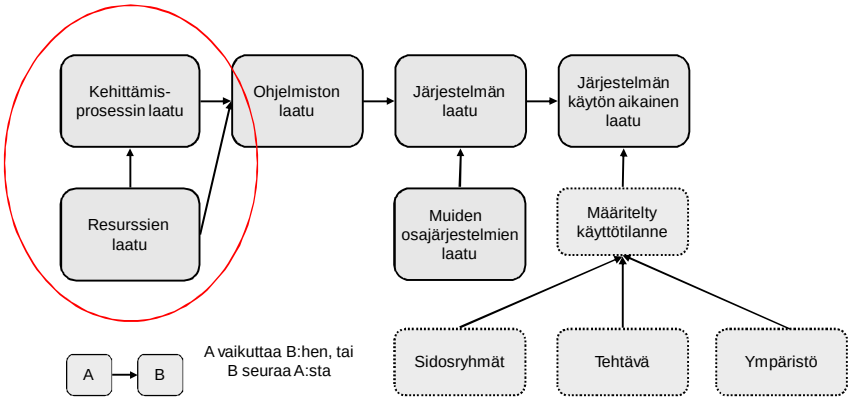
PROSESSIT JA LAATU

Laadun vaikutusketju



[Nev13] Risto Nevalaisen esitys "Järjestelmän ja ohjelmiston mittaaminen". http://www.sfsedu.fi/materiaalit (kts. IT-Standardit)

Kehitysmenetelmät ja -resurssit

- Palvelun toteutuksessa käytettävät resurssit ja menetelmät vaikuttavat kaikkiin laadun alueisiin
  - Käytettävyyssuunnittelijoiden ja ohjelmisto-arkkitehtien sekä –kehittäjien *tiedot, taidot* ja *kokemus* - ja ennen muuta *asenne* (myös johdon)
  - Aikataulu ja budjetti suhteessa toim. määrään
  - Käyttäjien *osallistaminen* kehitykseen ja käyttäjien *palautteen* systeemaattinen *kerääminen* ja *hyödyntäminen*
  - Laatutavoitteiden esilletuonti ja jatkuva arviointi

Prosessit ja laatu

- Kurssin lopuksi perehdymme siihen, miten erityyppisissä ohjelmistokehitysprosesseissa laatu rakentuu ja miten sitä ohjataan haluttuun suuntaan
- Prosessimalleja on seuraavalla noin sadalla dialla käsitelty aika yksityiskohtaisesti
  - Ne käydään luennolla ja ryhmätyöskentelyssä läpi valikoiden ja laatuasioihin keskittyen
  - Eri prosessityyppien perusasiat oletetaan tunnetuiksi aikaisemmilta kursseilta (esimerkiksi Ohjelmistotuotanto)

PERSONAL SOFTWARE PROCESS

Personal Software Process 1

W.S. Humphrey: *Introduction to the Personal Software Process*. Addison-Wesley, 1997.

- henkilökohtaisen ”ohjelmistoprosessin” (PSP) mittaaminen
- oman ohjelmistokehitystyön (”projektin”) mittaaminen
- projektisuunnittelun parantaminen mittaustulosten perusteella
- työn laadun parantaminen mittaustulosten avulla
- *Team Software Process* tuo nämä perusideat kehitystiimin tasolle (edellyttää PSP –käytäntöjä)
  - Capers Jonesin mukaan yksi parhaita malleja hyvän laadun kannalta

Personal Software Process 2

Suunnittelu

- ajankäytön mittaaminen ja kirjanpito
- työajan ja tehtävien jakaminen osiin
- työmäärän arviointi
- ohjelmarivien (LOC) mittaaminen
- tuottavuuden (LOC / h) mittaaminen
- ajankäytön hallinta (mittaaminen ⇒ muutokset)
- tehtävien priorisointi
- aikataulun laatiminen, seuranta ja hallinta (GANTT-kaavio)
- projektisuunnitelman laatiminen, seuranta ja hallinta

