Sertifioitu testaaja
Certified Tester

Perustason sertifikaattisisältö
Ketterä testaaja

Foundation Level Syllabus
Agile Tester

Versio 2014
Käännösversio 2015

Perustuu englanninkieliseen versioon 31.5.2014
Based on English version 31st of May, 2014

International Software Testing Qualifications Board

Perustason sertifikaattisisällön Ketterä testaaja työryhmä: Rex Black (Puheenjohtaja), Bertrja Cornanguer (Varapuheenjohtaja), Gerry Coleman (Oppimistavoitteet), Debra Friedenberg (Koe), Alon Linetzki (Liiketoiminnalliset tulokset ja markkinointi), Tauhida Parveen (Julkaisija) ja Leo van der Aalst (Kehitys).

Tekijät: Rex Black, Jaers Claesson, Gerry Coleman, Bertrja Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh ja Stephan Weber.

Sisäiset katselmoijat: Mette Bruhn-Pedersen, Christopher Clements, Alessjaro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael ja Erik van Veenendaal; 2013-2014.

Suomenkielisen version toteutukseen ja katselmointiin ovat osallistuneet seuraavat henkilöt: Kimmo Hakala, Minna Aalto, Kari Kakkonen, Marko Rytkönen, Lare Lekman, Soile Sainio ja Tommi Välimäki.

Versiohistoria

Versio	Päivä	Huomiot
Syllabus v0.1	26.7.2013	Itsenäiset osat
Syllabus v0.2	16.9.2013	Työryhmän katselmoinnin kommentit versiosta 01 sisällytetty
Syllabus v0.3	20.10.2013	Työryhmän katselmoinnin kommentit versiosta 02 sisällytetty
Syllabus v0.7	16.12.2013	Alfakatselmoinnin kommentit versiosta 03 sisällytetty
Syllabus v0.71	20.12.2013	Työryhmäpäivitykset versiosta 07
Syllabus v0.9	30.1.2014	Beetaversio
Syllabus 2014	31.5.2014	GA -versio
Syllabus 2014	30.9.2014	Kirjoitusvirheitä korjattu
suomenkielinen versio 2015	27.4.2015	Käännösversio GA -versiosta

Sisällysluettelo

Kiitokset

Tämän dokumentin tuotti International Software Testing Qualifications Board Foundation -tason työryhmä.

Ydintiimi kiittää katselmointitiimiä ja kansallisia hallituksia ehdotuksistaan sekä palautteestaan.

Ydintiimin jäsenet: Rex Black (Puheenjohtaja), Bertrja Cornanguer (Varapuheenjohtaja), Gerry Coleman (Oppimistavoitteet), Debra Friedenberg (Koe), Alon Linetzki (Liiketoiminnallinen tulos ja markkinointi), Tauhida Parveen (Julkaisija) ja Leo van der Aalst (Kehitys).

Tekijät: Rex Black, Jaers Claesson, Gerry Coleman, Bertrja Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh ja Stephan Weber.

Sisäiset katselmoijat: Mette Bruhn-Pedersen, Christopher Clements, Alessjaro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael ja Erik van Veenendaal.

Ydintiimi haluaa kiittää myös seuraavia henkilöitä kansallisista hallituksista ja Agile expert yhteisöstä, jotka osallistuivat katselmointiin, kommentointiin sekä tämän sertifikaattisisällön äänestyksen toimittamiseen: Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sjaberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic ja Terry Zuo.

Tämä dokumentti on muodollisesti hyväksytty julkaistavaksi ISTQB®:n yleiskokouksessa 31. Toukokuuta 2014.

Suomenkielisen version toteutukseen ja katselmointiin ovat osallistuneet seuraavat henkilöt: Kimmo Hakala, Minna Aalto, Kari Kakkonen, Marko Rytkönen, Lare Lekman, Soile Sainio ja Tommi Välimäki.

0. Johdatus tähän sertifikaattisisältöön

0.1 Tämän dokumentin tarkoitus

Tämä sertifikaattisisältö muodostaa pohjan International Software Testing Qualification -vaatimusten perustasolle ketterään testaukseen. ISTQB® antaa tämän sertifikaattisisällön käyttöön seuraavasti:

- Kansallisille hallituksille käännettäväksi paikalliselle kielelle ja valtuutetuille koulutuksentarjoajille. Kansalliset hallitukset voivat muokata sertifikaattisisältöä paikallisen kielen tarpeiden mukaisesti sekä muokata lähdeviittauksia sovittaen ne paikallisiin julkaisuihin.
- Kansallisille tutkintolautakunnille pohjaksi tuotettaessa koekysymyksiä paikalliselle kielelle sovitettuna sertifikaattisisällön oppimistavoitteisiin.
- Koulutuksen tarjoajille kurssimateriaalin tuottamiseen ja sopivien oppimismenetelmien määrittämiseen.
- Sertifiointikandidaateille kokeeseen valmistautumiseen (osana harjoituskurssia tai itsenäisesti).
- Kansainväliselle ohjelmisto- ja systeemitekniikan yhteisölle kehittämään ohjelmisto- ja systeemitestauksen ammattia sekä perustaksi kirjoille ja artikkeleille.

ISTQB® voi sallia muita entiteettejä käyttämään tätä sertifikaattisisältöä toisiin tarkoituksiin antamalla etukäteen kirjallisen luvan.

0.2 Yleiskatsaus

Perustason sertifikaattisisällön yleiskatsausdokumentti [ISTQB_FA_OVIEW_FI] sisältää seuraavat tiedot:

- Liiketoiminnalliset tulokset sertifikaattisisällölle
- Tiivistelmä sertifikaattisisällöstä
- Syysuhteet sertifikaattisisältöjen välillä
- Oppimistavoitteiden kuvaukset (K -tasot)
- Liitteet

0.3 Tentittävät oppimistavoitteet

Oppimistavoitteita käytetään Perustason sertifikaatin Ketterä testaaja oppisisällön tenttimiseen sertifikaatin saavuttamiseksi. Yleisesti, tämän sertifikaattisisällön kaikki osat ovat K1 -tason tentittäviä ja se tarkoittaa, että kokelas tunnistaa, muistaa ja pystyy palauttamaan mieleensä termin tai käsitteen. Määrätyt oppimistavoitteet tasoilla K1, K2 ja K3 kuvataan asiaankuuluvan luvun alussa.

1. Ketterä ohjelmistokehitys - 150 minuuttia

Avainsanat

Ketterä manifesti, ketterä ohjelmistokehitys, vaiheittainen kehittämismalli, iteratiivinen kehittämismalli, ohjelmiston elinkaari, testausautomaatio, testauksen kohteen kuvaus, testauslähtöinen ohjelmistokehitys, testioraakkeli, käyttäjätarina

Oppimistavoitteet ketterälle ohjelmistokehitykselle

1.1 Ketterän ohjelmistokehityksen perusteet

FA-1.1.1 (K1) Muistat ketterän ohjelmistokehityksen käsitteen, joka perustuu ketterään manifestiin
FA-1.1.2 (K2) Ymmärrät tiimiperusteisen lähestymistavan edut
FA-1.1.3 (K2) Ymmärrät aikaisen sekä jatkuvan palautteen merkityksen

1.2 Näkökulmia ketteriin lähestymistapoihin

FA-1.2.1 (K1) Muistat ketterän ohjelmistokehityksen näkökulmia
FA-1.2.2 (K3) Osaat kirjoittaa testattavia käyttäjätarinoita yhteistyössä kehittäjien sekä liiketoiminnan edustajien kanssa
FA-1.2.3 (K2) Ymmärrät kuinka retrospektiivejä voidaan hyödyntää ketterien projektien prosessikehityksessä
FA-1.2.4 (K2) Ymmärrät jatkuvan integraation käytön sekä tarkoituksen
FA-1.2.5 (K1) Tiedät miten iteraation ja julkaisun suunnittelu eroavat toisistaan. Ymmärrät kuinka testaaja tuo lisäarvoa niissä tapahtuviin tehtäviin

1.1 Ketterän ohjelmistokehityksen perusteet

Ketterän projektin testaaja työskentelee eri tavalla kuin testaaja perinteisessä projektissa. Testaajan täytyy ymmärtää ketteriä projekteja tukevat arvot ja periaatteet. Testaajan täytyy myös ymmärtää merkityksensä osana tiimiperustaista lähestymistapaa yhdessä kehittäjien ja liiketoiminnan edustajien kanssa. Ketterässä projektissa työskentelevien henkilöiden kommunikointi on aikaista ja jatkuvaa, mikä auttaa havaitsemaan vikoja aikaisin ja kehittämään laadukkaan tuotteen.

1.1.1 Ketterä ohjelmistokehitys ja ketterän ohjelmistokehityksen manifesti

Vuonna 2001 joukko laajimmin käytettyjen kevyiden ohjelmistokehitysmenetelmien edustajia sopi yhteisestä arvolauseiden ja periaatteiden joukosta, joka tuli tunnetuksi Ketterän ohjelmistokehityksen julistuksena tai Ketteränä manifestina. Ketterä manifesti sisältää neljä lausetta. Kokemuksemme perusteella arvostamme:

Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota
Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
Muutokseen reagoimista enemmän kuin pitäytymistä suunnitelmassa

Ketterä manifesti esittää, että vaikka edellä mainittujen lauseiden oikealla puolellakin olevilla käsitteillä on arvoa, vasemmalla puolella olevia käsitteitä arvostetaan kuitenkin enemmän.

Yksilöt ja kanssakäyminen

Ketterä ohjelmistokehitys on hyvin ihmiskeskeistä. Tiimit muodostuvat henkilöistä, jotka kehittävät ohjelmiston. Tiimin tehokkaassa työskentelyssä on kyse pikemminkin jatkuvasta kommunikoinnista ja kanssakäymisestä kuin turvautumisesta työkaluihin ja prosesseihin.

Toimiva ohjelmisto

Asiakkaan näkökulmasta toimiva ohjelmisto on paljon hyödyllisempi ja arvokkaampi kuin hyvin tarkka dokumentaatio ja se tarjoaa mahdollisuuden antaa kehittäjille nopeaa palautetta. Lisäksi toimiva ohjelmisto (vaikkakin rajoitetuilla toiminnoilla) on saatavilla paljon aikaisemmin kehityksen elinkaaressa, mikä voi nopeuttaa merkittävästi tuotteen julkaisua markkinoille. Näin ollen ketterä ohjelmistokehitys on erityisen hyödyllinen nopeasti muuttuvissa liiketoimintaympäristöissä, joissa ongelmat ja/tai ratkaisut ovat epäselviä, tai joissa liiketoiminta toivoo innovoivansa uusilla ongelma-alueilla.

Asiakasyhteistyö

Asiakkaan on usein hyvin vaikea määritellä haluamaansa järjestelmää ja tiivis yhteistyö asiakkaan kanssa parantaa todennäköisyyttä ymmärtää asiakkaan vaatimukset. Vaikka asiakkaan kanssa tehdyn sopimuksen merkitys on suuri, säännöllinen ja läheinen yhteistyö asiakkaan kanssa auttaa todennäköisesti onnistumaan paremmin projektissa.

Vastaaminen muutokseen

Muutos on väistämätöntä ohjelmistoprojekteissa. Lainsäädännöllä, kilpailijoiden toiminnalla, tekniikan kehittymisellä sekä muilla liiketoimintaympäristön tekijöillä on suuri vaikutus projektiin ja sen tavoitteisiin ja näihin tekijöihin on myös kehitysprosessin mukauduttava. Sinänsä työkäytäntöjen joustavuus muutoksen omaksumisessa on tärkeämpää kuin se, että noudatetaan jäykästi suunnitelmaa.

Periaatteet

Ketterän ohjelmistokehityksen julistuksen takana on 12 periaatetta.

- Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.
- Muuttuvat vaatimukset ovat tervetulleita myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.
- Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein ja suosimme tiheämpää aikaväliä.
- Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.
- Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jotka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.
- Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.
- Toimiva ohjelmisto on edistymisen ensisijainen mittari.
- Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.
- Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.
- Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.
- Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.
- Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan ja mukauttaa toimintaansa sen mukaisesti.

Eri ketterät menetelmät tarjoavat ohjaavia toimintatapoja julistuksen periaatteiden käytännön toteuttamiseen.

1.1.2 Tiimiperusteinen lähestymistapa

Tiimiperusteinen lähestymistapa tarkoittaa sitä, että kaikki joilla on projektin menestyksen kannalta välttämätöntä tietoa ja osaamista, osallistuvat. Tiimiin kuuluu asiakkaan sekä muita liiketoiminnan sidosryhmien edustajia, jotka määrittävät tuotteen ominaisuudet. Tiimin olisi hyvä olla suhteellisen pieni; menestyviä tiimejä on tunnistettu aina kolmen henkilön tiimeistä yhdeksän henkilön tiimeihin. Ihanteellisesti koko tiimi jakaa saman työtilan, sillä yhteinen fyysinen sijainti edesauttaa voimakkaasti kommunikointia ja vuorovaikutusta. Tiimiperusteista lähestymistapaa tuetaan päivittäisillä seisoen pidettävillä palavereilla (katso luku 2.2.1), johon osallistuvat kaikki tiimin jäsenet ja joissa kommunikoidaan työn edistyminen ja esteet nostetaan esille. Tiimiperusteinen lähestymistapa edistää tehokkaampaa ja suorituskykyisempää tiimidynamiikkaa.

Tiimiperusteisen lähestymistavan käyttäminen tuotekehityksessä on yksi ketterän kehityksen keskeistä hyödyistä. Sen hyödyt ovat seuraavat:

- Parantaa kommunikointia ja yhteistyötä tiimissä
- Mahdollistaa erilaisten taitojen valjastamisen projektin hyödyksi
- Laatu on jokaisen vastuulla

Ketterissä projekteissa laadusta vastaa koko tiimi. Tiimiperusteisen lähestymistavan ydin perustuu siihen, että testaajat, kehittäjät ja liiketoiminnan edustajat työskentelevät yhdessä jokaisessa kehitysprosessin vaiheessa. Testaajat työskentelevät läheisesti sekä kehittäjien että liiketoiminnan edustajien kanssa taatakseen sen, että haluttu laatutaso saavutetaan. Tähän sisältyy liiketoiminnan edustajien tukeminen sekä yhteistyö heidän kanssaan, jotta he pystyvät laatimaan sopivat hyväksymistestit. Siihen kuuluu myös työskentely kehittäjien kanssa, jotta saavutetaan yhteinen käsitys testausstrategiasta, sekä päättäminen testausautomaation lähestymistavoista. Näin ollen testaajat voivat siirtää ja levittää testausosaamista muille tiimin jäsenille sekä vaikuttaa tuotteen kehittämiseen.

Koko tiimi osallistuu jokaiseen konsultointiin tai tapaamisiin, jossa tuotteen ominaisuuksia esitetään, analysoidaan tai arvioidaan. Ajatusta jossa kaikkiin tuotteen kehitystä koskeviin keskusteluihin osallistuvat testaajat, kehittäjät ja liiketoiminnan edustajat, kutsutaan kolmen voimaksi [Crispin08].

1.1.3 Aikainen ja säännöllinen palaute

Ketterillä projekteilla on lyhyet iteraatiot, mikä mahdollistaa sen, että projektitiimi saa tuotteen laadusta jatkuvaa palautetta aikaisin läpi koko kehityksen elinkaaren. Yksi tapa antaa nopeaa palautetta on jatkuva integraatio (katso luku 1.2.4).

Kun käytetään peräkkäistä kehitysprosessia, asiakas ei useinkaan näe tuotetta kuin vasta lähellä projektin loppua. Siinä vaiheessa kehitystiimin on usein liian myöhäistä ottaa enää käsiteltäväksi mahdollisia asiakkaan ongelmia. Saamalla säännöllistä asiakaspalautetta projektin edistyessä ketterät tiimit voivat sisällyttää useimmat uudet muutokset tuotekehitysprosessiin. Aikainen ja säännöllinen palaute auttaa tiimiä keskittymään ja toimittamaan asiakkaalle ensiksi ne ominaisuudet, joilla on korkein arvo liiketoiminnalle tai ominaisuuteen liittyvä riski. Se auttaa myös johtamaan tiimiä paremmin, koska tiimin kyvykkyys on läpinäkyvää kaikille. Esimerkiksi kuinka paljon työtä voidaan tehdä sprintissä tai iteraatiossa? Mikä voisi auttaa edistymään nopeammin? Mikä estää meitä tekemästä niin? Aikaisen ja säännöllisen palautteen hyödyt ovat:

- Vältytään vaatimuksiin liittyviltä väärinymmärryksiltä, joista aiheutuvia ongelmia ei välttämättä löydetä ennen kuin vasta myöhemmin kehitysprosessissa, jolloin ne ovat kalliimpia korjata.

- Selkeytetään asiakkaan ominaisuuksiin liittyviä pyyntöjä ja saadaan ominaisuudet aikaisin asiakkaan käytettäväksi. Tällä tavalla tuote heijastaa paremmin asiakkaan tarpeita.
- Löydetään (jatkuvan integraation kautta), eristetään ja ratkaistaan laatuongelmat aikaisin.
- Tarjotaan tietoa ketterälle tiimille sen tuottavuudesta ja toimituskyvystä.
- Edistetään tasaista projektin etenemisvauhtia.

1.2 Näkökulmia ketteriin lähestymistapoihin

Organisaatiot käyttävät useita erilaisia ketteriä lähestymistapoja. Yleisiä käytäntöjä useimmissa ketterissä organisaatioissa ovat yhteistyössä tapahtuva käyttäjätarinoiden luonti, retrospektiivit, jatkuva integraatio sekä jokaisen iteraation kuin myös koko julkaisun suunnittelu. Tässä aliluvussa käydään läpi muutamia ketteriä lähestymistapoja.

1.2.1 Ketterän ohjelmistokehityksen lähestymistapoja

Ketteriä lähestymistapoja on useita ja niistä jokainen toteuttaa Ketterän ohjelmistokehityksen julistuksen arvoja ja periaatteita eri tavalla. Tässä sertifikaattisisällössä tarkastellaan kolmea eri lähestymistapaa: Extreme Programming (XP), Scrum ja Kanban.

Extreme Programming (XP)

XP:n toi alun perin julkisuuteen Kent Beck [Beck04] ja se on ketterä ohjelmistokehityksen lähestymistapa, jota kuvaavat tietyt arvot, periaatteet sekä ohjelmistokehityksen käytännöt.

XP sisältää viisi kehitystä ohjaavaa arvoa: kommunikointi, yksinkertaisuus, palaute, rohkeus ja kunnioitus.

XP sisältää myös joukon periaatteita täydentäviä arvoja: ihmisyys, talous, yhteinen hyöty, itsesimilaarisuus, parantaminen, moninaisuus, reflektio, virtaus, mahdollisuus, redundanssi, epäonnistuminen, laatu, pienet askeleet ja vastuullisuus.

XP kuvaa 13 pääkäytäntöä: yhdessä istuminen, koko tiimi, informatiivinen työtila, innostettu työskentely, pariohjelmointi, tarinat, viikoittainen sykli, neljännesvuosittainen sykli, verkkaisuus, kymmenen minuutin koonti, jatkuva integraatio, testilähtöinen ohjelmointi ja vaiheittainen suunnittelu.

