Scrum

- *Scrum* on IID-pohjainen prosessimalli, jonka painopiste on projektinhallinnan arvoissa ja tekniikoissa.
  - Nimi ”Scrum” tulee rugbysta ja tarkoittaa pelin aloitusryhmitystä.
- Scrum on tällä hetkellä yleisin ohjelmistoteollisuudessa käytetty ketterä prosessimalli. Scrumin suosio perustuu sen yksinkertaisuuteen ja eleganttiin tapaan suhtautua projektinhallintaan.
- Scrum on joustava prosessimalli, jossa kiinnitetään vain joitain projektinhallintaan liittyviä tekniikoita.
- Scrum-projektissa voidaan käyttää hyväksi muiden ketterien prosessimallien tekniikoita. Esimerkiksi useimmat XP:n tekniikat sopivat Scrum-projektiin.

Scrumin elinkaari

- Scrumin elinkaari rakentuu neljästä vaiheesta:
  - valmistelu (planning)
  - järjestäminen (staging)
  - kehitystyö (development)
  - julkaisu (release)
- Valmistelu ja järjestäminen ovat Scrum-projektin esivaiheita.
- Kehitystyö tehdään iteraatioina
  - alun perin iteraation pituus on ollut tasan 30 päivää, mutta nykyisin Scrum suosii 15-30 päivän mittaisia iteraatioita
- Julkaisussa asiakkaalle toimitetaan ohjelmiston toimiva versio.

Scrum: Valmistelu 1

- Valmisteluvaihe vastaa lineaaristen prosessien esi- tai kelpoisuusselvitystä (feasibility study). Siinä selvitetään tuotettavan ohjelmiston ja projektin taustoja ja varmennetaan, että ohjelmisto kannattaa toteuttaa.
- Tarkoitus:
  - asetetaan tavoite
  - selvitetään odotukset
  - varmistetaan rahoitus

Scrum: Valmistelu 2

- Tehtävät:
  - kirjataan tavoite
  - kirjataan budjetti
  - alustetaan *työlista* (backlog)
    - q työlista sisältää tuotteelle tähän mennessä löydetyt *vaatimukset*
    - q työlistaa päivitetään sitä mukaa kun uusia vaatimuksia ja niihin liittyviä tehtäviä syntyy ja vanhat vaatimukset muuttuvat
    - q työlistan vaatimuksia ja tehtäviä *priorisoidaan* projektin kuluessa
  - tarvittaessa tehdään ongelmakenttää selventävä ohjelmistosuunnitelma ja/tai prototyyppi

Scrum: Järjestäminen

- Järjestäminen vastaa perinteistä vaatimusten kartutusta ja analyysia. Siinä selvitetään alustavat tuotteen vaatimukset ja kuvataan ongelmakenttä.
  - Sekä vaatimukset että ongelmakenttä päivittyvät projektin aikana. Jokaisen iteraation jälkeen tuotteesta on syntynyt aiempaa selkeämpi käsitys.
- Tarkoitus:
  - tunnistetaan tuotteen vaatimuksia ja priorisoidaan niitä riittävästi ensimmäistä iteraatiota varten
- Tehtävät:
  - vaatimusten kartutus (elicitation)
  - alustava ohjelmistosuunnitelma ja/tai prototyyppi

Scrum: Kehitystyö

- Kehitystyössä ohjelmistoa toteutetaan iteratiivisesti IID:n periaatteiden mukaisesti
- Tarkoitus:
  - toteutetaan julkaisukelpoinen ohjelmisto *pyrähdyksinä* (sprint)
    - q yksi pyrähdys tarkoittaa yhtä aikaikkuna-iteraatiota
- Tehtävät:
  - pyrähdyksen suunnittelu(kokous)
  - pyrähdyksessä toteutettavien työlistan tehtävien valinta, priorisointi ja kirjaaminen *pyrähdyksen työlistaan* (sprint backlog)
  - jäljellä olevan työmäärän arviointi
  - päivittäiset Scrum-kokoukset
  - *Scrum-katselmointi* (Scrum review) pyrähdyksen lopussa: katsotaan (demotaan) tuotteen omistajalle, mitä tiimi on saanut pyrähdyksessä aikaan
  - mahdollinen tiimin itsearviointi (retrospective), jossa otetaan oppia pyrähdyksestä seuraavaa pyrähdystä varten

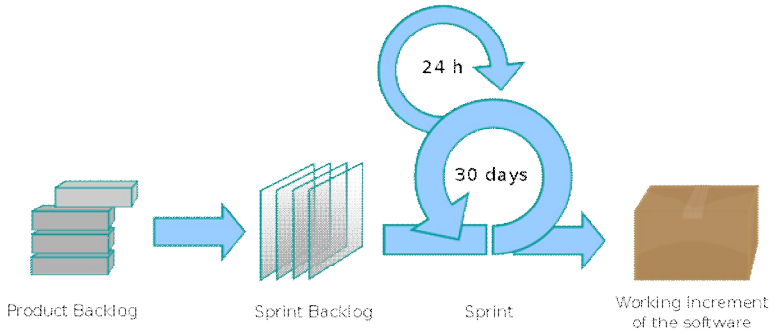
Scrum: Julkaisu

- Julkaisussa loppukäyttäjälle tarkoitettu ohjelmisto annetaan tuotantokäyttöön. Projektin aikana voi olla monta julkaisua.
- Tarkoitus:
  - julkaistaan valmis ohjelmisto
- Tehtävät:
  - dokumentointi
  - koulutus
  - myynti
  - jne.

