

# Ohjelmistotuotanto

Matti Luukkainen ja ohjaajat Kalle Ilves, Silva Perander, Topias Pyykönen, Jussi Laisi, Petrus Peltola, Kristian Krok

syksy 2020

Luento 1

26.10.2020

?

Johdanto *ohjelmistotuotantoon*, eli systemaattiseen tapaan tehdä hieman laajempia ohjelmistoja useamman hengen tiimissä ulkoiselle asiakkaalle

Johdanto *ohjelmistotuotantoon*, eli systemaattiseen tapaan tehdä hieman laajempia ohjelmistoja useamman hengen tiimissä ulkoiselle asiakkaalle

Erityinen paino ns ketterissä ohjelmistotuotantomenetelmissä

# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa*

# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa* suoritettuaan kurssin opiskelija

- ▶ tuntee ohjelmistoprosessin vaiheet (vaatimusmäärittely, suunnittelu, toteutus ja laadunhallinta)

# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa*

suoritettuaan kurssin opiskelija

- ▶ tuntee ohjelmistoprosessin vaiheet (vaatimusmäärittely, suunnittelu, toteutus ja laadunhallinta)
- ▶ tietää miten vaatimuksia hallitaan ketterässä ohjelmistotuotantoprosessissa



# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa*

suoritettuaan kurssin opiskelija

- ▶ tuntee ohjelmistoprosessin vaiheet (vaatimusmäärittely, suunnittelu, toteutus ja laadunhallinta)
- ▶ tietää miten vaatimuksia hallitaan ketterässä ohjelmistotuotantoprosessissa
- ▶ ymmärtää suunnittelun, toteutuksen ja testauksen vastuut ja luonteen ketterässä ohjelmistotuotannossa

# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa*

suoritettuaan kurssin opiskelija

- ▶ tuntee ohjelmistoprosessin vaiheet (vaatimusmäärittely, suunnittelu, toteutus ja laadunhallinta)
- ▶ tietää miten vaatimuksia hallitaan ketterässä ohjelmistotuotantoprosessissa
- ▶ ymmärtää suunnittelun, toteutuksen ja testauksen vastuut ja luonteen ketterässä ohjelmistotuotannossa
- ▶ ymmärtää laadunhallinnan perusteet

# Kurssin oppimistavoitteet

Tiedolliset ja tekniset valmiudet toimia *juniorikehittäjän roolissa* pienessä ohjelmistotiimissä, esim. *ohjelmistotuotantoprojektissa*

suoritettuaan kurssin opiskelija

- ▶ tuntee ohjelmistoprosessin vaiheet (vaatimusmäärittely, suunnittelu, toteutus ja laadunhallinta)
- ▶ tietää miten vaatimuksia hallitaan ketterässä ohjelmistotuotantoprosessissa
- ▶ ymmärtää suunnittelun, toteutuksen ja testauksen vastuut ja luonteen ketterässä ohjelmistotuotannossa
- ▶ ymmärtää laadunhallinnan perusteet
- ▶ osaa toimia ympäristössä, jossa ohjelmistokehitys tapahtuu hallitusti ja toistettavalla tavalla

# Kurssin rakenne

- ▶ Luennot ma ja ti 12-14
  - ▶ Ohjelmistotuotantoon liittyvää käsitteistöä ja teoriaa
  - ▶ Neljä nauhoitettua vierailuluentoa: Marko Klemetti Eficode, Niko Laitinen Nitor, Juha Viljanen Smartly ja Hannu Kokko Elisa

# Kurssin rakenne

- ▶ Luennot ma ja ti 12-14
  - ▶ Ohjelmistotuotantoon liittyvää käsitteistöä ja teoriaa
  - ▶ Neljä nauhoitettua vierailuluentoa: Marko Klemetti Eficode, Niko Laitinen Nitor, Juha Viljanen Smartly ja Hannu Kokko Elisa
- ▶ Laskarit
  - ▶ teoriaa kertaavat *monivalintatehtävät*, deadline su 23.59.
  - ▶ versionhallintaa, testaamista ja ohjelmistojen konfigurointia käsittelevät, deadline *maanantaina klo 23:59*.

