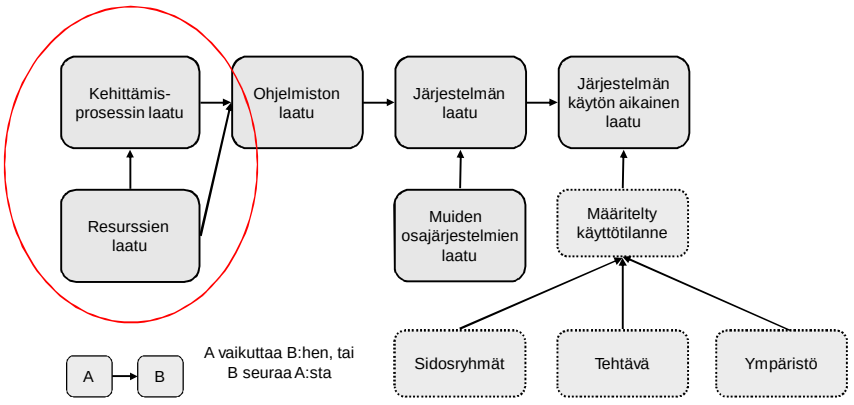
# PROSESSIT JA LAATU



## 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 Þ 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



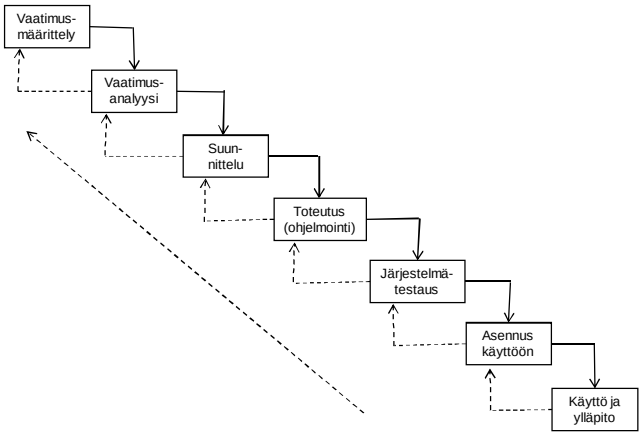
# ITERATIIVINEN OHJELMISTO-KEHITYS

## 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 kehitysvaihe 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 (järjestellä) 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.

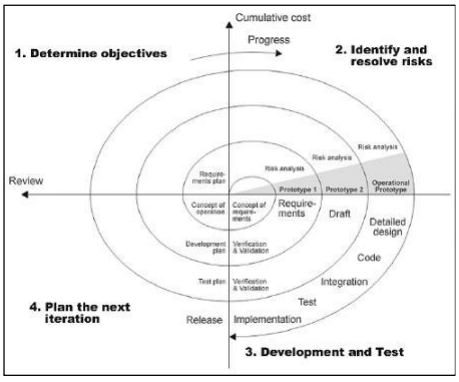
# Iteratiivinen ja lisäävä kehitystyö

- 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)




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

Kehitys tapahtuu sykleissä, joissa

1. selvitetään projektin menestymisen kannalta kriittisten sidosryhmien asettamat tavoitteet
2. Mietitään vaihtoehtoisia tapoja saavuttaa tavoitteet ja hallita niihin liittyvät riskit
3. Kehitetään ja testataan valittu riskit minimoiva ratkaisu (v & v)
4. Suunnitellaan seuraava kierros ja haetaan sille sidosryhmien hyväksyntä ja aloituslupa

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

## 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.