Ohjelmistotuotanto

Luento 10

4.12.

Laajan mittakaavan ketterä ohjelmistokehitys

Laajan mittakaavan ketterä

- Ketterät ohjelmistotuotantomenetelmät ovat alunperin tarkoitettu pienten tiimien hallintaan
  - Esim. Scrum mainitsee kehitystiimin koostuvan 3-10:stä henkilöstä
- Entä jos on kyseessä tuote, joka edellyttää suurempaa kehittäjäjoukkoa?
- Perusperiaatteena on edelleen pitää tiimit pieninä, mutta kasvattaa tuotantokapasiteettia käyttämällä useampia tiimejä
- Tämä taas edellyttää, että tiimien välistä työtä on koordinoitava jollain tavalla
- Jo kauan käytetty tapa Scrumin skaalaamiseen on ns **Scrum of Scrums**
- Ideana on muodostaa koordinoiva tiimi, johon kuuluu esim. yksi tai tarvittaessa useampikin jäsen jokaisesta Scrum-tiimistä
  - Perinteisin tapa lienee koostaa koordinointitiimi scrum mastereista
  - Joissain tilanteissa parempi henkilö koordinointiin voi kuitenkin olla ns lead developer, eli Scrum-tiimin kokeneempi sovelluskehittäjä
- Scrum of Scrums -tiimi voi tavata joka päivä tai jos se ei ole tarpeen niin esimerkiksi viikoittain

Scrum of Scrums

- Scrum of Scrums -periaate on jo hyvin vanha
- Artikkelissa *Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies* toinen Scrumin kehittäjistä Jeff Sutherland kertoo soveltaneensa Scrum of Scrumia jo vuonna 1996
  - Sutherland sovelsi periaatetta firmassa, missä oli satoja sovelluskehittäjiä ja kymmeniä Scrum-tiimejä, joiden vastuulla oli useita eri tuotteita
  - Jokaisen tuotteen tiimejä koordinoi oma, kerran viikossa kokoontuva Scrum of Scrums -tiimi
  - Koko firman tuoteportfolioa (eli kaikkien tuotteiden kokonaisuutta) hallinnoi ”management Scrum” eli Scrum of Scrum of Scrums -tiimi, joka kokoontui kuukausittain
  - Ylimmän tason management Scrum -tiimi koostui yrityksen johdosta, tuotepäälliköistä ja johtavista ohjelmistoarkkitehdeistä
  - https://www.researchgate.net/publication/290823579_Agile_Can_Scale_I
- Sutherlandin kuvaus ei ole kovin seikkaperäinen, ja se ei anna viitteitä miten esim. backlogien suhteen tulisi toimia laajemman skaalan Scrumissa

Laajan mittakaavan ketterä

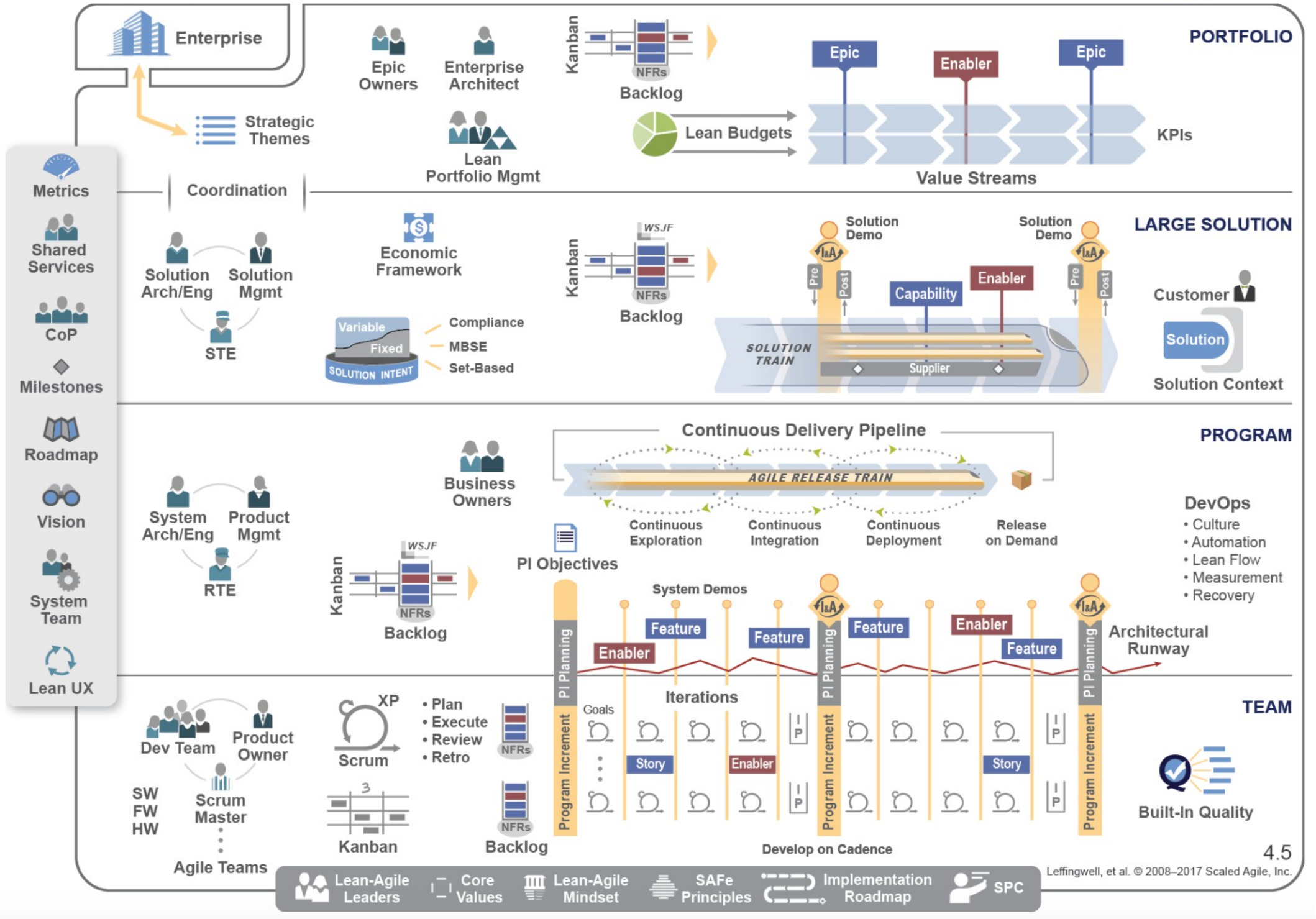
- Viimeisen kymmenen vuoden aikana ketterän skaalaamiseen on alettu kiinnittämään enemmän huomiota ja on esitelty useita laajan mittakaavan ketteriä menetelmiä
- Näistä eniten huomiota saaneet ovat
  - Scaled Agile Framework eli SAFe®
    - http://www.scaledagileframework.com
  - Large Scale Scrum eli LeSS
    - https://less.works
  - Disiplined Agile eli DA
    - http://www.disciplinedagiledelivery.com
- Yhteistä näille on se, että ne laajentavat ketteryyttä ottamalla mukaan lean-ajattelua, eli viime viikolla käsittelemäämme Toyota Productions Systemistä kehittynyttä tuotanto- ja tuotekehitysfilosofiaa
- Toisin kuin ketterät menetelmät, lean on lähtökohtaisesti tarkoitettu toimimaan todella suuressa skaalassa ja se sisältääkin enemmän koko organisaation toimintaa ohjaavia periaatteita kuin perinteinen ketterä
- Käsitellään nyt hieman tarkemmin SAFea ja LeSS:iä (DA on listatuista menetelmistä vähimmälle huomiolle jäänyt)

SAFe®

- SAFe®:n pääasiallinen kehittäjä on David Leffingwell joka toimi Nokia Mobile Phonesissa konstulttina 2000-luvulla
- SAFe® on syntynyt pitkälti Nokialla tehdyn työn pohjalta
  - NPM:lla olikin käytössä eräänlainen esiversio SAFe:sta
  - SAFe®:n virallinen ensimmäinen version julkaistiin 2011
- Kärjistetysti sanoen SAFe® yhdistää *kaikki* viimeisen 20 vuoden aikana kehitetyt ketterän ja leanin ohjelmistokehityksen parhaat käytänteet sekä joukon suurien yrityksien tuotteiden hallinnointiin suunnattuja käytänteitä
- SAFe® tarjoaa suuren määrän käytänteitä, henkilö- ja tiimirooleja sekä käsitteitä
- SAFe® sanoo olevansa framework eli menetelmäkehys, tarkoitus on että yritykset räätälöivät itselleen sopivanlaisen prosessin käyttäen SAFe®:n tarjoamia työkaluja
- SAFe® tarjaa myös 4 erikokoista valmiiksi räätälöityä konfiguraatiota
  - Näistä pienin *Essential SAFe*® on tarkoitettu pienemmille yrityksille ja SAFe®:n soveltamisen alkuvaiheeseen
  - Konfiguraatiosta suurin Full SAFe® taas soveltuu massiivisten, useita eri tuotteita hallitsevan jopa tuhansien sovelluskehittäjien yrityksen käyttöön
- Seuraavalla kalvolla oleva kaavio havainnollistaa Full SAFe®:m käsitteistöä

