VAASAN YLIOPISTO
TEKNILLINEN TIEDEKUNTA
TIETOTEKNIIKKA

Jouko Suvanto

OHJELMISTOKEHITYKSEN LAATUAJATTELU

Tietotekniikan
pro gradu-tutkielma

VAASA 2012

SISÄLLYSLUETTELO

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Jouko Suvanto
Tutkielman nimi: Ohjelmistokehityksen laatuajattelu
Ohjaajan nimi: Anja Jousranta
Tutkinto: Kauppatieteiden maisteri
Oppiaine: Tietotekniikka
Opintojen aloitusvuosi: 2006
Tutkielman valmistumisvuosi: 2012 **Sivumäärä:** 84

TIIVISTELMÄ:

Tutkielmassa on selvitetty ja kuvattu sitä, mihin ohjelmistokehityksen laatuajattelu perustuu, miten se on kehittynyt 1800-luvulta nykyiseen tilaansa ja miten sen pohjalta ohjelmistoprosessia voidaan parantaa.

Tieteellisfilosofisen viitekehyksen tutkielmalle muodostavat teoriat paradigmojen muuttumisesta, kolmen maailman objekteista ja tiedonintresseistä. Ymmärtääkseen ohjelmistojen laatuajattelua tulee tuntea se historiallinen tausta, tapahtumien ketju, jonka tuloksena ajattelu on kehittynyt. Tutkielmassa on kartoitettu tätä ketjua kolmen eri kertomuksen kautta: teollisen valmistuksen historian, laatuajattelun historian ja ohjelmistokehityksen historian.

Viimeisen kahdensadan vuoden aikana tapahtuneen teknologisen kehityksen myötä ovat myös työn tekemisen ja johtamisen suhteet teollisuudessa ovat muuttuneet. Vanhasta kokonaisvaltaisesta työn hallinnasta on siirrytty enemmän kapea-alaisen vaihetyön suorittamiseen. Samanlaista kehitystä on voitu havaita ohjelmistokehityksessä, johon on kopioitu mekaanisesti teollisuuden valmistusmenetelmiä ja – käytäntöjä.

Laatuajattelun perustan muodostavat tilastollinen laadunohjaus ja laatutyön trilogia. Toiminnan kehittäminen edellyttää organisaatiolta myös yhteistä tietämysperustaa, syvällisiä tietoja niistä keskeisistä teorioista, jotka ohjaavat käytännön parantamistyötä. Uusien tuotteiden kehittämisessä muodostaa organisaation tietämyksen hallinta usein kriittisen kilpailutekijän.

Ohjelmistokehityksessä on viimeisten kymmenen vuoden aikana perinteisempien suunnitelmapohjaisten kehitysmallien rinnalle tullut ketterämpiä, inkrementaalisia ja iteratiivisia malleja, joilla pyritään paremmin sopeutumaan toimintaympäristön nopeisiin muutoksiin ja parantamaan ohjelmistokehityksen tuottavuutta.

AVAINSANAT: Ohjelmistokehitys, ohjelmistoprosessin parantaminen, laadun parantaminen, laadun historia

UNIVERSITY OF VAASA
Faculty of technology
Author: Jouko Suvanto
Topic of the Master's Thesis: Quality Thinking in the Software Development
Instructor: Anja Jousranta
Degree: Master of Science in Economics and Business Administration
Major subject: Computer Science
Year of Entering the University: 2006
Year of Completing the Masters Thesis: 2012 **Pages:** 84

ABSTRACT:

This Master's thesis aim is to survey the profound elements of quality thinking in the software development. The study describes how the quality thinking has developed from its roots in 18th century to its present state and how the software process improvement can be performed on the basis of today's quality thinking approach.

The philosophical framework for this thesis is based on the following theories: the paradigm shifts, Popper's three worlds and the knowledge interests. To understand the software quality thinking of today one must understand its historical background, the chronology of those events of which this thinking is a result. My thesis depicts this background through three big stories: the history of the industrial manufacturing, the history of quality thinking and the history of the software development.

The technology development that has taken place during last two hundred years has also changed the relationships between the management and the work force in the industry. From the total control of work we have gone over to the narrower phase work where the content of work is depleted. Similar progress can be detected in the software development, into which the production methods and practices has been copied directly from the industry.

The foundation of quality thinking is formed by the statistical quality control and the trilogy of quality work. The development of operations requires that the organisation shares the same profound knowledge. This profound knowledge must cover those theories that guides and support the practical improvement actions. In the new products development the organisational knowledge is often a critical competitive factor.

During the last ten years traditional plan-based software development models have been replaced by more agile, incremental and iterative models. With these new models software organisations try better to response to the changes taking place in their business environment and improve their productivity.

KEYWORDS: Software development, software process improvement, quality improvement, the history of quality

LYHENTEET

ASD	Adaptive Software Development
B2B	Business-to-Business
B2C	Business-to-Customers
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
CSF	Critical Success Factor
DMAIC	Define, Measures, Analyse, Improve, Control
FDD	Feature-Driven Development
FMS	Flexible Manufacturing System
HR	Human Resources
HRD	Human Resource Development
HW	Hardware
IS	Information Systems
ISD	Information Systems Development
ICT	Information and Communication Technology
JIT	Just-In-Time
KM	Knowledge Management
KPA	Key Process Area
LSWD	Lean Software Development
NPD	New Product Development
OD	Organization Development
OL	Organizational Learning
PDCA	Plan, Do, Check, Act
PDSA	Plan, Do, Study, Act
R&D	Research and Development
ROI	Return-on-Investment
SCM	Software Configuration Management
SEI	Software Engineering Institute (Carnegie Mellon)
SEL	Software Engineering Laboratory (NASA)
SPC	Statistical Process Control
SPI	Software Process Improvement
SWD	Software Development
TDD	Test-Driven Development
TPS	Toyota Production System
TQC	Total Quality Control
TQM	Total Quality Management
UCD	User Centric Design
XP	Extreme Programming