Työprosessi

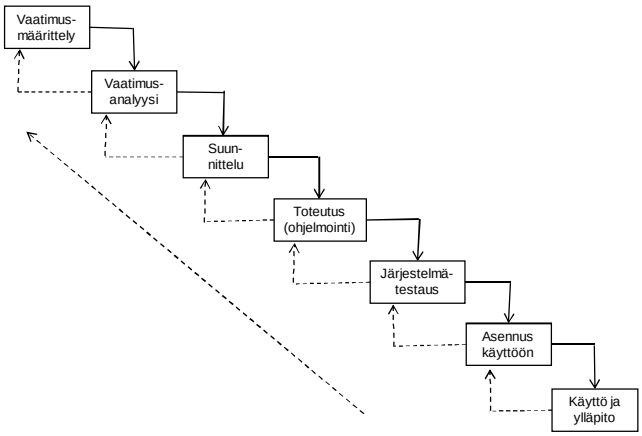
- oman työprosessin määrittely
- (ohjelmointi)virheiden luokittelu ja kirjaaminen
- virheenjäljitys ja -korjaus
- virheenjäljitys- ja korjausajan mittaaminen
- virhetiheyden (virheitä / KLOC) mittaaminen
- koodin katselmointi tarkistuslistan avulla
- laadun valvonta: virhemäärien ja virhetiheyden seuranta vaiheittain
- prosessin valvonta: katselmointiaika vs. testausaika vs. ohjelmointiaika

ITERATIIVINEN OHJELMISTO-KEHITYS

Iteratiivinen ohjelmistokehitys

- Yleinen (tai ainakin ”virallinen”) käsitys 1990-luvun loppupuolelle asti oli, että laatu saavutetaan huolellisella määrittelyllä ja hallinnalla.
- Käytännössä tämä tarkoitti, että kaikki oleelliset asiat kirjattiin noudatettaviksi säännöiksi. Sääntöjä voitiin toki muuttaa, jos ne osoittautuivat huonoiksi, mutta muutosprosessi oli hidas.
- Ohjelmistoja suunniteltiin ja toteutettiin samoilla periaatteilla kuin mitä tahansa teollisuustuotteita. Tämä on *ennustettavissa olevaa valmistusta* (predicatble manufacturing), siis massatuotantoa.

Klassinen vesiputousmalli



Vesiputousmallin hyvät ja huonot puolet

- Hyvää: selkeä, yksinkertainen, ennalta suunniteltavissa, vakiintunut ohjelmistotuotannon vaihejako
- Huonoa: liian kankea muutoksille, epärealistinen, liian myöhäinen ja kallis ongelmien havaitseminen, liian myöhään testattavissa
- Huom – ”vesiputous” on esimerkki *plan-driven* eli *suunnittelun roolia* (teknistä ja projektissuunn.) ja hallittavuutta korostavista prosesseista. Esiintyy harvoin ”puhtaana”.

Massatuotanto

- Massatuotannon vaatimuksena on toimiva *spesifikaatio*: sinikopio (blue print) tehtävästä tuotteesta.
- Spesifikaatio ei tule tyhjästä, vaan sen saamiseksi tehdään paljon suunnittelua ja prototyyppejä yrityksen ja erehdyksen kautta. Tämä on *uuden tuotteen kehitystä* (new product development).
- Perinteisessä ohjelmistokehityksessä uuden tuotteen kehitystä tehtiin osana massatuotantoa. Tekniikka ei sopinut mihinkään ohjelmistoprojekteihin hyvin ja harvoihin edes kohtalaisesti.

Massatuotannon ja kehitystyön vertailu

Perinteistä ohjelmistotuotantoa lukuun ottamatta missään insinöörityössä ei yritetä yhdistää massatuotantoa ja uuden tuotteen kehitystä. Alueet eroavat selvästi toisistaan:

Massatuotanto	Kehitystyö
Lopullinen spesifikaatio saadaan käyttöön ennen tuotantoa.	Valmis muuttumaton spesifikaatio ei yleensä ole saatavilla.
Luotettava kustannus- ja työmääräarvio saadaan ennen tuotantoa.	Kustannus- ja työmääräarvio tarkentuu huomattavasti työn aikana.
Vaadittavat tehtävät voidaan tunnistaa, määritellä, skeduloida ja järjestää etukäteen.	Tehtävät tarkentuvat ja mukautuvat projektin aikana. Toteutus-palaute-sykli tarvitaan.
Tuotantoaikaisten muutosten määrä on suhteellisen pieni. Ennustamattomiin tapahtumiin ei juuri varauduta.	Jatkuva varautuminen ennustamattomiin muutoksiin. Muutosten määrä on suuri.