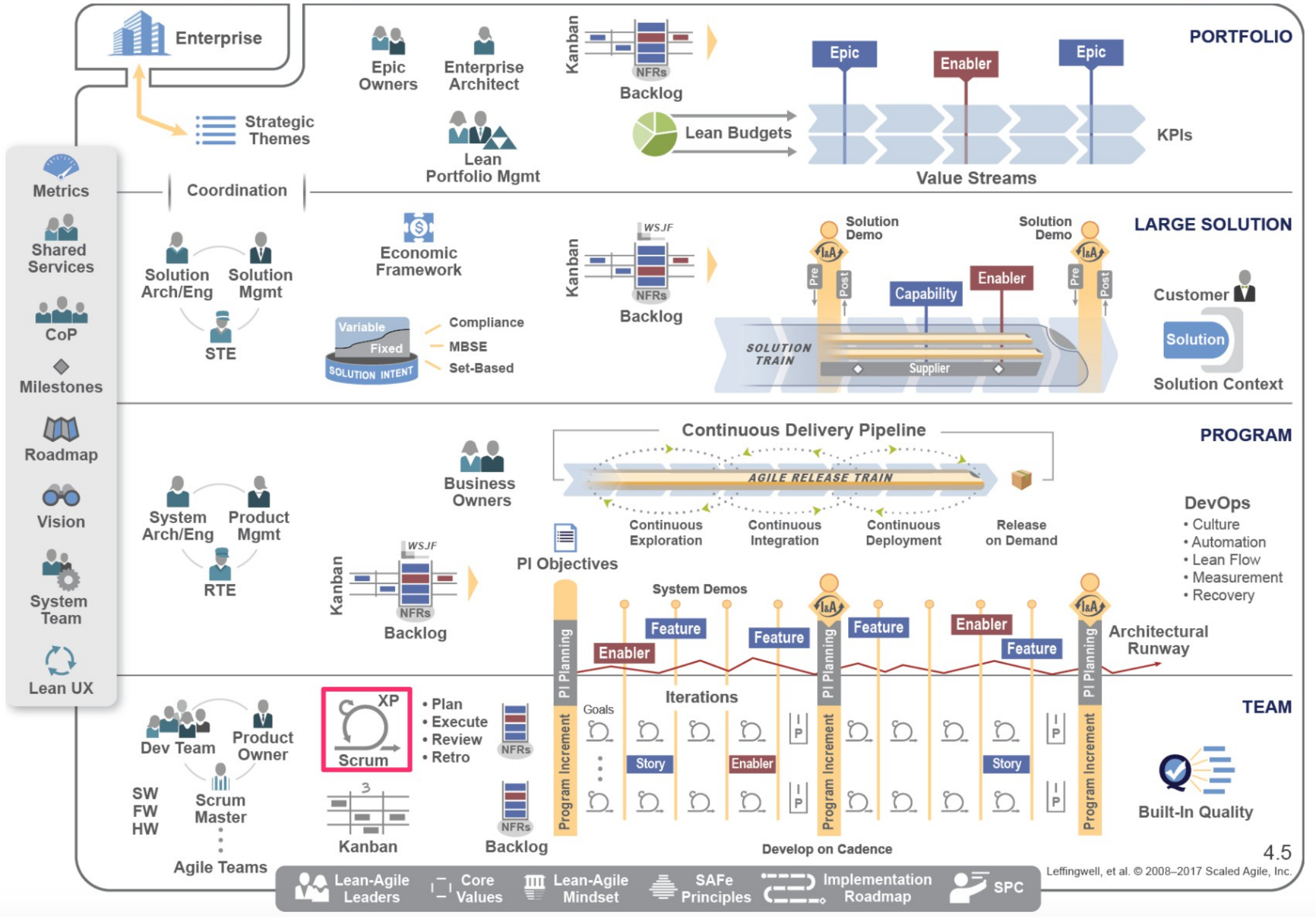
SAFe® for Lean Enterprises

Full SAFe



SAFe® for Lean Enterprises

Full SAFe



4.5

SAFe®

- Sovelluskehityksen ytimessä on SAFe®:n maustama Scrum
- Tiimien koordinointia hallitaan ylhäältä päin (top down) kokoamalla yhdestä tuotteesta vastaavien tiimien joukko käsitteen *release train* alle
  - Release trainin Scrum-tiimit toimivat synkroonissa toistensa kanssa tuottaen yhdessä isomman kokonaisuuden useammasta sprintistä koostuvan *product increment* -ajanjakson aikana
- Product incrementtejä ja niitä toteuttavia release traineja taas ohjaillaan yhä korkeammalta organisaatiosta erilaisten johtajien toimesta
  - SAFe® tarjoaa tähänkin paljon tukea käsitteistön ja määrittelemiensä roolien kautta
- SAFe® on dokumentoitu laajasti ja se antaa erittäin yksityiskohtaista ohjeistusta helpottamaan SAFe®:n käyttöönottoa ja noudattamista
  - Ohjeistusta antavat tietysti kallispalkkaiset konsultit ja räätälöidyt koulutuspaketit ja sertifiointi
- SAFe® vaikuttaa olevan firmojen johdon suosiossa
  - Toisin kuin ketterät menetelmät yleensä SAFe® tarjoaakin firman managementille sopivasti tekemistä roolien ja käytänteiden muodossa

SAFe®:n kritiikkiä

- Kuten aiemmin mainittu, sisältää SAFe® käytännössä kaikki mahdolliset ketterän ja lean-ohjelmistokehityksen parhaat käytänteet
  - Kaikki vieläpä selkeästi ja yksityiskohtaisesti dokumenoituna
- SAFe® onkin eräänlainen agile/lean-kehityksen supermarket, kaikki on helposti saatavissa valmiina pakatussa muodossa
  - Kerää tuotteet ostoskoriin, maksa kassalla, avaa paketti ja seuraa ohjetta...
- SAFe® käytetään paljon, myös Suomessa
- SAFe® on saanut osakseen myös paljon kritiikkiä
  - osa kritiikistä kohdistuu SAFe®:n määrittelemän prosessin raskauteen
  - Osa taas SAFe®:n top down -management luonteeseen
- Esim. Scrumin kehittäjä Ken Schwaber on kyseenalaistanut sen että onko SAFe® ylipäätään *ketterä* menetelmä ketteryyden alkuperäisen idean mukaan
  - Agile Manifestossa on periaate *Individuals and Interactions Over Processes and Tools* ja SAFe® taas prosessina vaikuttaa kovin raskaalta

Large Scale Scrumin eli LeSS

- LeSS:in taustalla on Craig Larman ja Bas Vodde jotka työskentelivät konsultteina 2000-luvun alussa mm. Nokia Siemens Networksilla
- Toisin kuin SAFe®, LeSS on erittäin yksinkertainen, hyvin vahvasti Scrumiin pohjautuva
  - Uusia rooleja, artifakteja ja palavereja ei ole
- LeSSistä on kaksi eri versiota
  - *LeSS* tilanteisiin, missä tuotetta tekee 2-8 Scrum-tiimiä
  - *LeSS Huge* tilanteisiin, missä tiimejä tarvitaan suurempi määrä
- Sekä LeSS että LeSS Huge perustuvat seuraaville oletuksille
  - Kehitetään yhtä tuotetta, jolla on yksi Product Owner ja yksi Product Backlog
  - Kaikilla tiimeillä on samaan aikaan etenevät Sprintit
  - Tiimit tekevät sprintin aikana yhdessä tuotteesta yhden uuden version
    - one shippable product increment
- Yksi LeSS-toteutus on tarkoitettu siis yhden tuotteen kehittämiseen. Jos yrityksellä on useita ei tuotteita, niitä kutakin varten on oma LeSS-toteutuksensa
  - Eli oma erillinen product backlog ja Product owner kullekin LeSS-toteutukselle
  - LeSS ei ota kantaa siihen miten firma hallinnoi tuoteperheitään

LeSS is Scrum

- Large-Scale Scrum (LeSS) isn't new and improved Scrum.
  - It's not Scrum at the bottom for each team, and something different layered on top
- **It's about figuring out how to apply the principles, elements, and elegance of Scrum in a large-scale context, as simply as possible**
  - Like Scrum and other truly agile frameworks, LeSS is "barely sufficient methodology"
- **applied to many teams** cross-functional, cross-component, full-stack feature teams of 3–9 learning-focused people that do all to create done items and a shippable product
- **working together** The teams are working together because they have a common goal to deliver one common shippable product at the end of a common Sprint, and each team cares about this because they are a feature team responsible for the whole, not a part
- **one product** A broad complete end-to-end customer-centric solution that real customers use
  - It's not a component, platform, layer, or library