KUVALUETTELO

1. JOHDANTO

1.1. Yleistä

Olen itse ollut tekemisissä tietokoneiden kanssa yli 30 vuotta. Harppaus ensimmäisistä ohjelmoitavista laskimista nykyisiin kannettaviin ja älykkäisiin päätelaitteisiin on ollut suuri. Keskeistä osaa tietokoneiden käytössä ja toiminnassa ovat näytelleet ohjelmistot, joiden avulla koneiden toimintaa ohjataan ja muokataan. Aluksi ohjelmien teko oli lähinnä insinöörimäistä kytkentätyötä, mutta myöhemmin, runsaan 50 vuoden aikana, se on muuttunut ja jakautunut eri suuntiin. Yksittäisen ohjelmoijan (kytkentäinsinöörin) sijaan on tullut joukko ihmisiä, joilla on erilaisia rooleja ja tehtäviä. Nämä roolit muotoutuivat aluksi suurkoneympäristössä vastaamaan ympäristön ja sen aikaisen toiminnan vaatimuksia. Entiset roolit (kuten atk-operaattori ja reikäkorttilävistäjä) ovat kadonneet tietokoneiden rajapintojen ja työkalujen muuttuessa samalla kun uudet roolit ovat korvanneet ne. Tällaisia uusia rooleja ovat esimerkiksi käyttökokemusasiantuntija, sovellusarkkitehti, testaussuunnittelija, konfiguraatiopäällikkö ja Master Data Specialist.

Tänään vallalla ovat fyysisesti huomattavasti pienemmissä koneissa suoritettavat ohjelmat sekä sulautetut, älykkäisiin elektroniikkalaitteisiin integroidut ohjelmistot. Ohjelmistotyötä teetetään suureksi osaksi alihankkijoilla ja usein jopa toisella puolella maapalloa. Integroitavuudesta ja verkko-ominaisuuksista on tullut niitä kriittisiä kehitysalueita, jotka määrittävät viime kädessä nykyisten ohjelmistojen arvoa ja menestystä. Ohjelmiston kehittämisestä on tullut monien eri toimijoiden ja osapuolten yhteistyötä ja työ, joka vaaditaan toimivan ja käyttökelpoisen ohjelmiston aikaansaamiseksi, on vahvasti fragmentoitunut sekä jakautunut useiden eri ihmisten tai organisaatioiden suoritettavaksi.

Erään SEI:n raportin mukaan (Gibson 2006) jopa 70 % parannushankkeista epäonnistuu tavalla tai toisella. Nämä hankkeet eivät saavuta tavoitteitaan: ne eivät pysy aikataulussa, ylittävät budjettinsa tai sitten niiden tuloksilla ei olekaan enää mitään arvoa muuttuneessa ympäristössä. Yleisimpinä epäonnistumisen syinä mainitaan

yrityksen johdon puutteellinen osallistuminen ja epärealistiset odotukset koko SPI -toiminnalle.

Ohjelmistoyritysten on jatkuvasti oltava valmiina reagoimaan markkinoiden muutoksiin, uusiin asiakasvaatimuksiin, teknologioihin liittyviin innovaatioihin ja uusiin kilpailijoihin. Menestyäkseen tällaisissa dynaamisissa oloissa yritysten ohjelmistoprosessin parantamishankkeiden pitää olla hyvin organisoituja, johdettuja ja toteutettu siten, että niiden on helppo tunnistaa ja vastata ympäristön odotettuihin ja odottamattomiin tapahtumiin. Ohjelmistoalan tutkijat ja toimijat ovat muutaman viime vuoden aikana laajalti omaksuneet ketterän toiminnan periaatteita ja käytäntöjä, joiden avulla pyritään ja pystytään nopeammin mukautumaan asiakastarpeissa tapahtuviin muutoksiin. On huomattava, ettei kuitenkaan löydy sellaista tutkimusta, joka selvittäisi, miten nämä periaatteet ja käytännöt todellisuudessa selkeästi ja yksiselitteisesti parantavat organisaatioiden kykyä kehittää korkealaatuisia ohjelmistoja.

1.2. Tutkimuksen toteutus

Pro graduni on teoreettisfilosofinen katsaus, jossa on osittain mentaalihistoriallinen tutkimustapa eli asioiden kehittymistä kuvataan henkilöiden ja ryhmien mentaliteettien kautta. Mentaliteetti on yleisen tulkinnan mukaan osittain tiedostettu, elämäntapaan ja maailmankuvaan liittyvä itsestäänselvyys, joka ohjaa ihmisten elämää ja valintoja. Se käännetään suomessa usein sanalla ”mielenlaatu”. Lähteinä olen käyttänyt kirjoja, artikkeleja ja internetissä julkaistuja tietoja.

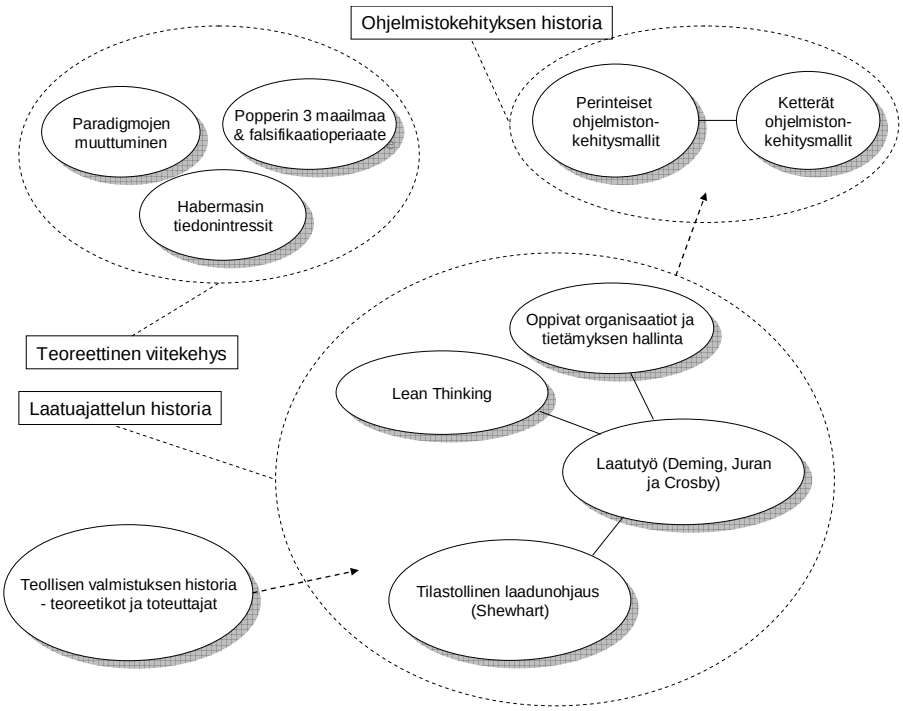
Keskeisiä kysymyksiä, joihin olen pyrkinyt löytämään vastauksia:

1. Millaisten tarinoita ohjelmistojen laadun suuri kertomus sisältää?
2. Millaisten vaiheiden kautta ohjelmistokehityksen nykytilanteeseen on tultu?
3. Mikä on se viitekehys/lähestymistapa, jonka pohjalta ohjelmistojen ja niiden suunnittelu- ja toteutusprosessin laatua tulisi parantaa?

Tutkimukseni keskeisiksi käsitteiksi ja teorioiksi olen valinnut seuraavat:

1. Kuhnin teoria paradigmojen muuttumisesta

2. Popperin kolme maailmaa ja falsifikaatioperiaate
3. Habermasin teoria tiedonintresseistä
4. Teollisen valmistuksen historia - teoreetikot ja toteuttajat
5. Tilastollinen laadunohjaus (Shewhart)
6. Laatutyö (Deming, Juran ja Crosby) ja prosessien kehittäminen
7. Oppivat organisaatiot ja tietämyksen hallinta
8. Perinteiset ja ketterät ohjelmistonkehitysmallit



Kuva 1. *Tutkielman keskeiset käsitteet ja teoriat.*

Yllä oleva käsitekuva esittää sitä kehityskulkua, miten teollisen valmistuksen historiasta on päädytty laatuajattelun historian kautta ohjelmistokehityksen historiaan ja nykytilanteeseen. Olen tarkastellut näitä historioita ja ohjelmistokehityksen nykytilannetta tutkielmani teoreettisen viitekehyksen kautta. Popperin väitteen mukaan tutkimuksen teoreettinen perspektiivi perustuu aina jokaisen tutkijan henkilökohtaiseen valintaan eikä sellaisenaan voi olla rationaalisen kritiikin kohteena. Kritiikki tulee hänen mukaansa kohdistaa sen sijaan aina tutkimuksen tuloksena syntyneitä tuloksia ja johtopäätöksiä kohtaan (Kragh 1989:56).

2. TEOREETTINEN VIITEKEHYS

2.1. Paradigmat ja niiden muuttuminen

Aiemmin tieteen kehityksestä oli vallalla käsitys, jonka mukaan uudet ”totuudet” rakentuivat vanhojen faktojen päälle. Uusi tietämys täydensi aina vanhaa tietämystä ja mitään ristiriitaa ei ollut koskaan muodostunut. Tiedon rakenteet olivat kunnossa ja tieteen varastoihin kertyi jatkuvasti lisää tietoa, joten tiede kehittyikin tasaisesti ja lineaarisesti.

Kuhn hylkäsi vuonna 1962 julkaistussa teoksessaan ”The Structure of Scientific Revolutions” tällaisen ajattelun ja esitti sen sijaan oman teoriansa siitä, miten tieteen kehitys etenee sykäyksittäin ja miten se sisältää usein epäjatkuvuuskohtia. Hänen mukaansa tapahtuu aika ajoin sellaisia suuria murroksia, ”vallankumouksia", joissa vallalla oleva tieteellinen ajattelu ja lähestymistapa (*paradigma*) kyseenalaistetaan, koska se ei pysty selittämään kaikkia tutkimuksessa tehtyjä havaintoja. Vähitellen löydetään enemmän ja enemmän paradigman kanssa ristiriitaisia anomalioita ja sitten jonkin ajan kuluttua koko vanha paradigma korvataankin uudella. Kun tieteen harjoittaja siirtyy uuden paradigman kannattajaksi, tapahtuu yksilön kohdalla kokemus (*Gestalt switch*), jonka jälkeen maailma näyttäytyy tarkkailijalleen systemaattisesti eri tavalla (Fuller 2003:21). Kuhnin mukaan tällaisessa tieteen vallankumouksessa tapahtuu kyseisen tieteenalan tiedossa ja teorioissa kaiken mullistava laadullinen muutos, paradigman vaihtuminen. Usein myös johtavat tieteen harjoittajat vaihtuvat, koska kaikki vanhan paradigman kannattajat eivät siirry uuden paradigman piiriin (Fuller 2003:19–20).

Kuhnin paradigma-käsitteellä voidaan ajatella olevan kaksi keskeistä merkitystä, joista ensimmäinen (1.) merkitys edustaa Kuhnin alkuperäistä, suppeampaa paradigman käsitettä ja jälkimmäinen (2.) merkitys on myöhemmin tehty laajennus, joka sisältää myös ensimmäisen:

1. konkreettiset malliesimerkit ("*exemplars*"), joita esitetään kyseistä paradigmaa noudattavan tieteenalan oppikirjoissa historiallisesti tärkeinä ongelmien ratkaisuina ja joiden kautta uudet tulokkaat perehdytetään tieteenalaan.
2. tieteenalan matriisi ("*disciplinary matrix*"), joka tarkoittaa sitä symbolisten yleis-tysten tai lakien, malliesimerkkien sekä yhteisten arvojen muodostamaa kokonai-suutta, joka vallitsee kyseistä paradigmaa noudattavassa tiedeyhteisössä.

