Ketterä testaus ja testaus ketterässä ohjelmistokehityksessä

Esitys kertoo ketterästä testauksesta ja sen soveltamisesta sekä ketterässä että myös perinteisessä ohjelmistokehityksessä sekä yleisemmin testauksesta ketterässä ohjelmistokehityksessä.

UPDATE 2010

Matti Vuori, www.mattivuori.net

Sisällysluettelo 1/5

Sisällysluettelo 2/5

Sisällysluettelo 3/5

Sisällysluettelo 4/5

Sisällysluettelo 5/5

Kalvosetistä

- Aiheena on tiivistetysti 1) ketterä testaus ja 2) testaus ketterässä ohjelmistokehityksessä.
- Oletetaan, että kuulijat ymmärtävät testauksen perusteet ja ketterän kehittämisen perusteet ja tuntevat Scrum-menetelmän perusteet.
- Tämä ei ole ainoa kalvosetti. Lisäaineistoa löytyy mm.
  - Scrum:sta.
  - Tehokkaasta vianetsinnästä.
  - Testausyksikön ohjeet toiminnasta Scrum-projekteissa.
  - Ja tietenkin testauksen kaikista osa-alueista; suurin osa systemaattisen maailman osaamisesta pätee edelleen – esimerkiksi toiminnallisuustestauksen tekniikat ovat aina yhtä päteviä.

Ketteryys ohjelmistokehityksessä

- Laajojen ennakkosuunnitelmien sijaan eletään ajan hermolla.
- Suunnitelmia muutetaan nopeasti tilanteiden niin vaatiessa.
- Ajatus, että ohjelmistokehitys on dynaamista ja jatkuva oppimisprosessi.
- Etukäteen ei voida tarkkaan tietää, miten tuote kehittyy ja mitä sen kanssa toimiminen edellyttää.
- Käsitykset ohjelmistosta muuttuvat jatkuvasti.
- Muutos otetaan vastaan positiivisena ilmiönä.

Ketterä testaus

- Ketterä (agile) testaus voi tarkoittaa erilaista testausta:
  - Mitä tahansa testausta, joka ei perustu testitapaustason ennakkosuunnitelmiin.
  - Tutkivaa testausta (exploratory testing), jossa testaaja etenee ohjelmistosta saatavien havaintojen perusteella.
  - Joskus se tarkoittaa testausta ketterässä ohjelmistokehityksessä.
  - Myös ohjelmistokehittäjien tekemä testilähtöinen kehitys luetaan usein ketterään testaukseen johtuen sen nopeudesta ja syklisyydestä.
- Vastapaino vanhalle ideaalille "suunnitelmaohjattu testaus", jossa periaatteessa jo hyvin varhaisessa vaiheessa, määrittelyjen pohjalta päätetään, mitä testataan ja miten.

Tutkiva testaus: Miten?

- Tutkitaan ohjelmistoa sitä käyttämällä ja tehdään siitä havaintoja, yritetään oppia ja ymmärtää sitä.
- Tunnistetaan riskialttiita ja testaamista edellyttäviä alueita.
- Testataan ne saman tien, ilman formaaleja testitapauksia.
- Tai suunnitellaan niille tarkemmat testit ja suoritetaan ne.
- Tulkitaan tuloksia ja jatketaan iterointia.

Tutkiva testaus: Miksi? 1/2

- Testaus on hyvä perustaa siihen, mitä toteutus kertoo.
  - Ollaan täydellisen realisteja. Ei luoteta dokumentteihin, vaan siihen, mitä on oikeasti saatu aikaan. Mitä toimintoja oikeasti on mukana?
  - Määrittelyt ovat aina vajaita ja virheellisiä. Ei jäädä puuttuvien määrittelyjen ansaan.
  - (Tietenkin on tiedettävä, miten ohjelmiston pitäisi toimia.)
- Aikaa ei kulu testauksen suunnitteluun, vaan saadaan nopeammin tuloksia.
  - Nopea testauksen käynnistys.
  - Nopea reagointi muutoksiin.
  - Saadaan nopeasti palautetta kehittäjille.
- Tutkiva testaus sopii tilanteisiin, joissa vaatimukset muuttuvat jatkuvasti.
  - Suunnitelmat olisivat muuten aina vanhentuneita.

Tutkiva testaus: Miksi? 2/2

- Kun ollaan avoimin silmin liikkeellä, löydetään paremmin virheitä kuin jos käytetään vain valmiiksi mietittyjä testitapauksia.
  - Liiallinen suunnittelu luo ennakko-odotuksia ja sulkee silmiä havainnoilta.
- Osataan tunnistaa sellaisia piirteitä ohjelmistosta, jotka eivät ole määrittelyvaiheessa nousseet esille.
- Ohjelmistot ovat niin monimutkaisia, että testitapauspohjainen lähestymistapa ei riitä.
  - Kaikkia testitapauksia ei pysty määrittämään. Ei ole aikaa!
- Toimintojen tekninen riskitaso paljastuu vasta, kun toteutus alkaa, tutkimalla ohjelmistoa.
- Sen avulla opitaan järjestelmästä.
- Se sopii käyttäjien näkökulmaan.
- Testaus on joka kerta erilaista, joten kyetään löytämään uusia virheitä.

Tutkiva testaus: Ongelmia

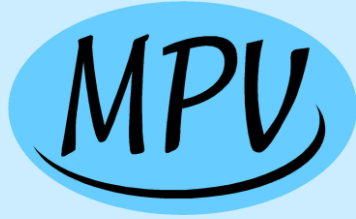
- Testauksen laadun todentaminen on vaikeaa.
- Tarvitsee taitoja ja ohjausta – muuten siitä on vähän hyötyä.
  - Huonosti tehtynä voi olla *hyvin* huonoa…
- Testauksen formaali osoittaminen vaikeaa.
  - Puuttuu speksejä, raportteja.
- Käytettävät tekniikat eivät kunnolla dokumentoituja.
- **Usein onkin hyvä, ettei rajoituta yhteen testaustyyliin – tutkiva testaus on vain kokonaisuuden osa.**

Tutkiva testaus: Esimerkkejä

- Tuotteen käyttäytymisen tutkiminen.
  - Miten juuri toteutettu uusi toiminnallisuus toimii?
- Ketterä savutestaus.
- Kattava ominaisuuksien testaus.
  - Järjestelmätestauksessa, hyväksymistestauksessa.
- Ketterä regressiotestaus. (Täydentämässä systemaattisia menettelyjä.)
- Oireiden taustalla olevien asioiden analysointi tarkemmalla tutkimisella.
  - Esimerkiksi kuormitustestauksen tulosten syiden selvittäminen tarkemmilla testeillä ja mittaroinnilla.
- Käytettävyystestauksen ensimmäinen vaihe, jossa pyritään ennen kaikkea ymmärtämään uutta ohjelmistokonseptia.

Tutkiva testaus: Sovellusalueita

- Ketterässä kehittämisessä.
- Systemaattisessa kehittämisessä.
- Asiakkaan hyväksymistestauksessa.
  - Eräät suomalaiset valtion virastot ovat siirtyneet pääasiassa ketterään testaukseen.



Tutkiva testaus perustuu strategioihin ja tietoon

- Ohjelmiston ymmärtämiseen.
- Havaintoihin – tunnistetaan ongelmien oireita ja alueita ohjelmistossa, joissa voi piillä virheitä?
- Strategioihin – millaisella mentaliteetilla virheitä etsitään?
  - Esim. ohjelmiston ”rikkominen”.
- Kokemustietoon – millaisia virheitä on aiemmin ollut ja millä alueilla?
- Testaus on intellektuaalista, haastavaa työtä.
- Testaaja on etsivä! Ei testitapauksia syöttävä robotti.
- Ks. kalvosetit ”Tehokkaan vianetsinnän periaatteet” ja ”Testaaja – robotti vai etsivä”.

Ohjelmiston ymmärtäminen kokeilemalla ja havainnoimalla

- Silmät avoinna kaikelle.
- Käyttöskenaarioiden kulku viitekehyksenä.
- Ohjelmiston erilaisten elementtien tunnistaminen.
- Tapahtumien tunnistaminen.
  - Miten ohjelma käyttäytyy, miten se reagoi.
- Tilojen tunnistaminen.
- Muutokset.
  - Tilasiirtymät.
  - Muutokset datassa.

Havaintojen tekeminen käyttäytymisestä

- Mikä on tuttua.
- Mikä on uutta, vierasta?
  - Millä logiikalla se toimii?
- Ohjelman reagointi
  - Käyttönopeus.
  - Erilaiset sekvenssit.
  - Erilainen data.
  - Ensimmäinen kerta ja sen jälkeen.
- Perinteiset ongelmat vastaavissa sovelluksissa ja tilanteissa.

Mentaliteetti testauksessa

- Kaikki on sallittua.
- Tiedetään, että virheitä on; ne on vain löydettävä.
- Pyritään rikkomaan ohjelmisto.
  - Ollaan kovakouraisia.
  - Bittejä saadaan aina lisää – antaa ohjelman kaatua.
- Hyödynnetään kokemusta.
- Hyödynnetään omaa ”vainua”.

Session lähtökohdat 1/2

- Tavoite.
  - Oppiminen vai ohjelmiston rikkominen vai jokin muu?
- Ohjelmiston ymmärtäminen.
  - Ohjelmiston tarkoitus ja käyttö.
  - Uusien toimintojen tarkoitus ja käyttö.
  - Mitkä asiat tässä tilanteessa tuottavat suurimman arvon asiakkaalle? Miten ne toimivat?
  - Siis: Mihin asioihin liittyy suurin riski siitä, että asiat eivät onnistu tai toimivat väärin?

Session lähtökohdat 2/2

- Ohjelmiston ja käyttäjän yhteistyön ymmärtäminen.
  - Millaisia käyttöskenaarioita on ja voi olla?
  - Mitä tiedetään tyypillisestä käytöstä?
  - Millaisia kaikkia poikkeuksia voi kuvitella.
  - Entä tahallinen väärinkäyttö?

