Hajaantuminen

- tällä hetkellä ohjelmistotuotantoa kuvaa *hajaantuminen ja erikoistuminen*
- perusperiaatteet ovat säilyneet ennallaan, mutta
- yritykset käyttävät omia *räätälöityjä* prosessimalleja, menetelmiä ja työkaluja

Erikoistuminen

- valmistettavat ohjelmistot eroavat toisistaan enemmän kuin koskaan
  - esim. kännyköihin ja mobiililaitteisiin tehtävät suhteellisen pienet ohjelmistot
  - toisaalta valtavia ohjelmistoja, joiden suorituskyky ja käyttöturvallisuus on määritelty erittäin korkealle.
- ohjelmistoyritysten täytyy erikoistua säilyttääkseen markkinansa

Ohjelmisto ja järjestelmä

Järjestelmä

- *Järjestelmä* (system)
  - Joukko toisiinsa liittyviä laitteistokomponentteja (ja mahdollisesti ohjelmistokomponentteja), jotka toimivat yhdessä jonkin päämäärän saavuttamiseksi
  - käyttäjät, käyttöympäristöt (sosiotekninen järjestelmä)
- Systems engineering
  - toimintatavat, joiden tarkoituksena varmistaa, että järjestelmä tai palvelu, jota ollaan tuottamassa vastaa sille asetettuja vaatimuksia

Tietokonepohjainen järjestelmä

- *Tietokonepohjainen järjestelmä (*computer-based system)
  - Joukko laitteisto- ja ohjelmistokomponentteja, jotka toimivat yhdessä jonkin päämäärän saavuttamiseksi
  - Käyttäjät, käyttöympäristöt
  - Voi olla osa laajempaa (sosioteknistä) järjestelmää

Osajärjestelmät ja käyttäjät

- Yleensä järjestelmä koostuu useasta osajärjestelmästä.
  - Kukin osajärjestelmä on itsenäinen kokonaisuus, joka toimii yhteistyössä muiden osajärjestelmien kanssa.
  - Osajärjestelmä rakentuu laitteisto- ja ohjelmistokomponenteista, jotka toteuttavat yhteistyössä osajärjestelmän tehtävät.
  - Käyttäjät käyttävät järjestelmää yhden tai useamman osajärjestelmän kautta (kälit).

Ohjelmisto ja sen ympäristö

- *Ohjelmisto* (software) on kokoelma yhteistyössä toimivia tietokoneohjelmia, ohjelmien käyttämiä tiedostoja ja niihin liittyviä dokumentteja.
- Ohjelmistoa käytetään aina jossakin ympäristössä
  - Ohjelmisto- ja laitteistoympäristö
  - Käyttöympäristö: missä käytetään, ketkä käyttävät

Järjestelmän ja ohjelmiston laadinta

- Järjestelmää ja ohjelmistoa on laadittava yhteistyössä:
  - Ohjelmistotuotantoa ei voi tehdä ilman tietoa ympäristöstä, johon ohjelmistoa ollaan tekemässä
  - Yhteistyö edellyttää ohjelmiston laatijoilta valmiutta kommunikoida sovellusalueen asiantuntijoiden kanssa

Mitä vaaditaan hyvältä ohjelmistolta?

- Hyvällä ohjelmistolla on seuraavat keskeiset ominaisuudet:
  - *Ylläpidettävyys:* tuotetta voidaan muokata vastaamaan muuttuviin vaatimuksiin.
  - *Luotettavuus:* tuote ei aiheuta fyysisiä eikä taloudellisia vahinkoja missään tilanteessa.
  - *Tehokkuus:* tuote ei tuhlaa resursseja.
  - *Käytettävyys:* tuote on looginen ja helppokäyttöinen.

Ohjelmistotuotannon prosessimalleja

Ohjelmistotuotantoprosessi

- *Ohjelmistotuotantoprosessi* (software process) on joukko toimintoja, jotka johtavat ohjelmiston valmistumiseen
- Prosessi on ohjeisto, joka kertoo työvaiheet ja niiden väliset suhteet
- Prosessin ilmentymä on *projekti:* projektissa toteutetaan prosessin työvaiheet
- Käytännössä jokainen ohjelmistoyritys käyttää itselleen räätälöityjä prosesseja.