Paradigma ja sitä noudattava tieteenala/tiedeyhteisö ovat tiukasti toisiinsa yhteydessä. Ne eivät voi olla olemassa ilman toisiaan. Tiedeyhteisö on ikään kuin paradigman omistaja. Tieteellisen totuuden määrittäminen jonain tiettynä ajanhetkenä ei voi Kuhnin mukaan tapahtua pelkästään objektiivisten kriteerien pohjalta vaan sen määrittää kyseisen tieteenalan yhteisö subjektiivisesti. Paradigma taas on yhteisön jatkuvan tutkimuksen ja kehittämisen kohde, ja ei koskaan ole lopullinen ja valmis, vaan sitä tarkennetaan jatkuvasti. Paradigmat ovat kuitenkin määrätyllä aikavälillä tarkasteltuina varsin pysyviä ja voivat siksi toimia muuttumattomina lähtökohtina tieteelliselle tutkimukselle.

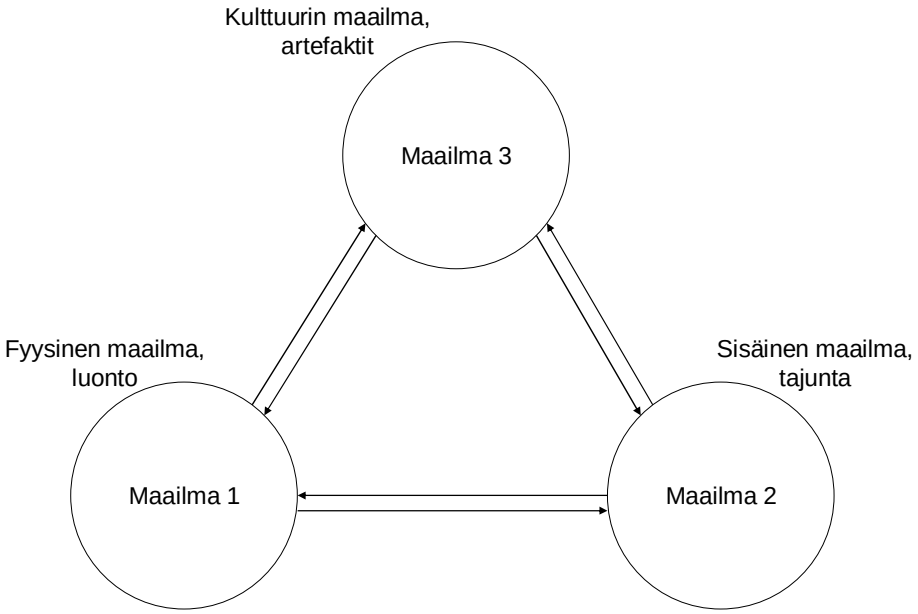
2.2. Popperin kolme maailmaa ja falsifikaatioperiaate

Popperin kolmen maailman teoria selittää olemassaolon todellisuutta kolme maailman ja niiden objektien kautta. Maailma tarkoittaa Popperin teoriassa oikeastaan samankaltaisten objektien muodostamaa joukkoa eikä suinkaan mitään rajattua paikkaa tai tilaa. Todellisuus muodostuu hänen mukaansa kolmen eri maailman objekteista ja niiden välisistä suhteista:

1. Maailma 1, joka sisältää fyysisen maailman objektit ja tapahtumat, jotka noudattavat fysiikan ja kemian lakeja. Näitä elementtejä ovat konkreettiset esineet, eläimet, kasvit, kivet, jne.(Niiniluoto 1990: 15)
2. Maailma 2, joka muodostuu elävien olentojen subjektiivisista kokemuksista. Tämän maailman elementtejä ovat tietoisuuden tilat ja mentaaliset objektit (Niiniluoto 1990: 17).

3. Maailma 3, joka on ihmisten ja yhteisöjen rakentama ja sisältää hänen toimintansa tulokset, artefaktit. Tällaisia elementtejä ovat tieteen ja taiteen tuotokset, rakennukset, koneet ja laitteet (Niiniluoto 1990: 19).

Popperin mukaan maailmassa 1 on olemassa useita eri tasoja alkaen alkeishiukkasista, atomeista ja molekyyleistä, aina nesteisiin, kiinteisiin kappaleisiin, orgaanisiin yhdisteisiin, viruksiin, yksi- ja monisoluisiin organismeihin sekä näiden populaatioihin. Popperin maailma 2 voidaan liittää vain riittävän kehittyneen keskushermoston omaaviin aineellisiin olentoihin eli ihmisiin ja eläimiin. Maailman 3 olioilla on oma historiansa ja ne ovat kiinni samassa ajassa kuin muidenkin maailmojen oliot. Maailmaa 3 ei voisi olla olemassa ilman maailmoja 1 ja 2, eikä maailmaa 2 voisi olla olemassa ilman maailmaa 1. Maailmojen väliset suhteet on mahdollista ajatella emergentin materialismin mukaan, jolloin maailma 1 edustaa materialistista maailmaa ja maailmat 2 ja 3 ovat vastaavasti sen evolutionaarisia tuotteita (Niiniluoto 1990: 20).



Kuva 2. *Popperin kolme maailmaa (Niiniluoto 1990: 23)*

Esimerkkinä kolmen eri maailman objektista Popper tuo esiin kirjallisen teoksen (Shakespeare’s Works, volume one). Kyseinen teos on ihmisen kirjoittama ja kuuluu

siten artefaktina selkeästi maailmaan 3, mutta fyysisenä kappaleena se kuuluu myös maailmaan 1. Kun ihminen pitää kädessään fyysistä kirjaa ja lukee sitä, muodostuu siitä samalla kuva hänen mieleensä. Tuo mieleen syntynyt fyysisen objektin kuva kuuluu silloin maailmaan 2 (Niiniluoto 1990: 21).