LeSSin taustalla olevat periaatteet: more with less

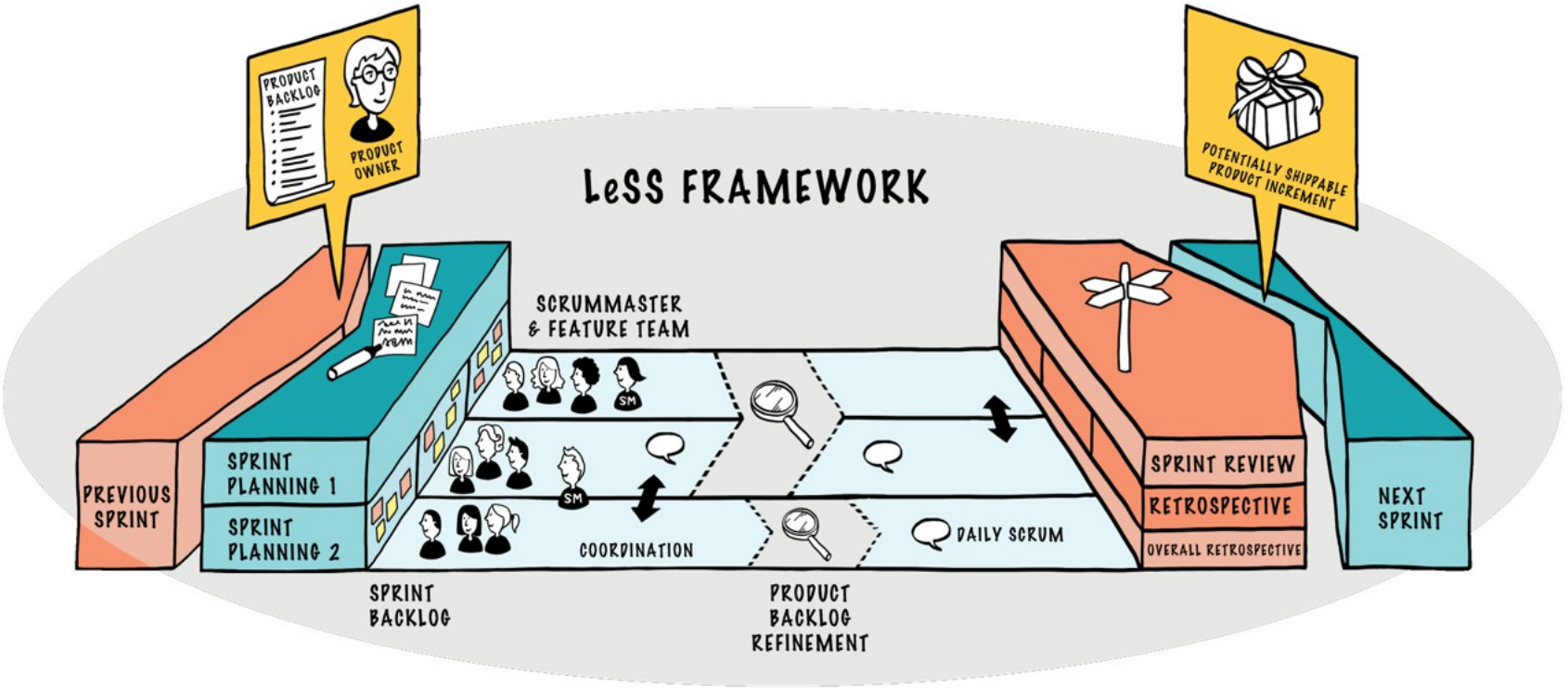
- LeSSin taustalla on joukko tuttuja ketterän ja lean-kehityksen periaatteita
- Periaatteet ovat lähes samat kuin SAFe:ssa, yksi periaatteista tekee kuitenkin selvää eroa menetelmien välille
- **More with less**
  - We don’t want more roles because more roles leads to less responsibility to Teams
  - We don’t want more artifacts because more artifacts leads to a greater distance between Teams and customers
  - We don’t want more process because that leads to less learning and team ownership of process
  - Instead we want more responsible Teams by having less (fewer) roles
  - We want more customer-focused Teams building useful products by having less artifacts
  - We want more Team ownership of process and more meaningful work by having less defined processes.
  - **We want more with less**

LeSS

- Katsotaan hieman tarkemmin LeSS:in pienempää konfiguraatiota
- The smaller LeSS framework is for one (and only one) Product Owner who owns the product, and who manages one Product Backlog worked on by teams in one common Sprint, optimizing for the whole product.
- The LeSS framework elements are about the same as one-team Scrum:
- **Roles**
  - one Product Owner
  - two to eight Teams
  - a Scrum Master for one to three Teams
- **Teams are feature teams**
  - true cross-functional and cross-component full-stack teams that work together in a shared code environment
  - each doing everything to create done items

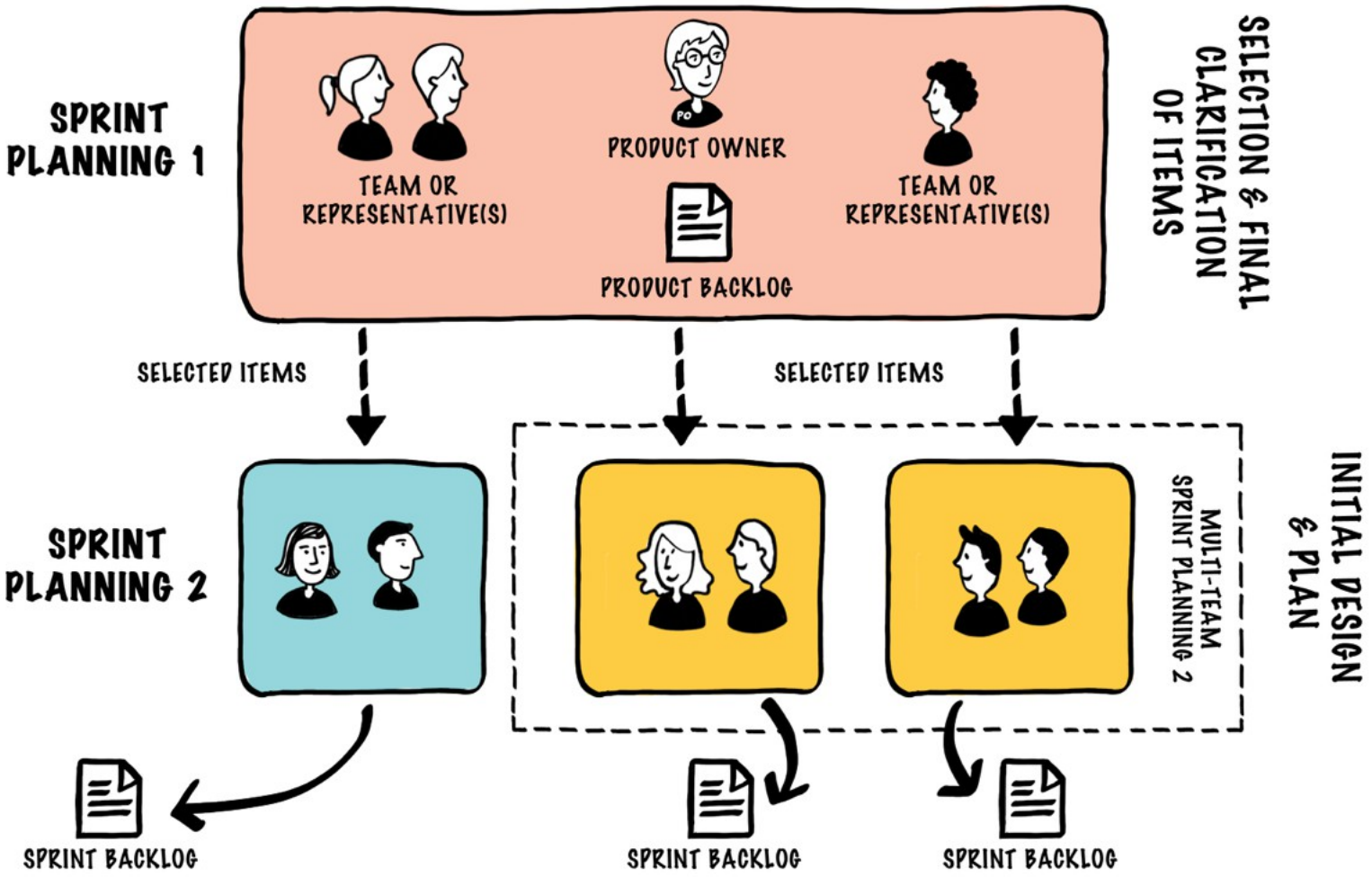
LeSS

- **Artifacts**
  - One potentially shippable product increment
  - one Product Backlog,
  - a separate Sprint Backlog for each Team.
- **Events**
  - One common Sprint for the whole product
  - it includes all teams and ends in one potentially shippable product increment