# Kurssin rakenne

- ▶ Luennot ma ja ti 12-14
  - ▶ Ohjelmistotuotantoon liittyvää käsitteistöä ja teoriaa
  - ▶ Neljä nauhoitettua vierailuluentoa: Marko Klemetti Eficode, Niko Laitinen Nitor, Juha Viljanen Smartly ja Hannu Kokko Elisa
- ▶ Laskarit
  - ▶ teoriaa kertaavat *monivalintatehtävät*, deadline su 23.59.
  - ▶ versionhallintaa, testaamista ja ohjelmistojen konfigurointia käsittelevät, deadline *maanantaina klo 23:59*.
  - ▶ oletettu kuormittavuus on noin 8 tuntia ensimmäisen kolmen viikon aikana ja 4 tuntia sen jälkeen
  - ▶ monivalintatehtäviin vastaaminen nopeaa, edellyttää osallistumista luennoille ja/tai viikon materiaalin lukemista

# Kurssin rakenne

- ▶ Luennot ma ja ti 12-14
  - ▶ Ohjelmistotuotantoon liittyvää käsitteistöä ja teoriaa
  - ▶ Neljä nauhoitettua vierailuluentoa: Marko Klemetti Eficode, Niko Laitinen Nitor, Juha Viljanen Smartly ja Hannu Kokko Elisa
- ▶ Laskarit
  - ▶ teoriaa kertaavat *monivalintatehtävät*, deadline su 23.59.
  - ▶ versionhallintaa, testaamista ja ohjelmistojen konfigurointia käsittelevät, deadline *maanantaina klo 23:59*.
  - ▶ oletettu kuormittavuus on noin 8 tuntia ensimmäisen kolmen viikon aikana ja 4 tuntia sen jälkeen
  - ▶ monivalintatehtäviin vastaaminen nopeaa, edellyttää osallistumista luennoille ja/tai viikon materiaalin lukemista
- ▶ Miniprojekti
  - ▶ alkaa kurssin 4. viikolla
  - ▶ yhdistää teorian ja käytännön

# Miniprojekti

- ▶ kurssin viikoilla 4-7
- ▶ ryhmätyö: harjotellaan projektinhallintaa sekä eräitä laadunhallintatekniikoita



# Miniprojekti

- ▶ kurssin viikoilla 4-7
- ▶ ryhmätyö: harjotellaan projektinhallintaa sekä eräitä laadunhallintatekniikoita
- ▶ ryhmässä 4-6 opiskelijaa, ryhmällä on myös asiakas, jota tavataan viikoittain
- ▶ ensimmäisellä viikolla asiakastapaamiseen tulee varata 90 minuuttia, jälkimmäisillä 30 minuuttia
- ▶ kurssin lopussa on miniprojektien yhteinen 2h kestoinen demotilaisuus

# Miniprojekti

- ▶ kurssin viikoilla 4-7
- ▶ ryhmätyö: harjotellaan projektinhallintaa sekä eräitä laadunhallintatekniikoita
- ▶ ryhmässä 4-6 opiskelijaa, ryhmällä on myös asiakas, jota tavataan viikoittain
- ▶ ensimmäisellä viikolla asiakastapaamiseen tulee varata 90 minuuttia, jälkimmäisillä 30 minuuttia
- ▶ kurssin lopussa on miniprojektien yhteinen 2h kestoinen demotilaisuus
- ▶ miniprojekteissa työskentelyyn tulee varata aikaa noin 6 tuntia viikossa

*Kurssin läpäisyn edellytyksenä on hyväksytysti suoritettu tai hyväksiluettu miniprojekti*

# Miniprojektin hyväksilukeminen

Vähintään 4 kuukauden työkokemus tiimityönä tehtävästä ohjelmistokehityksestä

Lähetä emailia välittömästi

Jaossa yhteensä 40 pistettä

- ▶ laskarit 11 pistettä
  - ▶ monivalintatehtävät 3 pistettä
  - ▶ viikoittaiset ohjelmointi/versionhallinta/konfigurointitehtävät 8 pistettä
- ▶ miniprojekti 9 pistettä
- ▶ koe 20 pistettä

Arvosanaan 1 riittää 20 pistettä, arvosanaan 5 tarvitaan 36 pistettä.

Läpipääsy edellyttää lisäksi miniprojektin hyväksyttyä suoritusta (tai hyväksilukua) ja vähintään puolia kokeen pisteistä

**Luennoilla** ohjelmistokehityksen teoriaa ja käsitteistöä

- ▶ laskarien *monivalintatehtävät* liittyvät kunkin viikon luentoihin

**Luennoilla** ohjelmistokehityksen teoriaa ja käsitteistöä

► laskarien *monivalintatehtävät* liittyvät kunkin viikon luentoihin

Versionhallintaa, konfigurointia, testausta ja ohjelmointia  
käsittelevien **teknisempien laskarien** aihepiirejä ei paljoa käsitellä  
luennoilla

**Luennoilla** ohjelmistokehityksen teoriaa ja käsitteistöä

► laskarien *monivalintatehtävät* liittyvät kunkin viikon luentoihin

Versionhallintaa, konfigurointia, testausta ja ohjelmointia  
käsittelevien **teknisempien laskarien** aihepiirejä ei paljoa käsitellä  
luennoilla