Useat nykyään käytössä olevista ketterän kehityksen lähestymistavoista ovat ottaneet vaikutteita XP:n arvoista ja periaatteista. Esimerkiksi ketterät tiimit scrumissa usein sisällyttävät XP:n käytäntöjä toiminnassaan.

Scrum

Scrum on ketterän johtamisen kehys ja se sisältää seuraavat keskeiset välineet ja käytännöt [Schwaber01]:

- Sprintti: scrum jakaa projektin iteraatioihin (kutsutaan sprinteiksi), jotka ovat kiinteän mittaisia (yleensä kahdesta neljään viikkoon)
- Tuoteversio: jokaisen sprintin tuotoksena syntyy potentiaalinen julkaistava/toimitettava tuote (kutsutaan tuoteversioksi).
- Tuotteen kehitysjono: Tuoteomistaja hallinnoi priorisoitua kehitettävien asioiden listaa (kutsutaan tuotteen kehitysjonoksi). Tuotteen kehitysjono kehittyy jokaisessa sprintissä (kutsutaan tuotteen kehitysjonon työstämiseksi).
- Sprintin kehitysjono: Jokaisen sprintin alussa scrumtiimi valitsee joukon korkeimman prioriteetin kehitettäviä asioita tuotteen kehitysjonosta sprinttiin (kutsutaan sprintin kehitysjonoksi). Koska

tuoteomistajan sijaan scrumtiimi valitsee sprintissä toteutettavat asiat, valintaa kutsutaan ennemminkin vetäväksi kuin työntäväksi periaatteeksi.
- Valmiin määritelmä: Varmistaakseen sen, että jokaisen sprintin jälkeen on potentiaalinen julkaistava versio, tiimi keskustelee ja määrittää sopivat kriteerit sprintin päättämiselle. Tämä keskustelu syventää tiimin ymmärrystä kehitettävistä asioista ja tuotevaatimuksista.
- Aikarajoittaminen: Vain ne tehtävät, vaatimukset tai ominaisuudet, jotka tiimi olettaa saavansa valmiiksi sprintin aikana, ovat mukana sprintin kehitysjonossa. Jos kehitystiimi ei pysty saamaan valmiiksi tehtävää sprintin aikana, tehtävään liittyvät ominaisuudet vedetään pois sprintistä ja kyseinen tehtävä siirretään takaisin tuotteen kehitysjonoon. Aikarajoittaminen ei koske ainoastaan tehtäviä, vaan myös muita tilanteita (esimerkiksi kokoukset pakotetaan alkamaan ja loppumaan ajallaan).
- Läpinäkyvyys: Kehitystiimi raportoi ja päivittää sprintin tilanteen päivittäisessä kokouksessa (kutsutaan päiväpalaveriksi). Se tekee meneillään olevan sprintin sisällön ja edistymisen mukaan lukien testitulokset, näkyväksi tiimille, johdolle ja kaikille kiinnostuneille osapuolille. Esimerkiksi kehitystiimi voi näyttää sprintin tilanteen kirjoitustaululta.

Scrum määrittelee kolme roolia:
- Scrummaster: Varmistaa sen, että scrumin käytännöt sekä säännöt otetaan käyttöön ja niitä noudatetaan. Hän selvittää kaikki rikkomukset, resurssiasiat tai muut esteet, jotka voivat estää tiimiä noudattamasta käytäntöjä ja ohjeita. Tämän roolin edustaja ei ole tiiminvetäjä vaan valmentaja.
- Tuoteomistaja: Edustaa asiakasta ja luo, ylläpitää ja priorisoi tuotteen kehitysjonon. Tämän roolin edustaja ei ole tiiminvetäjä.
- Kehitystiimi: Kehittää ja testaa tuotetta. Tiimi on itseorganisoitunut: ei ole tiiminvetäjää, joten tiimi tekee päätökset. Tiimi on myös osaamiseltaan monitahoinen (katso luvut 2.3.2 ja 3.1.4).

Scrum (toisin kuin XP) ei sanele mitään määrättyjä ohjelmistokehityksen tekniikoita (esimerkiksi testilähtöinen ohjelmistokehitys). Lisäksi scrum ei tarjoa opastusta siihen, miten testaus pitäisi tehdä scrum-projektissa.

Kanban

Kanban [Jaerson13] on johtamisen lähestymistapa, jota toisinaan käytetään ketterissä projekteissa. Sen yleinen tavoite on visualisoida ja optimoida kehitysjono arvoketjussa. Kanban hyödyntää kolmea välinettä [Linz14]:
- Kanban-taulu: Hallinnoitava arvoketju kuvataan Kanban-taululla. Jokainen sarake näyttää sijoituspaikan, jossa on siihen liittyvät tehtävät, kuten esimerkiksi kehitys ja testaus. Tuotettavien asioiden tai käsiteltävien tehtävien kuvaamiseen käytetään lappuja, joita siirretään vasemmalta oikealle sijoituspaikasta toiseen.
- Work-in-Progress Limit (WIP): Rinnakkaisten aktiivisten tehtävien määrä on tiukasti rajoitettu. Sitä kontrolloidaan asettamalla maksimiarvot sijoituspaikassa ja/tai taululla oleville lapuille. Jos sarakkeessa on vapaata kapasiteettia, työntekijä vetää lapun edellisestä sarakkeesta.
- Läpimenoaika: Kanbania käytetään optimoimaan jatkuva tehtävävirta minimoimalla (keskimääräinen) läpimenoaika arvovirrassa.

Kanban sisältää yhtäläisyyksiä scrumiin. Molemmissa aktiiviset tehtävät on visualisoitu, mikä tekee niiden sisällön ja tilanteen läpinäkyväksi. Ne tehtävät, joita ei ole vielä aikataulutettu, odottavat kehitysjonossa, josta ne siirretään Kanban-taululle heti, kun siellä on uutta tilaa (tuotantokapasiteettia).

Iteraatiot tai sprintit ovat Kanbanissa valinnaisia. Kanban-prosessi mahdollistaa tuotosten julkaisun yksi kerrallaan ennemmin kuin osana julkaisua. Siksi aikarajoittaminen synkronointimenetelmänä on valinnaista toisin kuin scrumissa, jossa kaikki sprinttiin sisältyvät tehtävät synkronoidaan.

1.2.2 Yhteistyössä tapahtuva käyttäjätarinoiden luonti

Heikko määrittely on usein suuri syy projektin epäonnistumiselle. Määrittelyongelmat voivat olla seurausta käyttäjien näkemyksen puutteesta heidän todellisiin tarpeisiinsa, kokonaisnäkemyksen puuttumisesta järjestelmään, tarpeettomista tai ristiriitaisista ominaisuuksista ja muusta erheellisestä viestinnästä. Ketterässä kehityksessä käyttäjätarinat kirjoitetaan tallentamaan vaatimukset kehittäjien, testaajien ja liiketoiminnan edustajien näkökulmasta. Peräkkäismallisessa kehityksessä tämä jaettu visio ominaisuuksista saavutetaan muodollisten katselmointien avulla sen jälkeen, kun vaatimukset on kirjoitettu; ketterässä kehityksessä jaettu visio saadaan aikaan tekemällä säännöllisesti vapaamuotoisia katselmointeja, kun vaatimuksia kirjoitetaan.

Käyttäjätarinoiden täytyy käsitellä sekä toiminnallisia että ei-toiminnallisia ominaisuuksia. Jokainen tarina sisältää hyväksymiskriteerit näille ominaisuuksille. Nämä kriteerit pitäisi määritellä yhteistyössä liiketoiminnan edustajien, kehittäjien ja testaajien kanssa. Ne tarjoavat kehittäjille ja testaajille ominaisuudesta laajemman vision, jonka liiketoiminnan edustajat kelpuuttavat. Ketterä tiimi pitää tehtävää suoritettuna sitten kun hyväksymiskriteerit on täytetty.

Tyypillisesti testaajan ainutlaatuinen näkökulma parantaa käyttäjätarinaa, koska hän havainnoi puuttuvia yksityiskohtia tai ei-toiminnallisia vaatimuksia. Testaaja voi osaltaan edistää työtä kysymällä liiketoiminnan edustajilta avoimia kysymyksiä käyttäjätarinaan liittyen, ehdottamalla tapoja käyttäjätarinan testaamiseksi ja varmistamalla sen hyväksymiskriteerit.

Yhteistyössä tapahtuvan käyttäjätarinan luonnissa voidaan käyttää sellaisia tekniikoita kuin aivoriihi ja mielekartta. Testaaja voi käyttää INVEST-tekniikkaa [INVEST]:

- Itsenäinen (Independent)
- Neuvoteltavissa oleva (Negotiable)
- Arvokas (Valuable)
- Arvioitavissa oleva (Estimable)
- Pieni (Small)
- Testattavissa oleva (Testable)

3C konseptin mukaan [Jeffries00] käyttäjätarina on yhdistelmä kolmesta elementistä:

- Kortti: Kortti on fyysinen media, joka kuvaa käyttäjätarinan. Se identifioi vaatimuksen, sen kriittisyyden, odotetun kehitys- ja testausajan sekä hyväksymiskriteerit tarinalle. Kuvauksen pitää olla täsmällinen, koska sitä tullaan käyttämään tuotteen kehitysjonossa.
- Keskustelu: Keskustelu selittää kuinka ohjelmistoa tullaan käyttämään. Keskustelu voi olla dokumentoitua tai suullisesti tapahtuvaa. Testaajien erilainen näkökulma kehittäjiin ja liiketoiminnan edustajiin verrattuna [ISTQB_FL_SYL] tuo arvokasta palautetta ajatusten vaihdolle, mielipiteille sekä kokemuksille. Keskustelu alkaa julkaisun suunnitteluvaiheessa ja jatkuu, kun tarina on aikataulutettu.
- Vahvistaminen: Hyväksymiskriteerit, joista on puhuttu keskusteluissa, käytetään vahvistamaan tarina tehdyksi. Nämä hyväksymiskriteerit voivat kattaa useita käyttäjätarinoita. Sekä positiivista että negatiivista testausta pitäisi käyttää kriteerien kattamiseksi. Vahvistamisen aikana eri osallistujat esittävät testaajan roolia. Heihin voi kuulua niin kehittäjiä kuin suorituskykyyn, tietoturvaan, yhteen toimivuuteen ja muihin laatuominaisuuksiin keskittyviä asiantuntijoita. Jotta tarina voidaan vahvistaa tehdyksi, määritetyt hyväksymiskriteerit pitäisi testata ja osoittaa kriteerien täyttyminen.

Ketterät tiimit eroavat sen mukaan, kuinka he dokumentoivat käyttäjätarinoita. Huolimatta siitä, mitä lähestymistapaa käyttäjätarinoiden dokumentoimisessa käytetään, dokumentoinnin pitäisi olla ytimekästä, riittävää ja tarpeellista.

1.2.3 Retrospektiivit

Ketterässä kehityksessä retrospektiivi tarkoittaa joka iteraation lopussa pidettävää kokousta, jossa keskustellaan siitä, missä onnistuttiin, mitä voisi parantaa sekä kuinka parannukset voidaan toteuttaa ja tähänastiset onnistumiset toistaa tulevissa iteraatioissa. Retrospektiivit kattavat sellaisia aiheita kuin prosessi, ihmiset, organisoitumiset, syysuhteet ja työkalut. Säännöllisesti pidetyt retrospektiivit, joihin liittyen toteutetaan sopivat seurantatoimet, ovat kriittisiä itseorganisoitumiselle sekä kehityksen ja testauksen jatkuvalle parantamiselle.

Retrospektiivien seurauksena voi olla testaukseen liittyviä parannuspäätöksiä liittyen testauksen tehokkuuteen, testauksen tuottavuuteen, testitapausten laatuun ja tiimityytyväisyyteen. Ne voivat käsitellä myös sovelluksen, käyttäjätarinoiden, ominaisuuksien tai järjestelmän rajapintojen testattavuutta. Vikojen juurisyyanalyysi voi ohjata testauksen ja kehityksen parannuksiin. Tiimien pitäisi ylipäänsä toteuttaa vain muutamia parannuksia yhteen iteraatioon. Tämä mahdollistaa jatkuvan parantamisen pysyvällä tahdilla.

Retrospektiivien ajoitus ja organisointi riippuvat siitä, mitä ketterää menetelmää seurataan. Liiketoiminnan edustajat ja tiimi ovat jokaisessa retrospektiivissä osallistujan roolissa ja fasilitaattori organisoi ja pitää kokouksen. Joissakin tapauksissa tiimi voi kutsua muita osallistujia kokoukseen.

Testaajien pitäisi olla tärkeässä roolissa retrospektiiveissä. Testaajat ovat osa tiimiä ja he tuovat mukaan ainutlaatuista näkökulmaa [ISTQB_FL_SYL], luku 1.5. Testausta tehdään jokaisessa sprintissä ja se vaikuttaa ratkaisevasti onnistumiseen. Kaikki tiimin jäsenet mukaan lukien testaajat ja ne, jotka eivät testaa, voivat tarjota työpanostaan sekä testauksen että testaukseen kuulumattomiin tehtäviin.

Retrospektiivit täytyy pitää ammattimaisessa ympäristössä, jossa vallitsee keskinäinen luottamus. Retrospektiivin menestystekijät ovat samoja, kuin minkä tahansa katselmoinnin menestystekijät, joita käytiin läpi luvussa 3.2 Perustason sertifikaattisisällössä [ISTQB_FL_SYL].

1.2.4 Jatkuva integraatio

Tuotteen inkrementin toimitus vaatii luotettavaa, toimivaa ja integroitua ohjelmistoa jokaisen sprintin lopussa. Jatkuva integraatio käsittelee tätä haastetta kokoamalla yhteen kaikki ohjelmistoon tehdyt muutokset ja integroimalla kaikki muutetut komponentit säännöllisesti, ainakin kerran päivässä. Kokoonpanonhallinta, käännös, ohjelmiston koonti, kehittäminen ja testaus on paketoitu yhdeksi automatisoiduksi toistettavaksi prosessiksi. Koska kehittäjät integroivat työnsä jatkuvasti, koostavat ja testaavat jatkuvasti, viat koodissa voidaan havaita nopeammin.

Kehittäjien tekemää koodausta, virheenjäljitystä ja koodin yhteiseen lähdekoodikirjastoon tallentamista seuraava jatkuvan integroinnin prosessi koostuu seuraavista automatisoiduista tehtävistä:

- Staattinen analyysi koodille: staattisen koodianalyysin suorittaminen ja tulosten raportointi
- Kääntäminen: koodin kääntäminen ja linkittäminen sekä ajettavien tiedostojen luominen
- Yksikkötestaus: yksikkötestien suorittaminen, koodikattavuuden tarkistaminen ja testitulosten raportointi
- Levittäminen: koonnin asentaminen testausympäristöön
- Integraatiotestaus: integraatiotestauksen suorittaminen ja tulosten raportointi
- Raportointi (näkymä): kaikkien edellä mainittujen toimintojen statuksen kirjaaminen julkiseen näkyvään paikkaan tai sähköpostitse lähettäminen

Automaattinen koonti ja testausprosessi tapahtuu päivittäin ja se havaitsee integraatiovirheet aikaisin ja nopeasti. Jatkuva integraatio antaa ketterille testaajille mahdollisuuden suorittaa automatisoidut testit säännöllisesti ja joissakin tapauksessa osana jatkuvan integroinnin prosessia, sekä lähettää nopeasti palautetta tiimille koodin laadusta. Nämä testitulokset ovat näkyvissä koko tiimille erityisesti silloin, kun automatisoidut raportit on integroitu prosessiin. Automaattinen regressiotestaus voi olla jatkuvaa läpi

iteraation. Hyvä automaattinen regressiotestaus kattaa niin paljon toiminnallisuudesta kuin on mahdollista, mukaan lukien aiemmissa iteraatioissa toimitetut käyttäjätarinat. Automaattisten regressiotestien hyvä kattavuus auttaa tukemaan laajojen integroitujen järjestelmien rakentamista (ja testausta). Kun regressiotestit on automatisoitu, testaajat vapautuvat keskittämään manuaalisen testauksensa uusiin ominaisuuksiin, tehtyihin muutoksiin sekä vikakorjausten varmistustestaukseen.

Automatisoitujen testien lisäksi jatkuvaa integraatiota noudattavat organisaatiot käyttävät tyypillisesti koontityökaluja jatkuvan laadun valvontaan. Yksikkö- ja integraatiotestien lisäksi nämä työkalut voivat suorittaa täydentäviä staattisia ja dynaamisia testejä, mitata ja profiloida suorituskykyä, luoda ja muotoilla dokumentaatiota lähdekoodista ja helpottaa manuaalisia laadunvalvontaprosesseja. Tämä jatkuva laadun valvonnan toteuttaminen tähtää tuotteen laadun parantamiseen ja sen toimittamiseen käytetyn ajan vähentämiseen korvaamalla perinteinen tapa toteuttaa laadunvalvontatehtävät vasta koko kehitystyön valmistumisen jälkeen.

Koontityökalut voidaan linkittää automaattisiin kehitystyökaluihin, jotka voivat poimia sopivan koonnin jatkuvasta integraatiosta tai koontipalvelimelta ja jakaa sitä yhteen tai useampaan kehitys-, testaus-, esitys- ja jopa tuotantoympäristöihin. Tämä vähentää virheitä ja viivästyksiä, jotka liittyvät siihen, että julkaisujen asentamisessa näihin ympäristöihin joudutaan turvautumaan tehtävään erikoistuneisiin henkilöihin tai kehittäjiin.

Jatkuva integraatio voi tarjota seuraavia etuja:

- Mahdollistaa aikaisemman integraatio-ongelmien sekä ristiriitaisten muutosten havaitsemisen sekä helpomman juurisyyanalyysin
- Antaa kehitystiimille säännöllistä palautetta siitä, toimiiko koodi
- Pitää päivän aikana testatun ohjelmiston version samana kehitetyssä versiossa
- Vähentää kehittäjän koodin refaktoroinnista aiheutuvaa regressioriskiä johtuen koodin nopeasta uudelleentestauksesta jokaisen pienen muutoksen yhteydessä
- Tarjoaa luottamuksen siihen, että jokapäiväinen kehitystyö perustuu kiinteään perustaan
- Tuo näkyvyyttä tuoteinkrementin valmistumisen lähestymiseen, mikä kannustaa kehittäjiä ja testaajia
- Poistaa isoihin integraatioihin liittyvän aikatauluriskin
- Tarjoaa suoritettavan ohjelmiston jatkuvan saatavuuden läpi sprintin testaus-, esitys- tai koulutustarkoituksiin
- Vähentää toistuvia manuaalisia testauksen tehtäviä
- Tarjoaa nopeaa palautetta tehdyistä laadun- ja testauksenparannuspäätöksistä

Jatkuvaan integraatioon liittyy kuitenkin myös riskejä ja haasteita:

- Jatkuvan integraation työkalut pitää ottaa käyttöön ja niitä pitää ylläpitää
- Jatkuvan integraation prosessi pitää olla määritetty ja vakiintunut
- Testausautomatisointi vaatii resursseja ja sen perustaminen voi olla monimutkaista
- Perusteellinen testikattavuus on olennaista, jotta saavutetaan automatisoidun testauksen hyödyt
- Toisinaan tiimit luottavat liikaa yksikkötesteihin ja suorittavat liian vähän järjestelmä- ja hyväksymistestejä

Jatkuva integraatio vaatii työkalujen käyttämistä, mukaan lukien testaustyökalut, automatisoidun koontiprosessin työkalut ja versionhallintatyökalut.