Mielentila

- Tutkiva testaus on aivotyötä ja edellyttää sopivia olosuhteita.
- Ei aikapaineita (vaikka testaukseen voikin olla annettu vain tietty aika – se on vain raami.
- Realistinen ajatus bugeista ja valmius niiden näkemiseen missä tahansa.
- Suuntautuminen siihen, millä on oleellista.
- Valmius käsitysten muuttamiseeen uusien havaintojen myötä.

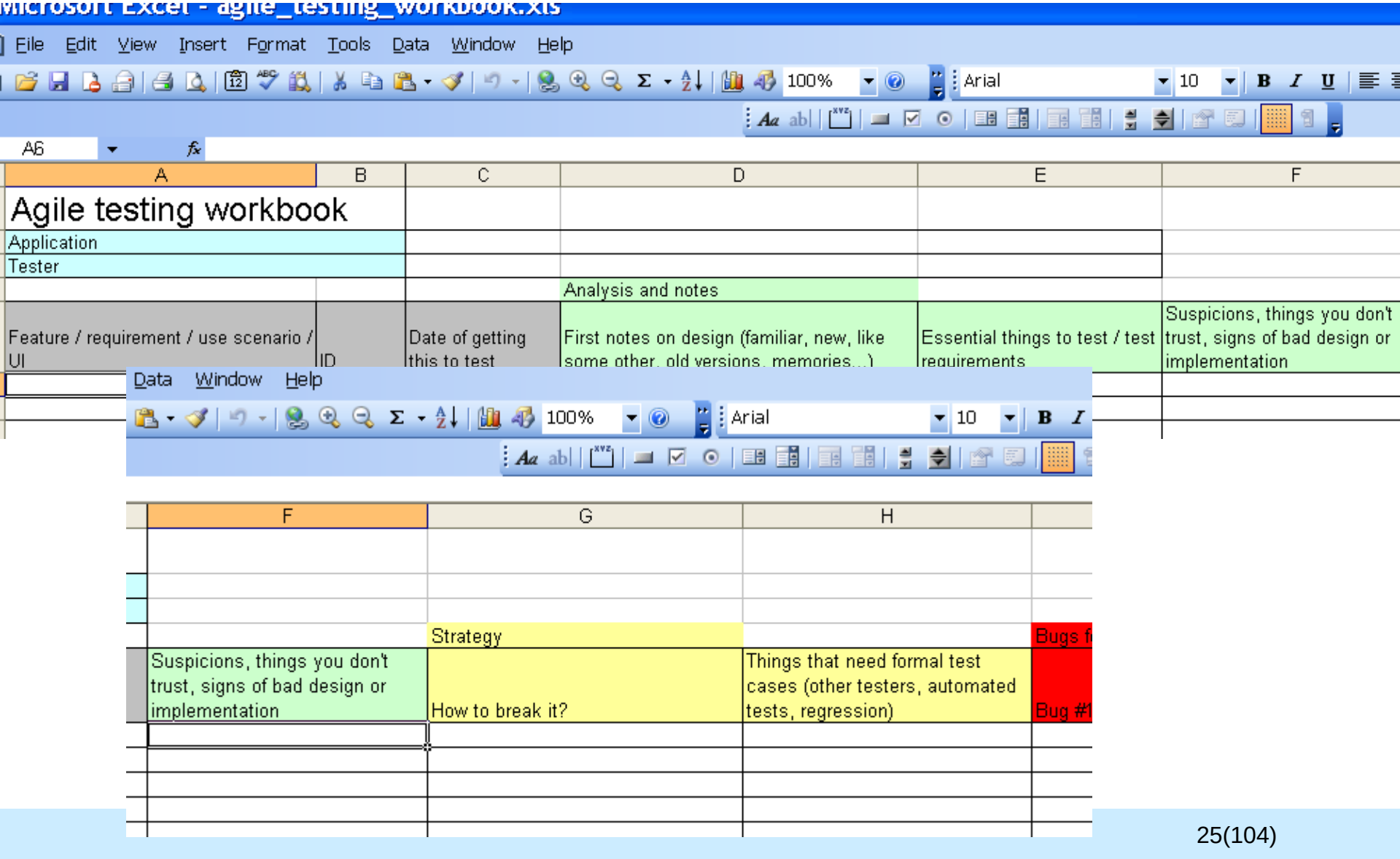
Session kulku

- Käyttöskenaariot, käyttötapaukset hyvä lähtökohta.
- Erityyppisten käyttäjien toimintamallit.
- Ei tarkkaa kuvausten antamaa ohjausta – vain viitekehys.
  - Tiukka seuraaminen olisi systemaattista testausta…
- Seurataan havaintoja, annetaan koetun suunnata testaksen kulkua.

Suorituksen dokumentointi?

- Testilokeja tarvitaan aina.
- Omien ajatusten ulkoistaminen edes kirjoittamalla tekee ajattelusta parempaa – ja parantaa testausta.
- Seuraavalla sivulla on yksi esimerkki ketterän testauksen logista.

Testauksen dokumentointilogi



Tutkiva testaus arjessa 1/2

- Ketterän testauksen tavat ovat väistämättä tärkeitä periaatteita.
- Käytännön ohjelmistoteollisuudessa ei voida toimia kirkasotsaisesti yksien periaatteiden mukaan – vaikka amerikkalaiset kirjailijat ja tutkijat niin ehdottaisivat.
- Tutkiva testaus on jo pitkään nähty tärkeänä osana hyvinkin suunniteltua testausprojektia.
  - Sitä on vain kutsuttu ad-hoc-testaukseksi.
- Hyvään testaustapaan kuuluu ylipäätään aina se, että testaustapoja muutetaan, kun saadaan uusia havaintoja tuotteesta ja vanhat tavat eivät enää paljasta virheitä.

Tutkiva testaus arjessa 2/2

- Olennaista on se, että tutkiminen myös tuottaa uusia testitapauksia, jotka otetaan mukaan päivitettyihin testispekseihin.
  - Uusi tieto saadaan jaettua kaikille testaajille.
  - Tämä vähentää riskejä.
- Termit ovat tärkeitä.
  - ”Tutkiminen” on helpompi hyväksyä systemaattisessa ohjelmistokehityksessä kuin minkäänlainen "ad-hoc"-toiminta.
- Ollakseen tehokkainta, tämä testaustapa pitää hyväksyä tärkeänä testausmenetelmänä ja siihen pitää antaa aikaa osaavimmille testaajille.
- *Mutta hyvä testaus koostuu erilaisista tavoista ja tyyleistä ja yhden ei ole hyvä antaa dominoida – ainakaan kunnes sen tiedetään toimivan erinomaisesti.*

Uusien toimintojen nopea testisuunnittelu

- Interaktion taso:
  - Lähtökohtana käyttäjätarina, käyttötapaus.
  - Toimii suoraan tutkivan testauksen lähtökohtana.
- Logiikan taso:
  - Perinteiset testaustekniikat – ekvivalenssiositus, raja-arvoanalyysi, päätöspuut, tilamallipohjainen testaus.
- Fyysinen taso:
  - Tapahtumien monitorointi ohjelmallisesti osana tutkivaa testausta.

Ennakkovarautuminen

- Ketterässä testauksessa on hyvä olla ennakkokäsityksiä testattavista asioita.
  - Niiden avulla on heti luotavissa käsitys siitä, mitä kaikkea uudessa toiminnallisuudessa on hyvä testata.
  - Luettelo tietynlaisten sovellusten ja toiminnallisuuksien yleisistä testattavista asioista. (esim. lista ”Ohjelmistojen yleiset testattavat asiat”.)
  - Kaikki uudetkin systeemit ovat jossain määrin vanhan toistoa.
  - Listat tyypillisistä virheistä tietynlaisissa toteutuksissa ovat hyödyllisiä.

Ketterä testaus ei-ketterässä projektissa 1/4

- Nykyaikainen testaus sisältää aina ketteriä piirteitä.
- Ymmärretään että:
  - Projekti ei koskaan tapahdu täysin ennakkosuunnitelmien mukaan.
  - Testaus tuottaa uutta tietoa, johon on reagoitava.

Ketterä testaus ei-ketterässä projektissa 2/4

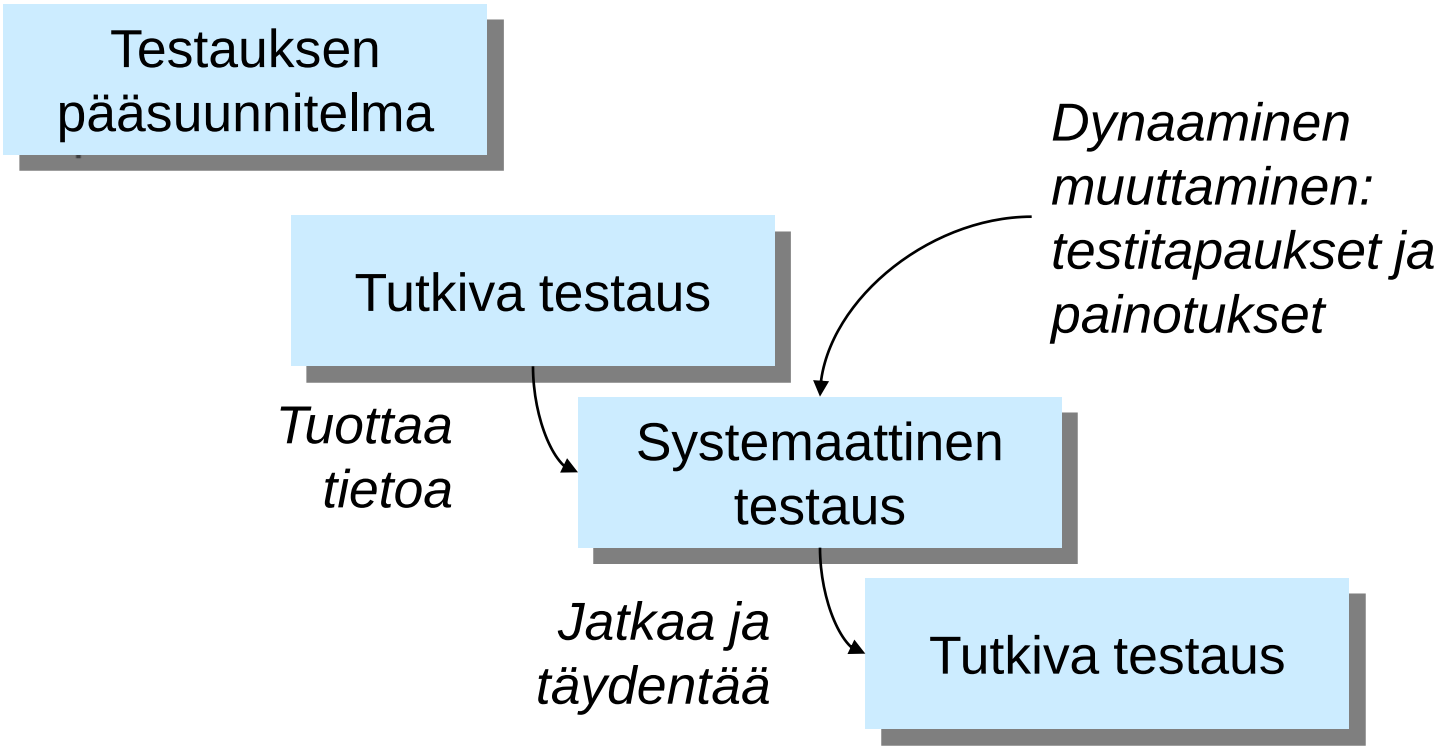
- Testaussuunnittelu
  - Testauksen W-malli, jossa testauksen peruspiirteet suunnitellaan projektin alussa, mutta tarkempi testaus sitten, kun tuote alkaa valmistua testattavaan kuntoon.
- Dynaaminen ohjaus
  - Vaikka tuotteella on buildaussuunnitelma, testaus sovitetaan ketterästi siihen, miten ohjelmiston osat valmistuvat.
  - Testauksen painotuksia eri osa-alueille muutetaan dynaamisesti sen perusteella, paljonko vikoja löytyy ja mikä on eri osa-alueiden riskitaso.
  - Jokaisen testikierroksen testisetti on viime kädessä tapauskohtainen valinta.
  - Testisettejä päivitetään asiakkailta ja sidosryhmiltä saatavien havaintojen perusteella.