**Miniprojekti** yhdistää luentojen teoria ja laskareissa käsitellyt  
teknisemmät asiat, ja soveltaa niitä käytännössä

**Luennoilla** ohjelmistokehityksen teoriaa ja käsitteistöä

- ▶ laskarien *monivalintatehtävät* liittyvät kunkin viikon luentoihin

Versionhallintaa, konfigurointia, testausta ja ohjelmointia  
käsittelevien **teknisempien laskarien** aihepiirejä ei paljoa käsitellä  
luennoilla

**Miniprojekti** yhdistää luentojen teoria ja laskareissa käsitellyt  
teknisemmät asiat, ja soveltaa niitä käytännössä

**Kokeessa** suurin painoarvo teoriassa ja sen soveltamisessa  
käytäntöön

- ▶ laskareiden teknisimpiä asioita ei kokeessa tulla kysymään
- ▶ tarkemmin kokeesta kurssin viimeisellä luennolla



Saat suoritusmerkinnän tekemällä kaikki kurssin  
versionhallintatehtävät ja suorittamalla hyväksytysti miniprojektin

The IEEE Computer Society defines software engineering as:

- ▶ *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*

The IEEE Computer Society defines software engineering as:

- ▶ *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*

Lähde *SWEBOK* eli *Guide to the Software Engineering Body of Knowledge*

- ▶ ison komitean yritys määritellä mitä ohjelmistotuotannolla tarkoitetaan ja mitä osa-alueita siihen kuuluu
- ▶ uusin versio vuodelta 2014

# Ohjelmistotuotannon osa-alueet

SWEBOK:in mukaan ohjelmistotuotanto jakautuu seuraaviin osa-alueisiin:

- ▶ Software requirements eli vaatimusmäärittely
- ▶ Software design eli suunnittelu
- ▶ Software construction eli toteutus/ohjelmointi
- ▶ Software testing
- ▶ Software maintenance eli ylläpito

# Ohjelmistotuotannon osa-alueet

SWEBOK:in mukaan ohjelmistotuotanto jakautuu seuraaviin osa-alueisiin:

- ▶ Software requirements eli vaatimusmäärittely
- ▶ Software design eli suunnittelu
- ▶ Software construction eli toteutus/ohjelmointi
- ▶ Software testing
- ▶ Software maintenance eli ylläpito
  
- ▶ Software configuration management
- ▶ Software engineering management
- ▶ Software engineering process eli ohjelmistotuotantoprosessi
- ▶ Software engineering tools and methods
- ▶ Software quality

Näiden osa-alueiden läpikäynti on myöskin tämän kurssin tavoite

# Ohjelmiston elinkaari (software lifecycle)

Riippumatta tyylistä ja tavasta, jolla ohjelmisto tehdään, käy ohjelmisto läpi seuraavat *vaiheet*

- ▶ Vaatimusten analysointi ja määrittely
- ▶ Suunnittelu
- ▶ Toteutus
- ▶ Testaus
- ▶ Ohjelmiston ylläpito ja evoluutio

# Ohjelmiston elinkaari (software lifecycle)

Riippumatta tyylistä ja tavasta, jolla ohjelmisto tehdään, käy ohjelmisto läpi seuraavat *vaiheet*

- ▶ Vaatimusten analysointi ja määrittely
- ▶ Suunnittelu
- ▶ Toteutus
- ▶ Testaus
- ▶ Ohjelmiston ylläpito ja evoluutio

Vaiheista muodostuu ohjelmiston “elinkaari”

Eri vaiheiden sisältöön palaamme myöhemmin tarkemmin

# Alussa (ja osin edelleen) code'n'fix

Historian alkuaikoina laitteet maksoivat paljon, ohjelmat olivat laitteistoihin nähden “triviaaleja”

- ▶ ohjelmointi konekielellä
- ▶ sovelluksen käyttäjä ohjelmoi itse ohjelmansa



# Alussa (ja osin edelleen) code'n'fix

Historian alkuaikoina laitteet maksoivat paljon, ohjelmat olivat laitteistoihin nähden “triviaaleja”

- ▶ ohjelmointi konekielellä
- ▶ sovelluksen käyttäjä ohjelmoi itse ohjelmansa

Vähitellen ohjelmistot alkavat kasvaa ja kehitettiin korkeamman tason ohjelmointikieliä (Fortran, Cobol, Algol)

- ▶ sovellusalue laajenee monille elämänaloille

# Alussa (ja osin edelleen) code'n'fix

Historian alkuaikoina laitteet maksoivat paljon, ohjelmat olivat laitteistoihin nähden “triviaaleja”