Scrum-kehitysprosessi lyhyesti

- Kunkin pyrähdyksen alussa työlistalta valitaan pyrähdyksen aikana toteutettavat tehtävät pyrähdyksen työlistalle.
  - Pyrähdyksen työlistalle ei lisätä uusia tehtäviä pyrähdyksen aikana.
  - Suositus: tehtävät on suunniteltava niin pieniksi, että kukin voidaan toteuttaa kahdessa työpäivässä.
- Jokainen työpäivä aloitetaan *Scrum-kokouksella* (Scrum meeting). Kokous pidetään aina samassa paikassa samaan aikaan.
  - Kokouksessa jokainen tiimin jäsen vastaa seuraaviin kysymyksiin:
    - Mitä olet tehnyt edellisen Scrum-kokouksen jälkeen?
    - Mitä aiot tehdä seuraavaan Scrum-kokoukseen mennessä?
    - Mitä esteitä olet kohdannut työssäsi?
  - Lisäksi oppikirjassa (Larman, 2004) suositellaan kahta lisäkysymystä:
    - Puuttuuko työlistasta tehtäviä? (Ei uusia vaatimuksia vaan vanhojen tarkennuksia.)
    - Oletko oppinut jotain sellaista, josta on hyötyä muille tiimiläisille?
- Pyrähdyksen päätteeksi tarkistetaan Scrum-katselmoinnissa, että tuotettu ohjelmistoversio täyttää sille asetetut vaatimukset.

Scrum-prosessimallin kaaviokuva



Kuva on lähteestä wikimedia.org.
Kuva on julkaistu GFDL-lisenssillä
(GNU Free Documentation License)

Scrumin sidosryhmät 1

Scrum-projektilla on neljä sidosryhmää:

1. *Tuotteen omistaja* (product owner)
   - Tuotteen omistaja edustaa projektin asiakasta:
     - vastaa siitä, että projektin työlista on ajan tasalla
     - valitsee seuraavassa pyrähdyksessä toteutettavat tehtävät
     - arvioi yhdessä muiden sidosryhmien kanssa jokaisen pyrähdyksen päättyessä tehdyn tuotoksen
   - Tuotteella voi olla monta yhteistyössä toimivaa omistajaa.
2. *Scrum-tiimi* (Scrum team)
   - Scrum-tiimi vastaa kuhunkin pyrähdykseen valittujen tehtävien toteutuksesta.
   - Tiimiläisille ei ole määritelty tämän tarkempia tehtäviä.
   - Tiimissä ei saisi olla enempää kuin seitsemän jäsentä. Tätä isommat projektit on organisoitava useiksi tiimeiksi.
3. *Kanat* (chickens)
   - Kaikkia muita projektin sidosryhmiä kutsutaan kanoiksi, myös esimerkiksi johtoportaan edustajia. Kanat saavat tarkkailla projektia mutta ne eivät voi vaikuttaa pyrähdysten sisältöön.

Scrumin sidosryhmät 2

4. *Scrum-mestari* (Scrum master)
   - Scrum-mestari vastaa projektin hallinnasta. Hän ei ole varsinainen projektipäällikkö, vaan pikemminkin projektin valmentaja.
   - Scrum-mestari on Scrum-tiimissä myös kehittäjänä, eikä siten projektissa pelkkänä hallintohenkilönä.
   - Scrum-mestarin tehtävät:
     - varmistaa, että projektin tavoitteet säilyvät projektin ajan
     - varmentaa, että projektissa seurataan Scrumin periaatteita
     - toimia Scrum-tiimin ja hallinnon yhteyshenkilönä
     - toimia Scrum-tiimiläisten valmentajana
     - tasoittaa ja poistaa esteitä
     - vastata Scrum-kokouksista
     - vastata Scrum-katselmoinneista (demoista)

Scrumin käytännöt 1

- Tärkeimmät Scrumin käytännöt:
  - Pyrähdykset
    - Alun perin 30 päivää, nykyisin 2-6 viikkoa. Pyrähdyksen aikana Scrum-tiimillä on vapaus tehdä pyrähdyksen työlistalla olevia tehtäviä ilman keskeytyksiä.
  - Itseorganisoituvat tiimit
    - Pyrähdyksen aikana asiakas, Scrum-mestari, managerit ja muut sidosryhmät eivät määrää tiimiä toimimaan kehitystyössä tietyllä tavalla. Kaikki päätökset tehdään pyrähdysten välillä.
  - Scrum-kokoukset
    - Jokainen työpäivä alkaa samaan aikaan ja samassa paikassa Scrum-kokouksella.
  - Rauhoitetut iteraatiot
    - Pyrähdyksen aikana sen työlistalle ei lisätä uusia tehtäviä. Jos jokin tehtävä on aivan pakko lisätä pyrähdyksen työlistalle, on jokin toinen tehtävä samalla poistettava listalta.

Scrumin käytännöt 2

- Päätökset tunnissa
  - Scrum-mestari pyrkii ratkaisemaan päätöstä vaativat kysymykset mieluimmin välittömästi tai viimeistään tunnin sisällä.
- Esteet pois päivässä
  - Scrum-kokouksissa esille tulevat ongelmat pyritään korjaamaan ennen seuraavaa Scrum-kokousta.
- Yhteinen työtila
  - Scrum-tiimi työskentelee yhteisessä projektihuoneessa. Yksityiset työtilat ovat muita tehtäviä varten.
- Päivittäinen integraatio
  - Kaikki varmennettu koodi integroidaan ja regressiotestataan ainakin kerran päivässä.