Teoksessaan ”Logik der Forschung” Popper arvosteli loogisen empirismin käsitystä tieteestä induktiivisena toimintana. Popper kannatti itse hypoteettis-deduktiivista tiedonhankintamallia, jonka mukaan tieteelliset teoriat ovat hypoteeseja eli oletuksia, joista johdetaan uusia ennusteita. Popperin mukaan tieteellisiä selitysjärjestelmiä on mahdotonta todistaa aukottomasti oikeiksi, koska induktiivisen päättelyn eli yksittäisistä tapauksista yleistävään etenevän päättelyn pohjalta voidaan päätellä itse asiassa vain se, että tähän mennessä ei ole tullut vastaan yksittäistapausta, joka sotisi vastaan voimassaolevaa selitystä tai teoriaa. Popperin lanseeraaman falsifikaatioperiaatteen mukaisesti on teorian vääräksi todistaminen kuitenkin aina mahdollista. Tieteellisestä teoriasta on oltava mahdollista johtaa ennuste tai seuraamus, jota sitten kokeellisesti testataan. Tieteellisten kokeiden avulla on täten mahdollista löytää ja tuoda esiin alkuperäisen teorian heikkoja kohtia ja virheellisyyksiä (Thornton 2009).

2.3. Habermasin tiedonintressiteoria

Habermasin tiedonintressiteorian perusväittämä lähtee siitä, että ei ole olemassa mitään intresseistä vapaata tietoa. Pyrkimys tiedon arvovapauteen katkaisee hänen mukaansa tärkeän yhteyden puhtaan teorian ja todellisen elämän väliltä. Habermasin mukaan tieto on aina riippuvainen historiallisista ja ideologisista seikoista. Tällainen intressi (tavoite, tarve, vaatimus) ei kuitenkaan ole mikään este tiedolle vaan muodostaa hänen mukaansa sillan pätevään tietoon. Kun tiedonintressi tunnetaan ja avoimesti tunnustetaan, tiedetään myös miten on mahdollista päästä kiinni pätevään tietoon eri tieteen alueilla. Toisin sanoen näiden intressien tiedostamisen kautta tieteiden ja tutkimuksen autonomia lisääntyy ja toimiva yhteys tietoteoreettisen subjektin ja objektin välillä on mahdollinen. (Bohman & Rehg 2011).

Habermas perustelee tiedon ja tiedon intressien suhteen toiminnalla, sillä hänen mukaansa tieto ja tietäminen liittyvät kiinteästi historiallisen ihmisen universaaliin pyrkimykseen luoda oma olemassaolonsa sekä uudistaa itseään lajina. Tarpeet tekniseen kontrolliin, praktiseen ymmärrykseen sekä emansipaatioon (pakosta vapautumiseen) kohosivat esiin luonnon oloista ja vaatimuksista. Nämä tarpeet määrittävät ne näkökulmat, joista käsin voimme käsittää todellisuuden sinänsä (*an Sich*). Habermasin teoria käsittää seuraavat tiedonintressit:

- **Tekninen tiedonintressi.** Ihmisten täytyy tuottaa välineellisen toiminnan avulla luonnosta kaikki mitä tarvitaan materiaalisen elämän turvaamiseen. Yhteiskunta tarvitsee teknistä tietoa manipuloidakseen luonnon objekteja kyseistä tarkoitusta varten. Tekninen intressi on läsnä vahvimmin empiiris-analyyttisissä tieteissä eli tyypillisesti luonnontieteissä. Nämä tieteet hankkivat tietonsa havaintojen kautta mahdollisimman objektiivisesti.
- **Praktinen tiedonintressi**. Yhteiskunta ei Habermasin mielestä voi olla pelkästään yksilön olemassaoloa tukeva ja suojeleva järjestelmä. Tekninen tiedon rinnalle tarvitaan praktista tietoa, joka määrittää miten tätä elämää eletään yhteisöllisesti. Praktista tiedonintressiä harjoittavat lähinnä historiallis-hermeneuttiset tieteet, jotka eivät hanki tietojaan empiiristen havaintojen kautta vaan kulttuurillisten merkitysten tulkinnan ja ymmärtämisen kautta.
- **Emansipatorinen tiedonintressi.** Tämä intressi liittyy ihmisten vapauttamiseen turhista perinteistä, yksisuuntaisesta ajattelusta ja olemisesta. Se koskee myös ihmisen vapautumista yhteiskunnallisesta pakotuksesta ja muiden rajoituksista. Emansipatorinen intressi on tyypillistä sekä kriittisille tieteille että filosofialle. Sillä ei ole selvää kohdealuetta, vaan se liittyy kaikkeen inhimilliseen ajatteluun ja tekemiseen ja sen päämääränä on vapautunut yhteiskunta. Tällainen yhteiskunta mahdollistaa jäsentensä kasvamisen kypsemmiksi ihmisiksi sekä sallii vapaan ja suoran kommunikaation kaikkien ihmisten kesken.

Habermasin mielestä nämä tiedonintressit muotoutuvat pääasiassa kolmen eri yhteiskunnallistamisen mekanismin eli työn, kielen ja vallan kautta. Tekninen intressi muotoutuu työn eli instrumentaalisen toiminnan kautta, praktinen kielen eli

kommunikaation kautta ja emansipatorinen vallan eli ulkoisen pakotuksen (*herruuden*) kautta (Bohman & Rehg 2011). Habermasin teoria tiedonintresseistä on myös vahvasti vaikuttanut pohjoismaisiin tutkijoiden, erityisesti Dahlbomin ja Mathiassenin (Dahlbom and Mathiassen 1997: 80–81) tutkimustyöhön heidän pyrkiessään luomaan poikkitieteellistä viitekehystä systeemisuunnitteluun kirjassaan ”Computers in Context: The Philosophy and Practice of System Design”.

Alla olevaan taulukkoon on koottu yhteenvedonomaisesti näitä tiedonintressejä vastaavat yhteiskunnallistamisen mekanismit, tavoitteet, tieteen alueet, maailmankäsitykset sekä näiden suuntausten vaikuttavimmat klassiset edustajat historiasta.