Ketterä testaus ei-ketterässä projektissa 3/4

- Tutkiva testaus
  - Testauksessa on aina mukana vapaamuotoinen osuus.
  - Uuteen versioon tutustuminen tehdään tutkivalla testauksella. Se tuottaa käsityksiä systemaattisen testauksen kohteista ja on siten myös osa testaussuunnittelua.
  - Kun systemaattinen testaus ei enää tuota tehokkaasti uusia vikoja, siirrytään vahvemmin tutkivaan testaukseen.
  - Havaintojen perusteella päivitetään myös systemaattisen testauksen testisettejä.

Ketterä testaus ei-ketterässä projektissa 4/4

- Kokonaisuus on siis yhdistelmä ennakkosuunnittelua ja ketterää reagointia.



Riskialttiiden asioiden nopea verifiointi

- Ketteryydessä on keskeistä kyky tarttua nopeasti riskialttiisiin asioihin.
- Jos esimerkiksi projektissa toteutetaan uudentyyppinen protokolla, sen toimivuutta ei verifioida pelkästään normaalin testauksen puitteissa, vaan verifiointi pyritään tekemään välittömästi erottamalla protokollan toteutus ja testaamalla se niin pian kuin mahdollista.
  - Näin saadaan osoitettua, että protokollaa voidaan käyttää.
  - Siihen liittyvä riski voidaan sulkea.

Eräs kaksivaiheinen tutkivan testauksen malli

1. Uusille toiminnallisuuksille ensimmäisillä testauskierroksilla tehtävä testaus, jolla opitaan tuotteen käyttäytymisestä.
2. Myöhempien testauskierrosten ketterä testaus, joka jatkaa virheiden etsimistä sitten, kun systemaattinen testisuunnittelu ei enää löydä virheitä tehokkaasti. Samalla vaihdetaan likaiseen testiympäristöön.

- Näiden välissä systemaattisempaa testausta.

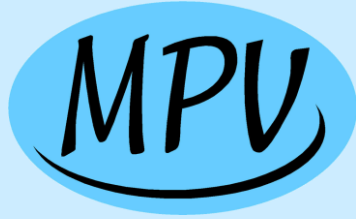
PET: Yleistä ja sovellustilanne

- Uuden toiminnallisuuden ensimmäisten testauskierrosten testaus: First round ad-hoc testing for new functionality – Preliminary Exploration Testing = PET testing
- Sovellustilanne:
  - Uutta buildia ei ole testattu systemaattisesti. Ei tiedetä, miten se toimii ja käyttäytyy.
  - Se on saattanut läpäistä automaattiset savutestit.

PET: Tavoitteet

- Opitaan uudet toiminnallisuudet.
- Tutkitaan ja ymmärretään sovellusta.
- Tarkkaillaan, miten se käyttäytyy.
- Havainnoidaan mahdollisia / tunnettuja ongelma-alueita.
- Löydetään virheitä.

PET: Taktiikka ja menettelyt



- Suoritetaan täysiä käyttötapauksia "kokeilevassa mielentilassa."
- Kokeillaan kaikkea ainakin kerran.
- Toimitaan puhtaassa testiympäristössä.
- Tehdään muistiinpanoja häiriöistä, hitaudesta, järjestelmän tilasta jne...
- Kokeillaan sovelluksen alueita, joilla on aiemmin ollut virheitä.
- Tunnista ja raportoi virheitä.
- Tarkistetaan myöhemmin muistiin (paperille ja mieleen) laitettujen asioiden pohjalta, että testispeksit kattavat kaikki epäilyttävät alueet.

FAT: Yleistä ja sovellustilanne

- Myöhäisten testikierrosten ad-hoc-testaus: Freeform Adaptive Testing = FAT testing
- Sovellustilanne
  - Uusi buildi on testattu systemaattisesti.

FAT: Tavoitteet

- Käytetään kokemusta ja virheiden arvausta uusien virheiden löytämiseen.

FAT: Taktiikka ja menettelyt 1/2

- Käytetään tutkivia tekniikoita virheiden löytämiseen.
- Käydään läpi alueita, joilla on ollut paljon virheitä.
- Testataan virallisten testitapausten "ympärillä" tavoitteena ohjelmiston "rikkominen".
- Suoritetaan kokonaisia käyttötapauksia.
- Testaus tehdään "likaisessa" testiympäristössä. Tämä auttaa ongelmiin törmäämisessä.
- Testausta muutetaan tuotteen käyttäytymisen perusteella. Keskitytään alueisiin, jotka ovat hitaita, joissa on odottamattomia ilmiöitä.

FAT: Taktiikka ja menettelyt 2/2

- Käytetään mielikuvitusta ohjelmiston monimutkaisilla alueilla.
- Aseta itsesi noviisin mielentilaan (joka ei ymmärrä mitään) ja expertin mielentilaan (joka kokeilee kaikkea, koska "siihen on oikeus"). Toimi kuin lapsi.
- Tee satunnaisia asioita. Tee samanaikaisia asioita. Keskeytä asioita (mobiililaitteen tapahtuma, PC:n tapahtuma, kollegan tekemä keskeytys!).
- Tee virheitä!
- Käytä muita periaatteita, jotka on kuvattu kalvosarjassa "Tehokkaan vianetsinnän periaatteet".
- Raportoi kaikki virheet.
- Tarkista testispeksit ja ehdota uusia testitapauksia.

FAT: Esimerkki karkean tason testisuunnitelmasta

1. Valitse rooli, jossa toimit (simuloi jotain käyttäjäryhmää käytettävien toimintojen osalta).
2. Valitse sen pohjalta testiympäristö.
3. Valitse käyttötapaukset. Priorisoi ne.
4. Tunnista prioriteetit
   - Uudet toiminnot, muutokset.
   - Millä alueilla on ollut virheitä.
   - Toimintojen prioriteetit (tuotteen ja valitun käyttäjäryhmän kannalta)
5. Suorita kokeileva testikierros
6. Suorita virheitä tekevä testikierros
7. Suorita kuormittava testikierros
8. Sovita toimintasi havaintoihisi. Improvisoi!

Ketterä ohjelmistokehitys

- Ketteriä malleja on monia erilaisia.
- Taustalla voi olla tiukkoja periaatteita kuten Agile Manifesto, mutta tosielämässä mallit muuttuvat realistisiksi ja kehittyvät.
  - Isojen projektien tarpeet ovat erilaisia kuin yhden tiimin tarpeet.
  - Esimerkiksi CMMI, ISO 9001 ja muut prosessivaatimukset eivät ole ketterän toiminnan vastapooli, vaan joukko järkeviä asioita, jotka on otettava huomioon myös ketterissä prosesseissa.
- Siksi on suhtauduttava varovasti niihin asioihin, jotka liitetään ketterään kehitykseen.
  - Niitä on sovitettava ja täydennettävä vastaamaan kunkin ohjelmistokehitystilanteen tarpeita.

Ketterän ohjelmistoprojektin karakterisoivat piirteet 1/4

- Inkrementaalisuus.
  - Kehitystä tehdään lyhyissä inkrementeissä, kenties kahden viikon tai 30 päivän stepeissä. (Stepeistä käytettävä termi vaihtelee eri malleissa, Scrumin puhuu ”sprinteistä”.)
- Syklisyys.
  - Inkrementit toteutetaan samanlaisena toistuvana prosessina.
- Suunnittelun lyhyt aikajänne.
  - Konkreettisia suunnitelmia tehdään vain seuraavaa inkrementtiä varten.
- Hyvin suunniteltu, yksinkertainen ohjelmistoprosessi.

Ketterän ohjelmistoprojektin karakterisoivat piirteet 2/4

- Tiimin sisäinen dynamiikka.
  - Tiimityö ja usein parityöskentely (esim. ohjelmistokehittäjä ja testaaja).
  - Koodin ja muun yhteinen omistajuus.
- Tuotteen pitäminen kunnossa, jotta on stabiili versio suunnanmuutoksia varten.
  - Jokaisen inkrementin jälkeen testattu, dokumentoitu jne…
  - Usein testauslähtöinen kehittäminen. (Esim: Tee yksikkötesti ensin, sitten vasta koodi.)
- Viestintäintensiivisyys.
  - Tilanne koko ajan selvillä.
  - Paljon keskustelua.
  - Tiimin seinätaulut ja muut tietojärjestelmät.
- Asiakas mukana kehittämisessä – jopa päivittäin.