Havaintoja

- **Ohjelmistotuotanto ei ole ennustettavaa rutiinimaista massatuotantoa vaan uuden tuotteen kehitystyötä**.
- Useimmissa ohjelmistotuotantoprojekteissa massatuotannon menetelmät eivät toimi. Ennalta määritellyt aikataulu, budjetti ja tehtävät eivät toteudu, koska valmista spesifikaatiota ei ole saatavilla ennen projektin alkua.
- Ohjelmistotuotannossa on varauduttava muutokseen. Lähes jokaisen ohjelmiston spesifikaatio muuttuu kehitysprojektin aikana.
- Kuitenkin: ohjelmistotuotantoon pyritään 2010-luvulla jälleen ottamaan mallia massatuotannosta, kun ”virtaviivainen” (lean) ohjelmistokehitys perustuu Japanissa 1950-luvulla luotuihin autonvalmistuksen periaatteisiin.

Valmiin spesifikaation ongelma

Entä jos spesifikaation saisi valmiina ja muuttumattomana? Tämä onnistuu valitettavan harvoin:

- sidosryhmät eivät tiedä, mitä haluavat ohjelmistolta
- heidän on vaikeaa ilmaista toiveitaan ja taustatietojaan
- suuri osa yksityiskohdista selviää vasta kehitystyössä
- yksityiskohtien määrä ja kompleksisuus ylittävät aivojen hahmotuskyvyn
- sidosryhmät muuttavat mieltään projektin aikana
- ulkoiset tekijät (kuten kilpailijat ja taloustilanne) vaikuttavat spesifikaatioon

Evoluutio ja iteratiivisuus

- Vaikka ohjelmistotuotannon pitää olla uuden tuotteen kehitystyötä, sen ei tarvitse olla hallitsematonta.
- Ohjelmistoprojektilla on aikataulu, budjetti, tehtävät ja spesifikaatio. Niitä ei kuitenkaan kiinnitetä projektin alussa, vaan ne elävät koko projektin ajan. Tämä on *evoluutiolähtöistä kehitystyötä* (evolutionary development).
- Yleensä tällaiset projektit ovat *iteratiivisia:* kehitystyö tehdään askel kerrallaan pieninä miniprojekteina, joista jokainen sisältävää perustehtäviä, kuten vaatimusmäärittelyä, suunnittelua, toteutusta ja **validointia**.

Ideoista prosessimalleihin

- *Iteratiivinen (ja evoluutio-)ohjelmistokehitys* (iterative and evolutionary software development) on prosessimallien perhe, jossa ohjelmiston elinkaari muodostuu useasta peräkkäisestä iteraatiosta.
  - Yhden iteraation suositeltava pituus on 1-6 viikkoa. Siihen kuuluu aina toteutusta eli uuden koodin kirjoittamista.
  - Useimmissa projekteissa on ainakin kolme iteraatiota.
- Iteratiivinen ohjelmistokehitys on vanha idea. Ensimmäiset iteratiiviset prosessit otettiin käyttöön 1960-luvulla, mutta 1970-luvulla ne menettivät teollisuudessa suosiota lineaarisille prosessimalleille.

Julkaisu

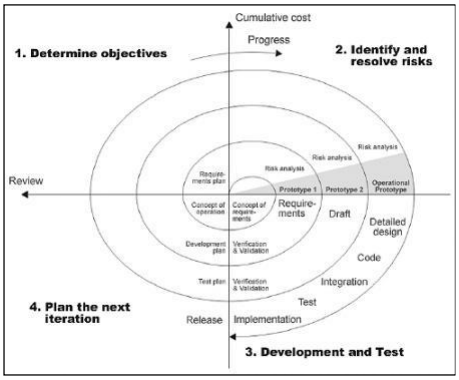
- Jokaisen iteraation tuloksena saadaan *julkaisu* (iteration release).
  - Julkaisu on lopullisen järjestelmän toimiva osa. Prototyyppi ei kelpaa julkaisuksi.
    - Käytännössä projekteissa on myös iteraatioita, joista ei synny uutta julkaisua. Näin käy silloin, kun koodia täytyy refaktoroida (eli siistiä) tai kun ohjelmistoa ei ole saatu testattua tarpeeksi aiemmissa iteraatioissa.
  - Kaikki julkaisut eivät ole julkisia eli loppukäyttäjälle tarkoitettuja.
  - Sisäiset julkaisut ovat kehitystiimin ja asiakkaan käytössä.
  - Julkiset julkaisut ovat loppukäyttäjälle tarkoitettuja versioita tuotteesta.
  - Joskus viimeinen julkaisu on lopullinen tuote, mutta nykyisin kehitystyö jatkuu yleensä tuotteen koko elinkaaren ajan. Tällöin tuotteesta tehdään useita julkisia julkaisuja.
  - Jos projektissa on monta kehitystiimiä, julkaisuun integroidaan kaikkien kehitystiimien tuotokset.