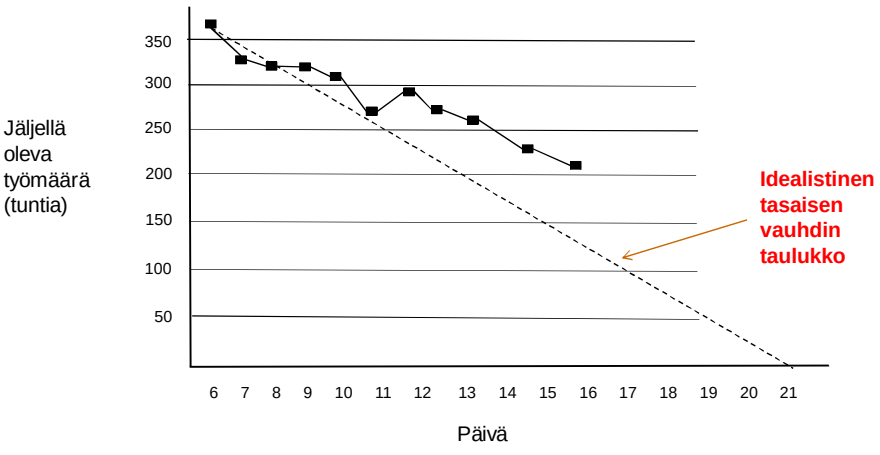
Pyrähdyksen työlista

- Pyrähdyksen työlista sisältää pyrähdyksen kannalta oleellista tietoa siinä tehtävistä töistä. Listalla on ainakin
  - tehtävän kuvaus: mitä tehdään
  - tehtävän esittäjä
  - tehtävästä vastuussa oleva tiimin jäsen
  - tehtävän status: aloittamaton, aloitettu, valmis
  - kesken olevan tehtävän vaatiman työajan arvio
- Jäljellä olevan työajan arvio on mielenkiintoinen. Puhtaassa Scrumissa ei arvioida tehtävien kestoja tai tehtävien välisiä riippuvuuksia. Riippuvuusverkkoa ja kriittistä polkua ei tarvita.
- Kun kaikki pyrähdyksen työlistan jäljellä olevan työajan arviot yhdistetään, saadaan jäljellä olevaa työmäärää kuvaava kaavio (*burndown chart*, *sprint backlog graph*). Se kertoo, paljonko pyrähdyksen kokonaistyömäärästä arvioidaan olevan jäljellä.

Työlistan tehtävät

- Työlistoissa voi olla monentyyppisiä tehtäviä, esimerkiksi:
  - asiakkaan (tuotteen omistajan) esittämiä vaatimuksia
  - tuotteeseen toteutettavien toimintojen kuvauksia
  - korjattavien virheiden kuvauksia (virheraportteja)
  - selvittämättömiä teknisiä ongelmia
- Vaatimukset ja toiminnot voidaan kuvata esimerkiksi käyttötapauksina (use case) tai käyttäjätarinoina (user story).
- Käyttäjätarina: tekstuaalinen, määärämuotoinen kuvaus käyttäjälle hyödyllisestä toiminnallisuudesta
  - ”<tietyssä roolissa> haluan <sitä sun tätä> <saavuttaakseni sen ja sen hyödyn>”
  - esim. ”sihteerinä haluan hakea ohjelmistosta henkilöiden puhelinnumeroita voidakseni soittaa heille ja ilmoittaa kokousten ajankohdat”
- Tehtävään kuluvan työmäärän arviointi: tiimin kokemukseen perustuvana yhteistyönä, esimerkiksi äänestämällä tai pelaamalla ”suunnittelupokeria” (planning poker).

Sprint backlog graph



Scrumin luonne

- Scrum on selkeästi hallinnollinen prosessimalli
  - sen käytännöissä otetaan kantaa siihen, miten projektiryhmä organisoituu ja toimii yhteistyössä
  - sen sijaan siinä otetaan hyvin vähän kantaa siihen, mitä tekniikoita projektiryhmä käyttää kehitystyössä
- Koska Scrum ottaa kantaa erityisesti hallintoon, se sopii hyvin erityyppisten projektien prosessimalliksi. Se sopii myös hyvin yhteen muiden prosessimallien käytäntöjen kanssa.

Scrumin huonoja puolia

- Scrum ei ota kantaa käytettäviin ohjelmistotuotannon menetelmiin, tekniikoihin tai työkaluihin, joten siitä ei saa tukea niiden valinnalle
- Scrum ei ota kantaa projektissa tuotettavaan dokumentaatioon, jolloin hyödyllisiä dokumentteja saatetaan jättää tekemättä
- Scrum edellyttää aktiivista tuotteen omistajaa, mutta asiakkaalta ei sellaista välttämättä saada
- Scrum-projekti nojaa tiimin yhteistyöhön ja ammattitaitoon, joten siihen ei voi ottaa ketä tahansa

Extreme Programming eli XP

- *Extreme Programming* (*XP*) toi ketterät menetelmät laajempaan tietoisuuteen 1990-luvun loppupuolella. Se on edelleen tunnettu ketterä prosessimalli, vaikka Scrum on vienyt siltä suosiota teollisuudessa.
- XP:n painopiste on kehitystyössä, kun Scrumin painopiste on projektinhallinnassa. Prosessimallit eivät ole ristiriidassa keskenään.
- XP:ssa korostetaan
  - yhteistyötä
  - tiheää ja aikaista julkaisutahtia
  - laadukkaita kehitystyön menetelmiä
- XP perustuu neljälle arvolle:
  - kommunikaatio
  - yksinkertaisuus
  - palaute
  - rohkeus

XP:n elinkaari

- XP:n elinkaari rakentuu viidestä vaiheesta:
  - kartoitus (exploration)
  - suunnittelu (planning)
  - iteraatiot ja ensimmäinen julkaisu (iterations to first release)
  - tuotteistus (productization, productionizing)
  - ylläpito (maintenance)
- Kartoitus ja suunnittelu ovat XP-projektin esivaiheita
- Kehitystyö tehdään IID-periaatteiden mukaan 1-3 viikon iteraatioissa
- Tuotteistus ja ylläpito ovat ohjelmistokehitysprojektin ulkopuolella, mutta kuuluvat XP-prosessiin

XP:n askeleet 1-2

1. XP-projekti voi alkaa kartoitusvaiheella. Siinä ohjelmiston alustavia ominaisuuksia kirjataan *tarinakorteille* (story cards) ja annetaan niille karkeita kokoarvioita.
2. *Julkaisun suunnittelupelissä* (release planning game) asiakkaat ja kehittäjät täydentävät tarinakortteja ja arvioita ja päättävät, mitä seuraavaan julkaisuun tulee.