Ketterän ohjelmistoprojektin karakterisoivat piirteet 3/4

- Arvostuksia (Agile Manifestosta, www.agilemanifesto.org)
  - Yksilöt ja kommunikointi > prosessit ja työkalut (on huomattava, että toteutustasolla ja testauksessa työkaluilla on aivan kriittinen rooli!).
  - Toimiva ohjelmisto > kattava dokumentaatio.
  - Asiakasyhteistyö > sopimusneuvottelu.
  - Muutokseen vastaaminen > suunnitelman noudattaminen.

Ketterän ohjelmistoprojektin karakterisoivat piirteet 4/4

- Agile Manifestosta poimittuja periaatteita:
  - Pyrkimys asiakastoimitusten nopeaan aloittamiseen ja säännölliseen jatkamiseen. Tavoitteena asiakkaan tyytyväisyys.
  - Muutoksia ei hyljeksitä, vaan ne ovat tervetulleita.
  - Liiketoimintaihmisten ja kehittäjien päivittäinen yhteistyö.
  - Ohjelmisto toimii tärkeimpänä mittarina.
  - Kehittäjien motivaatio. Hyvä ympäristö ja välineet.
  - Viestintä on tehokkainta kasvotusten.
  - Tekninen loistavuus ja hyvä suunnittelu parantavat ketteryyttä.
  - Yksinkertaisuus on oleellista.
  - Paras työ kumpuaa itseorganisoituvasta tiimistä.
  - Tiimin pitää arvioida toimintaansa ja kehittää sitä säännöllisin väliajoin.

Ketterän ohjelmistoprojektin laadunvarmistus 1/2

- Ohjelmistokehittäjillä päävastuu.
  - Laadunvarmistus integroitu kehittämiseen.
  - Kehittäjät toimittavat eteenpäin laadukasta koodia.
- Laadun rakentaminen pienissä palasissa.
- Tekninen laatu koko ajan kunnossa.
  - Testauslähtöinen kehitys.
  - Jatkuva integrointi (& integrointitestaus) .
  - Muut kurinalaiset käytännöt.
  - Automaattiset ”hyväksymistestit” (kehittämistiimin verifiointi, että asiakasvaatimukset toteutuvat).

Ketterän ohjelmistoprojektin laadunvarmistus 2/2

- Laadun seuranta tiheällä rytmillä.
  - Ei pääse syntymään patologisia ongelmia.
  - Laatu koko ajan näkyvillä.
  - Riskitaso alhainen ja koko ajan selvillä.
- Testit mittaavat edistymistä.
- Realistinen tuoteymmärrys.
  - Suullinen viestintä varmistaa, että ei luoteta virheellisiin dokumentteihin.
  - Keskustellen selviää, mitä asiat tarkoittavat.

Scrum?

- Scrum on noussut yhdeksi ketterän ohjelmistokehityksen de-facto-standardiksi.
- Usein kuvitellaan, että tietyt käytännöt, kuten jatkuva integrointi ja automatisoidut hyväksymistestit ovat osa Scrumia, mutta se ei ole totta.
- Scrum on ennen kaikkea projektinhallintamenetelmä.
- Se ei sisällä ohjelmistotuotannon tekniikoita tai suoria vaatimuksia niille.
- Kunkin kontekstin ammattilaisten on mietittävä, mitä Scrumin sprinttien puitteissa tehdään!
  - Eli testauksen näkökulmasta: Miten kaikki se testaus, jota oikeasti tarvitaan, saadaan tehtyä Scrum-pohjaisessa projektissa.

Scrum:ssa on paljon mahdollisuuksia testaukselle

- Scrum on testauksen kannalta erinomainen:
  - Sen avulla saadaan testausta tapahtumaan hallitusti projektin kuluessa, alusta alkaen.
  - Scrum-kulttuuri tukee testausta siellä, missä se on kriittisintä: yksikkö- ja integrointitasolla.
  - Järjestelmätestaukselle on paljon mahdollisuuksia rytmisen toiminnan ansiosta.
  - Käyttäjätestaukselle on monia loogisia paikkoja, jotka tarjoavat potentiaalia (julkistusten hyväksymistestaus ja arviointi).
- Monitasoisissa testausprosesseissa on kuitenkin vielä sovittamista. (Tätä käsitellään myöhemmin.)

Ketterien mallien testauksen hyviä piirteitä

- Panostus matalien testitasojen testaukseen.
  - Tehdään robustia koodia. Regressioriski hallitaan matalilla testitasoilla.
- Testauslähtöisyys.
  - Testit ovat aina olemassa.
  - Testisettiä kasvatetaan koko ajan.
  - Testejä ajetaan koko ajan.
- Testauksen automatisointi.
  - Automatisoidut testit yksikkö-, integrointi ja hyväksymistestauksessa ovat tehokkaita.
  - Regressiotestaus toimii.
- Testaus yhdistää eri näkökulmia luonnollisella tavalla.
  - Toteutus, integrointi, asiakas, regressio.

Ketterien mallien yleisiä testauksen patologisia ongelmia 1/2

- Huono testisuunnittelu.
  - Eri osapuolten yhteistyö ei edusta parasta osaamista, vaan huonoa kompromissia.
- Automatisointi-ajattelu ohjaa testisuunnittelua.
  - Testataan sitä, mitä voidaan testata automaattisesti.
- Unohdetaan testauksen tarvittavat tasot ja kokonaisprosessi.
  - Ylempien testaustasojen asianmukainen toteutus.
  - Testaus eri aikajänteillä.
  - Toimitusten ja tuotteenhallinnan edellyttämä testaus.
- Ammattilaistestaajien heikko hyödyntäminen.

Ketterien mallien yleisiä testauksen patologisia ongelmia 2/2

- Ei ymmärretä asiakkaan hyväksymistestauksen kokonaisuutta.
  - Automatisoidut hyväksymistestit ovat vain pieni osa hyväksymistä.
  - Joskus asiakkaan roolissa on vain ”product owner” tai asiakkaan yksi edustaja, jolloin taannutaan perinteiseen edustukselliseen kehittämiseen
- Käytettävyyden varmistamiseen ei riittävän hyvää panostusta.
  - Luotetaan asiakkaan tekemään testaukseen, mutta se ei riitä!

Ketterä testaus ketterässä ohjelmistokehityksessä

- Mennäänpä sitten ketterän testauksen haastavampaan maailmaan…
- Seuraavassa puhutaan ketteristä malleista yleensä, mutta jonkinlaisena viitekehyksenä on Scrum-prosessi.
- Scrumia ei kuitenkaan käsitellä sellaisenaan, koska kirjoittaja edustaa sitä koulukuntaa, että kaikki prosessimallit on aina tarpeen sovittaa kulloiseenkin kontekstiin.
  - Jokainen yritys ja yksikkö voi tarvita oman räätälöidyn prosessin.

Testauksen asennemuutos 1/3

- Testaus periaatteessa vastustaa muutosta, koska:
  - Se rikkoo suunnitelmat.
  - Tuottaa turhaa testaustyötä
  - Muutokset ovat aina huonosti dokumentoidut.
- …Mutta näin testaus on parantamista vastaan…
- => Ketterässä kehittämisessä on hyväksyttävä muutokset ja ymmärrettävä, että (hyvä) muutokset ovat tervetulleita.
- Inkrementtien sykli on hyvin ripeä.
  - On ehdittävät tehdä softan suunnittelu, toteutus, testaus, viankorjaus ja korjausten verifiointi.
- => Testauksen suunnittelu, toteutus ja suoritus ja viestintä on tehtävä aiempaa nopeammin.

Testauksen asennemuutos 2/3

- Viestintätapoja on kyseenalaistettava.
  - Jos ketterä projekti panostaa suulliseen viestintään, on päästävä siihen mukaan.
- => Erillään toimiva testaustiimi voi olla mahdoton (riippuu tiimin tehtävistä). Testaajien on elettävä tiimissä. Tämä uudistaa myös toimintatyyliä.
- Silti osa testausta tehdään kehittämistiimistä erillään.
- Kehittämistiimillä on vahva vuorovaikutus asiakkaan kanssa.
  - Mahdollisesti päivittäinen yhteistyö.
- => Myös testaajien on kyettävä toimimaan asiakkaan kanssa.

Testauksen asennemuutos 3/3

- Termit ja käsitteet muuttuvat.
  - Ketterässä prosessissa voi olla omanlaisensa merkitys vaikkapa hyväksymistestaukselle.
- => On hyväksyttävä se, että kehittämisprosessi ja projekti määrittävät käsitteet ja termit, joita käytetään.
- Mutta ketterät prosessit eivät ole ideaalisia!
  - Ne ovat usein jossain mielessä vajaita ja yksisilmäisiä.
  - Niitä kehittämässä ei ole ollut testauksen mestareita.
- => Testaajat voivat auttaa kehittämään malleja paremmiksi.

V-malli edelleen oleellinen

- Ketterä ohjelmistokehitys on luonteeltaan erilaista kuin vesiputousmalliin tai perinteinen inkrementaaliseen kehittäminen.
- Se on myös kehittäjälähtöistä ja painottaa robustin koodin tekemistä ja jatkuvaa komponenttitason integraatiota.
- Kaikki V-mallin esittämät testaustasot ovat edelleen oleelliset, esim.
  - Yksikkötestaus, integrointitestaus, järjestelmätestaus, järjestelmä-integrointitestaus, hyväksymistestaus.
  - Niitä voidaan toteuttaa uudella tavalla syklisesti, mutta tasojen olemassaolo ja vaatimukset on tärkeää ymmärtää.