- ▶ ohjelmointi konekielellä
- ▶ sovelluksen käyttäjä ohjelmoi itse ohjelmansa

Vähitellen ohjelmistot alkavat kasvaa ja kehitettiin korkeamman tason ohjelmointikieliä (Fortran, Cobol, Algol)

- ▶ sovellusalue laajenee monille elämänaloille

Pikkuhiljaa homma alkaa karata käsistä:

- ▶ budjetit ylittyivät ja projektit myöhästyivät aikatauluista
- ▶ ohjelmistot olivat tehottomia, niiden laatu oli huono ja ne eivät toimineet käyttäjien tarpeiden mukaan
- ▶ koodin ylläpito ja laajentaminen oli vaikeaa
- ▶ usein ohjelmistoja ei hyvistä aikeista huolimatta saatu ollenkaan toimitettua

Termi Software crisis lanseerataan kesällä 1968

- ▶ In essence, it refers to the difficulty of writing correct, understandable, and verifiable computer programs.

Termi Software crisis lanseerataan kesällä 1968

- ▶ In essence, it refers to the difficulty of writing correct, understandable, and verifiable computer programs.

Edsger Dijkstra:

- ▶ **as long as there were no machines, programming was no problem at all**

Termi Software crisis lanseerataan kesällä 1968

- ▶ In essence, it refers to the difficulty of writing correct, understandable, and verifiable computer programs.

Edsger Dijkstra:

- ▶ **as long as there were no machines, programming was no problem at all**
- ▶ when we had a few weak computers, programming became a mild problem

Termi Software crisis lanseerataan kesällä 1968

- ▶ In essence, it refers to the difficulty of writing correct, understandable, and verifiable computer programs.

Edsger Dijkstra:

- ▶ **as long as there were no machines, programming was no problem at all**
- ▶ when we had a few weak computers, programming became a mild problem
- ▶ **now we have gigantic computers, programming has become an equally gigantic problem.**

# Software development as Engineering

Termi **software engineering** määritellään ensimmäistä kertaa 1968:

- ▶ *The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*

# Software development as Engineering

Termi **software engineering** määritellään ensimmäistä kertaa 1968:

- ▶ *The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*

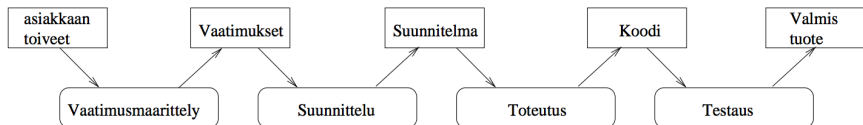
Ajatus siitä, että ohjelmistojen tekemisen tulisi olla kuin mikä tahansa muu insinöörityö

- ▶ ensin rakennettava artefakti *määritellään* (requirements)
- ▶ ja *suunnitell*an (design) aukottomasti
- ▶ tämän jälkeen *rakentaminen* (construction) on melko suoraviivainen vaihe

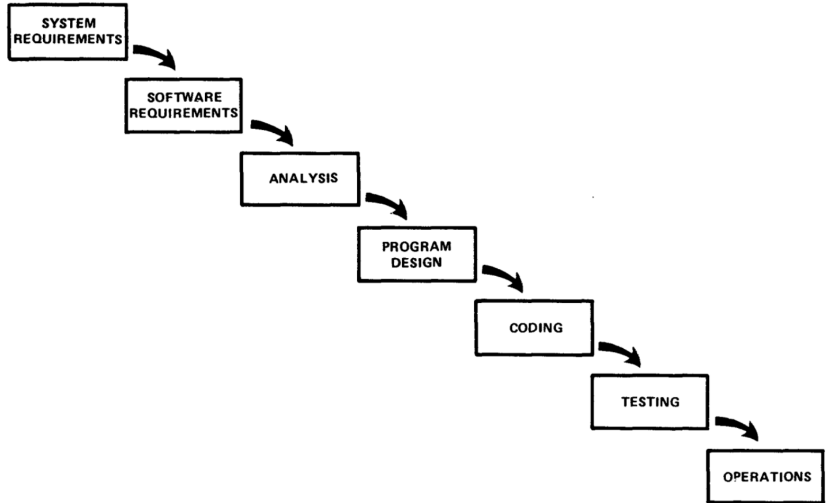


Winston W. Royce: Management of the development of Large Software, 1970

Sivulla 2 Royce esittelee yksinkertaisen prosessimallin, jossa elinkaaren vaiheet suoritetaan lineaarisesti peräkkäin:



# Vesiputousmalli Roycen artikkelista



# Vesiputousmallin suosion taustaa

Vesiputousmalli saavutti nopeasti suosiota

Yhdysvaltain puolustusministerö rupesi vaatimaan kaikilta alihankkijoiltaan vesiputousmallin noudattamista (Standardi DoD STD 2167)

