökalut tukevat testausta	K2					X									
6.2	**Testiautomaation hyödyt ja riskit**															
6.2.1	Muistaa testiautomaation hyödyt ja riskit	K1					X						X			

10. Liite C – Julkaisutiedot

ISTQB® Perustason Sertifikaattisisältö v4.0 on merkittävä päivitys, joka perustuu Perustason sertifikaattisisältöön (v3.1.1) ja Ketterä testaaja 2014 -sertifikaattisisältöön. Tästä syystä lukuja ja osia kohden ei ole yksityiskohtaisia julkaisuselosteita. Seuraavassa esitetään kuitenkin yhteenveto tärkeimmistä muutoksista. Lisäksi ISTQB® tarjoaa erillisessä julkaisuselosteessa jäljitettävyyden Perustason sertifikaattisisällön version 3.1.1, Ketterä testaaja -sertifikaattisisällön vuoden 2014 version oppimistavoitteiden ja uuden Perustason v4.0 -sertifikaattisisällön oppimistavoitteiden välillä, josta näkyy, mitkä oppimistavoitteet on lisätty, päivitetty tai poistettu.

Sertifikaattisisällön kirjoitushetkellä (2022-2023) yli miljoona ihmistä yli 100 maassa on suorittanut Perustason kokeen, ja sertifioituja testaajia on maailmanlaajuisesti yli 800 000. Jos lähdetään siitä, että he kaikki ovat lukeneet Perustason sertifikaattisisällön voidakseen läpäistä kokeen, se tekee Perustason sertifikaattisisällöstä todennäköisesti kaikkien aikojen luetuimman ohjelmistotestausdokumentin! Tämä merkittävä päivitys on tehty tätä perintöä kunnioittaen ja satojen tuhansien muiden ihmisten ISTQB®:n maailmanlaajuiselle testausyhteisölle tarjoamaa laatutasoa koskevan näkemyksen parantamiseksi.

Tässä versiossa kaikkia oppimistavoitteita on muokattu atomisiksi ja yhden-suhde-yhteen -jäljitettävyyden luomiseksi oppimistavoitteiden ja sertifikaattisisällön osien välillä, jolloin sertifikaattisisällössä ei ole sisältöä ilman siihen liittyvää oppimistavoitetta. Tavoitteena on tehdä tästä versiosta helpompi lukea, ymmärtää, oppia ja kääntää, ja siinä keskitytään käytännön hyödyllisyyden lisäämiseen sekä tietojen ja taitojen väliseen tasapainoon.

Tähän pääversioon on tehty seuraavat muutokset:

- Koko sertifikaattisisällön koon pienentäminen. Sertifikaattisisältö ei ole oppikirja, vaan dokumentti, jonka tarkoituksena on kuvata ohjelmistotestauksen peruskurssin peruselementit, mukaan lukien mitä aiheita tulisi käsitellä ja millä tasolla. Siksi erityisesti:
    - useimmissa tapauksissa esimerkkejä ei ole sisällytetty tekstiin. Koulutustarjoajan tehtävänä on antaa esimerkkejä ja harjoituksia koulutuksen aikana.
    - kirjoitettaessa noudatettiin "Sertifikaattisisällön kirjoittamisen tarkistuslistaa", joka antaa suositukset oppimistavoitteiden tekstin enimmäismäärille kullakin K-tasolla (K1 = enintään 10 riviä, K2 = enintään 15 riviä, K3 = enintään 25 riviä).
- Oppimistavoitteiden määrän vähentäminen verrattuna Perustason v3.1.1- ja Ketterä testaaja v2014 -sertifikaattisisältöihin:
    - 14 K1 oppimistavoitetta verrattuna 21:een oppimistavoitteeseen Perustaso v3.1.1:ssä (15) ja Ketterässä 2014 (6)
    - 42 K2 oppimistavoitetta verrattuna 53:een oppimistavoitteeseen Perustaso v3.1.1:ssä (40) ja Ketterässä 2014 (13)
    - 8 K3 oppimistavoitetta verrattuna 15:een oppimistavoitteeseen Perustaso v3.1.1 (7) ja Ketterässä 2014 (8).
- Tarjolla on kattavampia viittauksia klassisiin ja/tai arvostettuihin kirjoihin ja artikkeleihin ohjelmistotestauksesta ja siihen liittyvistä aiheista
- Merkittävät muutokset luvussa 1 (Testauksen perusteet):
    - testaustaitoja koskevaa osaa on laajennettu ja parannettu