Iteratiivinen ja lisäävä kehitystyö

- Ohjelmistotyötä, jossa tuotteeseen lisätään sykleittäin uusia ominaisuuksia, kutsutaan *lisääväksi ohjelmistokehitykseksi* (incremental development)
- Jos lisäykset liittyvät iteraatioihin, saadaan *iteratiivinen ja lisäävä ohjelmistokehitys* (Iterative and Incremental Development, IID)
- Lähes kaikki modernit ohjelmistoprosessit ovat IID-variantteja
- Erityisesti kaikki nykyiset *ketterät* prosessit ovat IID-variantteja

Spiraalimalli

Iteratiivisten ja lisäävien ohjelmistoprosessimallien klassikko:
spiraalimalli (spiral model, Boehm 1986)



Kehitys tapahtuu sykleissä, joissa
1. selvitetään projektin menestymisen kannalta kriittisten sidosryhmien asettamat tavoitteet
2. kehitetään ja evaluoidaan vaihtoehtoisia ratkaisuja
3. tunnistetaan ja ratkaistaan valittuun ratkaisuun liittyvät riskit
4. haetaan sidosryhmien hyväksyntä ja lupa seuraavaan kierroksen (syklin) käynnistämiseen

- *Eri sykleissä* voidaan noudattaa *eri menetelmiä* (vesip., agile)

http://en.wikipedia.org/wiki/Spiral_model

Iteraation pituus

- IID ei määrittele iteraation pituutta, mutta moderneissa iteratiivisissa prosessimalleissa iteraatiot ovat lyhyitä.
  - Vanhemmissa prosessimalleissa *syklin* (cycle) pituus on 3-6 kuukautta, joskus jopa enemmän.
    - Sykli on hiukan eri asia kuin iteraatio. Sykli tarkoittaa aikaa, missä käydään läpi kaikki työvaiheet vaatimusmäärittelystä testaukseen. Tuloksena ei välttämättä saada julkaisua vaan esimerkiksi prototyyppi. Iteraation tuloksena saadaan aina julkaisu.
  - Lyhyt sykli tarkoittaa nopeaa reagointia muutokseen. Toisaalta se myös tarkoittaa, että tehtävä ohjelmisto on ositettava hyvin pieniin toteutettaviin osiin ja että tuottamaton työ lisääntyy projektissa (syklin vaihtoon liittyvä työ).

Iteraation työvaiheet

- Iteraatiossa perustehtävien painotukset vaihtelevat sen mukaan, missä vaiheessa tuotteen elinkaarta ollaan:
  - ensimmäisillä iteraatioilla painotetaan vaatimusmäärittelyä ja vaatimusten validointia
  - myöhemmillä sykleillä vaatimusmäärittelyn paino vähenee ja suunnittelun paino lisääntyy
  - vielä myöhemmillä sykleillä vaatimusmäärittelyn ja suunnittelun paino vähenee ja toteutuksen ja järjestelmätestauksen paino lisääntyy
  - ylläpitovaiheessa toteutuksen (plus refaktoroinnin) ja regressiotestauksen paino lisääntyy

Riskilähtöinen ja asiakaslähtöinen iteraatiosuunnittelu

- Iteraatiossa toteutettavat ominaisuudet voidaan valita *riskilähtöisesti* (risk-driven) tai *asiakaslähtöisesti* (client-driven)
  - riskilähtöisessä kehitystyössä iteraatioon valitaan ne tehtävät, jotka ovat vaikeimmat ja riskialtteimmat toteuttaa
  - asiakaslähtöisessä kehitystyössä iteraatioon valitaan ne tehtävät, jotka asiakas asettaa korkeimmalle prioriteetille
- Riskilähtöinen kehitystyö helpottaa projektia, mutta asiakas ei välttämättä pidä lähestymistavasta. Usein syvällä olevat vaikeat ratkaisut eivät suoraan näy parantuneena toiminnallisuutena.
- Asiakaslähtöinen kehitystyö pitää asiakkaan tyytyväisenä, mutta voi johtaa raskaaseen refaktorointiin tai huonoon arkkitehtuuriin.