Prosessimalli

- *Prosessimalli* (process model) on abstrakti kuvaus prosessista.
  - Prosessimallista voidaan johtaa monta erilaista prosessia eri organisaatioihin ja sovellusalueille.
- Prosessimalli kuvaa yleisperiaatteen, ei yksityiskohtia. Näin prosessimalli määrittelee kehykset, joihin itse prosessit rakennetaan.

Prosessimallien perustehtävät

- Jokaiseen prosessimalliin sisältyy tavalla tai toisella seuraavat *perustehtävät*:
  - Vaatimusten keruu ja analyysi (vaatimusmäärittely),
  - Ohjelmiston suunnittelu,
  - Toteutus ja yksikkötestaus,
  - Integrointi ja järjestelmätestaus,
  - Käyttö ja ylläpito (evoluutio).

Vaatimusmäärittely

- Mitä vaatimuksia ohjelmistolle asetetaan:
  - mitä toimintoja sen pitää sisältää
  - liittyykö toimintaan laadullisia vaatimuksia (esim. nopeuden tai käyttötavan suhteen)
- Mitä rajoituksia ohjelmiston toimintaan liittyy:
  - esim. yhteistoiminta muiden ohjelmistojen kanssa

Ohjelmiston suunnittelu

- Suunnittelu kuvaa
  - ohjelmiston rakenteen
  - ohjelmiston tarvitsemat ja käsittelemät tiedot
  - komponenttien väliset rajapinnat
  - käytetyt algoritmit.
- Suunnittelu perustuu kerättyihin vaatimuksiin
- Suunnittelun tuloksena saadaan yksi tai useampi ohjelmiston toteutuksessa käytettävä malli ohjelmiston rakenteesta

Suunnittelun tasot

- Suunnitteluprosessi vaatii useita työvaiheita, jotka käsittelevät ongelmaa eri tarkkuustasoilla, esim.
  - Arkkitehtuurisuunnittelu
    - Tehdään järjestelmästä korkean tason malli, josta selviävät osajärjestelmät ja niissä käytettävät komponentit
  - Abstrakti määrittely
    - Määritellään kunkin osajärjestelmän tarjoamat palvelut ja rajoitteet
  - Rajapintasuunnittelu
    - Määritellään kunkin osajärjestelmän tarjoamat rajapinnat muille osajärjestelmille

Suunnittelun tasot 2

- Komponenttisuunnittelu
  - Määritellään osajärjestelmittäin palvelut toteuttavat komponentit ja suunnitellaan ne
  - Komponentit voivat olla esimerkiksi olioluokkia tai luokkaryhmiä
- Tietorakenteiden suunnittelu
  - Suunnitellaan järjestelmän toteutuksessa välttämättömät tietorakenteet
- Algoritmien suunnittelu
  - Suunnitellaan järjestelmän toteutuksessa käytettävät algoritmit

Toteutus ja yksikkötestaus

- Komponentit toteutetaan suunnitelman pohjalta
  - Ohjelmakomponentti: yhden asian (eikä muuta) kunnolla tekevä ohjelman osa, jolla on selkeä rajapinta ulospäin
  - Komponentti voi olla kooltaan mitä tahansa olioluokasta kokonaiseen ohjelmaan
- Toteutuksen osana *yksikkötestauksessa* varmistetaan, että komponentit toimivat luvatulla tavalla

Integrointi ja järjestelmätestaus

- Ohjelmakomponentit kootaan yhteen
- Komponenttien välinen yhteistyö testataan *integrointitestauksessa*
- Komponenteista kootaan osajärjestelmiä (itsenäisiä osia).
- Osajärjestelmien yhteistyö testataan
- Koko järjestelmä testataan todellisessa käyttöympäristössä ➔ *järjestelmätestaus*

Ohjelmiston käyttö ja ylläpito

- Ohjelmistoa on usein tarvetta muuttaa myöhemminkin:
  - Käytössä paljastuu virheitä
  - Käyttäjien vaatimukset täsmentyvät tai muuttuvat
  - Käyttöympäristö ja laitteisto muuttuvat
  - Käyttötavat muuttuvat
- *Ylläpito* = ohjelmiston muuttaminen käyttöönoton jälkeen
- *Evoluutio* = väistämättömät tekijät, jotka pakottavat ohjelmiston muuttumaan