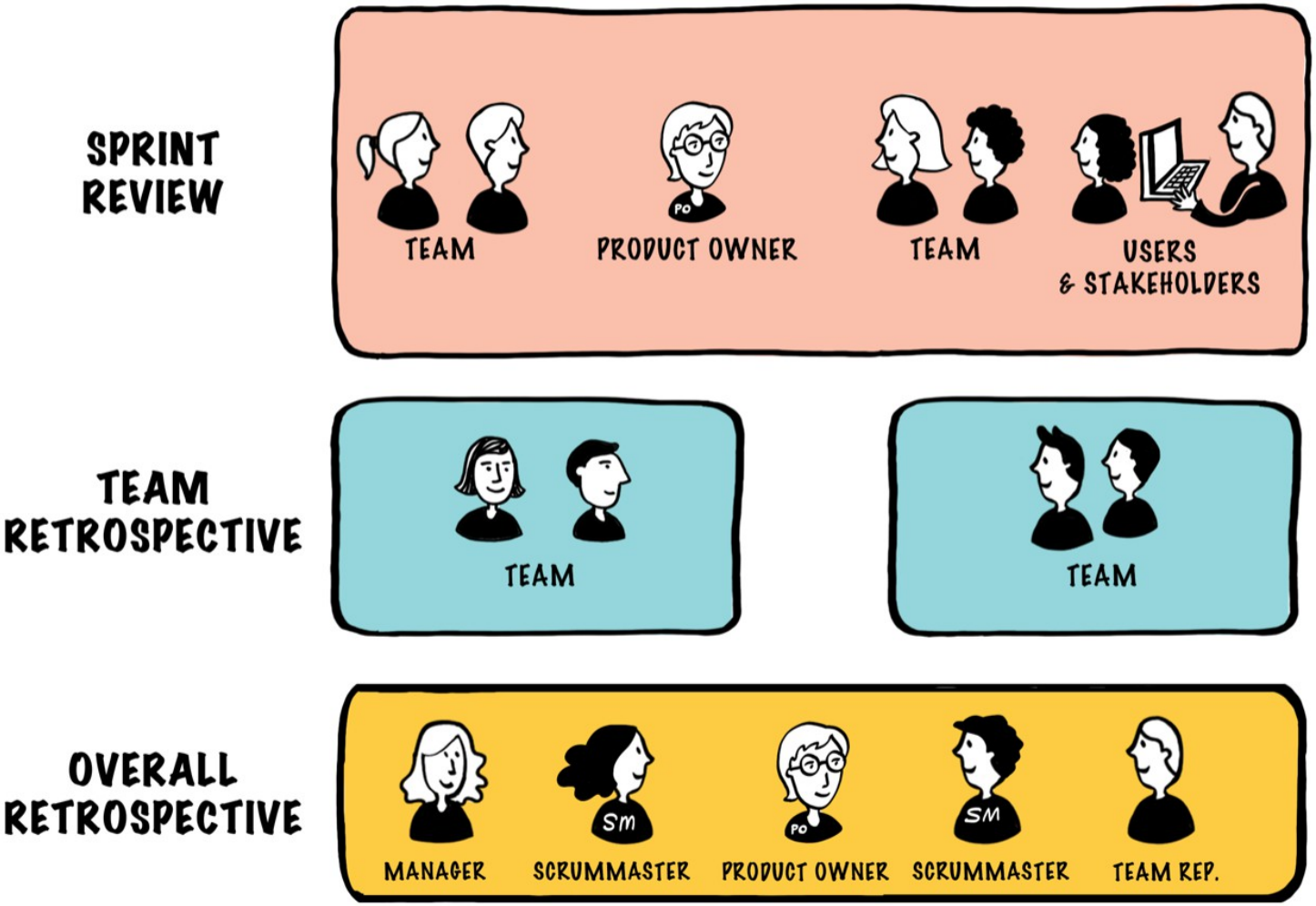
LeSS: kaksiosainen sprintin suunnittelu

- Sprintin suunnittelun ensimmäisessä osassa Product Owner ja tiimien edustajat valitsevat Product Backlogilta kaikille eri tiimille seuraavan sprintin storyt
- Suunnittelun toisessa osassa tiimit muodostavat kukin **oman sprint backlogin** joiden avulla sprintin sisäinen toiminta hallitaan normaalin Scrumin tapaan



LeSS: sprintin katselmointi ja kaksiosainen retrospektiivi

- Kaikkien tiimien yhteinen aikaansaannos (one shippable product increment) katselmoidaan yhdessä
- Retrospektiivi on kaksitasoinen: tiimikohtainen ja koko tuotteen valmistusprosessia tarkasteleva overall-retrospektiivi, missä on edustus kaikista tiimeistä ja mahdollisesti yrityksen johdosta



Muu tiimien välinen koordinointi

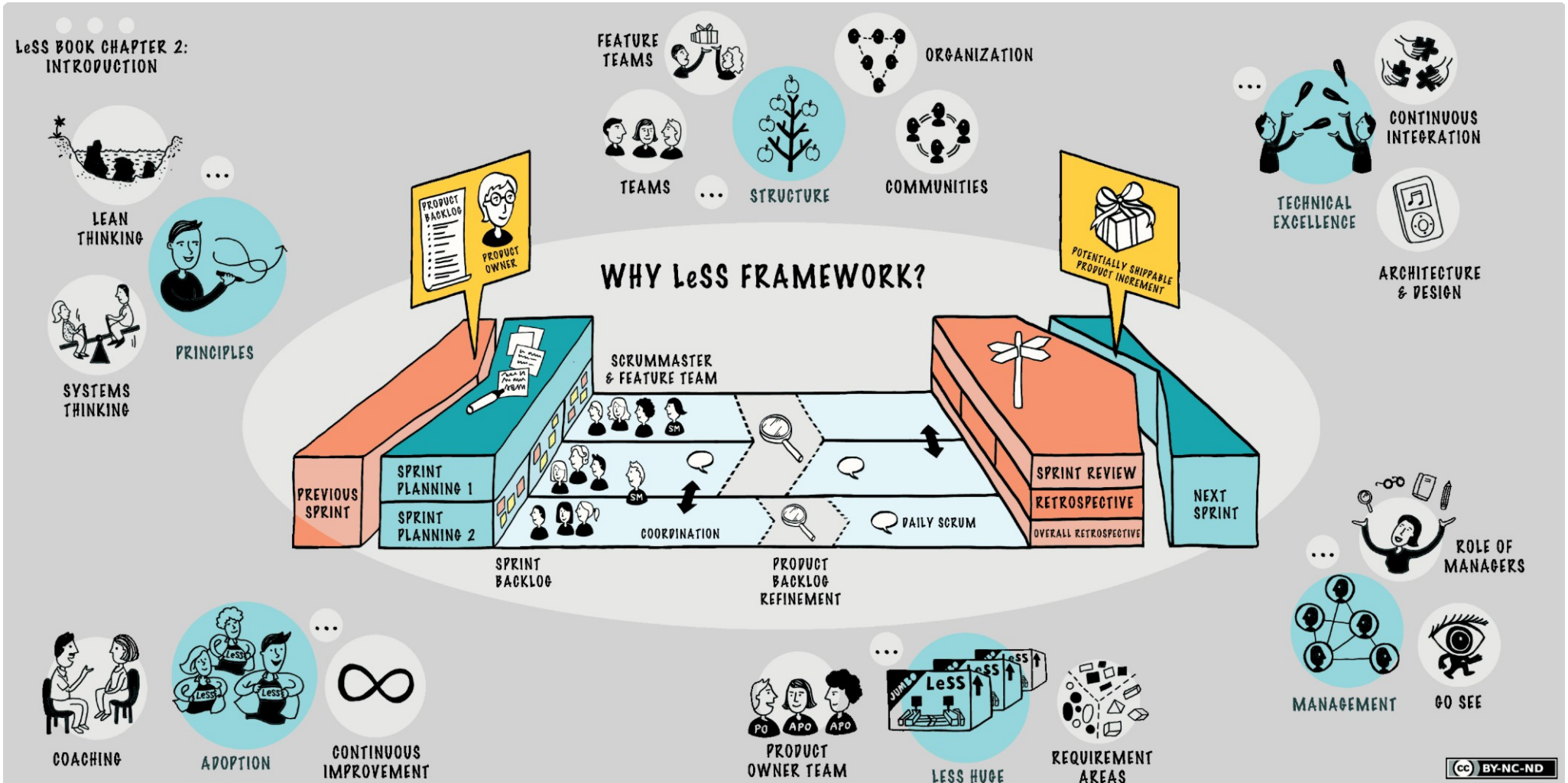
- Kaikille tiimeille yhteisen sprintin suunnittelun (ensimmäisen osan), Sprint reviewin ja overall-retrospektiivin lisäksi LeSS ei edellytä muita tiimien väliseen koordinaatioon tarkoitettuja yhteisiä tapaamisia
- LeSS antaa joukon aiheeseen liittyviä ohjeita ja suosituksia
- Cross-team coordination is decided by the teams
  - Prefer decentralized and informal coordination over centralized coordination
  - Emphasize *Just Talk* and informal networks via communicate in code, cross-team meetings, component mentors, travelers, scouts, and open spaces
  - *Just Talk* korostaa suoran kommunikoinnin tärkeyttä
  - *Communicate in code* tarkoittaa ryhmien välistä työskentelyä helpottava ohjelmointitapaa, mm. yhteisiä koodikäytänteitä ja jatkuvaa integraatiota
  - *Scouteilla* tarkoitetaan muiden tiimien daily scrumissa vierailemista
- LeSS mainitsee myös Scrum of Scrums -palaverit yhtenä mahdollisena tiimienvälisen koordinoinnin muotona, mutta suosittelee mielummin informaalimpia kommunikaation muotoja