Aikaikkunat 1

- IID:ssa iteraation kesto kiinnitetään etukäteen. Tätä kutsutaan *timeboxing*-tekniikaksi (aikaikkuna, aikasidonnaisuus).
- Aikaikkuna-tekniikassa iteraation kestoaika ei jousta. Jos allokoituja tehtäviä on jäljellä enemmän kuin iteraatiolle varattuun aikarajaan mahtuu, niitä karsitaan.
- Iteraation tuloksena saadaan aina sovittuna ajankohtana julkaisu eli lopullisen tuotteen toimiva osa.
  - Mutta joka ei ehkä sisällä aivan kaikkia aikaikkunan alussa aiottuja piirteitä (kts. edellinen kohta)

Aikaikkunat 2

- Aikasidonnaisuus ei tarkoita, että tiimiläisten täytyy tehdä ylitöitä pitääkseen iteraation takaraja
  - joskus ylityöt ovat ok, kunhan sekä tiimi että projektin johto hyväksyvät ne
  - pääsääntöisesti pitäisi kuitenkin seurata normaalia työviikkoa ja ylitöiden sijaan karsia toteutettavia ominaisuuksia
- Aikasidonnaisuus ei tarkoita, että kaikkien iteraatioiden olisi oltava saman mittaisia. Projektin iteraatioiden pituudet voivat vaihdella keskenään, mutta yhden iteraation pituus ei muutu sen aloituksen jälkeen.
- Kaikki yleisimmät ketterät prosessimallit suosittelevat vahvasti tai jopa vaativat aikaikkunoiden käyttöä.

Muutokset iteraatioissa

- IID:ssa varaudutaan muuttuviin vaatimuksiin, mutta ei koska tahansa. Muutoksiin varaudutaan iteraatioiden välillä, ei niiden sisällä.
- Iteraation käynnistymisen jälkeen ulkoiset sidosryhmät eivät vaikuta sen sisältöön:
  - kun tiimi on aloittanut sovitun iteraation toteutuksen, asiakas, projektin vastuuhenkilö tms. ei voi pyytää tai käskeä tiimiä tekemään ylimääräistä
  - tiimi itse voi vähentää iteraatiossa tehtäviä tehtäviä, jos aikaikkuna ei muuten pidä
  - jos iteraatio on menossa todella pahasti metsään, se voidaan keskeyttää ja aloittaa uusi iteraatio etuajassa

IID ja vaatimusmäärittely 1

- IID:n iteraatiossa toteutetaan sovitut tehtävät, mutta mistä ne saadaan?
- Tehtävät saadaan alun perin *vaatimusmäärittelystä* aivan kuten suunnitelmakeskeisissä prosesseissa.
- Vaatimusmäärittelyä tehdään rinnan ohjelmisto-kehityksen kanssa. Vaatimusmäärittely voi olla osa iteraatiota, mutta se voi myös olla iteraatioista erillään.
- IID:n kannalta riittää, että kussakin iteraatiossa tiedetään, mitä tehtäviä siihen kuuluu.

IID ja vaatimusmäärittely 2

- IID sopii projekteihin, joissa vaatimukset muuttuvat. Mutta
  - miten paljon vaatimukset muuttuvat?
  - mitkä vaatimukset saavat muuttua?
- Useimmissa projekteissa *suurin osa* vaatimuksista on melko stabiileja. Nämä vaatimukset tunnistetaan (ja toteutetaan) aikaisissa iteraatioissa.
- Osa vaatimuksista elää. Asiakas ei osaa heti kertoa, mitä tarvitaan, tai ongelmakenttä muuttuu. Nämä vaatimukset tunnistetaan ja toteutetaan projektin edetessä.
- Tärkeimpiä tunnistettavia vaatimuksia ovat **laatutekijöihin liittyvät vaatimukset**. Ne pitää tunnistaa mahdollisimman aikaisin, jotta ohjelmiston arkkitehtuuri saadaan räätälöityä niille sopivaksi. Niiden pitää myös olla mahdollisimman stabiileja.
  - Yleensä laatutekijöiden stabiilius ei ole ongelma. Lähinnä tehokkuus-laatutekijä voi tuottaa ongelmia.