Taulukko 1. Habermasin tiedonintressit, niiden pääasialliset mekanismit, tieteen alueet, maailmankäsitykset ja klassiset edustajat.

Tiedonintressi	**Meka-nismi**	**Tavoite**	**Tieteen alue**	**Maailman-käsitys**	**Edustajat (klassiset)**
tekninen	työ	hallinta	luonnontieteet	mekanistinen	Newton, Leibniz
praktinen	kieli	ymmärtäminen	humaaniset ja erityisesti hermeneuttiset tieteet	romanttinen	Dilthey, Heidegger
emansipatori-nen	valta	vapautuminen	kriittiset yhteiskunta-tieteet	kriittinen teoria, radikaali	Marx, Weber

3. TEOLLISEN VALMISTUKSEN HISTORIA

Tämä luku käsittelee seuraavia aiheita: työn muuttumista kotiteollisuudesta tehdastuotantoon, työn jakamisen ja teollisen valmistamisen teoriaa sekä tieteellistä työnjohtoa.

3.1. Adam Smith ja kansakuntien vauraus

Smithin mukaan työn jakaminen osiin ja erikoistuminen johti halvempaan työn hintaan ja halvempiin tuotteiden hintoihin. Halvemmat hinnat johtavat sitten markkinoiden kasvuun, koska yhä useammilla ihmisillä on varaa ostaa halvempihintaisia tuotteita. Smith sai tämän ajatuksensa Ranskassa ollessaan Francois Quesnaylta, joka oli ranskalainen ekonomisti. Hän tuli kuuluisaksi julkaistuaan vuonna 1758 teoksensa ”Tableau economique”. Teosta pidetään yhtenä ensimmäisenä kokonaistalouden toimintojen analyysinä ja tärkeänä lisänä taloudellisen ajattelun kehittymiseen. Fysiokratismi eli fysiokratia oli 1700-luvulla Ranskassa syntynyt talousteoria, jonka keskeinen ajatus oli maatalouden merkityksen korostaminen hyvinvoinnin alkulähteenä. Fysiokratismin mukaan tuottamatonta työtä piti aina välttää (Francois Desnay).

Smith vei tämän ajattelun pitemmälle ja ehdotti, että tuottavan työn kannattavuutta pitäisi parantaa entisestään jakamalla työ pienempiin osiin. Työn jakaminen kilpailun vallitessa johtaa halvempiin hintoihin ja laajentaa näin ollen markkinoita. Laajemmat markkinat ja kasvanut tuotanto mahdollistavat tuotannon uudelleen organisoinnin ja uusien tuotantotapojen keksimisen, mikä taas madaltaa hintoja entisestään, jne. Smithin keskeinen sanoma oli, että tällainen kasvu varmistaa kansakuntien varallisuuden. “*Jokaisen kansakunnan vuotuinen työmäärä on se perusta, joka alkuaan toimittaa sille kaikki ne elämän välttämättömyydet ja mukavuudet, jotka se vuoden aikana kuluttaa.”*

Työn jakaminen on aiheuttanut suuremman lisäyksen kansakuntien tuotannossa kuin mikään muu tekijä. Tällainen monipuolistuminen ja erilaisiin töihin erikoistuminen on ollut Smithin mukaan suurinta sellaisissa kansakunnissa, joissa on ollut enemmän teollisuutta, ja se vastuussa ”yleisestä runsaudesta” noissa kansakunnissa.

Maataloudessa ei työn jakaminen ole onnistunut niin hyvin kuin teollisuudessa. Näin ollen Smithin tarkoittamat rikkaat valtiot eivät suinkaan olleet niin paljon edellä köyhiä valtioita maataloudessa kuin teollisuudessa. Työn jakaminen ei ole syntynyt minkään viisauden hedelmänä, vaan ihmisen taipumuksesta ja halusta vaihtamiseen ja vaihteluun. Smithin mukaan ihmisten välillä esiintyvät luonnolliset lahjakkuuserot eivät suinkaan ole olleet erikoistumisen syy, vaan ovat tämän erikoistumiskehityksen tulos.

Valmistustyön jakaminen pienempiin osiin alkaa aina työprosessin analyysillä. Tämän analyysin avulla saadaan selville työn olennaiset osat, joiden kautta itse tavara, tuote valmistetaan. Jos käsityöläiseltä on tilattu esimerkiksi suurempi määrä jotain tavaraa, on hänen tietenkin tehokkaampaa ja nopeampaa valmistaa ne sarjana eri vaiheiden kautta, jolloin työkalujen ja materiaalien valmisteluun menevä aika yhtä tavarayksikköä pienenee. On huomattava, että työn jakaminen osiin ei vielä muuta käsityöammattilaista vaihetyöntekijäksi, mutta se on kuitenkin välttämätön edellytys tälle muutokselle

Adam Smith kuvaa kuuluisassa nuppineulan valmistusesimerkissään sitä, miten nuppineulan valmistaminen koostuu 18 eri operaatiosta. Joissain tehtaissa on nämä 18 eri valmistusvaihetta jaettu jokainen eri työntekijän tehtäväksi, kun taas joissain yksi työntekijä saattaa hoitaa 2-3 vaihetta. Näin ollen on joissain tapauksissa työn jakaminen viety vieläkin pitemmälle kuin alkuperäisessä esimerkissä peltisepän sarjatyöstä. Alkuaanhan yksi ja sama työntekijä teki kaikki työn osat, mutta äärimmilleen vietynä tekee jokainen työntekijä vain yhtä työvaihetta eikä sitten pitemmällä tähtäimellä osaakaan enää valmistaa varsinaista tuotetta. Näin sen vuoksi että hänen ammattitaitonsa kehittynyt vain sille tasolle, joka juuri ja juuri riittää hänen oman työvaiheensa suorittamiseen (Braverman 1977: 74–75). Vastaavanlaista kehitystä on ollut nähtävissä ohjelmistotuotannossakin: työn ja osaamisen fragmentoituminen johtaa helposti kapea-alaisuuteen ja vie vähitellen työntekijältä ammattiylpeyden ja työn ilon.