Yksikkötestaus ketterässä ohjelmistokehityksessä

- **Yleinen konsensus on sille, että hyvä yksikkötestaus, testauslähtöisesti tehtynä, on pakollinen edellytys ketterän ohjelmistokehityksen onnistumiselle.**
- Ketterät ohjelmistokehitysmallit ovat pääsääntöisesti testilähtöistä.
- Ennen koodia sille tehdään testit testausohjelmassa.
  - Yleensä luokkaa vastaava testiluokka.
- Testien ajo integroidaan osaksi kehittäjän henkilökohtaista buildausprosessia.
- Testaus aloitetaan ennen koodausta.
  - Ekalla kertaa ne failaavat.
  - Toteutuksen edistyessä testit alkavat mennä läpi.
  - Testien läpäisyaste kertoo toteutuksen edistymisen.
- Testausohjelma tuottaa automaattisia raportteja testien läpäisystä.

Integrointitestaus ketterässä ohjelmistokehityksessä

- Ketterät ohjelmistokehitysmallit soveltavat yleensä jatkuva tai hyvin taajaa integrointia (esim. päivittäinen).
- Muuten ohjelmisto ei pysy eheänä kehittämisen aikana.
- Regressioriskin hallitsemisen strategia: softa toimii kehittämisen ajan jatkuvasti.
- Integrointitestit on yleensä automatisoitu.
- Pääosa niistä on koottu kehittäjien yksikkötestauksen testiseteistä.
- Idea on se, että integrointiin tuotu koodi toimii ja ei tuota virheitä integrointitesteissä. Integrointitestejähän on jo simuloitu kehittäjän työasemassa.

Järjestelmätestaus ketterässä ohjelmistokehityksessä 1/4

- (Tässä käsitellään sitä yleensä ja toiminnallisuustestauksen kannalta, joitakin erityistestejä käsitellään erikseen.)
- Ketterässä ohjelmistokehityksessä järjestelmätestauksen pitää olla samantyyppistä kuin ei-ketterässä toiminnassa.
- Oleellista on suunnittelun aikajänne.
  - Suunnitelmat tehdään seuraavaa julkistusta varten – kenties kahden viikon päästä.
  - Koska seuraavan julkistuksen speksi on selkeä, testaussuunnittelu tehdään heti.
  - Testit suunnitellaan yhteistyössä kehittäjien kanssa. Kenties he osaavat automatisoida monet testit?
  - Suunnitelmat ovat pieniä, koska ne koskevat vain uusia ominaisuuksia.

Järjestelmätestaus ketterässä ohjelmistokehityksessä 2/4

- Aikajänteen vuoksi testaus aloitetaan mahdollisimman pian.
  - Heti, kun uusi ominaisuus on integroitu.
  - Testausta varten ei tuoteta uutta buildia, vaan joka päivä on kunnossa oleva buildi.
- Koska toiminta on dynaamista, käytetään toiminnallisuustestauksessa pääosin tiimiin integroituja testaajia erillisen tiimin sijaan.
  - Heti kun tiimi saa jotain aikaiseksi, sen testaus aloitetaan.
  - Suora vuorovaikutus viankorjauksessa kehittäjien kanssa.
  - Kuitenkin myös vikakanta tärkeä, jotta tiedetään tuotteen tila.
- Testaus on siis sekoitus ketterää ja ei-ketterää testausta.

Järjestelmätestaus ketterässä ohjelmistokehityksessä 3/4

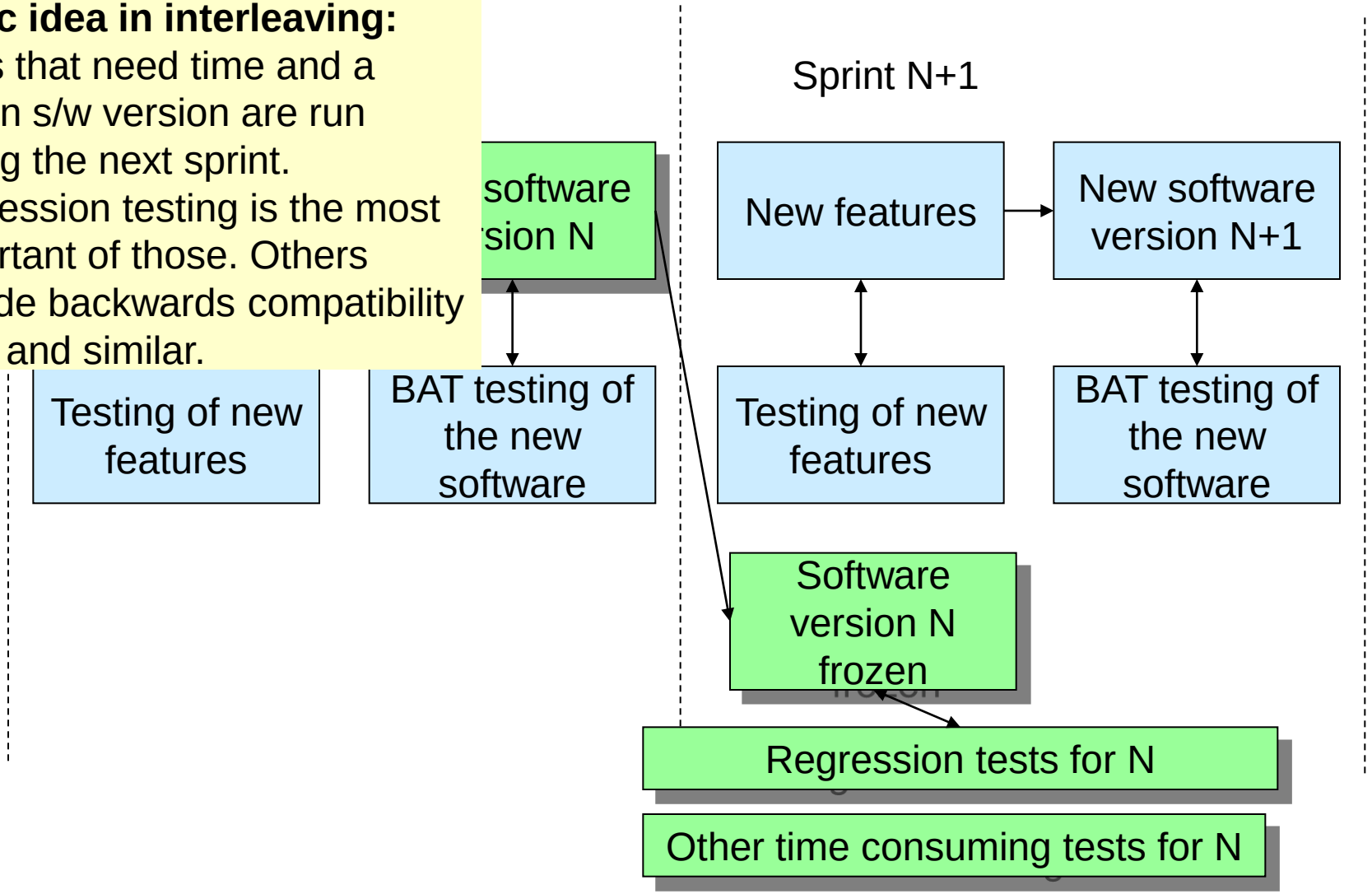
- Testauksen lomittaminen sprinttien yli.
  - Oleellista on säilyttää tekemisen rytmi.
  - Kaikki testaus, jossa ohjelmistokehittäjillä on rooli, on tehtävä toteuttavan sprintin aikana.
  - Mutta testaustiimin tekemä syvällisempi testaus ja erikoistestit voidaan tehdä seuraavan sprintin aikana.
  - Samalla vaihdetaan testausympäristöä. Tämä on oleellista!

Järjestelmätestaus ketterässä ohjelmistokehityksessä 4/4

- Testaajien tehtäviä samassa sprintissä:
  - Uuden toiminnallisuuden perustestaus ketterällä tekniikalla.
  - Heti, kun saadaan impulssi kehittäjältä, että jotain voisi testata.
  - Uuden toiminnallisuuden systemaattinen testaus, aloittaen positiivisilla testeillä.
  - Nopeasti, että kehittäjät saavat tukea työlleen.
  - Testaajien auttaminen testien automatisoinnissa.
- Seuraavassa sprintissä (jos on tarpeen):
  - Vaativampi negatiivinen testaus, yhteentoimivuustestaus, robustiustestaus, järjestelmätason regressiotestaus jne…

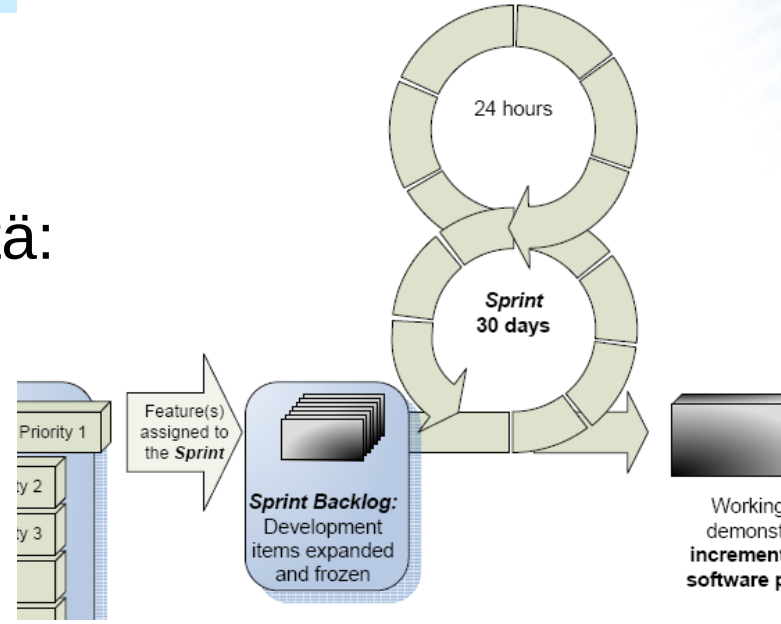
Yhteenveto testauksen lomittamisesta

Basic idea in interleaving:
Tests that need time and a frozen s/w version are run during the next sprint. Regression testing is the most important of those. Others include backwards compatibility tests and similar.



Välihuomio: Scrum ja kontrollin aikajänteet 1/2

- Yleinen ketterä projektinhallintamenetelmä
  Scrum tuntee kolme aikajännettä:
  - Päivä.
  - Sprintti.
  - Koko Scrum.
- Asioiden jakaminen useille sprinteille ei sovi sen filosofiaan.
- Mitä tehdä?