IID ja aikataulu

- Yksi tavallisimmista IID:n kritiikin kohteista on projektin aikataulutus. IID:n mukautuvan luonteen johdosta varsinkin projektin alussa on vaikea tehdä edes alustavaa aikataulua.
- Keskeiset termit ovat *projektin alussa* ja *alustava aikataulu*:
  - jo muutaman iteraation jälkeen tiimillä on useimmiten jo melko hyvä käsitys siitä, kuinka monta iteraatiota työ kaiken kaikkiaan vaatii
  - mikä tahansa projektin alussa tehty aikataulu on vain alustava, ja myös suunnitelmakeskeisten projektien alussa tehdyt aikataulut ovat summittaisia
- Ongelman ratkaisuna on
  - siirtää yleisen aikataulun tekoa projektin alusta muutaman iteraation päähän ja
  - tehdä yksityiskohtaista aikataulusuunnittelua korkeintaan muutaman iteraation verran eteenpäin

KETTERÄ OHJELMISTOKEHITYS

Ketterä ohjelmistokehitys

- *Ketterä ohjelmistokehitys* (agile development) ja *ketterät prosessimallit* (agile process models) ovat IID:n osajoukko. Ne sisältävät IID:n periaatteita, kuten aikaikkunat ja viivästetyn projektisuunnittelun, mutta niiden lisäksi ne korostavat *muutosten hyväksymistä* (embrace change).
- Ketterässä ohjelmistokehityksessä oleellista on *ohjautuvuus* (maneuverability). Se tarkoittaa, että ketterät projektit ovat valmiita vaihtamaan suuntaa aina kun tarve vaatii.

Ketterien prosessien muodollisuus

- *Muodollisuus* (ceremony) määrittelee, miten paljon prosessissa on dokumentaatiota, matemaattista formalismia, tarkastuksia yms.
- Ketteriin prosesseihin yhdistetään helposti epämuodollisuus, mutta tämä on myytti. Ketterät prosessit eivät ole sen epämuodollisempia kuin suunnitelmakeskeiset prosessit.
  - Esimerkiksi Scrumissa ei oteta kantaa siihen, miten muodollinen projektin tulee olla. Muodollisuuden aste jätetään projektin päätettäväksi.
  - XP:ssa dokumentaatio pidetään minimissä, mutta vastaavasti XP sisältää paljon muodollisia tekniikoita, kuten pariohjelmoinnin ja testivetoisen ohjelmistokehityksen.
  - Amblerin kehittämä menetelmäjoukko *Agile Modeling* (www.agilemodeling.com) lisää mallinnuksen ketteriin prosesseihin. Menetelmät sopivat käytännössä mihin tahansa ketterään prosessimalliin.

Agile Alliance

- Ketterille prosessimalleille ei ole yhtä ja ainoaa määritelmää. Parhaiten ketteriä prosessimalleja kuvaavat Agile Alliancen vuonna 2001 esittelemät *ketterien prosessimallien manifesti* (Agile Manifesto) ja *ketterät periaatteet* (agile principles).
- Agile Alliance voi edelleen hyvin. Organisaatiossa on kehitelty ketterien prosessimallien yleisiä periaatteita ja tehty ketteriä prosesseja tutuksi.
  - Agile Alliance löytyy osoitteesta http://www.agilealliance.com
- Ketterille prosessimalleille on mahdollisesti tulossa standardi.

Agile Manifesto

Koska ketterien prosessien manifesti menettäisi liikaa käännöksessä, se on tässä alkuperäisessä muodossa (Agile Alliance, 2001):

Agile Manifesto

Individuals and interactions	over processes and tools
Working software	over comprehensive documentation
Customer collaboration	over contract negotiation
Responding to change	over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Ketterät periaatteet 1

Myös Agile Alliancen ketterät periaatteet on paras antaa englanniksi:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Ketterät periaatteet 2

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinetely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Ketterien periaatteiden tulkintaa

Ketteristä periaatteista saadaan hyvä kuva siitä, mitä ketteriin prosesseihin sisältyy:

- projektit ovat joustavia (2, 3)
- toimiva kommunikaatio on erityisen tärkeää (4, 6, 11, 12)
- asiakastyytyväisyys on tärkein laadun mittari (1, 2, 3, 7)
- työntekijöiden tyytyväisyys on tärkein laadun tae (4, 5, 6, 8)
- kehitystyö on systemaattista (1, 3, 9, 10, 11, 12)

Ketterä projektinhallinta

- Ketterät prosessit korostavat tiimityöskentelyä. Tiimi toimii yhtenä tasa-arvoisena yksikkönä.
  - Perinteiset projektipäällikön tehtävät kuuluvat ketterissä prosesseissa koko tiimille. Koko tiimi esimerkiksi vastaa aikataulusta, resurssi- ja kustannusarvioista sekä riskienhallinnasta.
- Tiimeillä on toki johtaja, mutta ei perinteisessä mielessä.
  - Ketterän projektitiimin johtajan tehtäviä:
    - tiimiläisten valmennus
    - työtä hidastavien ongelmien (”töyssyjen”) tasoitus
    - resurssien hankkiminen
    - ketterien periaatteiden ylläpito projektissa
  - Hyvässä ketterässä projektissa johtajan ei juurikaan tarvitse käskeä tiimiläisiä.