# Vesiputousmallin suosion taustaa

Vesiputousmalli saavutti nopeasti suosiota

Yhdysvaltain puolustusministerö rupesi vaatimaan kaikilta alihankkijoiltaan vesiputousmallin noudattamista (Standardi DoD STD 2167)

Muutkin ohjelmistoja tuottaneet tahot ajattelivat, että koska DoD vaatii vesiputousmallia, tapa kannattaa omaksua itselleen

# Vesiputousmallin oletuksia

Vesiputousmalli perustuu vahvasti siihen, että *eri vaiheet ovat erillisten tuotantotiimien tekemiä*

- ▶ Tämän takia kunkin vaiheen tulokset *dokumentoidaan tarkoin*

# Vesiputousmallin oletuksia

Vesiputousmalli perustuu vahvasti siihen, että *eri vaiheet ovat erillisten tuotantotiimien tekemiä*

► Tämän takia kunkin vaiheen tulokset *dokumentoidaan tarkoin*

*Vaiheet tehdään peräkkäin:* esim. tekninen suunnittelu aloitetaan vasta sitten kun vaatimusmäärittely on valmis

# Vesiputousmallin oletuksia

Vesiputousmalli perustuu vahvasti siihen, että *eri vaiheet ovat erillisten tuotantotiimien tekemiä*

► Tämän takia kunkin vaiheen tulokset *dokumentoidaan tarkoin*

*Vaiheet tehdään peräkkäin:* esim. tekninen suunnittelu aloitetaan vasta sitten kun vaatimusmäärittely on valmis

Vesiputousmallin mukainen ohjelmistoprosessi on yleensä etukäteen *tarkkaan suunniteltu, resursoitu ja aikataulutettu*

# Vesiputousmallin oletuksia

Vesiputousmalli perustuu vahvasti siihen, että *eri vaiheet ovat erillisten tuotantotiimien tekemiä*

► Tämän takia kunkin vaiheen tulokset *dokumentoidaan tarkoin*

*Vaiheet tehdään peräkkäin:* esim. tekninen suunnittelu aloitetaan vasta sitten kun vaatimusmäärittely on valmis

Vesiputousmallin mukainen ohjelmistoprosessi on yleensä etukäteen *tarkkaan suunniteltu, resursoitu ja aikataulutettu*

**Vesiputousmallin mukainen ohjelmistotuotanto ei ole osoittautunut erityisen onnistuneeksi**



# Vesiputousmallin ongelmia

Asiakkaan *vaatimukset muuttuvat* usein matkan varrella:

- ▶ Asiakas ei tiedä tai osaa sanoa mitä haluaa/tarvitsee
- ▶ Asiakkaan tarve muuttuu projektin kuluessa
- ▶ Asiakas alkaa haluta muutoksia kun näkee lopputuotteen

# Vesiputousmallin ongelmia

Asiakkaan *vaatimukset muuttuvat* usein matkan varrella:

- ▶ Asiakas ei tiedä tai osaa sanoa mitä haluaa/tarvitsee
- ▶ Asiakkaan tarve muuttuu projektin kuluessa
- ▶ Asiakas alkaa haluta muutoksia kun näkee lopputuotteen

Vaatimusmäärittelyn, suunnittelun ja toteutuksen erottaminen ei järkevää

- ▶ Toteutusteknologiat vaikuttavat suuresti määriteltyjen ominaisuuksien hintaan
- ▶ Ohjelmaa on mahdotonta suunnitella siten, että toteutus on suoraviivaista
- ▶ Osa suunnittelusta tapahtuu pakosti vasta ohjelmoitaessa

# Vesiputousmallin ongelmia

Asiakkaan *vaatimukset muuttuvat* usein matkan varrella:

- ▶ Asiakas ei tiedä tai osaa sanoa mitä haluaa/tarvitsee
- ▶ Asiakkaan tarve muuttuu projektin kuluessa
- ▶ Asiakas alkaa haluta muutoksia kun näkee lopputuotteen

Vaatimusmäärittelyn, suunnittelun ja toteutuksen erottaminen ei järkevää

- ▶ Toteutusteknologiat vaikuttavat suuresti määriteltyjen ominaisuuksien hintaan
- ▶ Ohjelmaa on mahdotonta suunnitella siten, että toteutus on suoraviivaista
- ▶ Osa suunnittelusta tapahtuu pakosti vasta ohjelmoitaessa

*Lopuksi tapahtuva laadunhallinta* paljastaa ongelmat liian myöhään

- ▶ Korjaukset mahdollisesti kalliita: testaus voi paljastaa ongelmia jotka pakottavat muuttamaan ohjelmiston vaatimuksia