XP:n askeleet 3-5

3. *Iteraation suunnittelupelissä* (iteration planning game) asiakkaat valitsevat iteraatioon toteutettavat tarinat. Ohjelmistokehittäjät jakavat tarinat pieniksi arvioitavissa oleviksi tehtäviksi. Jos tehtäviä on liikaa yhtä iteraatiota kohti, niiden määrää karsitaan.
4. Kehittäjät toteuttavat valitut tarinat aikaikkuna-periaatteella. Asiakkaat ja kehittäjät käyvät yhteistyössä läpi testejä ja vaatimusten yksityiskohtia.
5. Jos seuraavaa julkaisua varten ei ole saatu kasaan tarpeeksi toteutettuja tarinoita, palataan askeleeseen 3.

XP-prosessimallin kaaviokuva

XP-prosessimallin kaaviokuvan selitystä

- Releaseplan
  - tehdään julkaisun tarinakortit
- Iteration plan
  - valitaan tarinakorteista seuraavassa iteraatiossa toteutettavat tehtävät
- Acceptance Test
  - suunnitellaan iteraation (asiakasvetoisia) hyväksymistestejä
- Stand up meeting
  - pidetään päivittäinen ryhmäpalaveri Scrumin tapaan
- Pair Negotiation
  - muodostetaan seuraavat *pariohjelmoinnin* parit
- Unit Test
  - kirjoitetaan yksikkötestit *kaikelle* (iteraatiossa kirjoitettavalle) koodille
- Pair Programming
  - tehdään *kaikki* koodaus pariohjelmointina (toinen ohjelmoi, toinen katselmoi ja suunnittelee uusia yksikkötestejä)

XP:n sidosryhmät 1

XP-projektilla on kuusi sidosryhmää:

1. *Asiakas* (customer)
   - tarinakorttien kirjoittaminen
   - vastuu hyväksymistesteistä
   - seuraavan julkaisun tai iteraation tarinakorttien valinta
2. *Koodaaja* (programmer)
   - testien kirjoittaminen
   - suunnittelu
   - ohjelmointi
   - refaktorointi (koodin muokkaus arkkitehtonisesti paremmaksi)
   - tarinoiden jako tehtäviksi ja tehtävien koon arviointi
3. *Testaaja* (tester)
   - asiakkaan avustaminen hyväksymistestien suunnittelussa ja kirjoittamisessa

XP:n sidosryhmät 2

4. *Valmentaja* (coach)
   - prosessin valvonta
   - prosessin mukauttaminen
   - väliintulo ongelmatilanteissa
   - koulutus
5. *Seuraaja* (tracker)
   - mittaustietojen keruu
   - edistymisen raportointi
   - palautteen anto huonoista mittausarvioista
6. *Konsultti* (consultant)
   - tekninen konsultaatio
   - XP-valmennus

XP:n käytännöt

Toisin kuin Scrumissa, XP:ssa on melko paljon toisiinsa sidoksissa olevia ydinkäytäntöjä:

- Suunnittelupeli
- Lyhyet jatkuvat julkaisut
- Järjestelmämetaforat
- Yksinkertainen suunnittelu
- Testaus
- Jatkuva refaktorointi
- Pariohjelmointi
- Koodin yhteisomistajuus
- Jatkuva integrointi
- Kestävä kehitystahti
- Tiimi samassa tilassa
- Koodausstandardit

Ydinkäytäntöjen lisäksi on joukko ”avustavia käytäntöjä”

XP:n käytäntöjen luonne

- XP:n käytännöt ovat vahvasti sidoksissa toisiinsa. Siksi on riskialtista muokata prosessia jättämällä siitä pois keskeisiä käytäntöjä.
  - Esimerkiksi lyhyet jatkuvat julkaisut tarkoittavat kevennettyä vaatimusmäärittelyä, mikä taas vaatii asiakkaiden jatkuvaa läsnäoloa ja työskentelyä yhteisessä tilassa.
- Muualla hyväksi havaitut käytännöt on viety äärimmilleen, mistä tulee XP:n termi ”Extreme”.
  - Testaus on hyvä käytäntö, joten tehdään kaikelle koodille yksikkötestit ja kaikille ominaisuuksille hyväksymistestit.
  - Koodikatselmoinnit ovat sitä parempia, mitä lähempänä koodin kirjoittamista ne pidetään, joten katselmoidaan koodia tosiajassa tekemällä pariohjelmointia.
  - Toimiva kommunikaatio on hyvä käytäntö, joten käytetään yhteistä työtilaa, tehdään pariohjelmointia ja otetaan asiakkaat mukaan suunnitteluun, ohjaukseen ja arviointiin.

XP:n vaatimusmäärittelyn käytännöt

- Suunnittelupelit
  - julkaisun ja iteraation suunnittelupelit
- Läsnäoleva asiakas
  - asiakkaan edustajat ovat kokopäiväisesti tiimin kanssa
  - asiakkaan edustajia voi olla useista eri sidosryhmistä
  - asiakas vastaa hyväksymistesteistä
- Hyväksymistestaus
  - jokaiselle ominaisuudelle täytyy olla automatisoidut asiakkaan kirjoittamat hyväksymistestit
  - kaikki testit ovat suoritettavissa binäärisellä hyväksytty/hylätty-tuloksella
  - yksittäisten testiajojen läpimenon tarkistamiseen ei tarvita ihmisiä, vaan ne ovat automatisoituja

XP:n suunnittelun käytännöt

- Järjestelmämetaforat
  - koko järjestelmän tai sen osan kuvaus helposti muistettavana rinnastuksena (eli metaforana)
  - XP:n versio arkkitehtuurisuunnitelmasta
- Yksinkertainen suunnittelu
  - suunnitellaan tätä hetkeä varten
  - ei spekuloida tulevien iteraatioiden vaatimilla piirteillä
  - yksinkertaiset luokat ja metodit
  - ei leikkaa-liimaa-koodausta
- Jatkuva refaktorointi
  - jatkuva prosessi yksinkertaistaa koodia ja suunnitteluratkaisuja, jolloin ylläpidettävyys paranee
  - pieniä muutoksia kerrallaan
  - mahdollisimman paljon valmiiden (suunnittelu)mallien ja refaktorointityökalujen käyttöä