- lisätty tiimiperustaista lähestymistapaa koskeva osa (K1)
- testauksen riippumattomuutta koskeva osa siirretty luvusta 5 lukuun 1.

- Merkittävät muutokset luvussa 2 (Testaus ohjelmistokehityksen elinkaaren aikana):
  - kohdat 2.1.1 ja 2.1.2 kirjoitettu uudelleen ja parannettu, vastaavia oppimistavoitteita muutettu
  - enemmän huomiota käytäntöihin, kuten: testit ensin -lähestymistapa (K1), shift-left (K2), jälkipalaverit (K2)
  - uusi osa testauksesta DevOpsin yhteydessä (K2)
  - integraatiotestaustaso jaettu kahteen erilliseen testaustasoon: komponentti-integraatiotestaus ja järjestelmäintegraatiotestaus.
- Merkittävät muutokset luvussa 3 (Staattinen testaus):
  - katselmointitekniikoita koskeva osa sekä K3 oppimistavoite (katselmointitekniikan käyttäminen) poistettu.
- Merkittävät muutokset luvussa 4 (Testianalyysi ja testien suunnittelu):
  - käyttötapaustestaus poistettu (mutta sisältyy edelleen Advanced Test Analyst -sertifikaattisisältöön)
  - enemmän huomiota yhteistyöhön perustuviin menetelmiin testauksessa: uusi K3 oppimistavoite ATDD:n käytöstä testitapausten laatimisessa ja kaksi uutta K2 oppimistavoitetta käyttäjätarinoista ja hyväksymiskriteereistä
  - päätöstestaus ja -kattavuus on korvattu haaratestauksella ja -kattavuudella (ensiksikin, haarakattavuutta käytetään käytännössä yleisemmin; toiseksi, eri standardit määrittelevät päätöksen eri tavoilla, toisin kuin "haaran"; kolmanneksi, tämä ratkaisee pienen, mutta vakavan virheen vanhassa 2018 Sertifikaattisisällössä, jossa väitetään, että "100% päätöskattavuus merkitsee 100% lausekattavuutta" - tämä väite ei ole totta sellaisten ohjelmien osalta, joissa ei ole päätöskohtia)
  - lasilaatikkotestauksen arvoa koskevaa osaa on parannettu.
- Merkittävät muutokset luvussa 5 (Testaustehtävien hallinta):
  - testausstrategioita/lähestymistapoja koskeva osa poistettu
  - uusi K3 oppimistavoite testauksen työmäärien arviointitekniikoihin liittyen
  - enemmän huomiota tunnettuihin ketteryyteen liittyviin käsitteisiin ja työkaluihin testauksen hallinnassa: iteraation ja julkaisun suunnittelu (K1), testipyramidi (K1) ja testausneljännekset (K2)
  - riskienhallintaa koskeva kohta on jäsennelty paremmin kuvaamalla neljä päätoimenpidettä: riskien tunnistaminen, riskien arviointi, riskien vähentäminen ja riskien seuranta.
- Merkittävät muutokset luvussa 6 (Testaustyökalut):
  - joihinkin testiautomaatiota koskeviin asioihin liittyvää sisältöä vähennetty perustasolle liian vaativana; työkalujen valintaa, pilottiprojektien toteuttamista ja työkalujen käyttöönottoa organisaatiossa koskeva kohta poistettu.