1.2.5 Julkaisun ja iteraation suunnitteleminen

Kuten Perustason sertifikaattisisällössä mainitaan [ISTQB_FL_SYL], suunnittelu on jatkuva tehtävä ja sama pätee myös ketteriin elinkaariin. Niissä tehdään kahdentyyppistä suunnittelua, sekä julkaisun että iteraation.

Julkaisun suunnittelu katsoo eteenpäin tuotteen julkaisuun, usein jo muutamia kuukausia ennen projektin alkamista. Julkaisun suunnittelu määrittelee ja uudelleen määrittelee tuotteen kehitysjonon ja siihen voi kuulua laajojen käyttäjätarinoiden jalostamista pienempien käyttäjätarinoiden kokoelmaksi. Julkaisun suunnittelu antaa perustan testauksen lähestymistavalle ja testauksen suunnittelulle läpi kaikkien iteraatioiden. Julkaisusuunnitelmat tehdään korkealla tasolla.

Julkaisua suunniteltaessa liiketoiminnan edustajat hyväksyvät ja priorisoivat julkaisuun liittyvät käyttäjätarinat yhdessä tiimin kanssa (ks. kappale 1.2.2). Näihin käyttäjätarinoihin perustuen identifioidaan projekti- ja laaturiskit ja suoritetaan korkean tason arviointia (katso luku 3.2).

Testaajat osallistuvat julkaisun suunnitteluun ja tuovat lisäarvoa erityisesti seuraaviin tehtäviin:

- Testattavien käyttäjätarinoiden määrittäminen, mukaan lukien hyväksymiskriteerit
- Projekti- ja laaturiskien analysointiin osallistuminen
- Käyttäjätarinoihin tarvittavan testauspanoksen arvioiminen
- Tarvittavien testitasojen määrittäminen
- Julkaisun testauksen suunnitteleminen

Sen jälkeen kun julkaisun suunnittelu on tehty, alkaa ensimmäisen iteraation suunnittelu. Iteraation suunnittelu katsoo eteenpäin yksittäisen iteraation loppuun asti ja se käsittelee iteraation kehitysjonoa.

Iteraation suunnittelussa tiimi valitsee käyttäjätarinat priorisoidusta julkaisun kehitysjonosta, tarkentaa niiden sisältöjä, suorittaa niille riskianalyysin ja arvioi jokaiselle tarinalle tarvittavan työmäärän. Jos käyttäjätarina on liian epäselvä eikä sitä onnistuta selkiyttämään, tiimi voi kieltäytyä hyväksymästä sitä ja käyttää prioriteettijärjestyksessä seuraavana olevaa käyttäjätarinaa. Liiketoiminnan edustajien täytyy vastata tiimin kysymyksiin jokaisesta käyttäjätarinasta niin, että tiimi voi ymmärtää mitä heidän pitäisi toteuttaa ja kuinka heidän pitäisi testata tarina.

Valittujen tarinoiden määrä perustuu tiimin vauhtiin ja valittujen käyttäjätarinoiden arvioituun kokoon. Sen jälkeen kun iteraation sisältö on lyöty lukkoon, käyttäjätarinat pilkotaan sopivien tiimin jäsenten toteutettaviksi tehtäviksi.

Testaajat osallistuvat iteraation suunnitteluun ja tuovat lisäarvoa erityisesti seuraavissa tehtävissä:

- Osallistumalla käyttäjätarinoiden yksityiskohtaiseen riskianalysointiin
- Määrittämällä käyttäjätarinoiden testattavuutta
- Luomalla hyväksymistestejä käyttäjätarinoille
- Pilkkomalla käyttäjätarinoita tehtäviksi (erityisesti testaustehtäviksi)
- Arvioimalla testauksen työmäärää kaikille testaustehtäville
- Identifioimalla testattavan järjestelmän toiminnallisia ja ei-toiminnallisia puolia
- Tukemalla ja osallistumalla testausautomaatioon useilla eri testaustasoilla

Julkaisusuunnitelmat voivat muuttua projektin edetessä, mukaan lukien tuotteen julkaisujonossa olevat yksittäiset käyttäjätarinat. Nämä muutokset voivat aiheutua sisäisistä tai ulkoisista tekijöistä. Sisäisiin tekijöihin kuuluvat toimituskyvykkyys, nopeus ja tekniset asiat. Ulkoisiin tekijöihin kuuluvat uusien markkinoiden ja mahdollisuuksien löytyminen, uudet kilpailijat tai liiketoiminnan uhkat, jotka voivat vaikuttaa julkaisun tavoitteisiin ja/tai tavoitepäiviin. Lisäksi iteraatiosuunnitelmat voivat muuttua iteraation aikana. Esimerkiksi tietty käyttäjätarina, jota oli pidetty suhteellisen yksikertaisena arvioinnissa, voi olla paljon odotettua monimutkaisempi.

Nämä muutokset voivat olla testaajille haasteellisia. Testaajien täytyy ymmärtää julkaisun koko kuva testauksen suunnittelua varten ja heillä täytyy olla joka iteraatiossa käytettävissään riittävä pohjamateriaali ja testioraakkeli testien kehitystarkoituksiin, kuten on kuvattu Perustason sertifikaattisisällössä kappaleessa 1.4. Tarvittavan tiedon täytyy olla aikaisin testaajien saatavilla, mutta

muutokset täytyy kuitenkin hyväksyä ketterien periaatteiden mukaisesti. Tämä pulma vaatii huolellisia päätöksiä liittyen testausstrategiaan ja testausdokumentaatioon. Lisää ketterästä testauksesta voi katsoa [Black09] luvusta 12.

Julkaisun ja iteraatioiden suunnittelun pitäisi sisältää testauksen suunnittelua sekä kehitykseen liittyvien tehtävien suunnittelua. Erityisesti testaukseen liittyviin käsiteltäviin asioihin kuuluvat:

- Testauksen laajuus, testauksen määrä valituilla alueilla, testauksen tavoitteet ja syyt edellä mainittuihin päätöksiin.
- Tiimin jäsenet, jotka toteuttavat testauksen tehtäviä.
- Tarvittava testiympäristö ja aineisto, milloin niitä tarvitaan ja onko niihin tulossa lisäyksiä tai muutoksia ennen projektia tai sen aikana.
- Toiminnallisten ja ei-toiminnallisten testaustehtävien ajoitus, järjestys, riippuvuudet ja esiehdot (esim. kuinka tiheään suoritetaan regressiotestausta, mitkä ominaisuudet riippuvat toisista ominaisuuksista tai testiaineistosta, jne.), mukaan lukien kuinka testauksen tehtävät liittyvät ja riippuvat kehityksen tehtävistä.
- Käsiteltävät projekti- ja tuoteriskit (ks. kappale 3.2.1).

Lisäksi laajemman tiimin työmääräarvion pitäisi ottaa huomioon myös vaadittujen testaustehtävien loppuun viemiseen tarvittava aika ja työpanos.

2. Keskeiset ketterän testauksen periaatteet, käytännöt ja prosessit – 105 minuuttia

Avainsanat
koonnin todentamistesti, kokoonpanon osa, kokoonpanonhallinta

Oppimistavoitteet keskeisille ketterän testauksen periaatteille, käytännöille ja prosesseille

2.1 Eroavaisuuksia perinteisten ja ketterien lähestymistapojen testauksessa

FA-2.1.1 (K2) Osaat kuvata testauksen tehtävien eroavaisuudet ketterien ja ei-ketterien projektien välillä
FA-2.1.2 (K2) Osaat kuvata kuinka kehityksen ja testauksen tehtävät muodostavat kokonaisuuden ketterissä projekteissa
FA-2.1.3 (K2) Osaat kuvata riippumattoman testauksen roolin ketterissä projekteissa

2.2 Testauksen asema ketterissä projekteissa

FA-2.2.1 (K2) Osaat kuvata työkaluja ja tekniikoita, joita käytetään ketterien projektien testauksen tilanteen viestimisessä, mukaan lukien testauksen edistyminen ja tuotelaatu
FA-2.2.2 (K2) Osaat kuvata testien kehitysprosessin läpi monien iteraatioiden ja osaat selittää miksi testausautomaatio on tärkeää ketterien projektien regressioriskin hallinnassa

2.3 Testaajan rooli ja taidot ketterässä tiimissä

FA-2.3.1 (K2) Ymmärrät testaajan taidot (ihmiset, toimintapiiri ja testaus) ketterässä tiimissä
FA-2.3.2 (K2) Ymmärrät testaajan roolin ketterässä tiimissä

2.1 Eroavaisuuksia perinteisten ja ketterien lähestymistapojen testauksessa

Kuten Perustason sertifikaattisisällössä [ISTQB_FL_SYL] sekä [Black09] kuvataan, testaustehtävät liittyvät toteutustehtäviin ja näin ollen testaus on erilaista eri elinkaarimalleissa. Testaajien täytyy ymmärtää testauksen eroavaisuudet perinteisten (esim. vaiheittainen malli, kuten v-malli tai iteratiivinen malli, kuten RUP) ja ketterien elinkaarimallien välillä työskennelläkseen tehokkaasti ja suorituskykyisesti. Ketterät projektit eroavat testauksen ja kehityksen tehtävien integroinnin, projektin tuotosten, nimeämisten, eri testaustasoilla käytettävien aloitus- ja lopetuskriteerien sekä työkalujen suhteen ja siinä, kuinka riippumatonta testausta voidaan käyttää tehokkaasti hyödyksi.

Testaajien pitäisi muistaa, että organisaatiot eroavat merkittävästi siinä, kuinka ne toteuttavat malleja. Poikkeaminen ketterien elinkaarimallien ihanteesta (katso kappale 1.1) voi edustaa älykästä käytäntöjen räätälöintiä ja sopeuttamista. Kyky sopeutua määrätyn projektin tilanteeseen ja siinä noudatettuihin ohjelmistokehityksen käytäntöihin on avaintekijä testaajien menestymiseen.

2.1.1 Testauksen ja kehityksen tehtävät

Yksi keskeisimmistä perinteisten ja ketterien elinkaarimallien välisistä eroavaisuuksista on ajatus hyvin lyhyistä iteraatioista, joista jokaisen tuloksena on toimiva ohjelmisto ja jossa on liiketoiminnan edustajille lisäarvoa tarjoavia ominaisuuksia. Projektin alussa on julkaisun suunnittelujakso, jota seuraavat peräkkäiset iteraatiot. Jokaisen iteraation alussa on iteraation suunnitteluvaihe. Kun iteraation puitteet on määritelty, valitut käyttäjätarinat toteutetaan, integroidaan järjestelmään ja testataan. Nämä iteraatiot ovat hyvin dynaamisia, ja toteutus-, integraatio- ja testaustehtäviä tapahtuu läpi jokaisen iteraation ja niissä on paljon rinnakkaisuutta ja päällekkäisyyttä. Testauksen tehtäviä suoritetaan koko ajan iteraation aikana eikä viimeisenä toimintona.

Testaajilla, kehittäjillä sekä liiketoiminnan edustajilla on kaikilla rooli testauksessa, kuten on perinteisissäkin elinaarimalleissa. Kehittäjät suorittavat yksikkötestejä, kun he kehittävät ominaisuuksia käyttäjätarinoista. Sitten testaajat testaavat nuo ominaisuudet. Myös liiketoiminnan edustajat testaavat tarinoita toteuttamisen aikana. Liiketoiminnan edustajat saattavat käyttää kirjoitettuja testitapauksia, mutta he voivat myös yksinkertaisesti kokeilla ja käyttää ominaisuutta antaakseen nopeasti palautetta kehitystiimille.

Joissain tapauksissa toteutetaan säännöllisesti korjaus- tai vakautusiteraatio, jossa pyritään ratkaisemaan pitkään auki olleita vikoja ja muita teknisen velan muotoja. Kuitenkin paras käytäntö on sellainen, että yhtään ominaisuutta ei pidetä valmiina, ennen kuin se on integroitu ja testattu järjestelmässä [Goucher09]. Toinen hyvä käytäntö on keskittyä seuraavan iteraation alussa edellisestä iteraatiosta jäljelle jääneisiin vikoihin osana kyseisen iteraation kehitysjonoa (kutsutaan ”korjaa viat ensin”). Jotkut kuitenkin valittavat, että tämä käytäntö johtaa tilanteeseen, jossa iteraation kokonaistyömäärää ei tiedetä ja on vaikeampi arvioida, milloin jäljellä olevat ominaisuudet voidaan toteuttaa. Peräkkäisten iteraatioiden lopuksi voi olla joukko julkaisuun liittyviä toimintoja, jotta ohjelmisto saadaan toimitusvalmiiksi, vaikka joissakin tapauksissa toimitus tapahtuu aina jokaisen iteraation lopuksi.

Kun käytetään riskipohjaista testausta yhtenä testausstrategiana, korkean tason riskianalyysi tapahtuu julkaisun suunnittelun aikana ja usein testaajat ohjaavat tämän analyysin. Jokaiseen iteraatioon liittyvät sille ominaiset laaturiskit tunnistetaan ja arvioidaan kuitenkin iteraation suunnittelun yhteydessä. Tämä riskianalyysi voi vaikuttaa kehitysjärjestykseen kuin myös ominaisuuksien testaamisen tärkeyteen ja syvyyteen. Se vaikuttaa myös jokaiseen ominaisuuteen tarvittavan testaustyön arviointiin (katso kappale 3.2).

Joissakin ketterissä käytännöissä (esim. XP) käytetään parityöskentelyä. Siihen voi kuulua paritestausta, jossa kaksi testaajaa yhdessä testaa ominaisuutta. Siihen voi myös kuulua myös se, että

testaaja työskentelee yhteistyössä kehittäjän kanssa ominaisuuden kehittämisessä ja testauksessa. Parityöskentely voi olla vaikeaa silloin, kun testaustiimi on hajautettu, mutta prosessit ja työkalut voivat helpottaa hajautettua parityöskentelyä (katso [ISTQB_ALTM_SYL], kappale 2.8).

Testaajat voivat toimia myös tiimin testaus- ja laatuvalmentajina ja jakaa testausosaamista ja tukea laadunvarmistustyötä tiimissä. Tämä edistää tuntemusta tuotteen laadun yhteisestä omistajuudesta.

Useissa ketterissä tiimeissä tapahtuu testausautomatisointia kaikilla testauksen tasoilla ja tämä voi tarkoittaa sitä, että testaajat käyttävät aikaa automatisoitujen testien luomiseen ja suorittamiseen sekä testien ja tulosten seurantaan ja ylläpitämiseen. Testiautomaation laajan käytön vuoksi suurempi osa ketterissä projekteissa tehtävästä manuaalisesta testauksesta tehdään yleensä käyttämällä kokemusperusteisia ja vikaperusteisia tekniikoita kuten ohjelmistohyökkäyksiä, tutkivaa testausta ja virheenarvausta (katso [ISTQB_ALTA_SYL], osat 3.3 ja 3.4 ja [ISTQB_FL_SYL], kappale 4.5). Kehittäjien keskittyessä luomaan yksikkötestejä, testaajien pitäisi keskittyä luomaan automatisoituja integraatio-, järjestelmä- ja järjestelmäintegraatiotestejä. Tämä johtaa ketterissä tiimeissä taipumukseen suosia testaajia, joilla on vahva tekninen ja testausautomaatiotausta.

Yksi ketteristä perusperiaatteista on se, että muutoksia voi tapahtua koko projektin ajan. Siksi ketterissä projekteissa suositaan hyvin kevyttä projektin työn tulosten dokumentointia. Olemassa olevien ominaisuuksien muutoksilla on vaikutuksia testaukseen ja erityisesti regressiotestaukseen. Automaattisen testauksen käyttäminen on yksi tapa hallita muutokseen liittyvän testaustyön määrää. Kuitenkin on tärkeää, että muutoksen määrä ei ylitä projektitiimin kykyä käsitellä muutoksiin liittyviä riskejä.

2.1.2 Projektin työn tulokset

Projektin työn tulokset, jotka ovat välitön mielenkiinnon kohde ketterille testaajille, jaetaan tyypillisesti kolmeen kategoriaan:

1. Liiketoimintapainotteiset työn tulokset, jotka kuvaavat sen, mitä tarvitaan (esim. vaatimusmäärittelyt) ja kuinka sitä käytetään (esim. käyttäjädokumentaatio)
2. Kehitystyön tulokset, jotka kuvaavat sen, miten järjestelmä rakennetaan (esim. tietokannan entiteettisuhdekaaviot), jotka itse asiassa toteuttavat järjestelmän (esim. koodi), tai jotka arvioivat yksittäistä koodin osaa (esim. manuaalinen ja automaattinen testaus)
3. Testaustyön tulokset, jotka kuvaavat sen, kuinka järjestelmä testataan (esim. testausstrategiat ja suunnitelmat), joilla testataan järjestelmä (esim. manuaaliset ja automaattiset testit), tai jotka esittävät testaustuloksia (esim. testauksen näkymä, josta puhutaan osassa 2.2.1)

Tyypillisessä ketterässä projektissa yleinen käytäntö on välttää tuottamasta hyvin suurta määrää dokumentaatiota. Sen sijaan keskitytään enemmän toimivaan ohjelmistoon yhdessä automaattisten testien kanssa, jotka osoittavat yhdenmukaisuuden vaatimuksiin. Tämä rohkaisu dokumentoinnin vähentämiseen koskee ainoastaan sellaista dokumentointia, joka ei tuo lisäarvoa asiakkaalle. Menestyvässä ketterässä projektissa tasapaino löytyy jostain siltä väliltä, että vähennetään dokumentaatiota tehokkuuden kasvattamiseksi ja että laaditaan riittävästi dokumentaatiota liiketoimintaan, testaukseen, toteutukseen ja ylläpitoon liittyvien tehtävien tukemiseksi. Tiimin täytyy tehdä julkaisun suunnittelun aikana päätös siitä, mitä työn tuloksia tarvitaan ja millä tasolla niitä dokumentoidaan.

Tyypilliset liiketoimintaorientoituneet työn tulokset ketterissä projekteissa käsittävät käyttäjätarinoita ja hyväksymiskriteerit. Käyttäjätarinat ovat ketterä muoto vaatimusmäärittelyistä, ja niiden pitäisi selittää se, kuinka järjestelmän pitäisi käyttäytyä yksittäisen, yhtenäisen ominaisuuden tai toiminnon suhteen. Käyttäjätarinan pitäisi määritellä tarpeeksi pieni ominaisuus, joka voidaan toteuttaa valmiiksi yksittäisessä iteraatiossa. Laajempaa kokoelmaa toisiinsa liittyviä ominaisuuksia tai kokoelmaa aliominaisuuksia, jotka muodostavat yksittäisen monimutkaisen ominaisuuden, voidaan kutsua eepoksiksi. Eepokset voivat sisältää käyttäjätarinoita eri kehitystiimeille. Esimerkiksi yksi käyttäjätarina

voi kuvata sen, mitä tarvitaan API-tasolla (väliohjelmisto) kun taas toinen voi kuvata sen, mitä tarvitaan käyttöliittymätasolla (sovellus). Nämä kokoelmat voidaan kehittää useiden sprinttien aikana. Jokaisella eepoksella ja sen käyttäjätarinoilla pitäisi olla hyväksymiskriteerit.