XP:n toteutuksen käytännöt

- Koodausstandardit
  - kaikki tiimiläiset noudattavat samaa koodaustyyliä
  - koodista ei pidä voida päätellä sen kirjoittajaa
- Pariohjelmointi
  - kaikki koodi tehdään parityönä: toinen ohjelmoi ja toinen katselmoi kirjoitettavaa koodia
  - koodausvuoro vaihtuu säännöllisesti
  - parit vaihtuvat säännöllisesti
  - kahta kokematonta ohjelmoijaa ei laiteta pariksi
- Koodin yhteisomistajuus
  - kaikki ovat vastuussa koko koodista
  - kuka tahansa voi muokata mitä tahansa osaa koodista
  - virheen löytäjällä on vastuu sen korjaamisesta

XP:n testauksen ja verifioinnin käytännöt

- Hyväksymistestaus
- Testivetoinen kehitystyö (Test-Driven Development, TDI) (yksikkötestauksessa)
  - yksikkötestit kirjoitetaan ennen niitä vastaavan ominaisuuden ohjelmointia
  - kaikkia yksikkötestejä suoritetaan automaattisesti tauotta
  - yksikkötestit vastaavat ominaisuuden spesifikaatiota
- Läsnäoleva asiakas

XP:n projektinhallinnan käytännöt

- Suunnittelupelit
- Lyhyet julkaisut
  - *evoluutiolähtöinen kehitystyö*: väärät päätökset ovat hyväksyttäviä
  - varautuminen muutokseen
- Kestävä kehitystahti
  - ei jatkuvia ylitöitä
  - tiimin ja asiakkaiden on voitava jaksaa samaa työtahtia periaatteessa rajattomasti
- Päivittäiset ryhmäpalaverit
  - tiimi käy joka aamu yhdessä läpi edellisenä päivänä kohdatut ongelmat ja niihin löydetyt ratkaisut
  - hyvin samanlainen kuin päivittäinen Scrum-kokous

XP:n versionhallinnan käytännöt

- Jatkuva integrointi
  - kaikki valmiiksi katsottu koodi integroidaan noin 15 minuutin välein erillisellä koneella, samalla suorittaen kaikki olemassa olevat yksikkötestit
  - jatkuva integrointi on käynnissä 24 tuntia vuorokaudessa jokaisena viikonpäivänä
- Yhteinen työtila
  - XP-projektit läpiviedään yhteisessä työtilassa, ei erillisissä toimistoissa
  - työtilan keskellä ovat pariohjelmointipöydät
  - seinät on varattu tauluille ja postereille yms.
  - tiimi ja asiakkaan edustajat työskentelevät samassa tilassa
- Suunnittelupelit

XP:n luonne

- XP on selkeästi ohjelmistokehittäjän prosessimalli
  - sen käytännöissä otetaan kantaa siihen, miten ohjelmistokehitys tehdään mahdollisimman tehokkaasti
  - siinä olevat hallinnolliset periaatteet ovat melko yleisiä ja erikoistettavissa eri projekteihin
- Scrumiin verrattuna XP on teknisempi, vaikeammin opittava ja kehitystyötä enemmän määrittelevä prosessimalli. Ehkä sen vuoksi XP on hävinnyt suosiossa Scrumille.

XP:n huonoja puolia

- XP edellyttää aktiivista asiakasta, mutta sellaista ei välttämättä saada projektiin
- XP ei tue ohjelmiston kokonaisarkkitehtuuria
- Useimmat ohjelmistokehittäjät eivät pidä pariohjelmoinnista
- XP ei ota kantaa projektissa tuotettavaan dokumentaatioon, jolloin hyödyllisiä dokumentteja saatetaan jättää tekemättä

XP ja Scrum

- XP:n käytännöt keskittyvät pääasiassa kehitystyöhön. Koska Scrumin käytännöt puolestaan keskittyvät pääasiassa projektinhallintaan, XP:n tekniikat ovat pitkälti yhteensopivat Scrumin kanssa.
- Vaikka Scrum-projektissa ei tarvitse seurata muita kuin Scrumin omia periaatteita, sopivien XP:n periaatteiden lisääminen tehostaa projektityöskentelyä.

Scrumiin sopivia XP:n käytäntöjä 1

Ainakin seuraavat XP:n käytännöt sopivat hyvin Scrum-projektiin:

- Automatisoitu testaus
  - kaikki testit on voitava suorittaa automaattisesti
  - kaikista testeistä on saatava yksiselitteinen läpi/kaatui-tulos (pass/fail)
  - hyväksymistestit kirjoitetaan yhdessä asiakkaan (tuotteen omistajan) kanssa
- Testivetoinen kehitystyö
  - Yksikkötestauksessa testit kirjoitetaan ennen koodia. Yksikkötestit vastaavat sekä testitapauksen kuvausta että testattavan ominaisuuden spesifikaatiota.
- Pariohjelmointi
  - Kaikki koodi kirjoitetaan pariohjelmointina yhden koneen ääressä. Pariohjelmoinnissa toinen kirjoittaa koodia ja toinen tekee koodille jatkuvaa laadunvarmistusta.