Välihuomio: Scrum ja kontrollin aikajänteet 2/2

- Ratkaisuja:
  - Vähennetään sprintin aikaisia tehtäviä, jotta kaikki tarpeellinen saadaan tehtyä. Joskus se onnistuu.
  - Käytännön on mentävä filosofian edelle.
  - Olennaista on tiimin dynamiikka, perustiimillä on yksi rytmi, mutta sen yläpuolella voi olla muita rytmejä ja jatkuvia prosesseja.
  - Muissa ketterissä prosesseissa voi olla erilaisia syklejä.
- Perusfilosofia on tärkein: Projekteissa on tehtävä tarpeelliset asiat ja perusprosesseja on sovitettava tarpeisiin.

Järjestelmäintegrointitestaus ketterässä ohjelmistokehityksessä

- Järjestelmäintegrointi tuottaa usein suurimmat ongelmat.
  - Se, mitä yksi Scrum-tiimi tuottaa, toimii hyvin, mutta tiimien tuotokset eivät toimi yhdessä tai muiden järjestelmien kanssa!
- Järjestelmäintegrointitestaus on pari järjestelmätestaukselle.
- Usein se on osa asiakkaan hyväksymistestaus.
- Vaatii aikaa, kokeilua ja harjoittelua ja ympäristöjen luomista ja hallintaa.
- Se pitää aloittaa heti ensimmäisessä sprintissä.
- Se on erillisten testaajien / tiimiin tehtävä.

Käytettävyystestaus ketterässä ohjelmistokehityksessä

- Käytettävyyden arviointi ja testaus on perinteisesti huonolla tolalla ketterissä malleissa.
- Luotetaan enemmän mukana olevaan asiakkaaseen, joka kuitenkin on usein vain yksilö, joka ei voi vastata kaikkia kohderyhmiä.
- Ketteriin malleihin voidaan sovittaa perinteisiä käytettävyyden arvioinnin ja testaamisen menettelyjä.
- Ennen ensimmäistä inkrementtiä tai sen aikana syntyy käyttöliittymäkonsepti, joka edellyttää analysointia.
- Uusille käyttöliittymille tehdään nopeat analyysit joka inkrementissä.
  - Myös käyttäjätestit silloin, kun luodaan uusi käyttöliittymä tai siihen tehdään suuria muutoksia.
- Käytettävyystestauksen testisuunnittelu ja –raportointi ovat perinteisesti hitaita.
  - On käytettävä sellaisia henkilöitä, jotka pystyvät toimimaan nopeasti.

Suorituskykytestaus ketterässä ohjelmistokehityksessä

- On oleellista se, että arkkitehtuuristen ratkaisujen päättämisen jälkeen todennetaan perusratkaisujen riittävä suorituskyky.
- Tämä merkitsee sitä, että ensimmäisissä inkrementeissä on tärkeää toteuttaa kriittiset arkkitehtuuriset kysymykset ja tehtävä niitä vastaava testaus.
- Testaus toistetaan arkkitehtuurin muuttuessa ja asiakastoimituksiin tähtäävien buildien toteutusvaiheessa.
- Kuormitustestauksen myöhempien vaiheiden tärkeyteen vaikuttaa se, millaiseen käyttöön otetaan väli-inkrementtejä – niiden onnistunut käyttö voi joskus testaamatta osoittaa, että suorituskyky on riittävä.
- Kuormitustestien automatisointi ja testien säännöllinen ajaminen sopii hyvin ketterän kehittämisen kulttuuriin.
  - Hyvin suunniteltuja perustestejä voidaan ajaa vaikkapa kerran päivässä.

Tietoturvatestaus ketterässä ohjelmistokehityksessä

- Tietoturvallisuuden varmistamisessa on olennaista käyttäjien toimintajärjestelmän taso.
- Siihen vastataan tietoriskianalyysillä, joka voidaan pitkälti tehdä jo konseptivaiheessa, ennen inkrementaalista kehittämistä.
- Tähän on palattava viimeisten inkrementtien aikana.
- Testattavat tieturvavaatimukset on tehtävä asiantuntijavoimin.
- Niiden verifiointi tehdään dynaamisesti niiden toteutuksen aikana.
- Pääosa vastuusta pitää olla ohjelmistokehittäjillä.
- Täydentäviä testejä tekevät tietoturvatestauksen ammattilaiset sopivassa kehitysvaiheessa.
- Pääperiaate: Jokaisen asiakastoimituksen tietoturvallisuuden pitää olla kunnossa.

Fyysisen turvallisuuden validointi ketterässä ohjelmistokehityksessä 1/2

- Koskee turvallisuuskriittisiä automaatiojärjestelmiä ja IEC 61508 –standardin soveltamista.
- Turvallisuuden varmistaminen edellyttää ennen kaikkea hyvää kokonaisprosessia, joka perustuu perinteisen maailman käsityksiin: suunnittelua, dokumentointi, validointia ja verifiointia, konfiguraationhallintaa jne…
- Siis ISO 9001:n, CMMI:n, IEEE-standardien maailmaa.
- Onnistuukohan sama ketterässä kehittämisessä?
  - Onnistuu varmasti, mutta työtä se vaatii…

Fyysisen turvallisuuden validointi ketterässä ohjelmistokehityksessä 2/3

Vaatimuksia validoinnille	Vaikutuksia prosessille
Järjestelmän turvallisuusominaisuuksien validointi turvallisuusvaatimuksia vasten	Vaatimukset on tunnistettava ja kuvattava. Pelkät XP-tyyliset käyttöskenaariot eivät riitä. Turvallisuusanalyysejä on tehtävä käyttöskenaarioille.
Kelpoistuksen kohteena on kokonaisjärjestelmä	Ohjelmisto tehdään hyvin, mutta sen toiminta systeemissä ratkaisee
Tunnettava konfiguraatio, joka validoidaan – muutosten hallinta	Kaikista muutoksista on pidettävä kirjaa. Niiden vaikutukset ja riskit on analysoitava ja testaus suunniteltava. Viimeisiä validointitestejä varten on softa jäädytettävä.
Koko laadunvarmistusketjun oltava kunnossa – analyysit, testit	Ketterään prosessiin on tuotava mm. koodikatselmoinnit, FMEA, laadukkaat järjestelmätason testit
Turvallisuustoimintojen oikea toiminta	Toiminnot on kuvattava ja toiminta on

Fyysisen turvallisuuden validointi ketterässä ohjelmistokehityksessä 3/3

Vaatimuksia validoinnille	Vaikutuksia prosessille
Turvallisuustoimintojen luotettavuus - erityisesti MTTF	Luotettavuus on analysoitava. Muutosten jälkeen analyysi on päivitettävä. Refaktorointien vaikutus selvitettävä.
Validointisuunnitelma, jossa kriteerit. Olennainen niistä on järjestelmän eheystaso	On tehtävä laadunvarmistussuunnitelma ja sen mukana validointikriteerit. Se voi elää projektin myötä, eli päivitettävä sprinteissä.
Testien suunnittelu ja dokumentointi tärkeää	Järjestelmätestaus ei voi olla pelkkää tutkivaa testausta. Järjestelmätestit on suunniteltava (testien kohteet ja testaustekniikat) ja dokumentoitava. Ketterä testaus sitten tämän lisäksi.

Regressiotestaus ketterässä ohjelmistokehityksessä 1/3

- Ketterälle projektille on ominaista:
  - Jatkuva muutos.
  - Toimivienkin asioiden refaktorointi, uudelleenrakentaminen tarvittaessa paremmiksi, ylläpidettävämmiksi jne.
- Tähän liittyy valtava regressioriski.
- Toisaalta, koko projekti on usein rakennettu hyvin regression sietäväksi:
  - Kehittäjien ja integroinnin automaattisesti suoritettavat testisetit saavat rikki menneet asiat kiinni.
  - Lisääntyvä käyttöliittymätason testausautomaatio ketterien menetelmien ”hyväksymistestauksessa” mahdollistaa regression tunnistamisen myös järjestelmätestaustasolla.

Regressiotestaus ketterässä ohjelmistokehityksessä 2/3

- Mutta entä jos testausta ei ole voitu riittävästi automatisoida – kuteen yleensä onkin.
- Tällöin tarvitaan vahva regressiotestauksen panostus.
  - Toteutusta seuraavan sprintin aikana.
  - Jatkuvana prosessina inkrementaalisen kehittämisluupin ympärillä.
- Jokaisen toimituksen regressiotestaus on aina oltava riittävän kattavaa ja laadukasta.

Regressiotestaus ketterässä ohjelmistokehityksessä 3/3

Testauksen automaatioaste		
Yksikkötestaus ja integrointitestaus automatisoitu Paljon järjestelmätason perustestejä (savutestit, perus UI-testejä, regressiotestejä)	Ei softan jäädytystä Sprintin lopussa Käsin tarvittaessa seuraavan aikana – järjestelmä ja järjestelmäintegrointi	Softan jäädytys Sprintin lopussa automaattiset Manuaaliset seuraavan aikana – järjestelmä ja järjestelmäintegraatio
Heikosti automatisoitu	Version jäädytys Manuaalisesti seuraavassa sprintissä	Version jäädytys Manuaalisesti seuraavassa sprintissä
	Ei erityisiä luotettavuus-, laatu- ja turvallisuusvaatimuksia	Kriittisen tason vaatimukset
		Softan vaatimustaso