Tyypillisiin kehitystyön tuloksiin ketterissä projekteissa kuuluu koodi. Ketterät kehittäjät luovat myös usein automaattisia testejä. Nämä testit saatetaan luoda koodin kehittämisen jälkeen. Joissakin tapauksissa kehittäjät kuitenkin luovat testejä vaiheittain jo ennen kunkin koodinosan toteuttamista ja luovat näin tavan varmistaa, että kun kyseinen osa toteutettu, se toimii tarkoitetulla tavalla. Vaikka tätä lähestymistapaa kutsutaan “testaa ensin” tai testiohjatuksi kehitykseksi, todellisuudessa nämä testit ovat enemmänkin suoritettavaan muotoon kirjoitettujen matalan tason suunnittelukuvausten kuin testien muoto [Beck02].

Tyypillisiin testaajan tuotoksiin ketterässä projektissa kuuluvat automatisoidut testit sekä erilaiset dokumentit, kuten testaussuunnitelmat, laaturiskiluettelot, manuaaliset testit, vikaraportit ja testilokit. Dokumentit pidetään niin kevyenä kuin mahdollista, mikä pätee näihin dokumentteihin usein myös perinteisissä elinkaarimalleissa. Testaajat tuottavat myös testimetriikkaa perustuen vikaraportteihin ja testituloslokeihin, edelleenkin painottaen kevyttä lähestymistapaa dokumentointiin.

Joissakin ketterissä toteutuksissa, erityisesti säädellyissä, turvallisuuskriittisissä, hajautetuissa, tai hyvin monimutkaisissa projekteissa tai tuotteissa, tarvitaan näiden työn tulosten lisäformalisointia. Esimerkiksi jotkut tiimit muuntavat käyttäjätarinoita ja hyväksymiskriteerejä formaalimmiksi vaatimusmäärittelyiksi. Vertikaalisia ja horisontaalisia jäljitettävyysraportteja voidaan tehdä, jotta vakuutetaan auditoijia, säädöksiä ja muita vaatimuksia.

2.1.3 Testitasot

Testitasot ovat testauksen tehtäviä, jotka liittyvät loogisesti toisiinsa, usein testattavan asian kypsyyden ja valmiuden osalta.

Perättäisissä elinkaarimalleissa testitasot on usein määritelty siten, että jonkin tason lopetuskriteerit ovat osa seuraavan tason aloituskriteerejä. Joissakin iteratiivisissa malleissa tämä sääntö ei päde ja testitasot limittyvät. Vaatimusmäärittely, suunnittelun määrittely ja kehitystehtävät voivat limittyä testitasojen kanssa.

Joissakin ketterissä elinkaarimalleissa limittymistä tapahtuu, koska muutoksia vaatimuksiin, suunnitteluun ja koodiin voi tapahtua missä iteraation vaiheessa tahansa. Vaikka scrum ei teoriassa salli muutoksia käyttäjätarinoihin iteraation suunnittelun jälkeen, käytännössä joskus niin tapahtuu. Iteraation aikana mikä tahansa tietty käyttäjätarina kehittyy tyypillisesti vaiheittain seuraavien testaustehtävien aikana:

- Yksikkötestaus, tyypillisesti kehittäjän suorittama
- Ominaisuuden hyväksymistestaus, joka joskus jaetaan kahteen tehtävään:
    - Ominaisuuden todentamistestaus, joka on usein automatisoitu, voidaan tehdä joko kehittäjien tai testaajien toimesta ja siinä testataan käyttäjätarinan hyväksymiskriteerejä vasten
    - Ominaisuuden kelpuutustestaus, joka on usein manuaalista ja siinä voi olla mukana kehittäjiä, testaajia ja liiketoiminnan edustajia, jotka toimivat yhteistyössä määrittääkseen, onko ominaisuus sopiva käyttöön, parantaakseen näkyvyyttä edistymisen suhteen ja saadakseen aitoa palautetta liiketoiminnan sidosryhmien edustajilta.

Lisäksi usein iteraatiossa on jatkuva rinnakkainen regressiotestausprosessi. Tämä sisältää automaattisten yksikkötestien ja ominaisuuksien todentamistestien uudelleenajamista nykyisestä ja aiemmista iteraatioista tavallisesti jatkuvan integraation puitteissa.

Joissakin ketterissä projekteissa voi olla järjestelmätestaustaso, joka käynnistyy, kun ensimmäinen käyttäjätarina on valmis kyseiseen testaukseen. Tämä voi sisältää toiminnallisten testien ajamista kuin myös ei-toiminnallisia testejä liittyen suorituskykyyn, luotettavuuteen, käytettävyyteen, sekä muita asiaankuuluvia testityyppejä.

Ketterät tiimit voivat käyttää usein hyväksymistestauksen eri muotoja (tässä käytetään termiä, joka on selitetty Perustason sertifikaattisisällössä [ISTQB_FL_SYL]). Sisäisiä alfatestejä ja ulkoisia betatestejä voidaan suorittaa joko kunkin iteraation päättymisen lähestyessä, iteraation valmistumisen jälkeen tai iteraatioiden sarjan jälkeen. Myös käyttäjän hyväksymistestejä, käyttöön soveltuvuuden hyväksymistestejä, säännöksiin perustuvia hyväksymistestejä sekä sopimuksiin perustuvia hyväksymistestejä voidaan tehdä lähellä kunkin iteraation päättymistä, iteraation valmistumisen jälkeen, tai iteraatioiden sarjan jälkeen.

2.1.4 Testaus ja kokoonpanonhallinta

Ketteriin projekteihin sisältyy usein automatisointityökalujen laajaa käyttöä kehityksessä, testauksessa ja ohjelmistokehityksen hallinnassa. Kehittäjät käyttävät työkaluja staattiseen analyysiin, yksikkötestaukseen ja koodikattavuuteen. Kehittäjät kirjaavat jatkuvasti koodia ja yksikkötestejä kokoonpanonhallintajärjestelmään automatisoitujen koonti- ja testikehystyökalujen avulla. Nämä kehystyökalut mahdollistavat uuden ohjelmiston integroinnin järjestelmään niin, että staattinen analyysi ja yksikkötestit toistetaan aina, kun uutta ohjelmistoa kirjataan sisään [Kubaczkowski].

Nämä automaattiset testit voivat myös sisältää toiminnallisia automatisoituja testejä integraatio- ja järjestelmätasoilla. Tällaisia toiminnallisia testejä voidaan luoda käyttämällä toiminnallisen testauksen testipetejä, avoimen lähdekoodin käyttöliittymän toiminnallisen testauksen työkaluja tai kaupallisia työkaluja. Nämä testit voidaan integroida automaattisiin testeihin, jotka ajetaan osana jatkuvaa integraatiota. Joissakin tapauksissa johtuen toiminnallisten testien kestosta, toiminnalliset testit erotetaan yksikkötesteistä ja ne voidaan ajaa harvemmin. Esimerkiksi yksikkötestit ajetaan aina silloin, kun uusi ohjelmisto kirjataan sisään, kun taas pidemmät toiminnalliset testit ajetaan ainoastaan muutaman päivän välein.

Yksi automaattisten testien tavoite on varmistaa se, että koonti toimii ja on asennettavissa. Jos yksikin automaattinen testi ei mene läpi, tiimin pitäisi korjata taustalla oleva vika ennen seuraavaa koodin sisään kirjaamista. Tämä edellyttää investointia reaaliaikaiseen testausraportointiin, jotta saadaan hyvä näkyvyys testituloksiin. Tämä lähestymistapa auttaa vähentämään kallista ja tehotonta sykliä “Koonti-asennus-epäonnistuminen-uudelleenkoonti-uudelleenasennus”, jota voi tapahtua useissa perinteisissä projekteissa, sillä koonnin rikkovat tai asennuksen epäonnistumisen aiheuttavat muutokset havaitaan nopeasti.

Automaattiset testaus- ja koontityökalut auttavat hallitsemaan monesti ketterissä projekteissa esiintyvää tiheisiin muutoksiin liittyvää regressioriskiä. Kuitenkin liika luottaminen siihen, että automaattinen yksikkötestaus yksistään hallitsee näitä riskejä, voi olla ongelma, koska yksikkötestien vianlöytötehokkuus on usein rajallinen [Jones11]. Automatisoituja testejä tarvitaan myös järjestelmä- ja integraatiotasolle.

2.1.5 Riippumattoman testauksen organisatoriset vaihtoehdot

Kuten Perustason sertifikaattisisällössä [ISTQB_FL_SYL] käsiteltiin, riippumattomat testaajat ovat usein tehokkaampia löytämään vikoja. Joissakin ketterissä tiimeissä toteuttajat luovat monet testeistä automatisoituina testeinä. Tiimiin voi sisältyä yksi tai useampi testaaja, jotka suorittavat monia testauksen tehtävistä. Näiden testaajien asema tiimissä luo kuitenkin riippumattomuuden ja objektiivisen arvioinnin menettämisen riskin.

Toiset ketterät tiimit käyttävät täysin riippumattomia, erillisiä testaustiimejä ja työllistävät testaajia tarpeen mukaan joka sprintin viimeisinä päivinä. Tämä voi säilyttää riippumattomuuden ja nämä testaajat voivat antaa ohjelmistosta objektiivisen, puolueettoman arvion. Kuitenkin aikataulupaineet, tuotteen uusien ominaisuuksien ymmärryksen puute sekä ihmissuhdeasiat liiketoiminnan edustajien ja kehittäjien kanssa johtavat usein ongelmiin tätä lähestymistapaa käytettäessä.

Kolmas vaihtoehto on käyttää riippumatonta erillistä testaustiimiä, josta testaajat tulevat työskentelemään pidempiaikaisesti ketterässä tiimissä heti projektin alusta lähtien, mikä sallii heidän säilyttää riippumattomuutensa, mutta samaan aikaan suo heille mahdollisuuden hankkia hyvä käsitys tuotteesta ja luoda vahvat suhteet muihin tiimin jäseniin. Lisäksi riippumattomassa testaustiimissä voi olla ketterän tiimin ulkopuolisia testauksen erityisosaajia, jotka voivat työskennellä pidempiaikaisesti ja/tai iteraatio-riippumattomissa tehtävissä, kuten esim. automatisoitujen testityökalujen kehittäminen, ei-toiminnallisen testauksen suoritus, testiympäristön ja -aineiston luominen ja tuki sekä sellaisten testitasojen suorittaminen, jotka eivät sovi hyvin sprinttiin (esim. järjestelmäintegraatiotestaus).

2.2 Testauksen asema ketterissä projekteissa

Ketterissä projekteissa muutoksia tapahtuu nopeasti. Tämä tarkoittaa sitä, että testauksen tilanne, testauksen edistyminen sekä tuotteen laatu kehittyvät jatkuvasti ja testaajien täytyy suunnitella tapoja saada tämä tieto tiimille, jotta he voivat tehdä päätöksiä onnistuakseen jokaisen iteraation toteuttamisessa. Lisäksi muutos voi vaikuttaa edellisistä iteraatiosta lähtöisin oleviin ominaisuuksiin. Sen vuoksi manuaalisia ja automaattisia testejä täytyy päivittää, jotta ne käsittelevät regressioriskin mahdollisimman tehokkaasti.

2.2.1 Testauksen tilanteen, edistymisen ja tuotelaadun kommunikointi

Ketterät tiimit etenevät tuottamalla joka iteraation loppuun mennessä toimivan ohjelmiston osan. Määrittääkseen sen, koska tiimillä on toimiva ohjelmisto, heidän täytyy valvoa kaikkien työtehtävien edistymistä iteraatiossa ja julkaisussa. Ketterien tiimien testaajat hyödyntävät erilaisia menetelmiä tallentaakseen testauksen edistymisen ja testauksen tilanteen mukaan lukien testausautomaation tulokset sekä tehtävätaululla olevien testaustehtävien ja tarinoiden edistyminen ja tiimin edistymisen näyttävät edistymiskäyrät. Nämä voidaan välittää muulle tiimille käyttäen erilaisia viestintämuotoja kuten wikikojelautaa tai wikityyppisiä sähköposteja, tai ne voidaan viestiä suullisesti seisoen pidettävissä päiväpalavereissa. Ketterät tiimit voivat käyttää työkaluja, jotka luovat automaattisesti tilanneraportteja testitulosten ja tehtävien edistymisen perusteella, ja nämä työkalut vuorostaan päivittävät wiki-tyyppisiä kojelautoja ja sähköposteja. Tämä kommunikointitapa kerää myös testausprosessista metriikkaa, jota voidaan käyttää prosessin kehittämiseen. Testauksen tilanteen viestiminen tällaisella automatisoidulla tavalla vapauttaa myös testaajien aikaa useampien testien suunnitteluun ja testitapausten ajamiseen.

Tiimit voivat käyttää edistymiskäyriä seuratakseen koko julkaisun ja iteraation aikaista edistymistä. Edistymiskäyrä [Crispin08] esittää jäljellä olevan työn määrää suhteessa julkaisuun tai iteraatioon varattuun aikaan.

Tarjotakseen välittömän ja yksityiskohtaisen esityksen koko tiimin nykytilanteesta mukaan lukien testauksen tilanne, tiimit voivat käyttää ketteriä tehtävätauluja. Tarinakortit, kehitystehtävät, testaustehtävät ja muut iteraation suunnittelun aikana luodut tehtävät (katso kappale1.2.5) esitetään tehtävätaululla käyttäen usein värillisiä kortteja, jotka esittävät eri tehtävätyyppejä. Iteraation aikana edistyminen hallitaan näiden tehtävien liikkumisella tehtävätaulun sarakkeissa kuten *tehtävänä*, *työn alla*, *todennettavana* ja *valmis*. Ketterät tiimit voivat käyttää työkaluja ylläpitääkseen tarinakortteja ja ketteriä tehtäväkortteja, jotka voivat automatisoida kojelautoja ja tilannepäivityksiä.

Tehtävätaululla olevat testaustehtävät liittyvät käyttäjätarinoille määriteltyihin hyväksymiskriteereihin. Kun testaustehtävän testiautomaatioskriptit, manuaaliset testit ja tutkivat testit on suoritettu hyväksytysti,

tehtävä liikkuu tehtävätaulun *valmis* -sarakkeeseen. Koko tiimi katselmoi tehtävätaulun tilanteen säännöllisesti, usein päiväpalaverien aikana varmistaakseen sen, että tehtävät liikkuvat taululla hyväksyttävällä vauhdilla. Jos jotkut tehtävät (testaustehtävät mukaan luettuna) eivät liiku tai liikkuvat liian hitaasti, tiimi katselmoi ja käsittelee kaikki ongelmat, jotka voivat estää kyseisten tehtävien edistymisen.

Päiväpalaveriin osallistuvat kaikki tiimin jäsenet testaajat mukaan lukien. Tässä tapaamisessa he kertovat nykyisen tilanteensa. Agenda jokaiselle osallistujalle on [Agile Alliance Guide]:

- Mitä olet saanut valmiiksi edellisen tapaamisen jälkeen?
- Mitä suunnittelet saavasi valmiiksi seuraavaan palaveriin mennessä?
- Mitä esteitä sinulla on?

Mikä tahansa ongelma, joka voi estää testauksen edistymisen, tuodaan esiin päiväpalaverin aikana. Näin koko tiimi on tietoinen ongelmista ja voi ratkaista ne sopivalla tavalla.

Parantaakseen tuotteen kokonaislaatua monet ketterät tiimit suorittavat asiakastyytyväisyyskyselyjä saadakseen palautetta siitä, täyttääkö tuote asiakasodotukset. Tiimit voivat tuotelaatua parantaakseen käyttää muita metriikoita, jotka ovat vastaavia kuin mitä käytetään perinteisissä kehitysprojekteissa, kuten testien läpäisyaste, vikojen löytämisaste, varmistus- ja regressiotestauksen tulokset, vikatiheys, löydetyt ja korjatut viat, vaatimuskattavuus, riskikattavuus, koodikattavuus ja koodimuutokset. Kuten missä tahansa elinkaarimallissa, kerättyjen ja raportoitujen metriikoiden pitäisi olla oleellisia ja niiden pitäisi tukea päätöksentekoa. Metriikoita ei saisi käyttää tiimin jäsenten palkitsemiseen, rankaisemiseen tai eristämiseen.

2.2.2 Regressioriskin hallinta manuaalisten ja automatisoitujen testitapausten kehittämisen avulla

Ketterissä projekteissa tuote kasvaa aina jokaisen iteraation valmistuessa. Siitä syystä myös testauksen laajuus kasvaa. Sen hetkisessä iteraatiossa tehtyjen koodimuutosten testaamisen lisäksi testaajien tarvitsee myös varmistaa se, että regressiota ei ole päässyt syntymään edellisissä iteraatioissa kehitettyihin ja testattuihin ominaisuuksiin. Ketterässä kehityksessä regression syntymisen riski on korkea laajojen koodimuutosten vuoksi (versiosta toiseen tapahtuva koodirivien lisääminen, muokkaus tai poistaminen). Koska muutokseen vastaaminen on yksi ketteristä avainperiaatteista, voidaan muutoksia tehdä aiemmin toimitettuihin versioihin liiketoiminnan vaatimusten täyttämiseksi. Jotta voidaan pitää yllä vauhtia hankkimatta suurta määrää teknistä velkaa, on kriittistä, että tiimit investoivat mahdollisimman varhain testausautomaatioon kaikilla testitasoilla. On myös kriittistä, että kaikki testauksen avut kuten automatisoidut testit, manuaaliset testitapaukset, testiaineisto sekä muut testauksen tuotokset pidetään ajan tasalla jokaisessa iteraatiossa. On erittäin suositeltavaa, että kaikkea testimateriaalia ylläpidetään kokoonpanonhallintatyökalussa, jotta mahdollistetaan versionhallinta, varmistetaan saatavuuden helppous kaikille tiimin jäsenille ja tuetaan muuttuvan toiminnallisuuden vaatimia muutoksia mutta silti säilytetään testimateriaalin historiatiedot.

Koska täydellinen, kaikkien testien toistaminen on harvoin mahdollista erityisesti tiukan aikataulun omaavissa ketterissä projekteissa, testaajien täytyy joka iteraatiossa varata aikaa aiempien ja nykyisen iteraation manuaalisten ja automatisoitujen testitapausten katselmointiin regressiotestijoukkoon ehdotettavien testitapausten valitsemiseksi sekä sellaisten testitapausten poistamiseksi, jotka eivät enää ole tarpeellisia. Määrättyjen ominaisuuksien todentamiseksi aikaisemmissa iteraatioissa laadituilla testitapauksilla voi olla myöhemmissä iteraatioissa vain vähäistä arvoa ominaisuuksien muutosten vuoksi tai siksi, että uudet ominaisuudet muuttavat kyseisten ominaisuuksien aiempaa käyttäytymistä.