Backlogin ylläpito

- Scrum olettaa, että noin 5-10% sprintin työskentelystä käytetään backlog groomingiin/refinementiin, jonka tarkoituksena siis pitää backlog DEEP:inä
- LeSS kiinnittää eksplisiittisesti huomioita backlogin ylläpitämiseen
  - The Product Owner shouldn’t work alone on Product Backlog refinement
  - He is supported by the multiple Teams working directly with customers/users and other stakeholders
  - All prioritization goes through the Product Owner
  - Clarification is as much as possible directly between the Teams and customer/users and other stakeholders
  - Product Backlog Refinement (PBR) is done per team for the items they are likely going to do in the future
  - Do multi-team and/or overall PBR to increase shared understanding and exploiting coordination opportunities when having closely related items or a need for broader input/learning
- LeSS siis rohkaisee voimakkaasti sovelluskehittäjien ja asiakkaiden/loppukäyttäjien läheiseen kanssakäymiseen

LeSS huge

- Edelleen yksi tuote ja yksi product backlog ja yksi vastuunalainen product owner
- Backlog jaetaan *vaatimusalueisiin* (requirement area)
  - Jokaiselle alueelle siitä vastaava *Area Product Owner*
    - Muodostetaan koko tuotetta hallinnoiva Product owner -tiimi
  - Vaikka kyseessä yksi backlog, tarjotaan siihen oma aluekohtainen näkymä



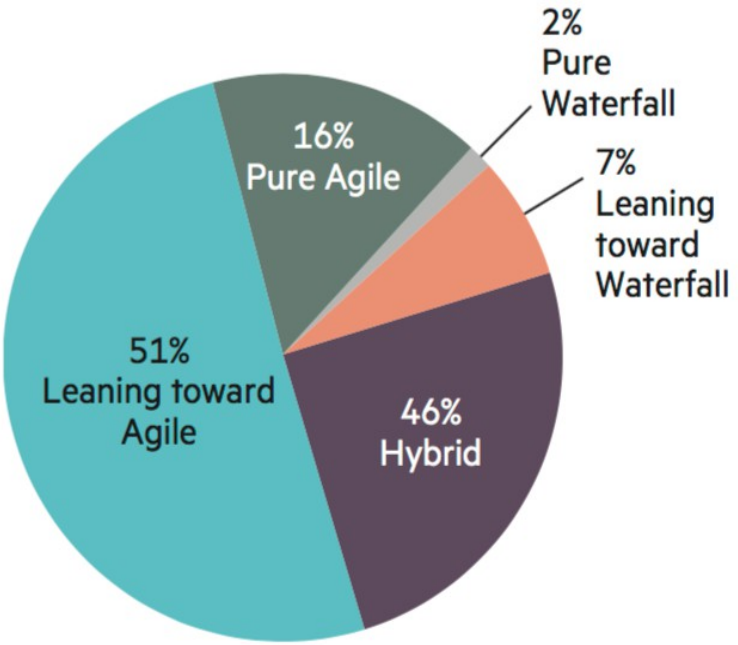
LeSS

- On erittäin mielenkiintoista että molemmat SAFe® ja LeSS ovat pitkälti syntyneet Suomessa ja Nokialla
- Nokian organisaatiorakenteen takia Nokia Mobile Phonesin (NMP) ja Nokia Siemens Networksin (NSN) ohjelmistokehitystapa oli kuitenkin täysin erilainen
- SAFe® (NMP) ja LeSS (NSN) samoista taustaperiaatteistaan ja yhteisestä syntykonsernista huolimatta ovat hyvin erilaisia menetelmiä
- Osoitteesta http://gosei.fi/blog/less-safe-comparison/ löytyy hyvä vertailu menetelmistä
- Kuten aiemmin todettiin SAFe® on suosittu yritysjohdon keskuudessa, mutta saanut paljon kritiikkiä arvovaltaistenkin ketterän kehityksen edustajien toimesta
- En tiedä kuvastaako se mitään menetelmien pitkän tähtäimen toimivuudesta, mutta SAFe®:n kotia Nokia Mobile Phonesia ei enää ole olemassakaan, Nokia (Siemens) Networks taas on nykyinen Nokia ja soveltaa edelleen LeSS-menetelmää

Ketterien menetelmien käyttö tutkimuksen valossa

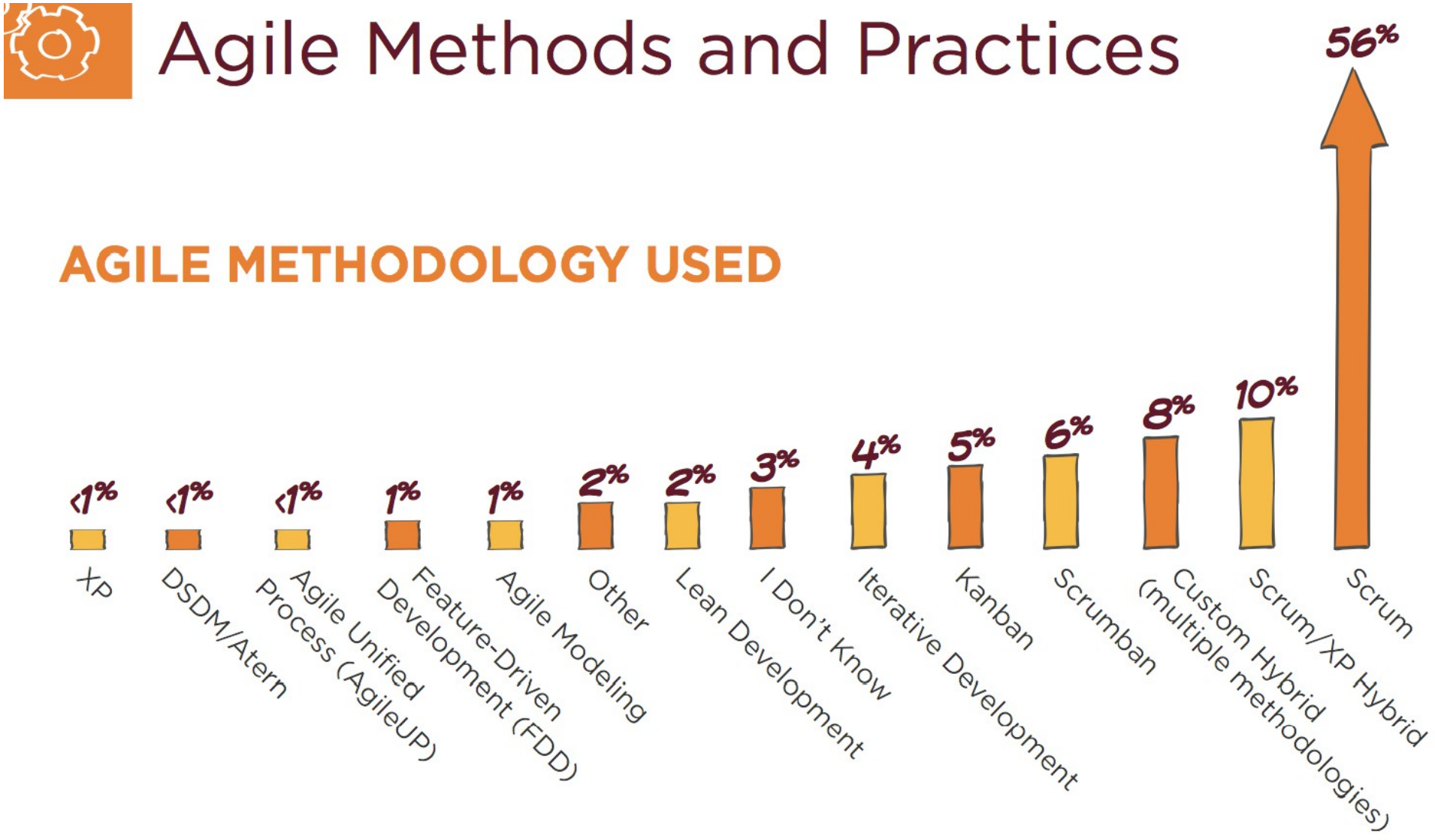
Miten laajalti ketteriä menetelmiä käytetään?