Prosessimallien tyypittelyä

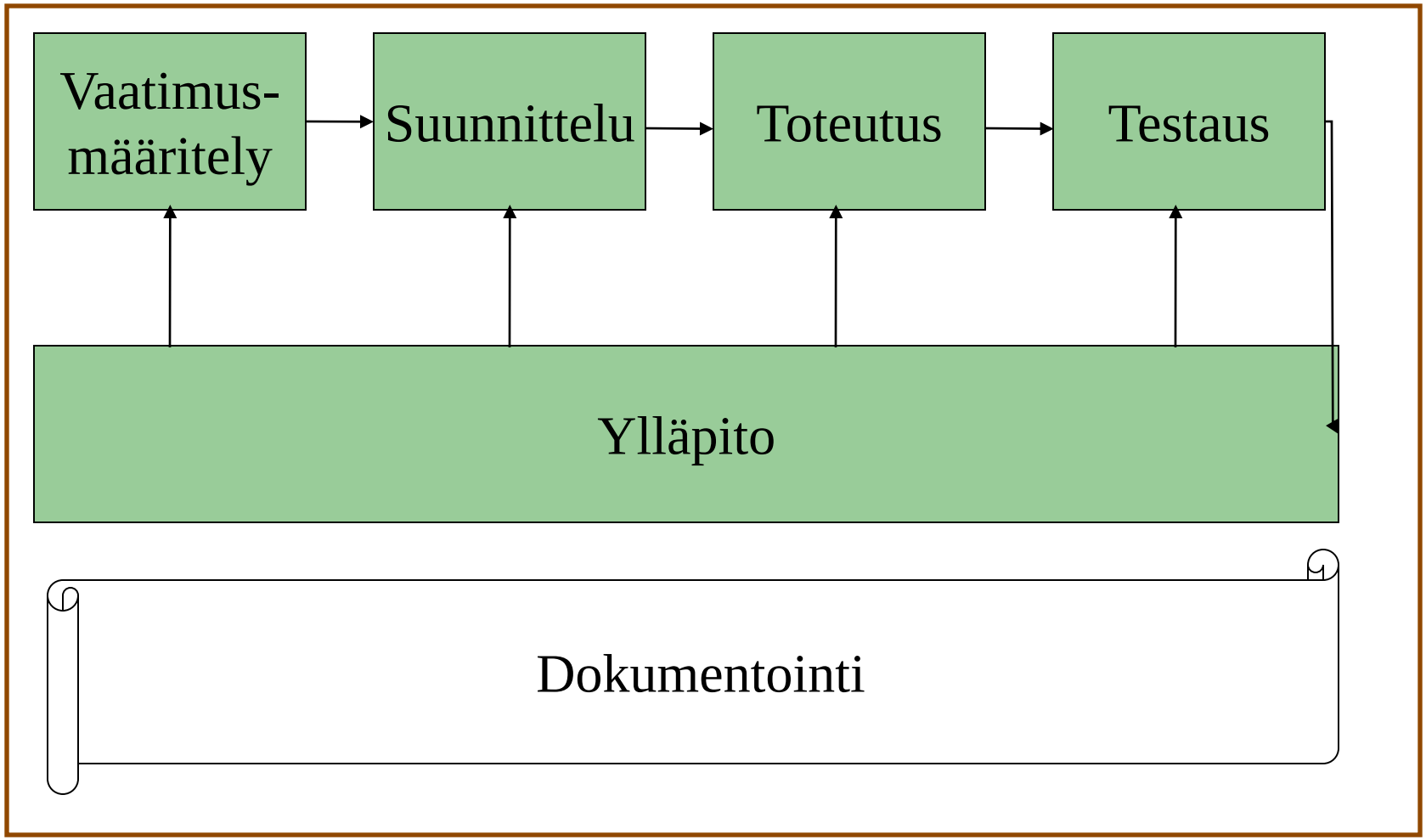
- *Lineaariset mallit:* vaihe kerrallaan
- *Komponenttimallit*: valmiiden osien käyttö
- *Prototyyppimallit (eli evoluutiomallit)*: prototyypit
- *Iteratiiviset ja inkrementaaliset mallit:* vaiheiden toistaminen, lisätään toiminnallisuutta vähän kerrassaan