Minimalismin periaate

- Ketterä periaate 10 yksinkertaisuudesta (”Simplicity is essential”) on helppo ymmärtää väärin. Sitä tulkitaan usein siten, että ketterät projektit ovat holtittomia ja hallitsemattomia. Tämä ei pidä paikkaansa. Tarvitaan ammattimaista asennetta ja laatutietoisuutta.
- Periaate 10 on minimalismin periaate. Sen mukaan asiat pitää tehdä niin yksinkertaisesti kuin mahdollista, *mutta ei yhtään yksinkertaisemmin.*
  - Periaate 10 koskee koko projektia, ei vain koodausta. Mikä on yksinkertaisin toimiva tapa kerätä projektin vaatimuksia? Mikä on yksinkertaisin toimiva tapa tehdä jatkuvaa testausta? Jos yksinkertaisin tapa on rakentaa testausta varten työkalu, niin sitten sellainen rakennetaan.

Ketterät prosessit ja systemaattisuus

- Ketterät prosessit eivät ole holtittomia tai hallitsemattomia. Niissä on paljon rakennetta, ohjausta ja hallintaa. Ne ovat systemaattisia.
- Ero suunnitelmakeskeisiin prosesseihin on siinä, että ketterissä prosesseissa lähtökohtana on *muutos*.
- Muutokseen varautuminen pakottaa joustaviin ratkaisuihin projektinhallinnassa, aikataulutuksessa, resurssien arvioinnissa ja dokumentaatiossa.
- Joustavuus saavutetaan sillä, että ketterien menetelmien periaatteet ovat vahvasti julkaisu- ja laatukeskeisiä. *Toimiva laadukas tuote on kattavaa dokumentaatiota tärkeämpi ketterän projektin tuotos*.

Säännöt vastaan periaatteet

- Ehkä tavallisin virhe siirryttäessä ketterään ohjelmistokehitykseen on määritellä suuri määrä prosessissa seurattavia sääntöjä
  - tuloksena saadaan jäykkä prosessi, joka ei ole joustava eikä työntekijäystävällinen
  - ajan myötä kommunikaatio saattaa jäädä sääntöjen tieltä vähemmälle
  - kommunikoinnin kärsiessä saadut julkaisut eivät vastaa asiakkaan tarpeita, jolloin asiakastyytyväisyys kärsii
  - lopulta toimimattomia sääntöjä noudatetaan epäsäännöllisesti, jolloin kehitystyön systemaattisuus kärsii
- Sääntöjen sijaan ketterissä prosesseissa suositaan ”periaatteita”. Ne määrittelevät raamit, joiden puitteissa projektit voivat toimia. Jos jotkut periaatteista eivät sovi tiimille, niistä pitää luopua.

Ketterien prosessien vaikutus ohjelmistokehitykseen

- Ketterät prosessimallit eivät luoneet iteraatioita, lyhyitä syklejä tai usean julkaisun esittelyä. Kaikki nämä keksittiin jo 1960-luvulla ennen vesiputousmallia.
- Royce ”esitteli” vesiputousmallin 1970, minkä jälkeen lineaarinen suunnitelmakeskeinen ohjelmistokehitys alkoi saada erityisesti yritysten johtotason suosiota.
  - Royce ei itse asiassa esitellyt lineaarista mallia. Artikkelissaan hän ehdotti mallia, jossa kehitystyö tehdään kahdessa iteraatiossa.
  - Ilmeisesti väärinkäsityksen johdosta Roycen artikkelia pidettiin todisteena johtajien suosimasta lineaarisesta mallista.
- Sen sijaan pariohjelmointi, testivetoinen ohjelmistokehitys ja jatkuva integrointi ovat melko tuoreita ja nimenomaan ketterään ohjelmistokehitykseen liittyviä tekniikoita.
- Tekniikoita tärkeämpää ketterissä prosesseissa on filosofia. Vaikka tekniikat kehitettiin aikaisemmin, vasta ketterissä prosessimalleissa ne yhdistettiin yhteisen filosofian alle.