Scrumiin sopivia XP:n käytäntöjä 2

- Jatkuva integrointi
  - kaikki hyväksytty koodi integroidaan ja testataan jatkuvasti erillisessä integrointijärjestelmässä
  - integrointijärjestelmä toimii 24/7-silmukassa, jossa käännetään koodi sekä suoritetaan kaikki yksikkötestit ja kaikki/useimmat hyväksymistestit
- Yhteinen koodin omistajuus
  - kuka tahansa tiimin jäsen on velvollinen korjaamaan löytämänsä koodivirheen
  - tiimiläisten on seurattava yhtenäistä koodausstandardia
- Yksinkertainen suunnittelu
  - koodi suunnitellaan ja toteutetaan nykyistä julkaisua varten
- Säännöllinen refaktorointi
  - koodia yksinkertaistetaan ja siivotaan säännöllisesti, tavoitteena minimaalinen ja helposti ymmärrettävä koodikanta

Lean

M. Poppendieck, T. Poppendieck: *Implementing Lean Software Development – From Concept to Cash*. Addison-Wesley, 2007.

- Viime vuosina käyttöön otettu seuraaja ketterille menetelmille
- Tavoitteena työn tehostaminen virtaviivaistamalla ja pelkistämällä työprosessia
- Perustuu Toyotan voittokulkuun johtaneisiin periaatteisiin ja käytäntöihin autoteollisuudessa

Lean-historiaa

- 1940- ja 1950-luvut: Toyota kehitti Japanissa nopeaa ”Just-in-Time”-tuotelinjaa kyetäkseen kilpailemaan tehokkuudessa amerikkalaisten autonvalmistajien kanssa
- 1962: *Toyota Production System* (TPS) otettiin käyttöön
- 1980-luku: TPS:n pohjalta kehitettiin ja määriteltiin yleiset *Lean*-tuotantoperiaatteet
- 1990-luku: TPS yleistettiin *Toyota Product Development System* -tuotekehitysmenetelmäksi
- 2000-luku: Lean-periaatteet otettiin käyttöön ohjelmistotuotannossa
- Kantateos: M. Poppendieck, T. Poppendieck: *Lean Software Development – An Agile Toolkit*. Addison-Wesley, 2003.

Ohjelmistotuotannon Lean-periaatteet 1

1. Jätteen poistaminen (eliminate waste)
   - v jäte: kaikki se, mikä ei tuota arvoa asiakkaalle
   - v tavoitteena poistaa ohjelmistokehityksestä kaikki turha työ
2. Jatkuva laadunvarmistus (build quality in)
   - v virheiden välttäminen koko kehitysprosessin ajan, ei vasta (erillisessä) testausvaiheessa
3. Tietämyksen luominen (create knowledge)
   - v ohjelmiston kehittäminen asteittain tarkentamalla
   - v vaatimusten, arkkitehtuurin, koodin yms. yksityiskohtia ei kiinnitetä ennen kuin kehitystiimillä on riittävästi siihen tarvittavaa tietämystä

Ohjelmistotuotannon Lean-periaatteet 2

4. Päätöksenteon lykkääminen (defer commitment)
   - (lopullisten) päätösten tekeminen vasta viime hetkellä, tarjolla olevien vaihtoehtojen perusteella
   - tavoitteena välttää jo tehdyn työn peruuttamista
5. Nopeus (deliver fast)
   - tuoteversioiden toimittaminen asiakkaille niin usein ja nopeasti, etteivät he ehdi muuttaa mieltään
6. Tiimityö (respect people)
   - ohjelmistojen kehittäminen autonomisissa tiimeissä, joilla on huipputason johtajat
7. Kokonaisuuden optimointi (optimize the whole)
   - ei yksittäisten vaan tuotantoprosessin kaikkien vaiheiden ja niiden välisten siirtymien tehostaminen

Lean-jäte 1

1. Osittain tehty työ (partially done work)
   - ohjelmistotuotteen osa tai prosessin välituotos, jota pitää jatkuvasti päivittää
   - tyypillisiä esimerkkejä: liian aikaisin jäädytetyt vaatimukset, integroimaton ohjelmakoodi, testaamaton koodi, dokumentoimaton koodi
2. Ylimäärä (extra features)
   - sellaiset ohjelmistoon lisätyt palvelut ja viritykset, joita asiakas ei tarvitse
3. Vanhan toisto (relearning)
   - joskus aiemmin tehdyn ja sittemmin unohdetun työn mieleen palauttamista ja uudelleen tekemistä

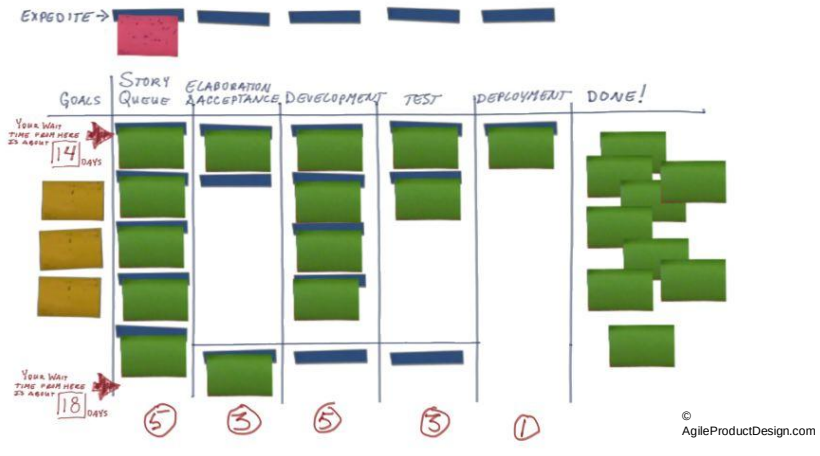
Lean-jäte 2

4. Täysi heprea (handoffs)
   - v sellaiset ohjeet tai dokumentit, joita niiden vastaanottaja ei pysty ymmärtämään
5. Poukkoilu (task switching)
   - v jatkuvaa siirtymistä hommasta toiseen tai useamman homman hoitamista yhtä aikaa
6. Viivästykset (delays)
   - v muuta työtä tekevien odottamista, joko antamaan neuvoja tai osallistumaan seuraavaksi vuorossa olevaan työtehtävään
7. Virheet (defects)
   - v vasta prosessin myöhemmissä vaiheissa havaittavat, etsittävät ja korjattavat virheet ohjelmakoodissa