- Nämä ovat pääluokkia: yksittäisissä prosessimalleissa näitä ominaisuuksia voidaan myös jossakin määrin yhdistellä

Vesiputousmalli

- Vesiputousmallissa on lineaarinen lähestymistapa:
  - Jo päätettyyn vaiheeseen ei enää palata myöhemmin
  - Jokainen vaihe saatetaan loppuun, sen tulokset hyväksytään ja jäädytetään ennen seuraavaan vaiheeseen siirtymistä
- Vesiputousmallin vaiheet ovat suoraviivaisesti prosessimallin perustehtävät

Vesiputousmallin tehtäväjako



Vesiputousmallin vaiheet

1. Vaatimusmäärittely
- mitkä ovat ohjelmiston
  - palvelut
  - rajoitukset
  - tavoitteet ?
- laaditaan ohjelmistoa kuvaavat mallit, joiden avulla voidaan tarkistaa vaatimusten järkevyys

2. Suunnittelu
- tarkennetaan vaatimus-määrittelyssä laadittuja malleja:
  - osajärjestelmät ja
  - niiden välinen yhteistyö
- tarkennetaan jokainen osajärjestelmä halutulle abstraktiotasolle asti
  - riittävä tarkkuus toteutuksen pohjaksi

Vesiputousmallin vaiheet 2

3. Toteutus (+yksikkötestaus)
   - suunnitelma toteutetaan osa kerrallaan
   - eri osat voidaan tehdä eri ohjelmointikielillä
   - kukin osa (yksikkö) testataan toteutuksen yhteydessä
   - yksikkötestauksen menetelmät

Malli perustuu vahvasti hyvään dokumentaatioon.

4. Integrointi ja testaus
   - toimivista osista kootaan osajärjestelmiä ja näistä edelleen koko järjestelmä
   - integrointitestaus: toimivatko rajapinnat?
   - koko järjestelmän verifiointi (selvitetään, että se toimii) ja validointi (selvitetään, että se tekee mitä halutaan)

5. Ylläpito
   - käyttöönoton jälkeen
   - korjaukset, laajennukset

Vesiputousmallin synnystä

- Royce 1970: *Managing the Development of Large Software Systems*
- Hieman tragikoomista on, että artikkelissa esiteltiin vesiputousmallin periaatteet, mutta todettiin ettei ohjelmistotuotantoa missään nimessä pidä tehdä sen mallin mukaisesti ☺
- Royce kannatti iteratiivista kehitystyötä
- Vesiputousmalli saavutti kuitenkin nopeasti suuren suosion, se vastasi oman aikansa ohjelmistoammattilaisten käsitystä oikeanlaisesta prosessista

Royce dokumentaatiosta

”At this point it is appropriate to raise the issue of – ”how much documentation?” My own view is ”quite a lot”; certainly more than most programmers, analysts, or program signers are willing to do if left to their own devices. The first rule of managing software development is ruthless enforcement of documentation requirements”

- ***Royce (1970): Managing the Development of Large Software Systems***

Vesiputousmalli nykyisin

- Edelleenkin suuri osa ohjelmistoyrityksistä käyttää vesiputousmallia tai jotakin sen varianttia (esim. *V-mallia*) kehitystyössään
- Valtava osa viimeisen 40 vuoden ohjelmistotekniikan tutkimustyöstä perustuu vesiputousmalliin

  → mallin teoria, vahvuudet ja heikkoudet tunnetaan hyvin
- Malli tarjoaa parhaan teoria- ja työkalutuen

Plussaa

- Selkeä ja helposti omaksuttava malli
- Tiukasti kontrolloitu prosessi sopii erityisesti hyvin laajoihin ja kriittisiin järjestelmiin
- Sopii hierarkkisiin organisaatioihin
- Ei vaadi välttämättä jatkuvaa ryhmätyötä (riippuu ehkä henkilöstä, onko tämä plussaa vai miinusta)
- Kiireisen asiakkaan ei tarvitse osallistua projektityöhön