- HP:n vuonna 2015 tekemä tutkimus julistaa ”Agile is the new normal”
  - Tutkimuksessa 601 vastaajaa
  - A hybrid: incorporate at least some Agile solutions and principles
  - Leaning towards agile jää määrittelemättä



- Ketteryys on Suomessa suosittua:
  - The results of the survey reveal that a majority of respondents' **organizational units are using agile and/or lean methods (58%)**
  - Markkula ym.: Survey on Agile and Lean usage in Finnish software industry, ESEM 2012 (ks. ACM digital library)
- Loppuvuodesta 2016 julkaistussa Brasiliassa, Suomessa ja Uudessa Seelannissa tehdyssä tutkimuksessa ”Adoption and Suitability of Software Development Methods and Practices” mainitaan seuraavat luvut
  - Scrum was most often utilized by respondents (71.2%), with Kanban (49.5%), Lean (39.7%) and Waterfall (35.3%) following in that order
  - http://ieeexplore.ieee.org/document/7890614/

Mitä ketteriä menetelmiä käytetään?



- VersionOnen ”internetin virallisesta” vuosiraportista
  - http://stateofagile.versionone.com

Ketterät käytänteet

- VersionOne:

80%	Daily standup
79%	Short iterations
79%	Prioritized backlogs
71%	Iteration planning
69%	Retrospectives
65%	Release planning
65%	Unit testing
56%	Team-based estimation
53%	Iteration reviews
53%	Taskboard
50%	Continuous integration
48%	Dedicated product owner
46%	Single team (integrated dev & testing)
43%	Coding standards
38%	Open work area
36%	Refactoring
34%	Test-Driven Development (TDD)
31%	Kanban
29%	Story mapping
27%	Collective code ownership
24%	Automated acceptance testing
24%	Continuous deployment
21%	Pair programming
13%	Agile games
9%	Behavior-Driven Development (BDD)

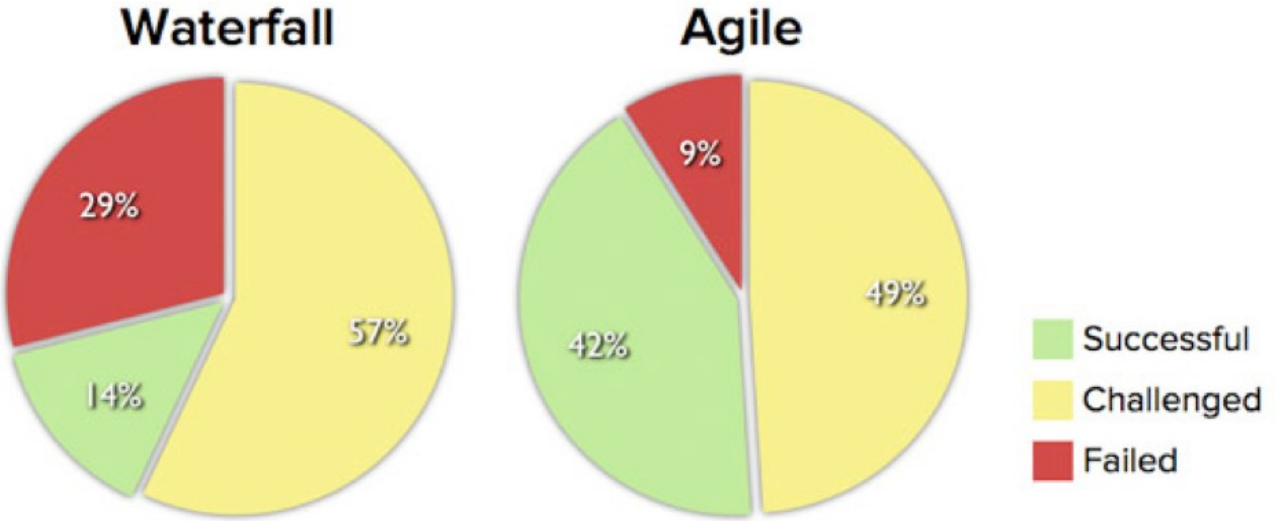
- Suomen tilanne: http://esem.cs.lth.se/industry_public/RodriguezetaI_ESEM2012_IndustryTra

Ketterät käytänteet Suomesta tehdyssä tutkimuksessa (n=225)

Practices	n	Mean	Median
Prioritized work list	204	4,2	4
Iteration/sprint planning	203	4,1	4
Daily stand-up meetings	209	3,7	4
Unit testing	199	3,7	4
Release planning	196	3,9	4
Active customer participation	196	3,5	4
Self-organizing teams	194	3,5	4
Frequent and incremental delivery of working software	189	4,1	4
Automated builds	185	3,5	4
Continuous integration	182	3,8	4
Test-driven development (TDD)	179	2,7	3
Retrospectives	177	3,6	4
Burn-down charts	174	3,2	3
Pair programming	174	2,4	2
Refactoring	163	3,4	3
Collective code ownership	159	3,3	3

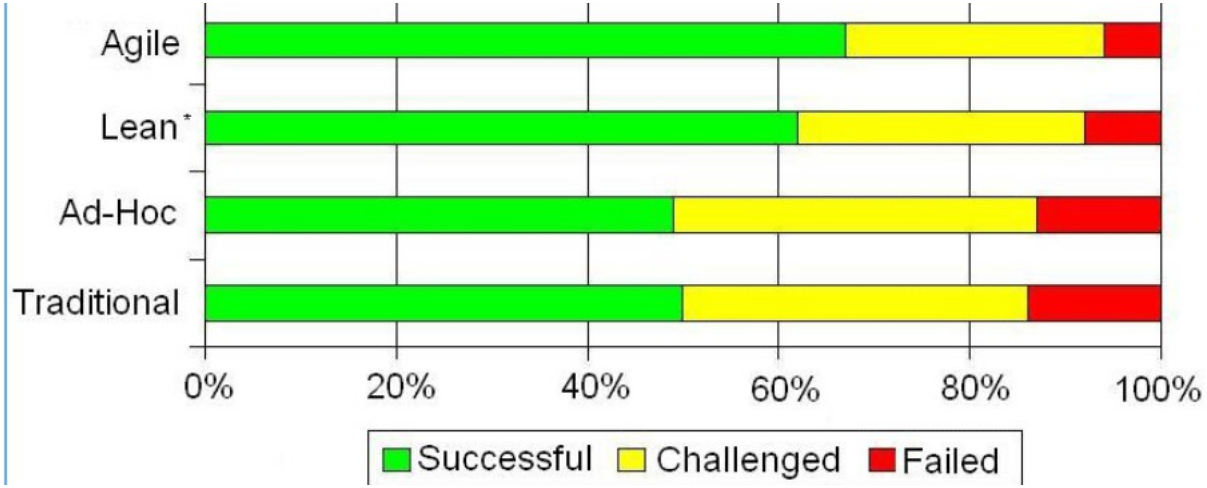
Projektien onnistuminen: ketterä vastaan perinteinen