Kanban

- v *Kanban*: työtehtävää kuvaava kortti
- v Kanban-taulu (Kanban board): tuotelinjaa, tehtäväketjua tai tuotantoprosessia kuvaava taulu
  - Ø sisältää kaikkia tehtäviä kuvaavat Kanban-kortit
  - Ø kortit on sijoitettu sarakkeisiin, jotka vastaavat tuotelinjan, tehtäväketjun tai tuotantoprosessin eri vaiheita
  - Ø kortteja siirretään sarakkeesta toiseen sitä mukaa kun työ etenee
  - Ø korteissa kuvataan tyypillisesti tehtävä, sen tekijät ja sen tekemiseen arvioitu työmäärä tai -aika
  - Ø sarakkeille on määritelty yläraja niihin sijoitettavissa olevien korttien määrälle (so., samassa vaiheessa yhtä aikaa olevien tehtävien määrälle)
  - Ø esimerkiksi korttien väreillä voi kuvata niitä vastaavien tehtävien vaihetta (aloitettu, kesken, valmis)

Kanban-taulu



Lean: käyttöönotto 1

Kokonaisuuden optimointi

1. Ota lean-periaatteet käyttöön koko prosessissa mm. nimeämällä arvoketjusta vastaava tiimi
2. Ota käyttöön arvoketjun mittarit
3. Vähennä vaiheesta toiseen siirtymisestä aiheutuvaa yleisrasitetta arvoketjussa ja prosessissa

Tiimityö

4. Kouluta tiiminjohtajia
5. Siirrä vastuut ja päätösvalta organisaatiossa mahdollisimman alas (tiimeille)
6. Korosta laatua ja ammattiylpeyttä mm. poistamalla rutiinityöt ja liian tiukat takarajat sekä antamalla aikaa testaukselle

Lean: käyttöönotto 2

Nopeus

7. Työskentele pienissä erissä ja lyhyissä iteraatioissa
8. Älä käynnistä uutta työvaihetta ennen kuin edellinen on valmis
9. Älä keskity resurssien tehokkaaseen käyttöön vaan mahdollisimman nopeaan tuotantoon

Päätöksenteon lykkääminen

10. Älä jää odottelemaan täydellistä vaatimusmäärittelyä vaan tuota vaatimuksia rinnakkain kehitystyön kanssa
11. Murra ohjelmiston arkkitehtoniset riippuvuudet
12. Kehitä rinnakkain useita mahdollisia ratkaisuvaihtoehtoja äläkä kiinnitä niistä yhtäkään ennen kuin olet täysin varma

Lean: käyttöönotto 3

Tietämyksen luominen

13. Varmista, että ohjelmistoa voidaan kehittää arkkitehtonisesti itsenäisinä moduuleina, joista kullakin on vastuutiimi
14. Anna tiimeille aikaa kehittää omaa toimintaansa ja järjestä tiimien yhteisiä kehittämistilaisuuksia
15. Järjestä ongelmanratkaisun koulutusta

Jatkuva laadunvarmistus

16. Kehitä ohjelmistoa testivetoisesti ja integroi koodia jatkuvasti
17. Automatisoi rutiinitehtävät, kuten testiajot ja versionhallinta
18. Siivoa ohjelmakoodia, testitapauksia ja dokumentaatiota refaktoroimalla niitä säännöllisesti

Lean: käyttöönotto 4

Jätteen poistaminen

19. Varmista, että tiimit varmasti ymmärtävät, mitä asiakkaat haluavat, ja keskittyvät oikeiden tuotteiden kehittämiseen
20. Älä tuota mitään muuta kuin arvoa asiakkaille
21. Koodaa vähemmän ja julista mutkikkuus pannaan

Agile vs. Lean 1

- Sekä agile että lean:
  - käyttäjätarinat
  - refaktorointi
  - testivetoinen kehitys ja automatisoitu testaus
  - pariohjelmointi
  - jatkuva koodin integrointi
  - versionhallinta
  - katselmoinnit ja tarkastukset
  - lisäävä (inkrementaalinen) kehitys
  - lyhyet iteraatiot
  - sisäisten ja ulkoisten julkaisujen erottaminen toisistaan
  - asiakaslähtöinen vaatimusten priorisointi
  - aikaikkunat
  - vastuulliset ja itsenäiset tiimit
  - yhteiset työtilat

Agile vs. Lean 2

- Agile muttei lean:
  - aina läsnä oleva asiakas
  - koodausstandardit
  - koodin yhteisomistajuus
  - (arkkitehtoninen) suunnittelupeli
  - 40 tunnin työviikko
  - päivittäiset kokoukset
- Lean muttei agile:
  - moduulien arkkitehtoninen riippumattomuus
  - arvoketju ja sen hallinta
  - pääsuunnittelija (chief engineer)
  - Kanban-taulu

Tutkittua tietoa

T. Dybå, T. Dingsøyr: Empirical Studies of Agile Software Development : A Systematic Review. *Information and Software Technology* 50, 2008, 833-859.

J.E. Hannay, T. Dybå, E. Arisholm, D.I.K. Sjøberg: The Effectiveness of Pair Programming: A Meta-Analysis. *Information and Software Technology* 51, 2009, 1110-1122.

D.I.K. Sjøberg, A. Johnsen, J. Solberg: Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software* 29, 5, 2012, 47-53.

M. Ikonen: Lean Thinking in Software Development: Impacts of Kanban on Projects. PhD thesis, A-2011-4, University of Helsinki, Department of Computer Science, 2011.