# Vesiputous oli väärinymmärrys

Paradoksaalista kyllä vesiputousmallin isänä pidetty Royce **ei suosittele** artikkelissaan suoraviivaisen lineaarisen mallin käyttöä

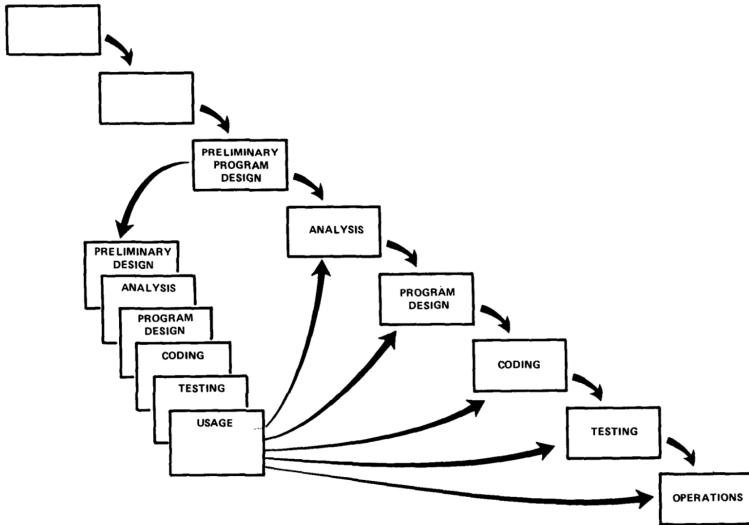
# Vesiputous oli väärinymmärrys

Paradoksaalista kyllä vesiputousmallin isänä pidetty Royce **ei suosittale** artikkelissaan suoraviivaisen lineaarisen mallin käyttöä

Royce esittelee lineaarisen vesiputousmallin **sivulla 2**, mutta toteaa että se **ei sovellu** monimutkaisiin ohjelmistoprojekteihin

Roycen mukaan sovelluksesta tulee ensin tehdä prototyyppi ja vasta siitä saatujen kokemusten valossa kannattaa suunnitella ja toteuttaa lopullinen ohjelmisto

# Roycen kahden iteraation malli



# Iteratiivinen ohjelmistokehitys

Vesiputousmallin ongelmiin reagoinut *iteratiivinen* tapa tehdä ohjelmistoja alkoi yleistyä 90-luvulla (mm. spiraalimalli, prototyyppimalli, Rational Unified Process)

# Iteratiivinen ohjelmistokehitys

Vesiputousmallin ongelmiin reagoinut *iteratiivinen* tapa tehdä ohjelmistoja alkoi yleistyä 90-luvulla (mm. spiraalimalli, prototyypimalli, Rational Unified Process)

- ▶ ohjelmistotuotanto jaetaan jaksoihin, eli *iteraatioihin*
- ▶ jokaisen iteraation aikana määritellään, suunnitellaan toteutetaan ja testataan ohjelmistoa
- ▶ eli ohjelmisto kehittyy vähitellen (inkrementaalisesti)

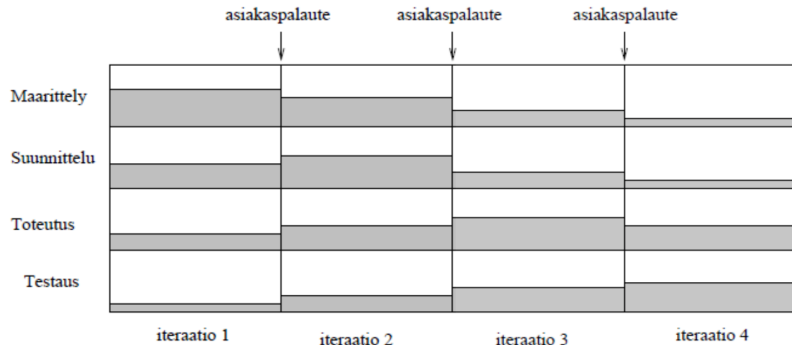


# Iteratiivinen ohjelmistokehitys

Vesiputousmallin ongelmiin reagoinut *iteratiivinen* tapa tehdä ohjelmistoja alkoi yleistyä 90-luvulla (mm. spiraalimalli, prototyypimalli, Rational Unified Process)

- ▶ ohjelmistotuotanto jaetaan jaksoihin, eli *iteraatioihin*
- ▶ jokaisen iteraation aikana määritellään, suunnitellaan toteutetaan ja testataan ohjelmistoa
- ▶ eli ohjelmisto kehittyy vähitellen (inkrementaalisesti)
- ▶ asiakasta tavataan jokaisen iteraation välissä
- ▶ asiakas näkee sen hetkisen version ohjelmasta ja pystyy vaikuttamaan seuraavien iteraatioiden kulkuun