- Standish CHAOS raport 2012



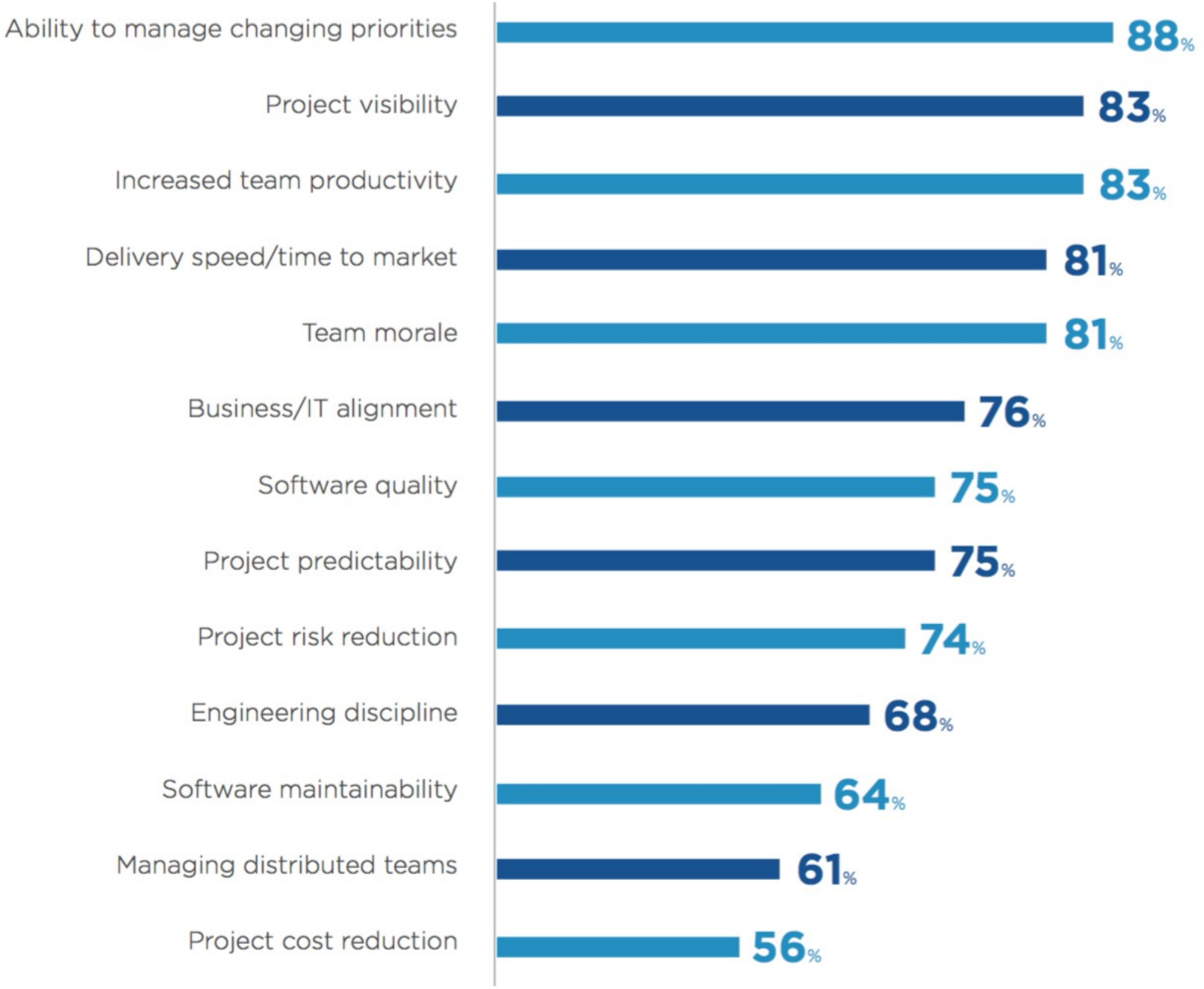
- Scott Ambler, Agile vs perinteinen 2011:

http://www.drdobbs.com/architecture-and-design/how-successful-are-it-p



Ketteryydellä saavutettuja etuja tarkemmin eriteltynä

- VersionOne 2016

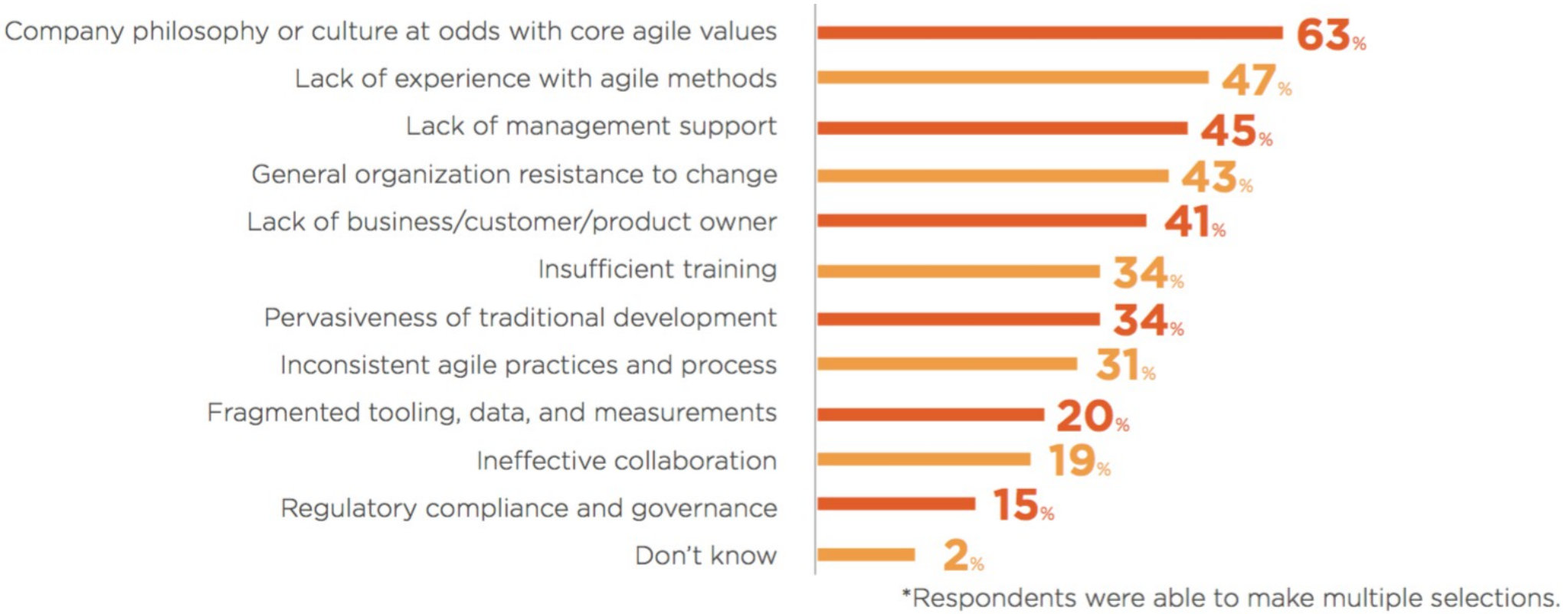


Ketteryydellä saavutettuja etuja Suomessa...

Effect	n	Mean	Median
Improved team communication	204	4,0	4
Enhanced ability to adapt to changes	203	3,9	4
Increased productivity	201	3,8	4
Enhanced process quality	198	3,7	4
Improved learning and knowledge creation	197	3,7	4
Enhanced software quality	196	3,8	4
Accelerated time-to-market/cycle time	192	3,7	4
Reduced waste and excess activities	190	3,5	4
Improved customer collaboration	190	3,7	4
Improved organizational transparency	187	3,5	4
Improved customer understanding	188	3,7	4

Suurimmat haasteet ketterien menetelmien käytössä

- VersionOne 2016



Evidenssiä on, mutta...

- Oikeastaan kaikki edelliset olivat kyselytutkimuksia
  - Käsitteitä ei ole kunnolla määritelty (esim. mitä ketteryydellä tai projektin onnistumisella tarkoitetaan)
  - Kyselyyn osallistuneet eivät välttämättä edusta tasaisesti koko populaatiota
  - Kaikkien kyselyjen tekijät eivät puolueettomia menetelmien suhteen (esim. Ambler ja VersionOne)
- Eli tutkimusten validiteetti on kyseenalainen
- Toisaalta kukaan ei ole edes yrittänyt esittää evidenssiä, jonka mukaan vesiputousmalli toisi systemaattisia etuja ketteriin menetelmiin verrattuna
- Myös akateemista tutkimusta on todella paljon (mm. Markkulan ym. kyselytutkimus) ja eri asioihin kohdistuvaa. Akateemisenkin tutkimuksen systemaattisuus, laatu ja tulosten yleistettävyys vaihtelee
  - Ohjelmistotuotannossa on liian paljon muuttujia, jotta jonkin yksittäisen tekijän vaikutusta voitaisiin täysin vakuuttavasti mitata empiirisesti
  - Menetelmiä soveltavat kuitenkin aina ihmiset, ja mittaustulos yhdellä ohjelmistotiimillä ei välttämättä yleisty mihinkään muihin olosuhteisiin
- Olemassa olevan evidenssin nojalla kuitenkin näyttää siltä, että ongelmistaan huolimatta ketterät menetelmät ovat useimmissa tapauksissa järkevä tapa ohjelmistokehitykseen

Koe

Koe

- Maanantaina 18.12. salissa A111 klo 09.00 salissa A111
- Kurssin pisteytys
  - Koe 20p
  - Laskarit 10p
  - Miniprojekti 10p
- Kurssin läpipääsy edellyttää
  - 50% pisteistä
  - 50% kokeen pisteistä
  - Hyväksyttyä miniprojektia
- Kokeessa on sallittu yhden A4:n kokoinen käsin, itse kynällä kirjoitettu lunttilappu