3.2. Charles Babbage ja koneellisen valmistuksen taloudesta

Andrew Ure ja Charles Babbage seurasivat Smithiä ja kirjoittivat 1830 - luvulla omat management-kirjansa. ”Management” käsitteenä on peräisin latinan sanasta ”manus”, joka tarkoittaa kättä. Termiä käytettiin alkuaan tarkoittamaan hevosen käsittelyä ja kouluttamista erilaisten liikkeiden ja temppujen suorittamiseen kentällä tai maneesissa eli ranskaksi ”manège” (Braverman 1977: 68–69). Management – käsitteen historia kertoo hyvin siitä ilmapiiristä, mikä 1800 - luvun tehtaissa vallitsi. Samalla tavalla kuin ratsastaja kouluttaa ja ohjaa hevostaan käyttämällä erilaisia välineitä ja avuja saadakseen hevosen suostumaan tekemään temppuja ja liikkeitä, niin vastaavasti pyrkii myös tehtaanomistaja käyttämään kaikkia keinoja saadakseen vallan ja kontrollin työväkeensä.

Bravermanin mukaan kontrolli ja hallinta muodostavat juuri sen keskeisen elementin eri yritysjärjestelmissä, jonka kaikki liikkeenjohtamisen tutkijat tunnustavat ja tunnistavat. Keskiaikaisessa käsityöverstaassa kontrolli pohjautui siihen kuuliaisuuteen, jota sen aikaisen näkemyksen mukaan kisällien ja oppipoikien tuli aina osoittaa mestarilleen, jonka palveluksessa he olivat. Uudessa tehdasympäristössä tämä kontrolli oli olennaisesti muuttunut ja perustui työnjohdon ja työntekijöiden väliseen vastakkainasetteluun. Kun ennen työntekijä oli ollut osallinen ja riippuvainen oman ”kotiyhtiönsä” menestyksestä, niin tehdasmaisessa tuotannossa ja uudessa työjärjestyksessä hän oli kiinnostunut tehtaanomistajan ja tehdasyhtiön menestyksestä vain siltä osin kuin se vaikutti hänen toimentuloonsa.

Teoksessaan ”On the Economy of Machines and Manufactures” vuodelta 1832 Babbage esittää oman teoriansa työn jakamisesta. Hän todistelee sitä, että kun työ jaetaan pienempiin osiin, joista jotkut osat ovat yksinkertaisempia kuin toiset, mutta jotka kaikki ovat yksinkertaisempia kuin kokonaisuus, tulee kokonaisuuden teettäminen halvemmaksi kuin aiemmin, koska voidaan käyttää entistä halvempia työntekijöitä. Näin ollen työ on jaettava pienimpiin mahdollisiin osiin, jotta sen teettäminen tulisi mahdollisimman halvaksi (Braverman 1977: 77).

Ruumiillisen työn jakamisen periaatteita voitiin hänen mukaansa soveltaa myös henkiseen ajattelutyöhön. Babbage kertoo kirjassaan tästä esimerkin vallankumousajan Ranskasta vuodelta 1793. Babbage oli tehnyt vuonna 1819 matkan Pariisiin ja tavannut tällöin ranskalaisia matemaatikoita, joiden kautta hän oli tutustunut ranskalaisiin logaritmi- ja trigonometriataulukoihin (Braverman 1977: 275). Ranskalainen insinööri ja matemaatikko de Prony toimi École Nationale des Ponts et Chaussées'n (*national school of roads and bridges*) pääinsinöörinä. Vuonna 1793 de Prony sai Ranskan kansalliskokoukselta tehtäväkseen uudistaa kaikki maan logaritmi- ja trigonometriset taulukot. Näiden taulukoiden uudelleenlaskeminen oli valtava urakka ja De Prony tajusikin varsin nopeasti, että hänellä ei ollut mitään reaalisia mahdollisuuksia onnistua tehtävässä noudattamalla perinteisiä työtapoja ja - menetelmiä. Erään kirjakaupan ohi kulkiessaan hän huomasi sen näyteikkunassa Smithin teoksen "Wealth of Nations" ja kiinnostui siitä suuresti. De Prony ymmärsi aika nopeasti, että kirjassa kerrottua esimerkkiä työn jakamisesta nuppineulojen valmistuksessa voitaisiin soveltaa myös matemaattisten taulukoiden laskentaan (Braverman 1977: 276).

Hän jakoi laskentatyön kolmelle eri työntekijäryhmälle. Hierarkian huipulle palkattiin 5-6 ranskalaista huippumatemaatikkoa, jotka loivat taulukoiden laskentaan tarvittavat kaavat, joiden perusteella taas kaksi alempaa ryhmää toteuttivat sitten itse laskennan. Ensimmäinen ryhmä ei tehnyt käytännön numeerisia laskentatehtäviä lainkaan. Toinen ryhmä koostui alle kymmenestä henkilöstä, joilla oli hyvät matemaattiset taidot ja joiden tehtävänä oli muuntaa ensimmäisen ryhmän kaavat numeerisiksi arvoiksi ja kehittää varsinaisten laskujen tarkistukseen tarvittavat menetelmät. Kolmanteen ryhmään, jonka koko vaihteli hankkeen aikana 60–80 henkilön välillä, kuului henkilöitä, jotka suorittivat yksinkertaisia yhteen - ja vähennyslaskuja.