Testitapauksien katselmoinnissa testaajien pitäisi pohtia testitapausten soveltuvuutta testausautomaatioon. Tiimin täytyy automatisoida aiemmista sekä nykyisestä iteraatiosta niin monta testiä, kuin on mahdollista. Tämä mahdollistaa sen, että automatisoidut regressiotestit pienentävät regressioriskiä pienemmällä työllä, kuin mitä manuaalinen testaus vaatisi. Tämä pienempi

regressiotestauksen työpanos vapauttaa testaajat testaamaan perusteellisemmin nykyisen iteraation uusia ominaisuuksia ja toimintoja.

On kriittistä, että testaajilla on mahdollisuus nopeasti identifioida ja päivittää aiempien iteraatioiden ja/tai julkaisujen testitapauksia, joihin nykyisen iteraation muutokset vaikuttavat. Sen määrittäminen, kuinka tiimi suunnittelee, kirjoittaa ja tallentaa testitapauksia, pitäisi tehdä julkaisun suunnittelun aikana. Hyvät testaussuunnittelun ja toteuttamisen käytännöt täytyy omaksua varhain ja niitä pitää käyttää johdonmukaisesti. Lyhempi testausaika ja jatkuva muutos jokaisessa iteraatiossa lisäävät huonojen testisuunnittelu- ja toteutuskäytäntöjen vaikutusta.

Testausautomaation käyttö kaikilla testitasoilla mahdollistaa sen, että ketterät tiimit tuottavat tuotteen laadusta nopeasti palautetta. Hyvin kirjoitetut automaattiset testit tarjoavat elävän dokumentaation järjestelmän toiminnasta [Crispin08]. Kirjaamalla automatisoidut testit ja niiden tulokset kokoonpanonhallintajärjestelmään yhdessä versioinnin ja tuotteen koontien kanssa, ketterät tiimit voivat katselmoida minkä tahansa koonnin testatun toiminnan ja testitulokset minä ajanhetkenä hyvänsä.

Automatisoidut yksikkötestit ajetaan ennen kuin lähdekoodi on kirjattu sisään kokoonpanonhallinnan päälinjalle ja niillä varmistetaan se, että koodimuutokset eivät riko ohjelmiston koontia. Koko tiimin edistymistä haittaavien koontirikkoutumisten vähentämiseksi koodia ei pitäisi kirjata sisään, ennen kuin kaikki automatisoidut testit on läpäisty. Automatisoitujen hyväksymistestien tulokset antavat palautetta tuotteen laadusta edellisen koonnin jälkeen tapahtuneen regression suhteen, mutta ne eivät kerro tuotteen kokonaislaadun tilaa.

Automatisoidut hyväksymistestit ajetaan säännöllisesti osana jatkuvaa integraatiota koko järjestelmän koonnissa. Nämä testit ajetaan vasten täyttä järjestelmäkoontia ainakin kerran päivässä, mutta yleensä niitä ei ajeta jokaisen koodin sisään kirjaamisen yhteydessä, koska niiden ajaminen kestää pitempään kuin automatisoitujen yksikkötestien ajaminen. Näin ollen ne voivat hidastaa koodin kirjaamisia sisään. Automatisoitujen hyväksymistestien tulokset tuottavat välitöntä palautetta tuotelaadusta regression osalta edellisestä koonnista lähtien, mutta niiden avulla ei ole mahdollista saada tilannetta tuotteen kokonaislaadusta.

Automatisoidut testit voidaan ajaa jatkuvasti järjestelmää vasten. Ensimmäinen osajoukko automatisoituja testejä, jotka kattavat järjestelmän kriittisen toiminnallisuuden ja integrointipisteet, pitäisi luoda heti sen jälkeen, kun uusi koonti on siirretty testiympäristöön. Nämä testit tunnetaan yleisesti koonnin todentamistesteinä. Koonnin todentamistestien tulokset mahdollistavat välittömän palautteen ohjelmistosta siirtämisen jälkeen, joten tiimit eivät hukkaa aikaa epävakaaseen koontiin.

Automatisoidut regressiotestijoukkoon kuuluvat testit ajetaan yleensä osana päivittäistä pääkoontia jatkuvan integraation ympäristössä ja uudestaan siinä vaiheessa, kun uusi koonti on siirretty testausympäristöön. Heti kun automatisoitu regressiotesti ei mene läpi, tiimi pysähtyy ja tutkii syyn epäonnistuneeseen testiin. Se, että testi ei mennyt läpi, voi johtua tarkoitetuista muutoksista nykyisen iteraation toiminnassa ja siinä tapauksessa testi ja/tai käyttäjätarina on ehkä päivitettävä vastaamaan uusia hyväksymiskriteereitä. Vaihtoehtoisesti testi on ehkä poistettava käytöstä, jos toinen testi on kehitetty kattamaan muutokset. Jos testi ei kuitenkaan mennyt läpi vian vuoksi, on hyvän käytännön mukaista korjata vika ennen etenemistä uusien ominaisuuksien kanssa.

Testausautomaation lisäksi seuraavat testaustehtävät voi olla myös hyvä automatisoida:

- Testiaineiston luonti
- Testiaineiston lataaminen järjestelmiin
- Koontien siirtäminen testiympäristöihin
- Testiympäristön palauttaminen (esim. tietokanta tai verkkosivuston tiedostot) lähtökohtaan
- Tulostietojen vertailu

Näiden tehtävien automatisointi vähentää kustannuksia ja luo tiimille mahdollisuuden käyttää aikaa uusien ominaisuuksien kehittämiseen ja testaamiseen.

2.3 Testaajan rooli ja taidot ketterässä tiimissä

Ketterässä tiimissä testaajien täytyy työskennellä läheisessä yhteistyössä muiden tiimin jäsenten ja liiketoiminnan edustajien kanssa. Tällä on lukuisia vaikutuksia niihin taitoihin, joita testaajalla pitää olla ja siihen, mitä tehtäviä he suorittavat ketterässä tiimissä.

2.3.1 Ketterän testaajan taidot

Ketterillä testaajilla pitäisi olla kaikki taidot, jotka on mainittu Perustason sertifikaattisisällössä [ISTQB_FL_SYL]. Näiden taitojen lisäksi ketterän tiimin testaajan pitää olla pätevä testausautomaatiossa, testiohjatussa kehityksessä, hyväksymistestiohjatussa kehityksessä, lasilaatikkotestauksessa, mustalaatikkotestauksessa ja kokemusperäisessä testauksessa.

Kun ketterät menetelmät riippuvat vahvasti yhteistyöstä, kommunikaatiosta ja vuorovaikutuksesta sekä tiimin jäsenten että tiimin ulkopuolisten sidosryhmien välillä, ketterissä tiimeissä testaajilla pitäisi olla hyvät ihmissuhdetaidot. Ketterissä tiimeissä testaajien pitäisi:

- Olla positiivisia ja ratkaisukeskeisiä yhdessä tiimin jäsenten ja sidosryhmien kanssa
- Osoittaa kriittistä, laatuorientoitunutta ja skeptistä ajattelua koskien tuotetta
- Kerätä aktiivisesti tietoa sidosryhmiltä (pelkästään vain kirjallisten määrittelyiden varaan jättäytymisen sijaan)
- Arvioida ja raportoida tarkasti testituloksia, testauksen edistymistä ja tuotelaatua
- Työskennellä tehokkaasti asiakkaan edustajien ja sidosryhmien kanssa testattavien käyttäjätarinoiden ja erityisesti hyväksymiskriteerien määrittämiseksi
- Tehdä yhteistyötä tiimissä työskentelemällä pareittain ohjelmoijien sekä muiden tiimin jäsenten kanssa
- Vastata muutokseen nopeasti mukaan lukien testitapausten muuttaminen, lisääminen ja parantaminen
- Suunnitella ja organisoida omaa työtänsä

Jatkuva taitojen (myös ihmissuhdetaitojen) kehittäminen on olennaista kaikille testaajille mukaan lukien testaajat ketterissä tiimeissä.

2.3.2 Testaajan rooli ketterässä tiimissä

Testaajan rooliin ketterässä tiimissä sisältyy tehtäviä, jotka generoivat ja tuottavat palautetta ei vain testauksen tilanteesta, testauksen edistymisestä ja tuotelaadusta vaan myös prosessin laadusta. Näiden, tässä sertifikaattisisällössä muualla kuvattujen tehtävien lisäksi näihin tehtäviin kuuluvat:

- Testausstrategian ymmärtäminen, toteuttaminen ja päivittäminen
- Testauskattavuuden mittaaminen ja raportointi kaikkien soveltuvien kattavuuslajien osalta
- Testaustyökalujen oikeanlaisen käytön varmistaminen
- Testausympäristöjen ja -aineiston konfigurointi, käyttäminen ja hallinnointi
- Vikojen raportointi ja tiimin kanssa työskentely niiden ratkaisemiseksi
- Muiden tiimin jäsenten valmentaminen testauksen olennaisilla osa-alueilla
- Sen takaaminen, että oikeanlaiset testauksen tehtävät aikataulutetaan julkaisun ja iteraation suunnittelun aikana
- Aktiivinen yhteistyö toteuttajien ja liiketoiminnan sidosryhmien kanssa vaatimusten selventämiseksi erityisesti testattavuuden, yhdenmukaisuuden ja valmiuden suhteen
- Osallistuminen proaktiivisesti tiimin retrospektiiveihin, ehdottamalla ja toteuttamalla parannuksia.

Ketterässä tiimissä jokainen tiimin jäsen on vastuussa tuotelaadusta ja jokaisella on rooli testaukseen liittyvien tehtävien suorittamisessa.

Ketterät organisaatiot voivat kohdata muutamia testaukseen liittyviä organisatorisia riskejä:

- Testaajat työskentelevät kehittäjien kanssa niin läheisessä yhteistyössä, että he menettävät oikeanlaisen testaajan ajattelutavan
- Testaajat alkavat sietää sisäisesti tai yleisesti tehottomia tai heikkolaatuisia toimintatapoja tiimissä tai pysyvät niistä hiljaa
- Testaajat eivät pysy saapuvien muutosten tahdissa aikarajoitteisissa iteraatioissa.

Lieventääkseen näitä riskejä organisaatiot voivat harkita vaihtelua säilyttääkseen riippumattomuuden, kuten keskusteltiin osassa 2.1.5.

3. Ketterät testausmenetelmät, tekniikat ja työkalut – 480 minuuttia

Avainsanat
hyväksymiskriteerit, tutkiva testaus, suorituskykytestaus, tuoteriski, laaturiski, regressiotestaus, testauksen lähestymistapa, testausohje, testauksen työmäärän arviointi, testin suoritusautomaatio, testausstrategia, testiohjattu kehitys, yksikkötestauskehys

Oppimistavoitteet ketterille testausmenetelmille, tekniikoille ja työkaluille

3.1 Ketterät testausmenetelmät

FA-3.1.1 (K1) Muistat testiohjatun kehityksen, hyväksymistestiohjatun kehityksen sekä käyttäytymisperustaisen kehityksen konseptit
FA-3.1.2 (K1) Muistat testipyramidin konseptin
FA-3.1.3 (K2) Osaat esittää testausneljännekset ja niiden yhteyden testitasoihin ja -tyyppeihin
FA-3.1.4 (K3) Osaat toimia testaajan roolissa scrumtiimissä

3.2 Laaturiskien arvioiminen ja testauksen työmäärän arviointi

FA-3.2.1 (K3) Osaat arvioida laaturiskejä ketterässä projektissa
FA-3.2.2 (K3) Osaat arvioida testauksen työmäärää iteraation sisällön ja laaturiskien perusteella

3.3 Ketterien projektien tekniikoita

FA-3.3.1 (K3) Osaat tulkita oikeanlaista tietoa testauksen tehtävien tukemiseksi
FA-3.3.2 (K2) Osaat selittää liiketoiminnan edustajille, kuinka määritetään testattava hyväksymiskriteeri
FA-3.3.3 (K3) Osaat kirjoittaa testitapauksia käyttäjätarinalle hyväksymistestiohjatussa kehityksessä
FA-3.3.4 (K3) Osaat kirjoittaa testitapauksia sekä toiminnalliselle että ei-toiminnalliselle käyttäytymiselle perustuen annettuihin käyttäjätarinoihin käyttäen mustalaatikkosuunnittelutekniikoita
FA-3.3.5 (K3) Osaat suorittaa tutkivaa testausta ketterässä projektissa

3.4 Ketterien projektien työkaluja

FA-3.4.1 (K1) Muistat saatavilla olevia testauksen työkaluja niiden tarkoituksen ja toimintojen mukaan ketterissä projekteissa

3.1 Ketterät testausmenetelmät

On olemassa määrättyjä testauskäytäntöjä, joita voidaan noudattaa kaikissa kehitysprojekteissa (olivatpa ne ketteriä tai ei) laadukkaiden tuotteiden tuottamiseksi. Niihin kuuluvat etukäteen tehtävä testitapauksien kirjoittaminen kuvaamaan oikeanlaista toimintaa ja ne keskittyvät aikaisin tapahtuvaan vikojen ehkäisyyn, havaitsemiseen ja poistamiseen, sekä sen varmistamiseen, että oikeat testityypit suoritetaan oikeaan aikaan osana oikeaa testitasoa. Ketterien toimijoiden tavoitteena on aloittaa nämä käytännöt aikaisin. Ketterien projektien testaajat ovat avainasemassa näiden testauskäytäntöjen käytön opastamisessa läpi elinkaaren.

3.1.1 Testiohjattu kehitys, hyväksymistestiohjattu kehitys ja käyttäytymisperustainen kehitys

Testiohjattu kehitys, hyväksymistestiohjattu kehitys ja käyttäytymisperustainen kehitys ovat kolme toisiaan täydentävää tekniikkaa, joita ketterissä tiimeissä käytetään testauksen toteuttamisessa useilla testaustasoilla. Jokainen tekniikka on esimerkki testauksen perusperiaatteesta, aikaisessa vaiheessa tapahtuvien testaus- ja laadunvarmistustehtävien eduista, sillä testit määritellään ennen kuin koodi kirjoitetaan.

Testiohjattu kehitys
Testiohjattua kehitystä (TDD) käytetään kehittämään koodia perustuen automatisoituihin testitapauksiin. Testiohjatun kehityksen prosessi on:

- Lisää testi, joka kuvaa ohjelmoijan ajatuksen pienen koodin palasen halutusta toiminnasta
- Aja testi, jonka pitäisi epäonnistua, koska koodia ei ole olemassa
- Kirjoita koodi ja aja testiä tiheinä kierroksina, kunnes testi menee läpi
- Refaktoroi koodi sen jälkeen, kun testi on mennyt läpi ja aja testi uudelleen sen varmistamiseksi, että se läpäisee edelleen refaktoroidun koodin
- Toista tämä prosessi seuraavalle pienelle koodin palaselle ja aja aiemmat testit samoin kuin myös lisätyt testit.

Kirjoitetut testit keskittyvät pääasiassa yksikkötasolle ja koodiin, vaikkakin testejä voidaan kirjoittaa myös integraatio- ja järjestelmätasoille. Testiohjattu kehitys saavutti suosionsa XP:n kautta [Beck02], mutta sitä käytetään myös muissa ketterissä menetelmissä ja joskus peräkkäiselinkaarimalleissa. Se auttaa kehittäjiä keskittymään selvästi määriteltyihin odotettuihin tuloksiin. Testit automatisoidaan ja niitä käytetään jatkuvassa integraatiossa.

Hyväksymistestiohjattu kehitys
Hyväksymistestiohjattu kehitys [Adzic09] määrittelee hyväksymiskriteerit ja testit käyttäjätarinoiden luomisen aikana (katso kappale 1.2.2). Hyväksymistestiohjattu kehitys on yhteistyökeskeinen lähestymistapa, joka mahdollistaa sen, että jokainen sidosryhmä ymmärtää kuinka ohjelmistokomponentin pitää käyttäytyä ja mitä kehittäjät, testaajat ja liiketoiminnan edustajat tarvitsevat tämän toiminnan varmistamiseen. Hyväksymistestiohjatun kehityksen prosessi on kuvattu osassa 3.3.2.

Hyväksymistestiohjattu kehitys luo uudelleenkäytettäviä testejä regressiotestaukseen. Tietyt työkalut tukevat näiden testien luomista ja ajamista usein jatkuvan integraation prosessin aikana. Nämä sovellukset voivat muodostaa yhteyden sovelluksen aineistoon ja palvelukerroksiin, mikä mahdollistaa testien ajamisen järjestelmä- tai hyväksymistasolla. Hyväksymistestiohjattu kehitys mahdollistaa vikojen nopean ratkaisemisen ja ominaisuuden toiminnan kelpuuttamisen. Se auttaa määrittämään sen, täyttääkö ominaisuus hyväksymiskriteerit.

Käyttäytymisperustainen kehitys
Käyttäytymisperustainen kehitys [Chelimsky10] mahdollistaa sen, että kehittäjä voi keskittyä koodin testaamiseen ohjelmiston odotetun käyttäytymisen perusteella. Koska testit perustuvat osoitetulle ohjelmiston käyttäytymiselle, muiden tiimin jäsenten ja sidosryhmien on helpompi ymmärtää ne.

Käyttäytymisperustaista kehitysrunkoa voidaan käyttää määrittämään hyväksymiskriteerit, jotka perustuvat *kun/jos/sitten* (given/when/then) formaattiin:

Kun on joku lähtötilanne,
Jos tapahtuu tämä tapahtuma,
Sitten näiden tulosten pitäisi syntyä.

Käyttäytymisperustaisen kehityksen kehys luo näistä vaatimuksista koodia, jota kehittäjät voivat käyttää testitapauksien tekemiseen. Käyttäytymispohjainen kehitys auttaa kehittäjiä toimimaan yhteistyössä muiden sidosryhmien kanssa, testaajat mukaan lukien, ja määrittämään täsmällisiä liiketoiminnan tarpeisiin keskittyviä yksikkötestejä.

3.1.2 Testipyramidi

Ohjelmistoa voidaan testata eri tasoilla. Tyypilliset testitasot pyramidin pohjalta huipulle ovat yksikkö-, integraatio-, järjestelmä- ja hyväksymistaso (katso [ISTQB_FL_SYL], kappale 2.2). Testipyramidi korostaa testitapausten suurempaa määrää alemmilla tasoilla (pyramidin pohja), ja kehityksen siirtyessä ylemmille tasoille (pyramidin huippu) testit vähenevät. Yleensä yksikkö- ja integraatiotestit automatisoidaan ja ne luodaan käyttämällä API-perustaisia työkaluja. Järjestelmä- ja hyväksymistasoilla automaattiset testit luodaan käyttämällä GUI-perustaisia työkaluja. Testipyramidin konsepti perustuu aikaisen laadunvarmistuksen ja testauksen testausperiaatteeseen, eli viat poistetaan mahdollisimman aikaisin elinkaaressa.

3.1.3 Testausneljännes, testitasot ja testityypit

Brian Marickin määrittämät testausneljännekset [Crispin08] liittävät testitasot sopiviin testaustyyppeihin ketterässä metodologiassa. Testausneljännesmalli ja sen muunnelmat auttavat varmistamaan, että kaikki tärkeät testityypit ja testitasot sisältyvät kehityksen elinkaareen. Tämä malli tarjoaa myös tavan erottaa ja kuvata testityypit kaikille sidosryhmille, mukaan lukien kehittäjät, testaajat ja liiketoiminnan edustajat.