# Iteratiivinen ohjelmistokehitys



# Iteratiivinen ohjelmistokehitys ei ole uusi keksintö

Yhdysvaltojen puolustusministeriön 2000 julkaisema standardi (MIL-STD-498) alkaa suositella iteratiivista ohjelmistoprosessia:

- ▶ There are two approaches, evolutionary (*iterative*) and single step (*waterfall*), to full capability. An *evolutionary approach is preferred...*

# Iteratiivinen ohjelmistokehitys ei ole uusi keksintö

Yhdysvaltojen puolustusministeriön 2000 julkaisema standardi (MIL-STD-498) alkaa suositella iteratiivista ohjelmistoprosessia:

- ▶ There are two approaches, evolutionary (*iterative*) and single step (*waterfall*), to full capability. An *evolutionary approach is preferred...*

Iteratiivinen ohjelmistokehitys on paljon vanhempi idea kun vesiputosmalli

- ▶ esim. NASA:n ensimmäisen amerikkalaisen avaruuteen vieneen Project Mercuryn ohjelmisto kehitettiin iteratiivisesti
- ▶ avaruussukkuloiden ohjelmisto tehtiin vesiputousmallin valtakaudella, mutta sekin kehitettiin iteratiivisesti (8 viikon iteraatioissa, 31 kuukaudessa)

# Iteratiivinen ohjelmistokehitys ei ole uusi keksintö

Yhdysvaltojen puolustusministeriön 2000 julkaisema standardi (MIL-STD-498) alkaa suositella iteratiivista ohjelmistoprosessia:

- ▶ There are two approaches, evolutionary (*iterative*) and single step (*waterfall*), to full capability. An *evolutionary approach is preferred...*

Iteratiivinen ohjelmistokehitys on paljon vanhempi idea kun vesiputosmalli

- ▶ esim. NASA:n ensimmäisen amerikkalaisen avaruuteen vieneen Project Mercuryn ohjelmisto kehitettiin iteratiivisesti
- ▶ avaruussukkuloiden ohjelmisto tehtiin vesiputousmallin valtakaudella, mutta sekin kehitettiin iteratiivisesti (8 viikon iteraatioissa, 31 kuukaudessa)

Roycen artikkelikin ehdotti *kahden iteraation* menetelmää ohjelmistojen tekemiseen

# Ketterien menetelmien synty

Perinteisissä prosessimalleissa korostettiin huolellista projektisuunnittelua, formaalia laadunvalvontaa, yksityiskohtaisia analyysi- ja suunnittelumenetelmiä ja täsmällistä, tarkasti ohjattua ohjelmistoprosessia

# Ketterien menetelmien synty

Perinteisissä prosessimalleissa korostettiin huolellista projektisuunnittelua, formaalia laadunvalvontaa, yksityiskohtaisia analyysi- ja suunnittelumenetelmiä ja täsmällistä, tarkasti ohjattua ohjelmistoprosessia

*Tukivat erityisesti laajojen, pitkäikäisten ohjelmistojen kehitystyötä*

- ▶ pienten ja keskisuurten ohjelmistojen tekoon turhan jäykkä

# Ketterien menetelmien synty

Perinteisissä prosessimalleissa korostettiin huolellista projektisuunnittelua, formaalia laadunvalvontaa, yksityiskohtaisia analyysi- ja suunnittelumenetelmiä ja täsmällistä, tarkasti ohjattua ohjelmistoprosessia

*Tukivat erityisesti laajojen, pitkäikäisten ohjelmistojen kehitystyötä*

- ▶ pienten ja keskisuurten ohjelmistojen tekoon turhan jäykkää

Pyrittiin työtä tekevän yksilön merkityksen minimoimiseen

- ▶ yksilö on “tehdastyöläinen”, joka voidaan helposti korvata toisella ja tällä ei ole ohjelmiston kehittämiseen vaikutusta



# Ketterien menetelmien synty

Perinteisissä prosessimalleissa korostettiin huolellista projektisuunnittelua, formaalia laadunvalvontaa, yksityiskohtaisia analyysi- ja suunnittelumenetelmiä ja täsmällistä, tarkasti ohjattua ohjelmistoprosessia

*Tukivat erityisesti laajojen, pitkäikäisten ohjelmistojen kehitystyötä*

- ▶ pienten ja keskisuurten ohjelmistojen tekoon turhan jäykkää

Pyrittiin työtä tekevän yksilön merkityksen minimoimiseen

- ▶ yksilö on “tehdastyöläinen”, joka voidaan helposti korvata toisella ja tällä ei ole ohjelmiston kehittämiseen vaikutusta