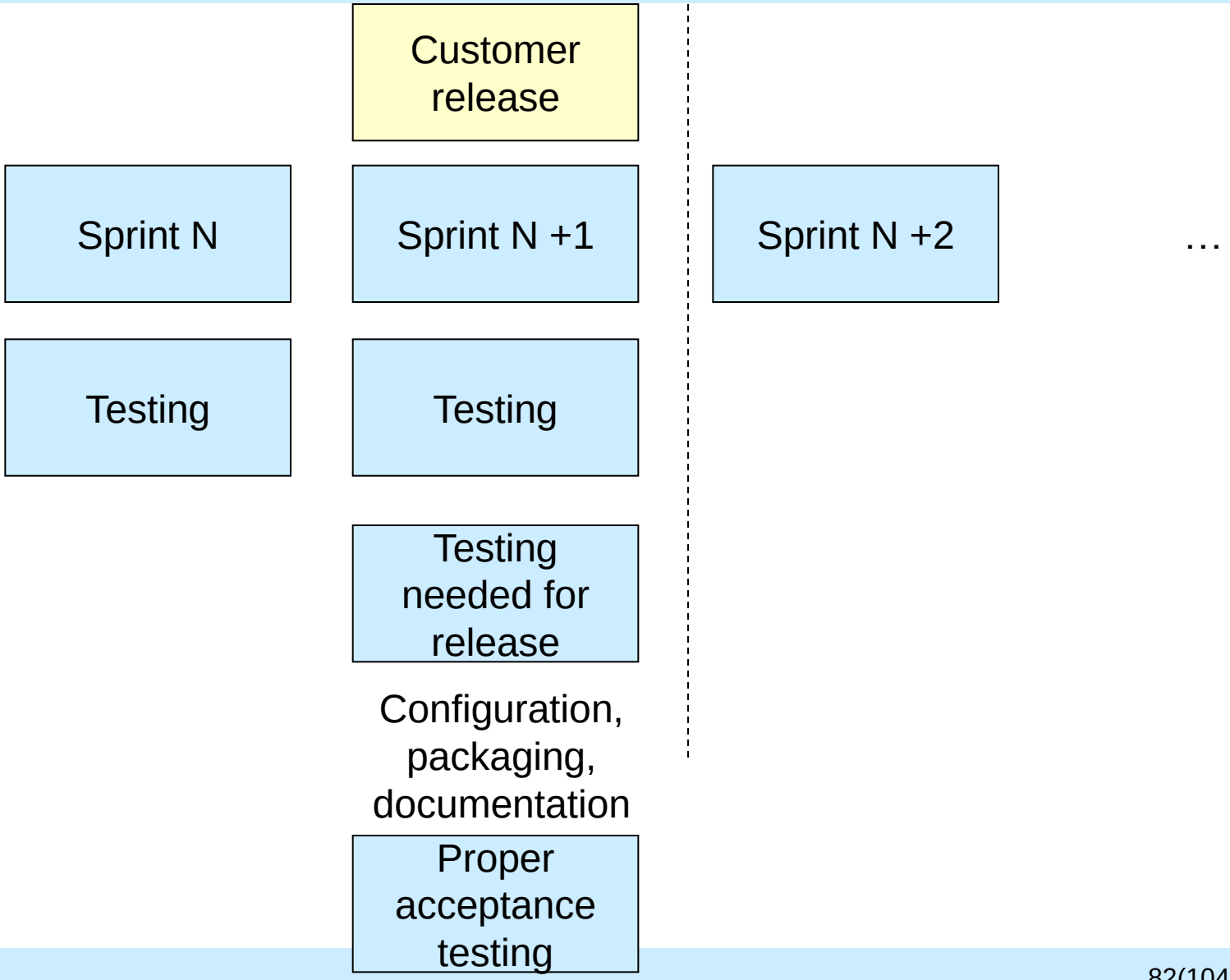
Ketterä regressiotestaus?

- Regressiotestaus on yleensä hyvin systemaattista.
  - Testit jotka kohdennetaan juuri muuttuneisiin asioihin tai niiden analysoituihin mahdollisiin vaikutuksiin.
  - Ennalta suunniteltu vakioitu regressiotestisetti.
- Nämä on joko automatisoitu tai toteutettu manuaalisina testitapauksina.
- Näitä voidaan täydentää ketterällä testauksella. Esim.
  - Asiat, jotka on vaikeaa ilmaista testitapauksina.
  - Täydet käyttöskenaariot.
- Osaavan testaajan havainnointi on aina etu missä tahansa testauksessa, koska systemaattiset testit ovat aina vain mekanistisia näytteitä.

Erityyppiset inkrementit 1/2

- Inkrementaalinen, ohjelmistokehityksen välivaiheiden testaus voi olla hyvin ketterää, mutta tuotantoon tarkoitettujen asiakastoimitusten (varsinkin viimeisen toimituksen) testauksen pitää olla hyvin suunnitelmallista.
  - Silloin on fokuksena eheän tuotteen kattava testaus.
- Koska rytmi on oleellista, kannattaa asiakastoimituksetkin rytmittää, esim. kolmen sprintin välein.
  - Silloin niiden edellyttämät erityistoimenpiteet voidaan rytmittää sopivasti.
- Tarvitaan siis ”julkaisusuunnitelma”, jossa on oleellisena piirteenä julkaisun rytmi – sisältöjä ei tunneta etukäteen.

Erityyppiset inkrementit 2/2



Hyväksymistestaus ketterässä ohjelmistokehityksessä 1/4

- Jokaiseen sprinttiin liittyy jonkinasteinen asiakkaan hyväksyntä vähintään katselmoinnilla.
- Asiakkaalle toimitettuihin julkistuksiin liittyy hyväksymistestaus.
- Perinteinen ketterien menetelmien tapa on se, että asiakkaan edustaja määrittää hyväksymistestit ja ne tekee asiakas, testaajat tai kehittäjät. Niitä pyritään joskus myös automatisoimaan.
- **Tämä ajattelu on täysin riittämätön.**

Hyväksymistestaus ketterässä ohjelmistokehityksessä 2/4

- Nämä testit lähinnä validoivat vaatimuksen toteutuksen.
  - Jokin raportti syntyy korrektisti.
  - Lomakkeella saadaan tietoa tietokantaan.
- Kyse on hyväksynnästä vain tässä kontekstissa.
  - Ei asiakasorganisaation hyväksynnästä!
  - Vasta oikeastaan savutesti – sen jälkeen alkaa kunnollinen testaus.

Hyväksymistestaus ketterässä ohjelmistokehityksessä 3/4

- Aito hyväksymistestaus merkitsee asiakkaan ja käyttäjien hyväksyntää.
- Se on välttämätöntä tehdä avoimesta näkökulmasta, riittävän monipuolisesti ja ei-mekanistisesti.
- Se ei saa jäädä asiakkaan ”edustajan” tehtäväksi, vaan mukaan on saatava asiakkaan organisaatio tekemään:
  - Professionaalista testisuunnittelua.
  - Professionaaliset testit.
  - Pätevät päätökset hyväksynnästä.
- Tietojärjestelmäkehityksessä asiakkaalta tarvitaan usein myös – kenties monitasoinen – hyväksymistestausympäristö.

Hyväksymistestaus ketterässä ohjelmistokehityksessä 4/4

- Hyväksymistestaus on suhteellisen jatkuva aktiviteetti.
- Koska sprintit tuottavat säännöllisesti uusia versioita asiakkaalle, niiden testaus tapahtuu monimuotoisesti:
  - Uusien ominaisuuksien hyväksymistestaus.
  - (…)
  - Uuden version testaus käytännössä, tuotantokäytössä.
- Esimerkiksi tietojärjestelmillä testaus voi olla monitasoista ja uusi järjestelmäversio läpäisee monet asiakkaan testitasot ja testiympäristöt, ennen kuin se on todettu tuotantokelpoiseksi.

Testauksen organisointi

- Pääosa testauksesta on integroitu kehittämistiimiin.
- Testaajat tiimin osana.
- Testaustiimejä tarvitaan / voidaan käyttää esim.:
  - Kattava toiminnallisuustestaus järjestelmätestaustasolla.
  - Järjestelmäintegrointitestaus.
  - Yhteentoimivuustestaus.
  - Toimituksiin liittyvät testaukset mm. konfiguraation testaus.
  - Kaikki erikoistestit.

Testaajan erilaiset tehtävät

1) Testauskonsultti projektille.
2) Ohjelmistokehittäjän tuki.
   - Miten inkrementin tuotoksia testataan, mitä testejä automatisoidaan.
3) Ketterä uusien toiminnallisuuksien testaus.
   - Nopeasti virheet esille.
4) Uusien toiminnallisuuksien systemaattinen testaus.
   - Perusasiat sprintin aikana.
   - Aikaa vievät seuraavan sprintin aikana.
5) Regressiotestaus.
6) Ei-toiminnallisten vaatimusten testaus.

Testaajan rooli 1/2

- Usein kiinteä osa tiimiä.
  - Välttämätöntä, koska muuten ei pääse osaksi suullista viestintäketjua.
  - Mikä objektiivisuudessa ja riippumattomuudessa menetetään, saadaan takaisin vuorovaikutuksen kautta.
- Rooli yhteistyössä:
  - Tiiviissä aikataulussa korostuu kehittäjiä konsultoiva rooli: autetaan tekemään hyviä suunnitelmia ja toteutuksia.
  - Tiivis aikataulu ja kevyt dokumentaatio edellyttävätkin hyvää yhteistyötä.
  - Valvova rooli: jonkun pitää mm. vahtia, että vaatimukset ymmärretään oikein ja tarvittaessa kvantifioidaan testausta varten (esim. suorituskykyvaatimukset)

Testaajan rooli 2/2

- Näkökulma testaustasoihin:
  - Testaajien tärkeä näkökulma on edelleen järjestelmätestauksessa.
  - Scrum-prosessissa suunnittelee asiakkaan kanssa ”hyväksymistestit”.
- Näkökulma testaussuunnitteluun:
  - Avainhenkilö testauksen pääsuunnitelman tekemisessä.
  - Scrum-prosessissa suunnittelee asiakkaan kanssa ”hyväksymistestit”.

Ketterän testauksen vaatimuksia testaajalle 1/2

- Hyvä osaaminen.
- Vahva ammatillinen itsetunto,
  - Tasavertainen asema ohjelmistokehittäjien kanssa (ketterässä ohjelmistokehityksissä) tiimissä.
  - Rohkeus tehdä nopeita päätöksiä.
- Monenlaisten testaustekniikoiden hallitseminen.
  - ”Työkalupakki”, josta valitaan sopiva menetelmä tarpeen mukaan.
- Dynaamisuus suunnitelmallisen ja tutkivan lähestymistavan välillä.
- Käyttäjien ajattelumallien ymmärtäminen ja kyky vapaasti simuloida erilaisia käyttäjiä.
- Buildi-/toimitus-orientaatio – varmistettava asiakasjulkistusten eheys.