Mitä kokeessa **ei** tarvitse osata

- Git
- Gradle
- Travis
- JUnit
- Mockito
- Cucumber
- Selenium

Reading list – eli lue nämä

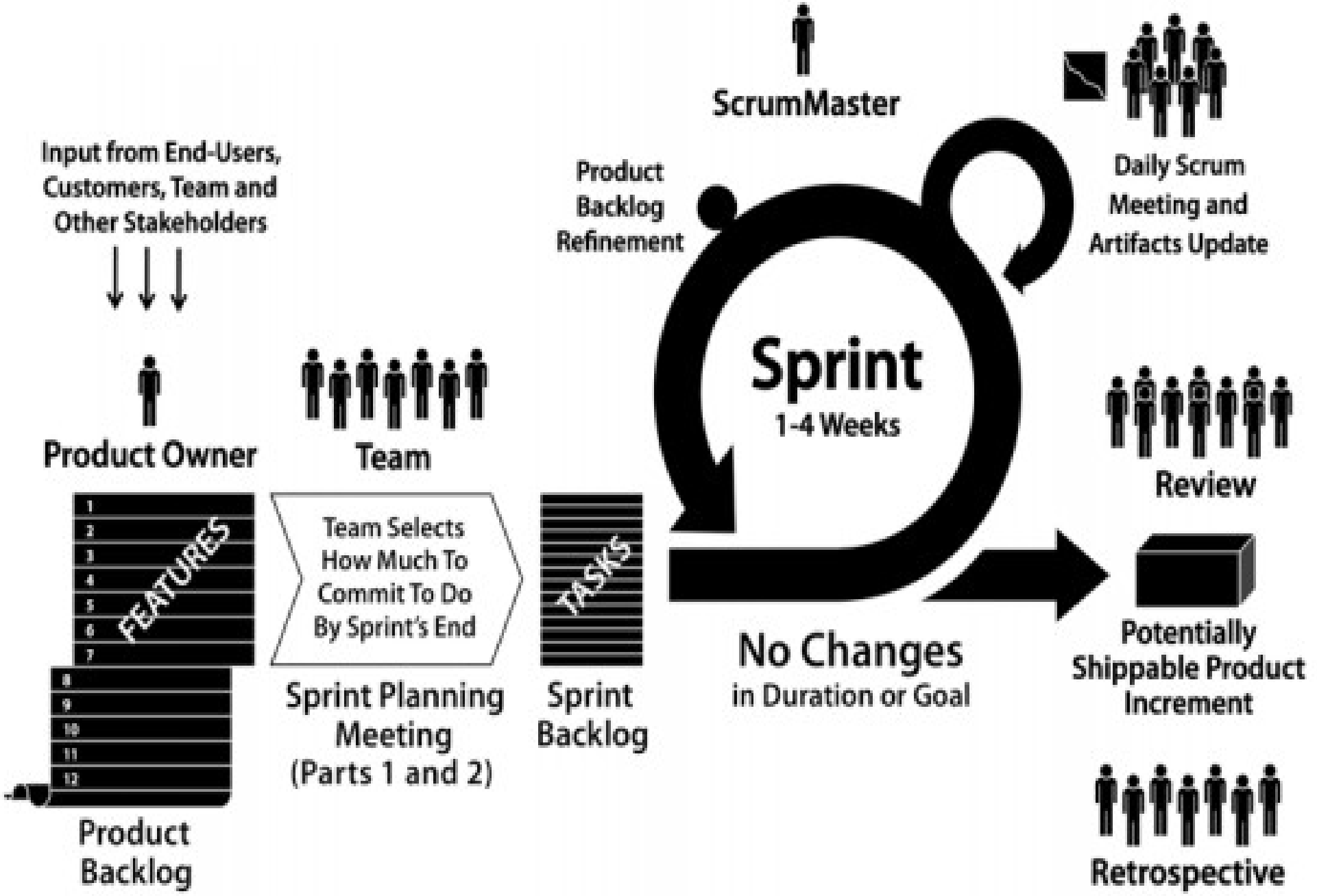
- Luentokalvot ja laskarit (paitsi edellisellä sivulla mainittujen osalta)
- Oliosuunnittelun itseopiskelumateriaali
  - https://github.com/mluukkai/ohjelmistotuotanto2017/blob/master/web/olios
- Ja seuraavat
  - http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf
  - http://www.infoq.com/minibooks/scrum-xp-from-the-trenches
    - Sivut 1-86 (painos 1), sivut 1-92 (painos 2)
  - http://www.leanprimer.com/downloads/lean_primer.pdf
  - http://sourcemaking.com/design_patterns
    - Tarpeellisissa määrin

Tärkeät teemat vielä pikakelauksella

Luento 1

- Termi software engineering
  - Mitä pitää sisällään
- Prosessimallit
  - Vaiheet
    - Vaatimusmäärittely
    - Suunnittelu
    - Toteutus
    - Testaus
    - Ylläpito
  - vesiputous/lineaarinen/BUFD
  - Iteratiivinen
  - Ketterä
- Motivaatio prosessimallien kehittymiselle

Luento 2 - Scrum



Luento 3 - vaatimusmäärittely

- Vaatimukset jakautuvat
  - Toiminnallisiin
  - Ei-toiminnallisiin (rajoitteet ja laatuvaatimukset)
- Vaatimusmäärittelyn luonne ja vaiheet
  - oldschool vs. moderni
- Ketterä vaatimustenhallinta
  - User story
    - Arvoa tuottava toiminnallisuus
    - ”Card, conversation, confirmation”
    - INVEST
    - Estimointi

Luento 4

- Ketterä vaatimustenhallinta
  - Product backlog
    - DEEP
  - Julkaisun suunnittelu
  - Velositeetti
- Sprintin suunnittelu
  - Storyjen valinta
  - Storyistä taskeihin
- Sprint backlog
  - Taskboard
  - Burndown
- Wip-rajoitteet
  - Scrumban-prosessi

Luento 5 - laadunhallinta

- Validointi ”are we building the right product”
  - Katselmointi ja tarkastukset
  - Vaatimusten validointi (ketterä vs. trad)
  - Koodin katselmointi
- Verifiointi ”are we building the product right”
  - Vastaako järjestelmä vaatimusmäärittelyä
- Verifiointi tapahtuu yleensä testauksen avulla
  - Testauksen tasot:
    - Yksikkö-, Integraatio-, Järjestelmä-, Hyväksymätestaus
  - Käsitteitä:
    - black box, white box, ekvivalenssiluokka, raja-arvo, testauskattavuus
  - regressiotestaus
  - Ohjelman ulkoinen laatu vs. sisäinen laatu

Luento 6 - ketterä laadunhallinta

- Testaus ketterissä menetelmissä
  - Automaattiset regressiotestit tärkeät
- TDD
  - Red – green – refactor
  - Enemmän suunnittelua kun testausta, testit sivutuotteena
- Storytason testaus / ATDD / BDD
- Jatkuva integraatio ja jatkuva käyttöönotto
  - ”integraatiohelvetti” → Daily build / smoke test → jatkuva integraatio → continuous delivery → continuous deployment
  - CI/CD ei ole pelkkä työkalu vaan workflow ja mentaliteetti
  - Deployment pipeline
- Tutkiva testaus
  - ”Exploratory testing is simultaneous learning, test design and test execution”

Luento 7

- Tuotannossa tapahtuva laadunhallinta
  - Blue-green-deployment
  - Canary releaset
  - Feature toggle
- DevOps
- Ohjelmiston arkkitehtuuri
  - Arkkitehtuurin määritelmiä
  - Arkkitehtuurimallit
    - kerrosarkkitehtuuri

Luento 8 – oliosuunnittelu

- Arkkitehtuurimallit
  - mikropalveluarkkitehtuuri
- Arkkitehtuurin kuvaaminen
  - Monia näkökulmia, erilaisia kaavioita
- Arkkitehtuuri ketterissä menetelmissä
  - Ristiriita arkkitehtuurivetoisuuden ja ketterien menetelmien välillä
  - Inkrementaalinen arkkitehtuuri: Edut ja haitat
- Helposti ylläpidettävän eli sisäiseltä laadultaan hyvän koodin tunnusmerkit ja laatuattribuutit
  - kapselointi, koheesio, riippuvuuksien vähäisyys, toisteettomuus, selkeys, testattavuus

- Suunnittelumallit itseopiskelumateriaalissa
  - https://github.com/mluukkai/ohjelmistotuotanto2017/blob/master/web/ol

Luento 9

- Käsitteet tekninen velka, koodihaju ja refaktorointi
- Lean
  - Peräisin Toyoa production systemistä
  - Tavoite, perusta, peruspilarit ja periaatteet
  - Lean-ohjelmistokehitys ja sen suhde ketteriin menetelmiin

Luento 10 - Laajan mittakaavan Scrum

- Scrum of scrums
- SAFe ja LeSS