Miinusta

- Vaatimukset *eivät saa* muuttua
- Valmista vasta projektin päätyttyä
- Myöhässä olevaa projektia vaikea saada aikatauluun
- Työntekijät saattavat turhautua dokumentointiin

Komponenttimalli

- Tavoitteena on käyttää mahdollisimman paljon jo valmiita komponentteja
- Valmiita komponentteja saadaan
  - omista aiemmista projekteista
  - ostamalla
  - ilmaisista Free Software –projekteista (vaarallisia, koska ylläpidosta ei ole takeita)
- Uudelleenkäytettäviä komponentteja pidetään komponenttikirjastossa

Komponenttimallin työvaiheet

- Komponenttimallin työvaiheet ovat:
  - Vaatimusmäärittely
    - Etsitään ja analysoidaan vaatimuksia
  - Komponenttianalyysi
    - Etsitään komponenttikirjastosta vaatimusanalyysin mukaiseen tuotteeseen sopivia komponentteja
  - Vaatimusten muokkaus
    - Täydennetään vaatimuksia ottaen huomioon löydetyt komponenttiehdokkaat

Komponenttimallin työvaiheet II

- Työvaiheet jatkuvat:
  - Suunnittelu ja komponenttien eristys
    - Eristetään komponenttikirjastoon omia kelvollisia komponenttiehdokkaiden suunnitelmia
  - Toteutus ja komponenttien eristys
    - Eristetään komponenttikirjastoon omia kelvollisia komponenttiehdokkaiden toteutuksia
  - Järjestelmän validointi
    - Varmistetaan, että tehtiin oikea järjestelmä

Iteratiiviset ja inkrementaaliset mallit

- Vesiputousmallissa ohjelma määritellään ja suunnitellaan yhtenä kokonaisuutena. Tämä ei usein toimi:
  - vaatimuksia ei tunneta kunnolla,
  - vaatimukset ovat epäselviä tai
  - vaatimukset muuttuvat projektin aikana
- Inkrementaalisissa malleissa ohjelma määritellään ja toteutetaan yleensä *iteratiivisesti sykleissä*: ytimestä erikoisosiin

Syklit

- Ohjelmisto jaetaan osiin, jotka toteutetaan omina sykleinään
  - Ensin toteutetaan ydin, johon seuraavien syklien tulokset voidaan liittää
- Vain yhden syklin vaatimukset käsitellään kerrallaan
- Toteuttamattomien syklien vaatimukset voivat muuttua

Etuja ja haittoja

- Asiakas saa alusta lähtien jossakin määrin käyttökelpoisen tuotteen
  - Tärkeimmät piirteet toteutetaan ensin
- Virheelliset määritykset huomataan ajoissa
- Ohjelman rakenteen pitäminen loogisena ja selkeänä voi olla ongelmallista

Prototyyppimallit

- *Throwaway prototyping*
  - Kehitetään hyvin nopeasti prototyyppi ohjelmistosta
  - Pyrkimyksenä päästä varhaisessa vaiheessa selvyyteen asiakkaan vaatimuksista ikään kuin iteroimalla vaatimusmäärittelyä asteittain tarkemmaksi
  - Asiakas näkee nopeasti kuinka hänen vaatimuksensa realisoituvat todelliseksi ohjelmistoksi ja pystyy antamaan palautetta
  - Konkreettisen prototyypin avulla pyritään siis ymmärtämään ja tarkentamaan asiakkaan toiveita ja vaatimuksia

Prototyyppimallit II

- *Evolutionary prototyping*
  - Prototyypin tarkoitus olla osittain valmis ohjelma, joka jalostetaan askeleittain tuotantoversioksi
  - Aloitetaan parhaiten ymmärretyistä osista
  - Asiakas saa alusta lähtien käyttökelpoisen ohjelmiston
  - Asiakkaan kanssa neuvotellen lisätään prototyyppiin toiminnallisuutta

Prototyypin tehtävä

- Tarkoituksena voi olla tarkentaa asiakkaan toiveita ja vaatimuksia (*throwaway prototyping*)