S. Mäkinen: Driving Software Quality and Structuring Work through Test-Driven Development. Pro gradu, HY/TKTL, 2012.

V. Kettunen et al.: A Study on Agility and Testing Processes in Software Organizations. In Proc. ISSTA'10. ACM, 2010.

Tutkittua tietoa 1

+ XP-projektit myöhästyvät hieman vähemmän kuin ”perinteiset” projektit
+ XP-projektit ylittävät budjettinsa selvästi vähemmän kuin ”perinteiset” projektit
+ pienten XP-tiimien tuottavuus on useimmiten parempaa kuin vastaavien ”perinteisten” projektiryhmien tuottavuus
+ läsnä oleva asiakas parantaa asiakasyhteistyötä
+ XP-projektien asiakkaat ovat tyytyväisempiä kuin ”perinteisten” projektien asiakkaat
+ XP-projekteissa ohjelmistokehittäjät ovat tyytyväisempiä kuin ”perinteisissä” projekteissa
+ XP toimii parhaiten, kun ohjelmistokehittäjät ovat kokeneita ja hallitsevat sekä toimialan että kehitystyökalut

Tutkittua tietoa 2

+ ketterästi kehitettyjen ohjelmistojen laatu on parempi kuin vastaavien ”perinteisesti” kehitettyjen ohjelmistojen laatu
+ 40 tunnin työviikko vähentää ylitöiden tekemistä

+ testivetoinen kehitys vähentää virheiden määrää
+ testivetoinen kehitys parantaa testauksen koodikattavuutta
+ testivetoinen kehitys pienentää koodin mutkikkuutta (complexity)
+ testivetoinen kehitys lisää ohjelmiston ylläpidettävyyttä
+ testivetoisesti kehitetyn ohjelmiston kokonaiskustannukset ovat pienemmät kuin ”perinteisesti” kehitetyn ohjelmiston, mikäli sen elinkaari on riittävän pitkä

Tutkittua tietoa 3

+ testauksen aloittaminen projektin aikaisessa vaiheessa helpottaa testausresurssien tarpeen arviointia
+ jatkuva ja aikaisin aloitettu testaus antaa enemmän aikaa testauksen tekemiseen
+ jatkuva ja aikaisin aloitettu testaus helpottaa testauksen fokusointia ohjelmiston tiettyihin osiin
+ myös testaus mukautuu helpommin muutoksiin ketterässä kehityksessä

Tutkittua tietoa 4

+ pariohjelmointi parantaa koodin laatua yleisesti ottaen hiukan, kokemattomilla ohjelmoijilla huomattavasti
+ pariohjelmointi nopeuttaa koodin tuottamista
+ pariohjelmointi on parhaimmillaan vaikeissa ja korkeaa laatua vaativissa tehtävissä, joilla ei ole tiukkaa aikataulua
+ ketterät projektit kehittävät opiskelijoiden kommunikointi-, sitoutumis- ja yhteistyötaitoja
+ ohjelmistokehitys on Kanbanilla nopeampaa kuin Scrumilla
+ tehtävät valmistuvat Kanbanilla nopeammin kuin Scrumilla
+ työn tuottavuus on Kanbanilla suurempi kuin Scrumilla
+ Kanban visualisoi hyvin työn etenemistä
+ Kanban tukee tilannekohtaista analyysia ja muutosta

Tutkittua tietoa 5

- työmäärän arviointi menee pahasti metsään XP-projektien alussa (mutta paranee projektien edetessä)
- XP-projekteissa ei kiinnitetä tarpeeksi huomiota ohjelmiston suunnitteluun eikä arkkitehtuuriin
- XP-projektien 40 tunnin työviikko uuvuttaa kehittäjät
- XP ei sovellu suuriin projekteihin
- Scrum-projekteissa pitää kouluttaa myös asiakasta
- Scrumin aikaikkunat ovat keinotekoisia, koska työmäärät yleensä aliarvioidaan ja vastaan tulee ennakoimattomia ongelmia
- Scrumin aikaikkunat johtavat liian nopeasti kehitettyyn ja huonolaatuiseen koodiin
- Scrumin pyrähdyksissä työ jakaantuu epätasaisesti: esimerkiksi testaajilla on pyrähdyksen alussa luppoaikaa ja lopussa liian kiire
- Scrumin järjestämisvaihe (staging) koetaan jätteeksi

Tutkittua tietoa 6

- ketterästi kehitettyjen ohjelmistojen käyttöliittymä on huonompi kuin ”perinteisesti” kehitettyjen ohjelmistojen käyttöliittymä
- ketterissä tiimeissä on vaikeampi vaihtaa ohjelmistokehittäjiä kuin ”perinteisissä” projektiryhmissä
- ketterien projektin onnistuminen ei riipu pelkästään teknisistä vaan myös sosiaalisista taidoista
- aina ketterästi läsnä oleva asiakas stressaantuu ja uupuu

Tutkittua tietoa 7

- testivetoinen kehitys lisää hieman koodin kytkentää (coupling)
- testivetoisesti kehitetyssä ohjelmistossa on lähes yhtä paljon yksikkötestauskoodia kuin varsinaista toiminnallista koodia
- useimmiten testivetoinen kehitys vie enemmän aikaa kuin ”perinteinen” kehitys, lisää kokonaistyömäärää ja kasvattaa kustannuksia
- testivetoinen kehitys vähentää työn tuottavuutta

- pariohjelmointi lisää kokonaistyömäärää
- pariohjelmointi vähentää kokeneiden ohjelmoijien tuottavuutta ja laskee heidän koodinsa laatua
- pariohjelmointi ei sovellu tehtäviin, joilla on tiukka aikataulu, vähäiset resurssit ja korkeat laatuvaatimukset

Tutkittua tietoa 8

- testaukseen erikoistuneille henkilöille on edelleen tarvetta (kehittäjien tekemä testaus ei aina riitä)
- kehityksen rinnalla tehty testaus voi olla vaikea suorittaa ja vaatii erityistä harkintaa ja ohjausta
- dokumentaation puute muodostuu ongelmaksi, jos testausta pitää ulkoistaa tai testaajilla ei ole riittävää kokemusta tuoteympäristöstä
- ketterässä kehityksessä testaajilta vaaditaan parempaa teknistä osaamista

Tutkittua tietoa 9

– testivetoinen kehitys on vaikeaa monille opiskelijoille

– virheitä korjataan Kanbanilla hitaammin kuin Scrumilla
– Kanban ei tarjoa apua ongelmien varsinaiseen ratkaisemiseen
– Kanban-taulu on liian rajoittunut työkalu suuriin projekteihin

± XP, Scrum, lean ja Kanban vaativat johtamista ja hyviä johtajia onnistuakseen; ilman niitä lopputuloksena on useimmiten kaaos ja sekuli