Ketterän testauksen vaatimuksia testaajalle 2/2

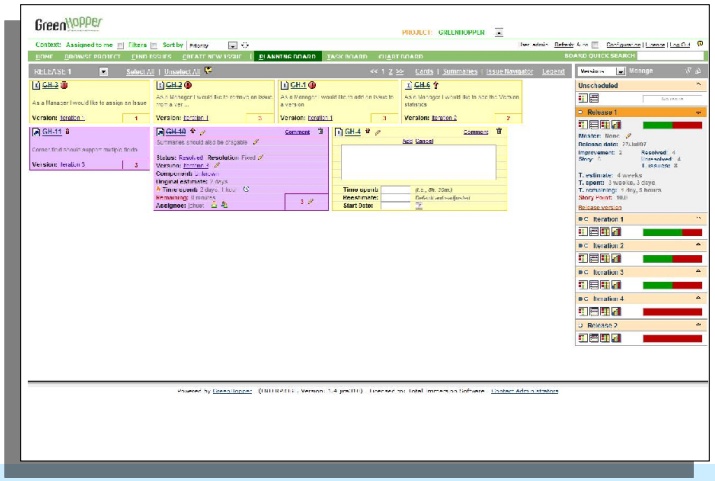
- Kyky toimia epävarmuudessa ja jatkuvassa muutoksessa.
- Yhteistyökyky ohjelmistokehittäjien ja asiakkaiden kanssa.
- Tiimiosaaminen.
- Kyky muuttaa toimintatyyliä projektin vaatimusten mukaiseksi.
- Tuoteteknologian ja tuotekonseptin ymmärtäminen.
- Kyky omaksua nopeasti uusia asioita.
- Kyky oivaltaa uusien asioiden olennaiset testattavat asiat.
- Älykkyys.
- Toiminnan nopeus – inkrementtien parissa tehtävä työ on saatava heti käyntiin – ja virheet korjattua ja uudelleentestattua.

Testaustiimit?

- Testaustiimejä tarvitaan edelleen.
- Vastapaino kehittäjäkeskeisyydelle.
- Experttien yhteistyö erityistestien suunnittelussa ja testauksen kokonaisuuden hallinnassa.
- Testaustasot järjestelmätestauksesta ylöspäin ovat testaustiimien tehtäviä.

Testitiimin tiedonsaanti

- Edustaja mukana kehitystiimin palavereissa.
  - Videoyhteys?
- Tiimin sähköinen toiminnanohjaus.
  - Seinätaulujen sijaan tietojärjestelmät.
  - JIRA taskien hallintaan.
  - Seinätauluja korvaavat ohjelmistot (esim. GreenHopper plugin for JIRA http://www.greenpeppersoftware.com/confluence/display/GH/FEATURES
- Yhteistyö kehittäjien kanssa.
  - Kenties työparien määrittely, vaikka ollaankin etänä?



Testaussuunnitelmat

- Ketteräkin testaus tarvitsee stabiilit, suunnitellut puitteet:
  - Testauksen pääsuunnitelma.
  - Laadunvarmistussuunnitelma.
  - Projektisuunnitelma.
  - Näiden taustalla käytettävä prosessimalli, joka määrittää testauksen yleiset periaatteet.
- Jokainen inkrementti voi tarvita oman tiiviin testaussuunnitelmansa, joka perustuu siihen, mitä ko. inkrementissä toteutetaan.
- Asiakastoimitukset tarvitsevat suunnitelman ja tarkistuslistoja.

Testaussuunnittelun yhteistyö

- Monissa malleissa kehittäjät, testaajat ja asiakas suunnittelevat yhdessä testit.
- Ns. ”hyväksymistestit.”
- Testit kuvaavat, miten uusien ominaisuuksien pitäisi toimia.
- Testit myös dokumentoivat yksityiskohtia.
- Ketterässä ideaalissa tuloksena automatisoituja hyväksymistestejä.
- Vaarana, että testisuunnittelu jää tähän – haastavien negatiivisten testien suunnittelu vaatii testaussuunnittelijan erityisosaamista.
- On vaarana degeneroitua perinteiseen ”myyntipäällikkö määrittää testit” –periaatteeseen!

Riskianalyysit 1/2

- Perinteisessä ohjelmistokehityksessä tehdään
  - Projektin riskianalyysi, ottaen myös vaatimusmäärittely huomioon.
  - Testausprojektin riskianalyysi (järjestelmätestaustiimi).
- Projektin riskianalyysi on yhtä oleellinen myös ketterässä kehityksessä.
- Testausprojektin riskianalyysin sijaan painottuvat validoinnin ja verifioinnin yleiset riskit:
  - Ovatko projektin puitteet tältä osin kunnossa?
  - Sopiiko prosessi tuotteelle ja julkistuksille?

Riskianalyysit 2/2

- Jokaiselle inkrementille on hyvä tehdä nopea riskianalyysi.
  - Inkrementin riskit.
  - Uusien ominaisuuksien riskit.
  - Inkrementin uusien asioiden testattavuus.
  - Välineenä esimerkiksi inkrementin riskikartta.

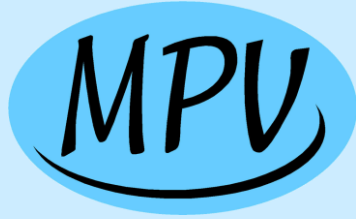
Testauksen priorisointi

- Esimerkiksi Scrum perustuu priorisointiin.
  - Ensimmäisissä sprinteissä toteutetaan asiakkaalle tärkeimmät asiat.
  - Samoin pitäisi toteuttaa muutenkin riskialttiimmat asiat.
- Testauksen priorisointi seuraa tätä päälinjaa.
- Keskeiset priorisointiparadigmat:
  - Toteutusta ohjaa suurin arvo asiakkaalle – toteutuuko se.
  - Riskipohjaisuus.

Testauksenhallinta

- ”Toteutusläheisessä” testauksessa se on integroituna ketterän kehittämisen prosessin / taskien hallintaan.
  - Työt eivät ole koko tiimin kannalta tehtyjä, ennen kuin uusi toiminnallisuus on myös testattu.
  - Esim. Scrumissa osa burndow-chartia.
- Tämä kattaa tiimin puitteissa tehtävän testauksen.
- Perinteinen testauksenhallinta on toimivaa ja tarpeellista edelleen.

Testauksenhallinnan avainperiaatteet



- Kriittisten asioiden testaus ensimmäisissä inkrementeissä.
- Varmistaminen, että yksikkö- ja integrointitestaus toimii.
- Automatisoitujen hyväksymistestien valvonta.
- Kehittämistaskeihin pitää liittää testaustaskit.
- Regressiotestauksen varmistaminen.
- Huolehtiminen, että kaikki ominaisuudet saavat riittävän järeän testauksen, yli automatisoitujen testien.
- Sprinttien yli lomitettujen testaustehtävien hallinta.
- Panostus muutoksenhallintaan.

Vianhallinta

- On kiusaus tehdä vioista taskeja projektin backlogiin.
- On kuitenkin erotettava viat ja niiden systemaattinen hallinta ja taas niiden edellyttävät ohjelmistokehittäjille avattavat taskit.
- Siis: on syytä käyttää ihan normaalia vikakantaa.
- Vian elinkaarimalli on sovitettava projektin tarpeisiin.

Testilogit

- Ketterässä toiminnassa on testilogeille uusia tarpeita.
- Koska testausta suunnitellaan kevyemmin ja dynaamisemmin, ei ole vanhan maailman tilannetta, jossa logeissa vain kuitataan tietyt asiat tehdyiksi.
  - Nyt ne kuvaavat tekemisen substanssia.
- Testilogien osana on hyvä olla päiväkirjatyyppinen kirjaus, jossa dokumentoidaan tehtyjä testaustehtäviä.

Testauksen ja kehityksen synkronointi

- Ketterissä projekteissa on rytmiltään erilaisia synkronoinnin tapoja
  - Prossessin palaverit erityisesti sprinttien alussa ja lopussa
  - Kahdenvälinen viestintä
- Synkronointiin on panostettava, jotta testaus koko ajan kohdentuu oikeisiin asioihin
- Joissakin organisaatioissa on viikoittaisia testauksen ja kehityksen synkronointipalavereja
- Ja tiimin sisällä tietenkin synkronointi on jatkuvaa

Ketterän testauksen vaatimuksia organisaatiolle 1/2

- Mietitty, vakiinnutettu ohjelmistoprosessi ja sen osana testausprosessi.
- Vahva osaaminen – ellei asioita hallita, ketteryys muuttuu kaaokseksi.
- Voimakas laatumyönteisyys.
- Ymmärrys hyvän laadun lähtökohdista – robusti koodi, asiakasnäkökulma, arkkitehtuuri.
- Ketterän kehittämisen menestystekijöiden ymmärtäminen ja tukeminen.
- Testausmyönteinen ohjelmistokehitys.
- Kaiken tarvittavan testauksen ymmärtäminen.

Ketterän testauksen vaatimuksia organisaatiolle 2/2

- Ohjelmisto koko ajan ”kunnossa”, jotta voidaan vaihtaa suuntaa helposti. Vikamäärän pitäisi ideaalisesti olla nolla.
  - Vanhat viat korjataan ennen kuin kehitetään uutta.
- Hyvä viestintä.
- Jatkuva dokumentointi.
- Valmiita malleja erilaisten asioiden testaamiseen.
- Kevyet testauksen dokumentointimallit.
  - Suunnitelmapohjat, joilla ohjelmiston pienenkin inkrementin testaussuunnittelu onnistuu kevyellä panostuksella.

Yhteenveto

- Ketterät menetelmät ovat tärkeitä, mutta ne ovat vain osa monipuolisen ammattimaisen testauksen keinovalikoimaa.
- Ketteryydestä tulee huonosti toteutettuna hallitsematon kaaos.
- Ketteryys edellyttää siksi vastapainokseen vahvaa projektinjohtamista ja jämäkkää testaussuunnittelua.
- Henkilökohtaiset taidot korostuvat ketterässä toiminnassa.
- Ketterien prosessien ideaaleihin ei voida luottaa käytännössä, kun kaikki tarvittavat palaset ovat harvoin paikoillaan.