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)
    - työlista sisältää tuotteelle tähän mennessä löydetyt *vaatimukset*
    - työlistaa päivitetään sitä mukaa kun uusia vaatimuksia ja niihin liittyviä tehtäviä syntyy ja vanhat vaatimukset muuttuvat
    - 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)
    - 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

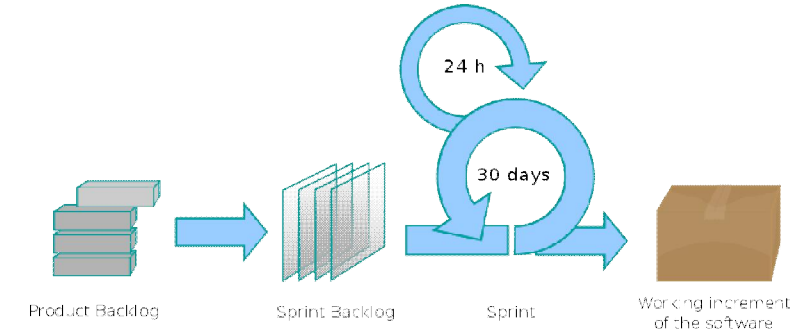
Scrum: Julkaisu

- 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.

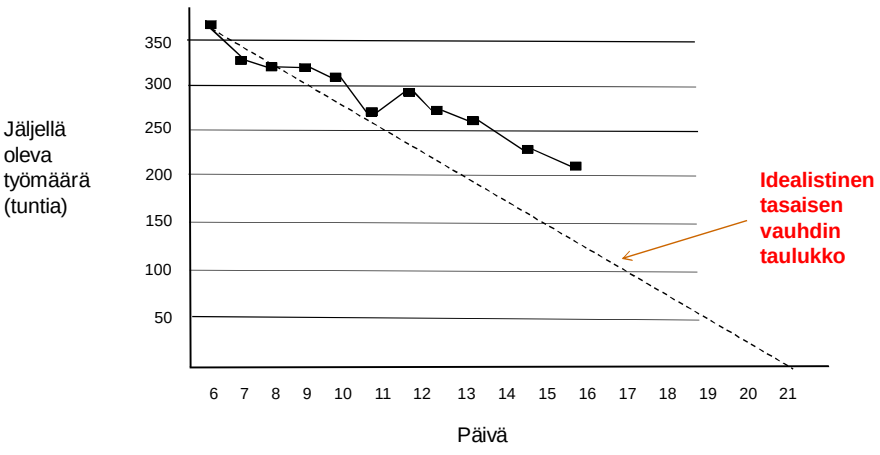
Pyrähdyksen työlista

- 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)
   - jäte: kaikki se, mikä ei tuota arvoa asiakkaalle
   - tavoitteena poistaa ohjelmistokehityksestä kaikki turha työ
2. Jatkuva laadunvarmistus (build quality in)
   - virheiden välttäminen koko kehitysprosessin ajan, ei vasta (erillisessä) testausvaiheessa
3. Tietämyksen luominen (create knowledge)
   - ohjelmiston kehittäminen asteittain tarkentamalla
   - 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

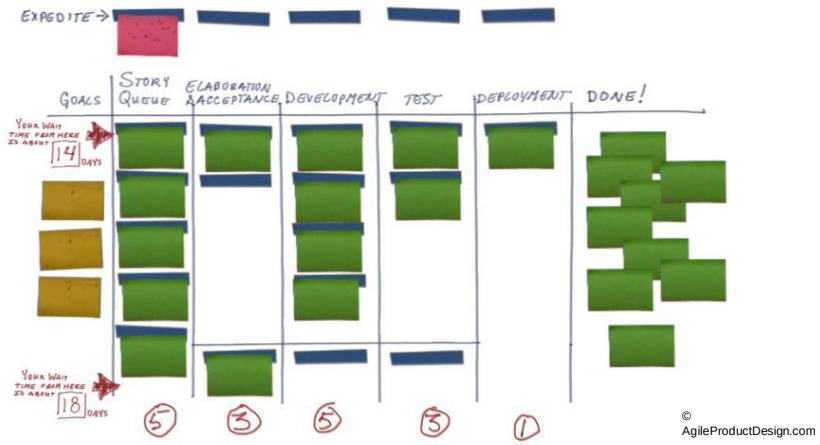
Lean-jäte 2

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

Kanban

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