Testausneljänneksissä testit voivat olla liiketoiminta- (käyttäjät) tai teknologia- (kehittäjä) suuntautuneita. Jotkut testit tukevat ketterän tiimin työtä ja varmistavat ohjelmiston käyttäytymistä. Toiset testit voivat todentaa tuotetta. Testit voivat olla täysin manuaalisia, täysin automatisoituja, yhdistelmä manuaalisia ja automatisoituja, tai manuaalisia, mutta työkalujen tukemia. Neljännekset ovat seuraavat:

- Neljännes Q1 on yksikkötaso (teknologiasuuntautunut) ja se tukee kehittäjiä. Tämä neljännes sisältää yksikkötestejä. Nämä testit pitäisi automatisoida ja sisällyttää jatkuvan integraation prosessiin.
- Neljännes Q2 on järjestelmätaso (liiketoimintasuuntautunut) ja se varmistaa tuotteen toimintaa. Tämä neljännes sisältää toiminnallisia testejä, esimerkkejä, tarinatestejä, käyttäjäkokemusprototyyppejä ja simulaatioita. Nämä testit tarkistavat hyväksymiskriteerit ja ne voivat olla manuaalisia tai automatisoituja. Ne luodaan usein käyttäjätarinoiden kehityksen aikana ja näin ollen ne parantavat tarinoiden laatua. Ne ovat käyttökelpoisia, kun luodaan automatisoituja regressiotestijoukkoja.
- Neljännes Q3 on järjestelmä- tai käyttäjän hyväksymistaso (liiketoimintasuuntautunut) ja se sisältää testejä, jotka arvioivat tuotetta käyttäen realistisia skenaarioita ja dataa. Tämä neljännes sisältää tutkivaa testausta, skenaarioita, prosessivirtoja, käytettävyystestausta, käyttäjän hyväksymistestausta, alfatestausta ja betatestausta. Nämä testit ovat usein manuaalisia ja käyttäjäkeskeisiä.
- Neljännes Q4 on järjestelmä- tai käyttöön soveltuvuuden hyväksymistaso (teknologiasuuntautunut) ja se sisältää testejä, jotka arvioivat tuotetta. Tämä neljännes sisältää suorituskyky-, kuormitus-, rasitus- ja skaalautuvuustestejä, tietoturvatestejä sekä

ylläpidettävyyden, muistin hallinnan, yhteensopivuuden ja yhteen toimivuuden, datamigraation, infrastruktuurin ja toipuvuuden testausta. Nämä testit ovat usein automatisoituja.

Minkä tahansa iteraation aikana voidaan tarvita testejä mistä tahansa tai jokaisesta neljänneksestä. Testausneljännekset soveltavat ennemmin dynaamista kuin staattista testausta.

3.1.4 Testaajan rooli

Tässä sertifikaattisisällössä on yleisesti viitattu ketteriin menetelmiin ja tekniikkoihin ja testaajaan rooliin erilaisissa ketterissä elinkaarimalleissa. Tässä osassa käydään läpi erityisesti testaajan roolia scrumissa [Aalst13].

Tiimityö

Tiimityö on ketterän kehityksen perusperiaate. Ketteryys korostaa tiimiperustaista lähestymistapaa, jossa kehittäjät, testaajat ja liiketoiminnan edustajat työskentelevät yhdessä. Seuraavat ovat organisatorisia ja käyttäytymiseen liittyviä parhaita käytäntöjä scrumtiimeissä:

- Poikkitoiminnallinen: Jokainen tiimin jäsen tuo erilaisia taitoja tiimiin. Tiimi työskentelee yhdessä liittyen testausstrategiaan, testaussuunnitteluun, testien määrittelyyn, testien suorittamiseen, testien arviointiin ja testitulosten raportointiin.
- Itseorganisoituva: tiimissä voi olla vain kehittäjiä, mutta kuten on mainittu osassa 2.1.5, ihannetilanteessa testaajia olisi yksi tai useampi.
- Yhteinen sijainti: testaajat istuvat yhdessä kehittäjien ja tuoteomistajan kanssa.
- Yhteistyössä tekeminen: testaajat tekevät yhteistyötä muiden tiimin jäsenten, muiden tiimien, sidosryhmien, tuoteomistajan ja scrummasterin kanssa.
- Valtuutettu: tekniset päätökset liittyen suunnitteluun ja testaukseen tekee koko tiimi yhdessä (kehittäjät, testaajat ja scrummaster) yhteistyössä tuoteomistajan ja tarvittaessa muiden tiimien kanssa.
- Sitoutunut: testaaja on sitoutunut kyseenalaistamaan ja arvioimaan tuotteen käyttäytymistä ja ominaisuuksia asiakkaiden ja käyttäjien odotuksien ja tarpeiden pohjalta.
- Läpinäkyvä: kehityksen ja testauksen eteneminen on näkyvissä ketterällä tehtävätaululla (katso kappale 2.2.1).
- Uskottava: Testaajan pitää varmistaa testaukseen, sen valmisteluun ja suoritukseen liittyvän strategian luotettavuus, koska muuten sidosryhmät eivät luota testauksen tuloksiin. Tämä tapahtuu usein tarjoamalla testausprosessista tietoa sidosryhmille.
- Avoin palautteelle: Palaute on tärkeä menestymiseen liittyvä seikka missä tahansa projektissa ja erityisesti ketterissä projekteissa. Retrospektiivit luovat tiimeille tilaisuuden oppia menestyksestä ja epäonnistumisista.
- Joustava: testauksen täytyy pystyä vastaamaan muutokseen, samoin kuin kaikkien muidenkin toimintojen ketterissä projekteissa.

Nämä parhaat käytännöt maksimoivat testauksen onnistumisen todennäköisyyden scrum-projekteissa.

Nollasprintti

Nollasprintti on projektin ensimmäinen iteraatio ja siinä tehdään useita valmistelevia toimia (katso kappale 1.2.5). Iteraation aikana testaaja tekee yhteistyötä tiimin kanssa seuraavissa tehtävissä:

- Projektin laajuuden identifioiminen (eli tuotteen kehitysjono)
- Ensimmäisen järjestelmäarkkitehtuurin ja korkean tason prototyyppien luominen
- Tarvittavien työkalujen suunnitteleminen, hankkiminen ja asentaminen (esim. testauksenhallinta, vianhallinta, testausautomaatio ja jatkuva integraatio)
- Ensimmäisen testausstrategian luominen kaikille testitasoille, käsittäen muun muassa testauksen laajuuden, tekniset riskit, testityypit (katso kappale 3.1.3) sekä kattavuustavoitteet.

- Ensimmäisen laaturiskianalyysin suorittaminen (katso kappale 3.2.1)
- Testimetriikkojen määrittäminen testausprosessin, projektissa tehdyn testauksen edistymisen ja tuotteen laadun mittaamiseksi.
- Määritelmän laatiminen käsitteelle ”valmis”
- Tehtävätaulun luominen (katso kappale2.2.1)
- Sen määrittäminen, koska testausta jatketaan, tai koska se pysäytetään ennen järjestelmän toimittamista asiakkaalle.

Nollasprintti asettaa suunnan sille, mitä testauksen tarvitsee saavuttaa ja miten läpi sprinttien.

Integraatio

Ketterissä projekteissa tavoitteena on toimittaa asiakkaalle jatkuvasti lisäarvoa (mieluiten jokaisessa sprintissä). Jotta tämä on mahdollista, integraatiostrategian pitäisi sisältää sekä suunnittelua että testausta. Jotta jatkuvan testauksen strategiaa voidaan käyttää toimitettujen toiminnallisuuksien ja ominaisuuksien osalta, on tärkeää tunnistaa kaikki riippuvuudet taustalla olevien toimintojen ja ominaisuuksien välillä.

Testauksen suunnittelu

Koska testaus on täysin integroitu ketterään tiimiin, niin testauksen suunnittelun pitäisi alkaa julkaisunsuunnittelutilaisuudessa ja se pitäisi päivittää jokaisessa sprintissä. Julkaisun ja jokaisen sprintin testaussuunnittelun pitää käsitellä asioita, joita käytiin läpi kappaleessa 1.2.5.

Sprintin suunnittelun tuloksena syntyy joukko tehtäviä, jotka laitetaan tehtävätaululle, jossa jokaisen tehtävän pitäisi olla pituudeltaan yksi tai kaksi työpäivää. Lisäksi kaikkia testauksen ongelmia pitäisi seurata, jotta testaus saadaan pidettyä tasaisesti käynnissä.

Ketterät testauskäytännöt

Useat käytännöt voivat olla hyödyllisiä scrumtiimin testaajille ja joitakin niistä ovat:

- Parityöskentely: kaksi tiimin jäsentä (esim. testaaja ja kehittäjä, kaksi testaajaa, tai testaaja ja tuoteomistaja) istuvat yhdessä työaseman äärellä ja suorittavat testausta tai muuta sprintin tehtävää.
- Vaiheittainen testisuunnittelu: testitapaukset ja testiohjeet rakennetaan vähitellen käyttäjätarinoista ja muista testauksen lähtökohdista aloittaen yksinkertaisista testeistä siirtyen kohti monimutkaisempia testejä.
- Ajatuskartta: ajatuskartta on hyödyllinen työkalu testauksessa [Crispin08]. Esimerkiksi testaajat voivat käyttää ajatuskarttaa identifioidakseen sen, mitä testejä tehdään, kuvaamaan testausstrategioita ja kuvaamaan testiaineistoa.

Nämä käytännöt täydentävät muita käytäntöjä, joista keskusteltiin Perustason sertifikaattisisällön luvussa 4 [ISTQB_FL_SYL].

3.2 Laaturiskien arviointi ja testauksen työmäärän arviointi

Tyypillinen testauksen tavoite projekteissa, ketterissä tai perinteisissä, on vähentää tuotteen laatuongelmien riskejä hyväksyttävälle tasolle ennen julkaisua. Testaajat ketterissä projekteissa voivat käyttää samankaltaisia tekniikoita kuin käytetään perinteisissä projekteissa laaturiskien (tai tuoteriskien) tunnistamiseen, niihin liittyvän riskitason arvioimiseen sekä riskien riittävään pienentämiseen tarvittavan työmäärän arvioimiseen, ja sen jälkeen he voivat pienentää näiden riskien vaikutusta testisuunnittelun, testien toteutuksen ja suorituksen avulla. Kuitenkin, johtuen lyhyistä iteraatioista ja muutoksen tahdista ketterissä projekteissa, vaaditaan jonkin verran näiden tekniikoiden muokkaamista.

3.2.1 Laaturiskien arviointi ketterissä projekteissa

Yksi monista testauksen haasteista on testattavien tilanteiden asianmukainen valinta, kohdentaminen ja priorisointi. Siihen kuuluu jokaisen testattavan tilanteen testeillä kattamiseen tarvittavan työmäärän arvioiminen sekä tuloksena syntyvien testien jaksottaminen sellaisella tavalla, että optimoidaan tehtävän testaustyön hyödyllisyys ja tehokkuus. Ketterien tiimien testaajat voivat käyttää riskien tunnistamis-, analysointi- ja pienentämisstrategioita auttaakseen määrittämään suoritettavien testitapausten hyväksyttävän määrän, vaikka monet vuorovaikutuksessa olevat rajoitteet ja muuttujat voivat vaatia kompromisseja.

Riski on negatiivisen tai ei-toivotun seurauksen tai tapahtuman mahdollisuus. Riskitaso saadaan arvioimalla riskin esiintymisen todennäköisyys ja riskin vaikutus. Kun potentiaalinen ongelma vaikuttaa pääasiassa tuotelaatuun, tarkoitetaan laatu- tai tuoteriskejä. Kun potentiaalisen ongelman päävaikutus kohdistuu projektin menestykseen, kyse on projekti- tai suunnitteluriskeistä [Black07] [vanVeenendaal12].

Ketterissä projekteissa laaturiskianalyysiä tehdään kahdessa vaiheessa:

- Julkaisun suunnittelu: Liiketoiminnan edustajat, jotka tietävät julkaisun ominaisuudet, antavat korkean tason yleiskuvan riskeistä. Koko tiimi, mukaan lukien testaaja(t), auttaa riskien tunnistamisessa ja arvioinnissa.
- Iteraation suunnittelu: koko tiimi tunnistaa ja arvioi laaturiskit.

Esimerkkejä järjestelmän laaturiskeistä ovat:

- Virheelliset laskutoimitukset raporteissa (toiminnallinen riski liittyen tarkkuuteen)
- Hidas vaste käyttäjän syötteeseen (ei-toiminnallinen riski liittyen tehokkuuteen ja vasteaikaan)
- Vaikeus ymmärtää näyttöjä ja kenttiä (ei-toiminnallinen riski liittyen käytettävyyteen ja ymmärrettävyyteen).

Kuten aiemmin todettiin, iteraatio alkaa iteraation suunnittelulla, joka kulminoituu arvioituihin tehtäviin tehtävätaululla. Nämä tehtävät voidaan priorisoida osittain niihin liittyvän laaturiskin perusteella. Ne tehtävät, joihin liittyy korkeampia riskejä, pitäisi aloittaa aiemmin ja niihin pitäisi kuulua enemmän testaustyötä. Ne tehtävät, joihin liittyy alhaisempia riskejä, pitäisi aloittaa myöhemmin ja niihin pitäisi kuulua vähemmän testaustyötä.

Seuraava luettelo on esimerkki siitä, kuinka laaturiskien analyysiprosessi voidaan suorittaa ketterässä projektissa iteraation suunnittelun aikana:

1. Kokoa ketterän tiimin jäsenet yhteen, mukaan lukien testaaja(t)
2. Luetteloi kaikki nykyisen iteraation kehitysjonon tehtävät (esim. tehtävätaululla)
3. Tunnista jokaiseen asiaan liittyvät laaturiskit ja ota huomioon kaikki olennaiset laatuominaisuudet
4. Suorita jokaisen tunnistetun riskin arviointi, johon kuuluu kaksi tehtävää: riskin luokittelu ja sen riskitason määrittäminen vikojen vaikutuksen ja todennäköisyyden perusteella.
5. Määrittele testauksen laajuus, joka on verrannollinen riskitasoon
6. Valitse kunkin riskin, riskitason ja olennaisten laatuominaisuuksien perusteella sopivat testaustekniikat riskin pienentämiseksi.

Tämän jälkeen testaaja suunnittelee, toteuttaa ja suorittaa testejä riskien pienentämiseksi. Tämä sisältää ominaisuuksien, käyttäytymisen, laatuominaisuuksien sekä asiakas-, käyttäjä- ja sidosryhmätyytyväisyyteen vaikuttavien ominaisuuksien muodostaman koko kokonaisuuden.

Ketterän tiimin pitäisi koko projektin ajan pysyä tietoisena lisäinformaatiosta, joka voi muuttaa riskejä ja/tai tunnettuihin laaturiskeihin liittyvää riskitasoa. Laaturiskianalyysiä pitäisi muokata aika ajoin, ja siitä

seuraa muutoksia testeihin. Muokkaamiseen kuuluu uusien riskien tunnistamista, olemassa olevien riskien uudelleenarviointia sekä riskiä lieventävien toimien tehokkuuden arviointia.

Laaturiskejä voidaan pienentää myös ennen testien suorituksen aloittamista. Jos esimerkiksi riskien tunnistamisen aikana löydetään ongelmia käyttäjätarinoista, projektitiimi voi riskien pienentämisstrategiana katselmoida käyttäjätarinat perusteellisesti.

3.2.2 Testauksen työmäärän arviointi perustuen sisältöön ja riskiin.

Julkaisun suunnittelun aikana ketterä tiimi arvioi tarvittavan työmäärän julkaisun saamiseksi valmiiksi. Arviointi kohdistuu myös testauksen työmäärään. Yleinen arviointimenetelmä ketterissä projekteissa on suunnittelupokeri, joka on yhteisymmärrykseen perustuva tekniikka. Tuoteomistaja tai asiakas lukee käyttäjätarinan arvioijille. Jokaisella arvioijalla on korttipakka, jonka arvot ovat samoja kuin Fibonaccin lukujonossa (esim. 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,…) tai missä tahansa muussa kasvavassa sarjassa (esim. paidan koot alkaen koosta XS kokoon XXL). Arvot edustavat tarinapisteiden määrää, työpäiviä tai muuta tiimin arvioinnissa käyttämää yksikköä. Fibonaccin lukujonon käyttäminen on suositeltavaa, koska lukujonon numerot heijastavat sitä, että epävarmuus kasvaa suhteessa tarinan kokoon. Korkea arvio tarkoittaa usein sitä, että tarinaa ei ole kunnolla ymmärretty, tai se pitäisi pilkkoa useisiin pienempiin tarinoihin.

Arvioijat keskustelevat ominaisuudesta ja kysyvät tarvittaessa tuoteomistajan kysymyksiä. Arvioinnissa ovat esillä sellaiset näkökulmat kuin kehityksen ja testauksen työmäärä, tarinoiden monimutkaisuus ja testauksen laajuus. Siksi onkin suositeltavaa ottaa arvioinnissa huomioon tuoteomistajalta määrittämän kehitettävän asian tärkeyden lisäksi myös asian riskitaso ennen suunnittelupokerin aloittamista. Kun ominaisuudesta on keskusteltu perusteellisesti, jokainen arvioija valitsee itse yhden kortin, joka edustaa hänen arviotaan. Sitten kaikki kortit paljastetaan yhtä aikaa. Jos kaikki arvioijat valitsivat saman arvon, siitä tulee arvio. Jos ei, arvioijat keskustelevat arvioiden eroavaisuuksista, minkä jälkeen pokerikierros uusitaan, kunnes hyväksyntä saavutetaan joko yhteisymmärryksellä tai käyttämällä sääntöjä (esim. käytetään mediaania, käytetään korkeinta pistemäärää), joilla rajoitetaan pelattavien pokerikierrosten määrää. Nämä keskustelut varmistavat luotettavan arvion työmäärästä, joka tarvitaan tuoteomistajan haluamien tuotteen kehitettävien asioiden valmiiksi saamiseen, ja auttavat lisäämään kollektiivista tietämystä siitä, mitä pitää tehdä [Cohn04].

3.3 Ketterien projektien tekniikoita

Monia perinteisissä projekteissa käytettäviä testitekniikoita ja -tasoja voidaan soveltaa myös ketterissä projekteissa. Ketterissä projekteissa on kuitenkin niille ominaisia seikkoja ja eroja testitekniikoissa, terminologiassa ja dokumentaatiossa, jotka pitäisi ottaa huomioon.

3.3.1 Hyväksymiskriteerit, riittävä kattavuus sekä muut testauksen tiedot

Ketterissä projekteissa alkuperäiset vaatimukset esitetään projektin alussa käyttäjätarinoiden priorisoituna kehitysjonona. Alussa vaatimukset ovat lyhyitä ja ne noudattavat tavallisesti etukäteen määriteltyä muotoa (katso kappale1.2.2). Ei-toiminnalliset vaatimukset, kuten käytettävyys ja suorituskyky, ovat myös tärkeitä ja ne voidaan määritellä omina käyttäjätarinoinaan tai ne voidaan yhdistää muihin toiminnallisiin käyttäjätarinoihin. Ei-toiminnalliset vaatimukset voivat noudattaa etukäteen määriteltyä muotoa tai standardia kuten [ISO25000] tai toimialakohtaista standardia.