  → prototyypin hylkääminen
- Tarkoituksena voi olla esitellä asiakkaalle ydinohjelmisto tai jokin ydinohjelmistoon liitetty toiminto (*evolutionary prototyping)*

  → prototyypin kehittäminen eteenpäin

- Asiakkaan voi olla vaikeaa ymmärtää, miksi prototyyppi ei kelpaa lopputuotteeksi.
  - Prototyyppi ei täytä valmiille ohjelmistolle asetettavia laatuvaatimuksia

Prototyyppimallien soveltuvuus

- Prototyyppimallit sopivat hyvin pienten ja keskisuurten järjestelmien (Sommervillen mukaan max 500 000 koodiriviä)
- Erityisesti suurten järjestelmien kohdalla
  - Projektihallinnon voi olla vaikeaa hahmottaa projektin tilaa ja etenemistä
  - Jatkuvat muutokset voivat rikkoa suunnitellun arkkitehtuurin
- Prototyyppien tekeminen vaatii sopivat välineet (sovelluskehitin) + niiden käyttötaidon

Ketterät prosessimallit

- 1980- ja 1990-luvun prosessimalleissa korostettiin
  - huolellista projektisuunnittelua,
  - formaalia laadunvalvontaa,
  - yksityiskohtaisia analyysi- ja suunnittelumenetelmiä,
  - CASE-työkalujen käyttöä ja
  - täsmällistä tarkasti ohjattua ohjelmistoprosessia.

Ketterien prosessimallien synty

- Prosessimallit tukivat erityisesti laajojen, pitkäikäisten ohjelmistojen kehitystyötä, mutta pienten ja keskisuurten ohjelmistojen tekoon ne osoittautuivat usein turhan jäykiksi.
- Ristiriidan seurauksena syntyi joukko *ketteriä prosessimalleja* (agile process models), jotka korostivat itse ohjelmistoa yksityiskohtaisen suunnittelun ja dokumentaation sijaan.
  - Näistä tunnetuimmat ovat *eXtreme Programming* (XP) ja *Scrum*

Ketterien prosessien luonne

- Kaikki ketterät prosessimallit ovat inkrementaalisia ja iteratiivisia
- Yksi iteraatiokierros on muutaman viikon mittainen
- Iteraation aikana käydään läpi vaiheet vaatimuksista luovutukseen
- Lyhyen syklin johdosta asiakas on jatkuvasti mukana kehitystyössä: asiakkaan palautetta tarvitaan jatkuvasti

Ketterien prosessien periaatteet

- Asiakkaan osallistuminen
  - Asiakas osallistuu aktiivisesti kehitystyöhön. Asiakas esittää ja priorisoi uusia järjestelmän vaatimuksia ja evaluoi syklin aikana saadut tulokset.
- Lisääntyvät ominaisuudet
  - Kullakin syklillä ohjelmistoon lisätään uusia ominaisuuksia, joiden vaatimukset saadaan asiakkaalta.

Ketterien prosessien periaatteet II

- Ihmiset prosessia tärkeämmät
  - Projektin jäsenten taidot ja toimintatavat ovat tärkeämmät kuin vahvasti ohjattu prosessi
- Muutoksen hyväksyminen
  - Vaatimukset muuttuvat, joten järjestelmä pitää suunnitella tukemaan muutosta
- Yksinkertaisuuden ylläpito
  - Sekä ohjelmisto että prosessi tulee pitää mahdollisimman yksinkertaisena (mutta ei yhtään yksinkertaisempana)

Extreme programming (XP)

- Pyritään viemään hyviä käytäntöjä ”äärimmäisyyksiin”
- Ohjelma kehitetään hyvin pieninä sykleinä
  - Sykli kerrallaan lisätään mahdollisimman pieni joukko järkeviä ominaisuuksia
  - Toteutusjärjestys määräytyy asiakkaiden tarpeista
  - Iteratiivinen kehitystapa viety äärimmilleen: uusia versioita päivittäin

Ihmisläheisyys

- Kommunikointi kasvotusten, ei dokumenttien välityksellä
- Asiakkaan jatkuva läsnäolo
- Vaatimukset selville keskustelemalla asiakkaan kanssa
- Järkevä etenemistahti, ei ylitöitä: kaoottinen kiire heikentää koodin laatua ja pitkän aikavälin tuottavuutta