Ristiriita synnytti joukon **ketteriä prosessimalleja** (agile)

- ▶ korostivat itse ohjelmistoa sekä ohjelmiston asiakkaan ja toteuttajien merkitystä yksityiskohtaisen suunnittelun ja dokumentaation sijaan

# Ketterä manifesti 2001

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ▶ **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

# Ketterä manifesti 2001

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ▶ **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

Manifestin laativat ja allekirjoittivat 17 ketterien menetelmien varhaista pioneeria, mm. Kent Beck, Robert Martin, Ken Schwaber ja Martin Fowler

# Ketterät periaatteet, osa 1

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

# Ketterät periaatteet, osa 1

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale

# Ketterät periaatteet, osa 1

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale

**Working software** is the primary measure of progress

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale

**Working software** is the primary measure of progress

Business people and developers must **work together daily** throughout the project

# Ketterät periaatteet, osa 1

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale

**Working software** is the primary measure of progress

Business people and developers must **work together daily** throughout the project

The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation



# Ketterät periaatteet, osa 1

Our highest priority is to satisfy the customer through **early and continuous delivery of valuable software**

**Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale

**Working software** is the primary measure of progress

Business people and developers must **work together daily** throughout the project

The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation

**Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

The best architectures, requirements, and designs emerge from **self-organizing teams**

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

The best architectures, requirements, and designs emerge from **self-organizing teams**

At regular intervals, the **team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

The best architectures, requirements, and designs emerge from **self-organizing teams**

At regular intervals, the **team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly

**Simplicity** – the art of maximizing the amount of work not done – is essential

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

The best architectures, requirements, and designs emerge from **self-organizing teams**

At regular intervals, the **team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly

**Simplicity** – the art of maximizing the amount of work not done – is essential

Continuous attention to **technical excellence and good design** enhances agility

## Ketterät periaatteet, osa 2

Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**

The best architectures, requirements, and designs emerge from **self-organizing teams**

At regular intervals, the **team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly

**Simplicity** – the art of maximizing the amount of work not done – is essential

Continuous attention to **technical excellence and good design** enhances agility

Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

# Ketterät menetelmät

Sateenvarjotermi useille ketterille prosessimalleille

- ▶ Näistä tunnetuimpia ovat eXtreme programming eli XP ja Scrum
- ▶ Molempiin, erityisesti Scrumiin tutustutaan kurssin aikana



# Ketterät menetelmät

Sateenvarjotermi useille ketterille prosessimalleille

- ▶ Näistä tunnetuimpia ovat eXtreme programming eli XP ja Scrum
- ▶ Molempiin, erityisesti Scrumiin tutustutaan kurssin aikana

Ketterä ohjelmistotuotanto on ottanut vaikutteita myös Toyota production systemin taustalla olevasta *lean-ajattelusta*

Viime vuosina syntynyt joukko “leaneja menetelmiä”

- ▶ Kanban
- ▶ Scrumban

# Ketterät menetelmät

Sateenvarjotermi useille ketterille prosessimalleille

- ▶ Näistä tunnetuimpia ovat eXtreme programming eli XP ja Scrum
- ▶ Molempiin, erityisesti Scrumiin tutustutaan kurssin aikana

Ketterä ohjelmistotuotanto on ottanut vaikutteita myös Toyota production systemin taustalla olevasta *lean-ajattelusta*

Viime vuosina syntynyt joukko “leaneja menetelmiä”

- ▶ Kanban
- ▶ Scrumban

sekä laajan mittakaavan leaniin ja ketterään kehitykseen tarkoitettuja menetelmiä kuten SaFe

Myös nämä kuuluvat kurssin aihepiiriin

# Viikon 1 laskarit

Ohjelmistokehityksen käytännön työkaluja

- ▶ Versionhallinta: git
- ▶ Automatisoitu testaus: JUnit
- ▶ Projektin riippuvuuksienhallinta ja “buildaus”: gradle
- ▶ CI- ja build-palvelinohjelmisto: Github Action

Tavoitteena mahdollistaa hallittu- ja toistettavissa oleva ohjelmistokehitys

# Viikon 1 laskarit

Ohjelmistokehityksen käytännön työkaluja

- ▶ Versionhallinta: git
- ▶ Automatisoitu testaus: JUnit
- ▶ Projektin riippuvuuksienhallinta ja “buildaus”: gradle
- ▶ CI- ja build-palvelinohjelmisto: Github Action

Tavoitteena mahdollistaa hallittu- ja toistettavissa oleva ohjelmistokehitys

*Deadline maanantaina klo 23:59*

**Muista myös teoriaa kertaavat monivalintatehtävät, deadline sunnuntaina klo 23:59**