Käyttäjätarinat toimivat tärkeänä testauksen lähdedokumentaationa. Muuta tärkeää testauksen pohjamateriaalia ovat:

- kokemukset aiemmista projekteista
- olemassa olevat järjestelmän toiminnot, ominaisuudet ja laatupiirteet
- koodi, arkkitehtuuri ja design
- käyttäjäprofiilit (konteksti, järjestelmän konfiguraatiot ja käyttäjien toiminta)
- Nykyisen ja aiempien projektien vikatiedot

- vikojen luokittelu vikaluokitusjärjestelmässä
- soveltuvat standardit (esim. [DO-178B] lentokone-elektroniikan ohjelmistoille)
- laaturiskit (katso kappale3.2.1).

Jokaisen iteraation aikana kehittäjät luovat koodia, joka toteuttaa käyttäjätarinassa kuvatut toiminnot ja piirteet sekä niihin liittyvät laatuominaisuudet, ja tämä koodi todennetaan ja kelpuutetaan hyväksymistesteillä. Ollakseen testattavia hyväksymiskriteerien pitäisi soveltuvin osin käsitellä seuraavia aiheita [Wiegers13]:

- Toiminnallinen käyttäytyminen: ulkoisesti havaittava toiminta käyttäjän tekemien syötteiden mukaan määrätyllä kokoonpanolla.
- Laatuominaisuudet: Kuinka järjestelmä suorittaa määrätyn toiminnan. Ominaisuuksilla voidaan tarkoittaa laatuattribuutteja tai ei-toiminnallisia vaatimuksia. Yleisiä laatuominaisuuksia ovat suorituskyky, luotettavuus, käytettävyys jne.
- Skenaariot (käyttötapaukset): Sarja toimintoja ulkoisen toimijan (usein käyttäjä) ja järjestelmän välillä, jotta saadaan toteutettua määrätty tavoite tai liiketoiminnallinen tehtävä.
- Liiketoimintasäännöt: Toimintoja, jotka järjestelmä voi suorittaa ainoastaan tietynlaisissa ulkopuolisten proseduurien ja rajoitteiden määrittämissä olosuhteissa (esim. vakuutusyhtiön proseduuri vakuutushakemusten käsittelemiseksi).
- Ulkoiset rajapinnat: Kehitettävän järjestelmän ja ulkopuolisen maailman välisten yhteyksien kuvaus. Ulkoiset rajapinnat voidaan jakaa eri tyyppeihin (käyttöliittymä, liittymät muihin järjestelmiin jne.).
- Rajoitteet: Kaikki suunnittelun ja toteutuksen rajoitteet, jotka rajoittavat kehittäjän vaihtoehtoja. Laitteissa, joissa on sulautettuja ohjelmistoja, täytyy usein huomioida fyysiset rajoitteet kuten koko, paino ja liitäntäyhteydet.
- Datan määritykset: Asiakas voi kuvata yksittäisen tietokokonaisuuden muodon, tietotyypin, sallitut arvot ja oletusarvot monimutkaisessa liiketoimintatietorakenteessa (esim. postinumero Yhdysvalloissa käytettävässä postiosoitteessa).

Käyttäjätarinoiden ja niiden hyväksymiskriteerien lisäksi muu tieto on merkityksellistä testaajalle:

- Kuinka järjestelmän oletetaan toimivan ja kuinka sitä oletetaan käytettävän
- Liittymät, joita voidaan käyttää ja joihin voidaan päästä järjestelmän testaamiseksi
- Onko nykyinen työkalutuki riittävää
- Onko testaajalla riittävät tiedot ja taidot suorittaa tarpeelliset testit.

Testaajat huomaavat usein iteraation aikana tarvitsevansa lisätietoa (esim. koodikattavuus) ja heidän tulisi työskennellä yhteistyössä muiden ketterän tiimin jäsenten kanssa tiedon saamiseksi. Asiaankuuluvalla tiedolla on merkitystä, kun määritetään, voidaanko jotakin tiettyä tehtävää pitää valmiina. Tämä valmiin määritelmän käsite on kriittinen ketterissä projekteissa ja sitä käytetään useilla eri tavoilla, joita käydään läpi seuraavaksi.

Testitasot

Jokaisella testitasolla on oma valmiin määritelmä. Seuraavassa luettelossa on esimerkkejä, jotka voivat olla olennaisia eri testitasoilla.

- Yksikkötestaus
  - 100% päätöskattavuus, missä mahdollista, sekä huolellinen toteuttamiskelvottomien polkujen katselmointi
  - Staattinen analyysi on suoritettu koko koodille
  - Ei ratkaisemattomia vakavia vikoja (luokiteltu tärkeyden ja vakavuusasteen mukaan)
  - Ei tunnettua hyväksymätöntä teknistä velkaa suunnittelussa eikä koodissa [Jones11]
  - Koko koodi, yksikkötestit ja yksikkötestitulokset on katselmoitu
  - Kaikki yksikkötestit on automatisoitu

- Tärkeät ominaisuudet ovat hyväksyttävissä rajoissa (esim. suorituskyky).
- Integraatiotestaus
  - Kaikki toiminnalliset vaatimukset on testattu, mukaan lukien sekä positiiviset että negatiiviset testit, ja testien määrä perustuu kokoon, monimutkaisuuteen ja riskeihin
  - Kaikki yksiköiden väliset rajapinnat on testattu
  - Kaikki laaturiskit on katettu sovitun testauksen laajuuden mukaan
  - Ei ole ratkaisemattomia vakavia vikoja (priorisoituna riskin ja tärkeyden mukaan)
  - Kaikki löydetyt viat on raportoitu
  - Kaikki regressiotestit on automatisoitu, mikäli mahdollista, ja kaikki automatisoidut testit on tallennettu yhteiseen tietovarastoon.
- Järjestelmätestaus
  - Käyttäjätarinoiden, ominaisuuksien ja toimintojen alusta loppuun testaaminen
  - Kaikki käyttäjäpersoonat on katettu
  - Järjestelmän tärkeimmät laatuominaisuudet on katettu (esim. suorituskyky, kestävyys, luotettavuus)
  - Testaus on tehty tuotannonkaltaisessa ympäristössä käsittäen kaikki laitteistot ja ohjelmistot kaikille tuetuille kokoonpanoille siinä laajuudessa kuin on mahdollista
  - Kaikki laaturiskit on katettu sovitun testauksen laajuuden perusteella
  - Kaikki regressiotestit on automatisoitu mikäli mahdollista, ja kaikki automatisoidut testit on tallennettu yhteiseen tietovarastoon
  - Kaikki viat on raportoitu ja mahdollisesti on korjattu
  - Ratkaisemattomia vakavia vikoja ei ole (priorisoituna riskin ja tärkeyden mukaan).

Käyttäjätarina

Valmiin määritelmä käyttäjätarinoille voidaan määritellä seuraavilla kriteereillä:

- Iteraatioon valitut käyttäjätarinat ovat valmiita, tiimi ymmärtää ne ja niille on yksityiskohtaiset ja testattavat hyväksymiskriteerit
- Kaikkien käyttäjätarinoiden elementtien määrittely ja katselmointi mukaan lukien käyttäjätarinan hyväksymistestit on saatu valmiiksi
- Tiimi on tunnistanut valittujen käyttäjätarinoiden toteuttamiseen ja testaamiseen tarvittavat tehtävät ja arvioinut tehtävien työmäärät.

Ominaisuus

Ominaisuuksien valmiin määritelmään, joka voi liittyä useaan käyttäjätarinaan tai eepokseen, voi kuulua seuraavia asioita:

- Kaikki olennaiset käyttäjätarinat hyväksymiskriteereineen ovat asiakkaan määrittämiä ja hyväksymiä
- Suunnittelu on valmis ja teknistä velkaa ole
- Koodi on valmista ja tunnettua teknistä velkaa eikä keskeneräistä refaktorointia ole
- Yksikkötestit on suoritettu ja määritetty kattavuustaso on saavutettu
- Määritettyihin kattavuuskriteereihin perustuvat ominaisuuden integraatio- ja järjestelmätestit on suoritettu
- Vakavia korjattavia vikoja ei ole
- Ominaisuuden dokumentaatio on valmis ja se voi sisältää julkaisuselosteen, käyttöohjeita ja online-käyttöohjetoimintoja.

Iteraatio

Valmiin määritelmä iteraatiolle voi sisältää seuraavia:

- Kaikki iteraation ominaisuudet ovat valmiita ja ne on yksittäin testattu ominaisuuden tasokriteerin mukaan.
- Kaikki ei-kriittiset viat, joita ei voida korjata iteraation rajoitteiden johdosta, on lisätty tuotteen kehitysjonoon ja priorisoitu
- Kaikkien iteraation ominaisuuksien integrointi on valmistunut ja testattu
- Dokumentaatio on kirjoitettu, katselmoitu ja hyväksytty.

Tässä vaiheessa ohjelmisto on potentiaalisesti julkaistavissa, koska iteraatio on saatu onnistuneesti loppuun mutta kaikkien iteraatioiden tuloksena ei ole julkaisua.

Julkaisu

Julkaisun valmiin määritelmä, joka voi liittyä useaan iteraatioon, voi sisältää seuraavia alueita:

- Kattavuus: Kaikki relevantit testauskohteen kuvaamisen elementit julkaisun koko sisällölle on katettu testauksella. Kattavuuden riittävyys määritellään sen mukaan, mikä on uutta tai muuttunut, kuinka monimutkaisia ja isoja muutokset ovat ja millainen vikaantumisen riski niihin liittyy.
- Laatu: Vikaintensiteetti (esim. kuinka monta vikaa löydetään päivässä tai tapahtumaa kohti), vikatiheys (esim. kuinka monta vikaa löydetään verrattuna käyttäjätarinoiden määrään, työmäärään ja/tai laatuattribuutteihin), arvioitu jäljellä olevien vikojen määrä on hyväksyttävissä rajoissa, ratkaisemattomien ja jäljellä olevien vikojen seuraukset (esim. vakavuus ja tärkeys) on ymmärretty ja ne ovat hyväksyttäviä, jäljelle jäävä riskitaso jokaisessa identifioidussa laaturiskissä on ymmärretty ja hyväksyttävä.
- Aika: Jos etukäteen määritetty toimituspäivä on saavutettu, on pohdittava julkaisemiseen tai julkaisematta jättämiseen liittyviä liiketoiminnallisia seikkoja.
- Kustannus: Arvioituja elinkaarikustannuksia pitäisi käyttää toimitettuun järjestelmään sijoitetun pääoman tuoton laskemiseksi (laskettujen kehitys- ja ylläpitokustannuksien pitäisi olla selvästi pienempiä kuin odotettu tuotteen kokonaismyynti). Tuotantoon päässeiden vikojen vuoksi pääosa elinkaarikustannuksista tulee ylläpidosta sen jälkeen, kun tuote on julkaistu.

3.3.2 Hyväksymistestiohjatun kehityksen soveltaminen

Hyväksymistestiohjattu kehitys on testauslähtöinen lähestymistapa, jossa testitapaukset luodaan ennen käyttäjätarinoiden toteuttamista. Testitapaukset luo ketterä tiimi mukaan lukien kehittäjät, testaajat sekä liiketoiminnan edustajat [Adzic09], ja ne voivat olla manuaalisia tai automatisoituja. Ensimmäinen vaihe on määrittelytyöpaja, missä kehittäjät, testaajat ja liiketoiminnan edustajat analysoivat käyttäjätarinan, keskustelevat siitä ja kirjoittavat sen. Kaikki vaillinaisuudet, tulkinnanvaraisuudet tai virheet korjataan tämän prosessin aikana.

Seuraava vaihe on testien luominen. Sen voi tehdä koko tiimi yhdessä tai jokainen testaaja itsekseen. Kaikissa tapauksissa riippumaton henkilö, kuten liiketoiminnan edustaja, kelpuuttaa testit. Testit ovat esimerkkejä, jotka kuvaavat käyttäjätarinan erityisominaisuudet. Nämä esimerkit auttavat tiimiä toteuttamaan käyttäjätarinat oikein. Koska esimerkit ja testit ovat samoja, näitä termejä käytetään usein vaihtaen niitä keskenään. Työ alkaa perusesimerkeillä ja avoimilla kysymyksillä.

Ensimmäiset testit ovat tyypillisesti positiivisia testejä, joilla varmistetaan oikea toiminta ilman poikkeus- tai vikatilanteita, ja ne muodostavat toimintojen sarjan, joka suoritetaan, jos kaikki menee odotetusti. Sen jälkeen, kun positiiviset testit on tehty, tiimin pitäisi kirjoittaa myös negatiivisia testejä ja kattaa ei-toiminnallisia attribuutteja (esim. suorituskyky, käytettävyys). Testit esitetään tavalla, jonka jokainen sidosryhmä pystyy ymmärtämään, ja ne sisältävät luonnollisen kielen lauseita, jotka kuvaavat tarvittavat esiehdot, mikäli niitä on, sekä syötteet ja niihin liittyvät tulokset.

Esimerkkien täytyy kattaa kaikki käyttäjätarinan ominaisuudet ja niiden ei pitäisi täydentää tarinaa. Tämä tarkoittaa, että sellaista esimerkkiä ei pitäisi olla, joka kuvaa sellaista käyttäjätarinan näkökulmaa, jota ei ole dokumentoitu itse käyttäjätarinassa. Lisäksi kahden esimerkin ei pitäisi kuvata samoja käyttäjätarinan ominaisuuksia.

3.3.3 Toiminnallinen ja ei-toiminnallinen mustalaatikkotestisuunnittelu

Ketterässä testauksessa testaajat luovat usein testejä samaan aikaan kehittäjien ohjelmointitehtävien kanssa. Aivan kuten kehittäjät ohjelmoivat käyttäjätarinoiden ja hyväksymiskriteerien perusteella, testaajat luovat testejä perustuen käyttäjätarinoihin ja niiden hyväksymiskriteereihin. (Jotkut testit, kuten tutkivat testit ja eräät muut kokemusperusteiset testit, luodaan myöhemmin testien suorituksen aikana, kuten kerrotaan kappaleessa 3.3.4.) Näitä testejä luodessaan testaajat voivat soveltaa perinteisiä mustalaatikkotestisuunnittelutekniikoita kuten ekvivalenssiositusmenetelmää, raja-arvoanalyysiä, päätöstauluja ja tilasiirtymätestausta. Raja-arvoanalyysiä voisi esimerkiksi käyttää valittaessa testiarvoja, kun asiakas voi valita rajoitetun määrän nimikkeitä ostoon.

Useissa tilanteissa ei-toiminnallisia vaatimuksia voidaan dokumentoida käyttäjätarinoina. Mustalaatikkotestisuunnittelutekniikoita (kuten raja-arvoanalyysi) voidaan myös käyttää, kun luodaan testejä ei-toiminnallisten laatuominaisuuksien testaamiseksi. Käyttäjätarina voi sisältää suorituskyky- tai luotettavuusvaatimuksia. Esimerkiksi tietty suoritus ei voi ylittää jotakin aikarajaa tai operaatioiden määrä saa epäonnistua korkeintaan tietyn määrän kertoja.

Lisää tietoa mustalaatikkotestisuunnittelutekniikoiden käytöstä on Perustason sertifikaattisisällössä [ISTQB_FL_SYL] sekä Jatkotason sertifikaattisisällön Testausasiantuntija-osassa [ISTQB_ALTA_SYL].

3.3.4 Tutkiva testaus ja ketterä testaus

Tutkiva testaus on tärkeää ketterissä projekteissa johtuen testianalyysin aikarajoitteista ja rajallisista käyttäjätarinoiden yksityiskohdista. Parhaiden tulosten saavuttamiseksi tutkiva testaus pitäisi yhdistää muiden kokemusperäisten tekniikoiden kanssa osaksi reaktiivista testausstrategiaa, joka yhdistetään muihin testausstrategioihin kuten analyyttiseen riskipohjaiseen testaukseen, analyyttiseen vaatimusperusteiseen testaukseen, mallipohjaiseen testaukseen ja regressiota ehkäisevään testaukseen. Testausstrategioita ja testausstrategioiden yhdistämistä käydään läpi Perustason sertifikaattisisällössä [ISTQB_FL_SYL].

Tutkivassa testauksessa testien suunnittelu ja suoritus tapahtuvat samaan aikaan ja niitä ohjaa etukäteen tehty testausohje. Testausohje kuvaa aikarajoitetun testaussession aikana katettavat testattavat tilanteet. Tutkivan testauksen aikana viimeisimpien testien tulokset ohjaavat seuraavaa testiä. Samoja lasi- ja mustalaatikkotekniikoita voidaan käyttää testien suunnittelemiseen kuin suoritettaessa etukäteen suunniteltua testausta.

Testausohje voi sisältää seuraavaa tietoa:

- Toimija: järjestelmän tarkoitettu käyttäjä
- Tarkoitus: ohjeen teema sisältäen nimenomaisen tavoitteen, jonka toimija haluaa saavuttaa (eli testattavat tilanteet)
- Järjestely: mitä pitää olla valmiina, jotta testien suoritus voidaan aloittaa
- Prioriteetti: kyseisen ohjeen suhteellinen tärkeys, joka perustuu siihen liittyvän käyttäjätarinan tärkeyteen tai riskitasoon
- Lähdeviite: määrittelyt (esim. käyttäjätarina), riskit tai muut tietolähteet
- Tietoaineisto: mitä hyvänsä tietoaineistoa tarvitaan ohjeen toteuttamiseksi
- Tehtävät: lista ideoita koskien sitä, mitä käyttäjä voi haluta tehdä järjestelmällä (esim. “Kirjaudu järjestelmään huippukäyttäjänä”) ja mitä voisi olla mielenkiintoista testata (sekä positiivisia että negatiivisia testejä)

- Oraakkelimuistiinpanot: kuinka arvioidaan tuotetta oikeiden tulosten määrittelemiseksi (esim. otetaan talteen, mitä tapahtuu näytöllä, ja verrataan sitä käyttöohjeessa kuvattuun)
- Muunnelmat: vaihtoehtoiset toiminnat ja arvioinnit, joilla täydennetään tehtävien alla kuvattuja ajatuksia.

Tutkivan testauksen hallinnassa voidaan käyttää menetelmää, jota kutsutaan nimellä sessioperusteinen testauksenhallinta. Sessio määritellään keskeytymättömäksi jaksoksi testausta, joka voi kestää 60-120 minuuttia. Testisessio sisältää seuraavat vaiheet:

- Tutkimussessio (opitaan, kuinka se toimii)
- Analysointisessio (toiminnan tai ominaisuuksien arviointi)
- Syvempi perehtyminen (rajatapaukset, skenaariot, vuorovaikutukset).

Testauksen laatu riippuu testaajien kyvystä kysyä relevantteja kysymyksiä liittyen siihen, mitä testataan. Esimerkkeinä voidaan mainita seuraavia:

- Mikä on kaikkein tärkeintä selvittää järjestelmästä?
- Millä tavalla järjestelmä voi epäonnistua?
- Mitä tapahtuu jos.....?
- Mitä pitäisi tapahtua kun.....?
- Täyttyvätkö asiakkaan tarpeet, vaatimukset ja odotukset?
- Onko järjestelmä mahdollista asentaa (ja poistaa, jos tarpeellista) kaikkia tuettuja päivityspolkuja käyttämällä?