Testivetoisuus ja integrointi

- *”Test-first”:* yksikkötestit uutta toiminnallista komponenttia varten toteutetaan jo ennen itse komponentin toteutusta
- Erittäin testivetoinen kehitys on XP:n merkittävimpiä innovaatioita
- Yksikkötestit kaikelle kirjoitetulle koodille, automatisoitu testaus
- Jatkuva integrointi: toteutetut toiminallisuudet integroidaan välittömästi koko järjestelmään
- Tällöin aina intensiivinen, automatisoitu testaus: eteenpäin ei jatketa ennen kuin kaikki testit läpäisty

Suunnittelun yksinkertaisuus

- Suunnittelua tehdään vain juuri sen verran kun on tarpeellista, ei enempää eikä vähempää
- Perinteisessä ohjelmistokehityksessä pyritään ennakoimaan tulevaisuudessa mahdollisesti tapahtuvia muutoksia
- XP:ssä pyritään välttämään tätä: ennakointi on usein turhaa ja hukattua energiaa
- Ennakoimisen välttäminen johtaa koodin laadun huononemiseen: muutosten toteuttaminen tulee koko ajan vaikeammaksi

→ XP:ssä tähän ongelmaan vastataan *refaktoroinnilla*

Refaktorointi

- Koodin sisäisen rakenteen parantaminen
- Koodin toiminnallisuuden säilyttäminen (ei uusia ominaisuuksia)
- Kaikki kehittäjät refaktoroivat koodia jatkuvasti aina kun parannusmahdollisuuksia löydetään
- *Pariohjelmoinnilla* ja *koodin yhteisomistajuudella* pyritään tekemään refaktoroinnista motivoivaa

Kehittäjäparit ja yhteisomistajuus

- Pariohjelmointi
  - Työskennellään pareittain, jolloin kaksi silmäparia aina valvomassa työn laatua
- Koodin yhteisomistajuus
  - Jokainen kehittäjäpari työskentelee kaikilla järjestelmän osa-alueilla, kaikki voivat muuttaa kaikkea

→ei pääse kehittymään asiantuntijuuden ”saarekkeita”

  - Yhteinen vastuu, *”egoless programming*”

Scrum

- Nimi rugbyn alkuasetelmasta: tiiviistä ”ihmiskeosta”
- Roolit: *tiimi*, *scrum-mestari*, *omistaja*
- Scrum-mestari
  - Eräänlainen projektipäällikkö, jolla ei kuitenkaan suoranaista käskyvaltaa tiimin jäseniin
  - Hoitaa tiimin ”rajapintaa” ulkomaailmaan
  - Esteiden poistaminen, työrauhan takaaminen tiimille
- Omistaja (product owner)
  - Edustaa asiakkaita
  - (periaatteessa) jatkuvasti mukana tiimin työssä

Pyrähdykset (sprint)

- Tiimistä riippuen muutamia viikkoja, usein n. 30 päivää
- Koko projekti koostuu pyrähdysten sarjasta: lyhyillä pyrähdyksillä pyritään joustavuuteen

- *Projektin työlistasta (product backlog)* valitaan pyrähdyksessä toteutettavat vaatimukset *pyrähdyksen työlistaan (sprint backlog)*

Pyrähdyksen työlista

- Ensisijaisesti omistaja määrittää mitä vaatimuksia valitaan pyrähdyksen työlistaan
- Pyrähdyksen työlistaan ei voi tehdä muutoksia pyrähdyksen aikana
- Pyrähdyksen päättyessä tuotteen tulisi sisältää uutta toiminnallisuutta (lopussa demotaan)

Scrum-kokoukset

- Scrum-kokoukset (Daily Scrum)
  - Päivittäinen enintään 15 minuuttia kestävä kokous
  - Kukin kertoo mitä tehnyt, mitä aikoo tehdä, mitä vaikeuksia
  - Pyritään varmistumaan, että kaikki edistyvät, kaikilla mielekästä tekemistä ja poistamaan esteet tekemisen tieltä
  - Esteiden poistaminen Scrum-mestarin tehtävä