Testauksen suorittamisen aikana testaaja käyttää luovuutta, intuitiota, kognitiota ja osaamistaan löytääkseen mahdollisia ongelmia liittyen tuotteeseen. Testaajalla tarvitsee myös olla hyvä tietämys ja ymmärrys testattavasta ohjelmistosta, liiketoiminta-alueesta, siitä, kuinka ohjelmistoa käytetään, ja kuinka määritetään, milloin järjestelmä epäonnistuu.

Heuristiikkojen sarjaa voidaan soveltaa testauksessa. Heuristiikka voi ohjata testaajaa testien suorituksessa ja tulosten arvioinnissa. [Hendrickson]. Esimerkkeihin kuuluvat:

- rajat
- CRUD: luo (Create), lue (Read), päivitä (Update), poista (Delete)
- kokoonpanon muunnelmat
- keskeytykset (esim. uloskirjautuminen, sammuttaminen tai uudelleen käynnistäminen).

Testaajan on tärkeää dokumentoida prosessia niin paljon kuin mahdollista. Muuten voisi olla vaikeata palata taaksepäin ja nähdä, kuinka järjestelmän ongelma havaittiin. Seuraava lista tarjoaa esimerkkejä tiedoista, joita voi olla hyödyllistä dokumentoida:

- Testikattavuus: mitä syötetietoja on käytetty, kuinka paljon on katettu ja kuinka paljon on jäljellä testattavaa
- Arviointimuistiinpanot: havaintoja testauksen aikana, onko järjestelmä ja testattava ominaisuus vakaa, löydettiinkö vikoja, mikä on suunniteltu seuraavaksi vaiheeksi nykyisten havaintojen mukaan ja muu lista ideoita
- Riski-/strategialista: mitä riskejä on katettu ja mitkä jäävät jäljelle tärkeimmistä riskeistä, tullaanko noudattamaan alkuperäistä strategiaa, tarvitaanko jotakin muutoksia
- Ongelmat, kysymykset ja poikkeamat: kaikki odottamaton käyttäytyminen, kaikki kysymykset liittyen lähestymistavan tehokkuuteen, kaikki huolet koskien ideoita/testiyrityksiä, testiympäristöä, testiaineistoa, toiminnon väärinymmärtämistä, testiskriptiä tai testattavana olevaa järjestelmää
- Todellinen käyttäytyminen: järjestelmän todellisen käyttäytymisen tallentaminen (esim. video, kuvakaappaukset, tuloksena olevat tiedostot)

Kirjattu informaatio pitäisi kerätä ja/tai vetää yhteen jonkinlaiseen hallintavälineeseen (esim. testauksenhallintatyökalut, tehtävänhallintatyökalut, tehtävätaulu) sellaisella tavalla, että sidosryhmien on helppo ymmärtää kaiken suoritetun testauksen senhetkinen tilanne.

3.4 Työkalut ketterissä projekteissa

Perustason sertifikaattisisällössä [ISTQB_FL_SYL] kuvatut työkalut ovat olennaisia ja niitä käyttävät testaajat ketterissä tiimeissä. Kaikkia työkaluja ei käytetä samalla tavalla ja joillakin työkaluilla on enemmän merkitystä ketterissä kuin perinteisissä projekteissa. Vaikka esimerkiksi testauksenhallinta- ja havaintojenhallintatyökaluja (vianhallintatyökaluja) voidaan käyttää ketterissä tiimeissä, jotkut ketterät tiimit valitsevat kaiken kattavan työkalun (esim. sovelluksen elinkaarenhallinta tai tehtävänhallinta), joka tarjoaa ketterälle kehitykselle oleellisia ominaisuuksia kuten tehtävätaulut, edistymiskäyrät ja käyttäjätarinat. Kokoonpanonhallintatyökalut ovat tärkeitä ketterien tiimien testaajille johtuen automatisoitujen testien suuresta määrästä kaikilla tasoilla ja tarpeesta varastoida sekä hallita niihin liittyviä automatisoituja testimateriaaleja.

Perustason sertifikaattisisällössä [ISTQB_FL_SYL] esiteltyjen työkalujen lisäksi ketterien projektien testaajat voivat myös hyödyntää seuraavissa kappaleissa esiteltäviä työkaluja. Koko tiimi käyttää näitä työkaluja varmistaakseen tiimin yhteistyön ja tiedon jakamisen, jotka ovat avain ketteriin käytäntöihin.

3.4.1 Tehtävänhallinta- ja seurantatyökalut

Joissakin tapauksissa ketterät tiimit käyttävät fyysisiä tarina-/tehtävätauluja (esim. valkotaulu, korkkitaulu) käyttäjätarinoiden, testien ja muiden tehtävien hallintaan ja seurantaan läpi jokaisen sprintin. Toiset tiimit voivat käyttää sovelluksen elinkaarenhallinta- ja tehtävänhallintaohjelmistoja mukaan lukien sähköiset tehtävätaulut. Nämä työkalut palvelevat seuraavia tarkoituksia:

- Tarinoiden ja niiden keskeisten kehitys- ja testaustehtävien tallentaminen sen varmistamiseksi, että mitään ei katoa sprintin aikana
- Tiimin jäsenten tehtäville antamien työmääräarvioiden tallentaminen ja tarinan toteuttamiseen vaadittavan työmäärän automaattinen laskenta tehokkaiden iteraation suunnittelupalavereiden tukemiseksi
- Samaan tarinaan liittyvien kehitys- ja testaustehtävien yhteenliittäminen kokonaiskuvan saamiseksi tarinan toteuttamiseen tiimiltä vaadittavasta työmäärästä.
- Kehittäjien ja testaajien tehtävien tilapäivitysten yhteen kokoaminen sitä mukaan, kun he saavat työnsä valmiiksi, ja sitä kautta automaattisesti lasketun tilannekatsauksen saaminen jokaisen tarinan, iteraation ja koko julkaisun tilasta
- Tarjoavat visuaalisen esityksen (metriikoiden, kaavioiden ja mittaritaulujen kautta) jokaisen käyttäjätarinan nykytilanteesta, iteraatiosta ja julkaisusta mahdollistaen kaikkien sidosryhmien (mukaan lukien maantieteellisesti hajautettujen tiimien) nopean tilanteen tarkistamisen
- Integroituvat kokoonpanonhallinnan työkalujen kanssa, joka voi mahdollistaa koodin sisään kirjaamisen automaattisen tallentamisen ja koonnin vasten tehtäviä ja joissakin tapauksissa automaattisen tilannepäivityksen tehtäville.

3.4.2 Kommunikoinnin ja tiedon jakamisen työkalut

Sähköpostin, dokumenttien ja suullisen kommunikoinnin lisäksi ketterät tiimit käyttävät usein kolmea eri täydentävää työkalua kommunikoinnin ja tiedon jakamisen tukemiseen: wikejä, pikaviestintää ja työpöydän jakamista.

Wikit mahdollistavat tiimien rakentaa ja jakaa internetissä luettavan tietämyskannan projektin eri näkökulmista sisältäen seuraavia:

- Tuoteominaisuusdiagrammeja, ominaisuuskeskusteluja, prototyyppidiagrammeja, valokuvia kirjoitustauluilla olevista keskusteluista sekä muuta tietoa
- Työkaluja ja/tai tekniikoita joita tiimin muut jäsenet ovat kokeneet hyödyllisiksi

- Metriikoita, kaavioita ja mittaritauluja tuotetilanteesta, joka on erityisen hyödyllistä silloin, kun wiki on integroitu muihin työkaluihin (kuten koontipalvelimeen ja tehtävänhallintajärjestelmään), koska työkalu voi päivittää tuotetilanteen automaattisesti
- Tiimin jäsenten välisiä keskusteluja, samantapaisia kuin pikaviestintä ja sähköposti, mutta sellaisella tavalla, että ne voidaan jakaa kelle tahansa tiimin jäsenelle.

Pikaviestintä, puhelinkokoukset ja videokeskustelutyökalut tarjoavat seuraavia etuja:

- Mahdollistavat tiimin jäsenten välisen reaaliaikaisen suoran kommunikoinnin, erityisesti hajautetuissa tiimeissä
- Hajautetut tiimit ovat mukana päiväpalavereissa
- Pienentävät puhelinlaskuja VOIP-tekniikan käytön johdosta ja poistavat kustannusrajoitteita, jotka voivat vähentää tiimin jäsenten välistä kommunikointia hajautetuissa ympäristöissä.

Työpöydän jakamiseen ja tallentamiseen tarkoitetut työkalut tarjoavat seuraavia etuja:

- Hajautetuissa tiimeissä voidaan tehdä tuote-esittelyjä, koodikatselmointeja ja jopa parityöskentelyä
- Tuote-esittelyjen tallentaminen jokaisen iteraation lopuksi, jotta ne voidaan lähettää tiimin wikiin.

Näitä työkaluja pitäisi käyttää täydentämään ja laajentamaan, ei korvaamaan, kasvokkain tapahtuvaa kommunikointia ketterissä tiimeissä.

3.4.3 Ohjelmiston koonnin ja levityksen työkalut

Kuten aiemmin tässä sertifikaattisisällössä kerrottiin, päivittäinen ohjelmiston koonti ja jakelu ovat avainkäytäntöjä ketterissä tiimeissä. Tämä vaatii jatkuvan integraation ja koonnin jakelun työkalujen käyttämistä. Näiden työkalujen käyttämistä, hyötyjä ja riskejä kuvattiin aiemmin kappaleessa 1.2.4.

3.4.4 Kokoonpanonhallinnan työkalut

Ketterissä tiimeissä kokoonpanonhallinnan työkaluja ei käytetä vain lähdekoodin ja automatisoitujen testien säilyttämiseen, vaan myös manuaalisten testien ja muiden töiden tulokset tallennetaan usein samaan tietovarastoon tuotteen lähdekoodin kanssa. Tämä tarjoaa jäljitettävyyden testauksessa käytettyjen ohjelmistoversioiden ja testeissä käytettyjen testitapausten välillä ja mahdollistaa nopean muutoksen historiatietoja menettämättä. Versionhallintajärjestelmien päätyyppeihin kuuluvat keskitetyt lähdekoodinhallintajärjestelmät ja hajautetut versionhallintajärjestelmät. Tiimin koko, rakenne, sijainti ja vaatimukset välineen integroinnille muiden työkalujen kanssa määrittelevät sen, mikä versionhallintajärjestelmä on oikea nimenomaiselle ketterälle projektille.

3.4.5 Testien suunnittelun, toteuttamisen ja suorittamisen työkalut

Jotkut työkalut ovat käyttökelpoisia ketterille testaajille määrätyssä ohjelmistotestausprosessin vaiheessa. Vaikka useimmat näistä työkaluista eivät ole uusia tai ominaisia ketterälle, ne tarjoavat ketterissä projekteissa tapahtuvien nopeiden muutosten kannalta tärkeitä ominaisuuksia.

- Testien suunnittelutyökalut: Käsitekarttojen ja muiden sen tyyppisten työkalujen käyttäminen on tullut suositummaksi uuden ominaisuuden nopeaan suunnitteluun ja testien määrittämiseen.
- Testitapausten hallintatyökalut: Ketterässä kehityksessä käytettävä testitapausten hallintatyökalu voi olla osa koko tiimin sovelluksen elinkaarenhallinta- tai tehtävienhallintatyökalua.
- Testiaineiston valmistelu- ja luontityökalut: Työkalut, jotka generoivat tietoaineistoa ja jolla täytetään sovelluksen tietokantaa, ovat hyvin hyödyllisiä silloin, kun sovelluksen testaamiseksi on välttämätöntä käyttää isoa tietomäärää ja erilaisia tietojen yhdistelmiä. Nämä työkalut voivat myös auttaa määrittämään uudelleen tietokannan rakennetta, kun tuote muuttuu ketterän projektin aikana, ja uudelleenfaktoroimaan skriptejä tietoaineiston luomiseksi. Tämä

mahdollistaa nopean testiaineiston päivittämisen, kun muutoksia tapahtuu. Jotkut testiaineiston valmistelutyökalut käyttävät tuotannosta peräisin olevaa tietoa raakamateriaalina ja arkaluontoinen aineisto poistetaan tai anonymisoidaan skriptien avulla. Toiset testiaineiston valmistelutyökalut voivat auttaa kelpuuttamaan laajoja määriä tietosyötteitä tai tulosteita.

- Testiaineiston lataamistyökalut: Sen jälkeen, kun testiaineisto on luotu testausta varten, se täytyy ladata sovellukseen. Manuaalinen testiaineiston lisääminen on usein aikaa vievää ja virhealtista, mutta on olemassa testiaineiston lataamistyökaluja, jotka tekevät prosessista luotettavan ja tehokkaan. Itse asiassa monet testiaineiston luontityökalut sisältävät integroidun komponentin testiaineiston lataamiseen. Muissa tapauksissa, myös massalataaminen tietokannanhallintajärjestelmää käyttämällä on mahdollista.
- Automaattiset testiensuoritustyökalut: On olemassa testiensuoritustyökaluja, jotka ovat enemmän linjassa ketterän testauksen kanssa. Saatavilla on sekä kaupallisia että avoimen lähdekoodin erityistyökaluja, jotka tukevat testauslähtöisiä lähestymistapoja, kuten käyttäytymisperustaista kehitystä, testiohjattua kehitystä sekä hyväksymistestiohjattua kehitystä. Näiden työkalujen avulla testaajat ja liiketoiminnan edustajat voivat kuvata odotettua järjestelmän käyttäytymistä taulukoilla tai luonnollisella kielellä avainsanoja käyttämällä.
- Tutkivan testauksen työkalut: Työkalut, jotka tallentavat ja kirjaavat sovelluksella suoritettuja toimintoja tutkivan testauksen aikana, ovat hyödyllisiä testaajalle ja kehittäjälle, koska ne tallentavat suoritetut toimenpiteet. Tämä on hyödyllistä vian löytämisen yhteydessä, koska ennen häiriön ilmaantumista suoritetut toimenpiteet on tallennettu ja niitä voidaan käyttää vian raportoimiseen kehittäjille. Tutkivan testauksen aikana suoritettujen vaiheiden kirjaaminen voi osoittautua hyödylliseksi, jos testi päätetään myöhemmin ottaa mukaan automatisoituun regressiotestauksen testijoukkoon.

3.4.6 Pilvilaskenta- ja virtualisointityökalut

Virtualisointi mahdollistaa yksittäisen fyysisen resurssin (palvelin) toimimisen kuten useat, pienet ja erilliset resurssit. Kun käytetään virtuaalikoneita tai pilvi-instansseja, tiimeillä on enemmän palvelimia käytössään kehitykseen ja testaukseen. Tämä voi auttaa välttämään viiveitä, joita esiintyy fyysisiä palvelimia odotellessa. Uuden palvelimen hankkiminen tai palvelimen palauttaminen on tehokkaampaa, sillä useimmissa virtualisointityökaluissa on sisäänrakennettuna ominaisuudet tilannekohtaisten tietojen tallentamista varten. Jotkut testauksenhallintatyökalut hyödyntävät nykyisin virtualisointiteknologioita tilannekuvan tallentamiseksi palvelimilta sillä hetkellä, kun häiriö havaitaan. Tämä mahdollistaa sen, testaajat voivat jakaa tilannekuvan vikaa tutkivien kehittäjien kanssa.

4. Viitteet

4.1 Standardit

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 ISTQB dokumentit

- [ISTQB_ALTA_SYL] ISTQB Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB_ALTM_SYL] ISTQB Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB_FA_OVIEW] ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus, Version 2011

4.3 Kirjat

[Aalst13] Leo van der Aalst ja Cecile Davis, “TMap NEXT® in Scrum,” ICT-Books.com, 2013.
[Adzic09] Gojko Adzic, “Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing,” Neuri Limited, 2009.
[Jaerson13] David Jaerson, “Kanban: Successful Evolutionary Change for Your Technology Business,” Blue Hole Press, 2010.
[Beck02] Kent Beck, “Test-driven Development: By Example,” Addison-Wesley Professional, 2002.
[Beck04] Kent Beck ja Cynthia Jares, “Extreme Programming Explained: Embrace Change, 2e” Addison-Wesley Professional, 2004.
[Black07] Rex Black, “Pragmatic Software Testing,” John Wiley ja Sons, 2007.
[Black09] Rex Black, “Managing the Testing Process: Practical Tools and Techniques for Managing Hardware ja Software Testing, 3e,” Wiley, 2009.
[Chelimsky10] David Chelimsky et al, “The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends,” Pragmatic Bookshelf, 2010.
[Cohn04] Mike Cohn, “User Stories Applied: For Agile Software Development,” Addison-Wesley Professional, 2004.
[Crispin08] Lisa Crispin ja Janet Gregory, “Agile Testing: A Practical Guide for Testers ja Agile Teams,” Addison-Wesley Professional, 2008.
[Goucher09] Adam Goucher ja Tim Reilly, editors, “Beautiful Testing: Leading Professionals Reveal How They Improve Software,” O'Reilly Media, 2009.
[Jeffries00] Ron Jeffries, Ann Jaerson, ja Chet Hendrickson, “Extreme Programming Installed,” Addison-Wesley Professional, 2000.
[Jones11] Capers Jones ja Olivier Bonsignour, “The Economics of Software Quality,” Addison-Wesley Professional, 2011.
[Linz14] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.
[Schwaber01] Ken Schwaber ja Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.
[vanVeenendaal12] Erik van Veenendaal, “The PRISMA approach”, Uitgeverij Tutein Nolthenius, 2012.
[Wiegers13] Karl Wiegers ja Joy Beatty, “Software Requirements, 3e,” Microsoft Press, 2013.

4.4 Ketterä terminologia

ISTQB-sanastosta löytyvät avainsanat on kuvattu jokaisen luvun alussa. Yleisissä ketterissä termeissä olemme turvautuneet seuraaviin yleisesti hyväksyttyihin määritelmiä tarjoaviin Internet -lähteisiin.

http://guide.Agilealliance.org/
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

Rohkaisemme lukijoita tarkistamaan nämä verkkosivut, jos he löytävät tästä dokumentista vieraita ketterän kehityksen termejä. Nämä linkit olivat aktiivisia tämän dokumentin julkaisun aikoihin.

4.5 Muita viitteitä

Seuraavat lähteet viittaavat Internetissä ja muualla saatavilla olevaan tietoon. Vaikka nämä lähdeviitteet on tarkistettu tämän sertifikaattisisällön julkaisun aikoihin, ISTQB:ta ei voida pitää vastuussa, jos ne eivät ole enää saatavilla.

- [Agile Alliance Guide] Various contributors, http://guide.Agilealliance.org/.
- [Agilemanifesto] Various contributors, www.agilemanifesto.org.
- [Hendrickson]: Elisabeth Hendrickson, “Acceptance Test-driven Development,” testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
- [INVEST] Bill Wake, “INVEST in Good Stories, ja SMART Tasks,” xp123.com/articles/invest-in-good-stories-ja-smart-tasks.
- [Kubaczkowski] Greg Kubaczkowski ja Rex Black, “Mission Made Possible,” www.rbcs-us.com/images/documents/Mission-Made-Possible.pdf.

5. Asiahakemisto