Kolmannen ryhmän laskemat tulokset annettiin toisen ryhmän tarkistettavaksi eli kolmannen ryhmän työ oli näin ollen puhtaasti rutiininomaista mekaanista laskemista. Myös toisen ryhmän työhön kuului käytännön laskutehtäviä, mutta se oli kuitenkin mielenkiintoisempaa, sillä työ sisälsi vaihtelua ja uusien haastellisten asioiden kehittämistä. Työn aikana huomattiin, että vaikka kolmannen ryhmän työntekijöistä ei juuri ollut minkäänlaista matematiikan taitoa (pois lukien yhteen - ja

vähennyslaskutaito) he tekivät vähemmän virheitä kuin muut ryhmät, joiden matematiikan osaaminen oli parempaa. Babbagen mukaan oli selvää, että osaavia ja taitavia henkilöitä ei pidä panna tekemään sellaisia yksinkertaisia rutiinitehtäviä, jotka voitiin teettää halvemmalla ja vähemmän ammattitaitoisilla henkilöillä. Hänen mielestään tällaisilla ihmisillä ei ollut päässään myöskään niin paljon häiritseviä ajatuksia, joten he keskittyivät työhönsä paremmin (Braverman 1977: 276–277). Joillain nykyajan esimiehilläkin tuntuu olevan samansuuntaisia ajatuksia – heidän mukaansa yliopistosta valmistuneet eivät esimerkiksi yleensä sovellu ohjelmointityöhön, koska he ajattelevat asioita liian monimutkaisesti.

Babbage uskoi myös, että tulevaisuudessa kolmannen ryhmän työt pystyttäisiin suorittamaan laskukoneella, jonka jälkeen voitaisiin puolestaan siirtyä yksinkertaistamaan ja jakamaan myös toisen ryhmän työtä yhä pienempiin osiin. Tulevaisuuden yhteiskunnassa olisi mahdollista muuttaa koko päätöksenteko mekaaniseksi prosessiksi, jota ohjaisivat ja valvoisivat ensimmäisen ryhmän kaltaiset miehet. Tuo ryhmä olisi silloin ainoa, jonka tarvitsee ymmärtää asioita ja tapahtumia syvällisesti tieteellisellä tasolla. Muiden ryhmien työ koostuisi tällöin vain datan käsittelystä ja koneellisista operaatioista (Braverman 1977: 277–278).

3.3. Andrew Ure ja teollisen valmistamisen filosofia

Andrew Ure rakensi teollisen valmistamisen teoreettista perustaa teoksessaan ”The Philosophy of the Manufacturers”, vuodelta 1835. Ure oli Glasgown yliopiston professori, joka ihaili suuresti uutta teollista valmistusjärjestelmää. Hänen ajatuksensa edustivat uutta omistavaa yhteiskuntaluokkaa, joka oli saanut varallisuutensa tehtaistaan työntekijöiden tekemän työn tulosten kautta. Ure uskoi, että teollisen valmistamisen hyödyt merkitsivät Britannialle suurta vaurastumisen ja hyvinvoinnin aikaa. Teoksessaan hän vertaili koneiden ja ihmisen tekemää mekaanista työtä ja uskoi fysiomekaanisen tieteen tuovan helpotusta ihmisten päivittäiseen elämään. Kun työntekijä valmistaisi enemmän tuotteita ja saisi siten myös enemmän palkkaa, niin

hyvinvointi leviäisi koko yhteiskuntaan. Ure oli ajatuksiltaan idealisti ja vaikuttaa uskoneen vilpittömästi ihmisten hyvyyteen, vaikka todellisuus ei hänen ympärillään kauniina ja oikeudenmukaisena näyttäytynytkään (Halsall 2006).

Höyrykoneen vastustajat väittivät, että höyrykone pyörittää tehtaan koneita (esimerkissä kangaspuita) sellaisella vauhdilla, että ihmiset menevät sekaisin tuosta pakkotahtisesta kiireestä. Uren mukaan koko tehtaan koneisto voitiin rakentaa ja säätää sellaiseksi, että työntekijöiden ei tarvitse suorittaa juuri ollenkaan lihastyötä. Näin työntekijöille suodaan mahdollisuus hyviin ja alenemattomiin palkkoihin sen lisäksi, että tehdasympäristö edisti monin tavoin heidän terveyttään. Kotona tehdyssä työssä jokainen työntekijä oli kuitenkin oma herransa, joka myi työnsä tulokset edelleen. Tehtaassa työntekijä oli koko ajan tehtaan johtajan sekä hänen edustajiensa, työnjohtajien, valvonnan ja käskyjen alainen. Niiden työntekijöiden, jotka eivät työskennelleet tehtaassa, oli tehtävä kaikki lihasvoimallaan, he kokivat työnsä rasittavaksi, heidän oli väsyttyään pakko pitää taukoja, jotka jokainen yksin laskettuina olivat lyhyitä, mutta yhteenlaskettuina muodostivat suuren osan työajasta. Näin ollen he eivät pystyneet saamaan samaa palkkaa kuin tehdastyöntekijät, heillä ei ollut varaa riittävän hyvään ruokaan, huonomman ruoan vuoksi heidän terveytensä heikkenee, jne.” Ure kuvasikin teoksessaan tarkasti tätä kurjistumisen mekanismia, jonka hän uskoi olevan estettävissä korvaamalla ihmisen kotona käsin tekemä valmistustyö tehtaissa koneiden avulla tapahtuvalla työllä. Ure näkikin tehdastyön tieteellisten menetelmien avulla tapahtuvan muuttamisen olevan selkeästi filantrooppista toimintaa (Halsall 2006).

Käsityön avulla tavaroita valmistettaessa on ihmistyövoima yleensä kallein tuotantotekijä. Kun tavaroita ryhdytään valmistamaan koneellisesti, ammattitaitoisen työvoiman tarve vähenee ja se työvoima, joka on tarpeen, koostuu pääasiassa koneiden toimintaa valvovista ja niitä ylläpitävistä halvemman kategorian työntekijöistä. Kun valmistuksen tieltä poistettaisiin lihastyön rajoitukset, voitaisiin tuotantonopeutta ja – määrää nostaa moninkertaiseksi. Ure näki ammattitaitoisen työntekijän ajattelukyvyn ja vapaan tahdon olevan kuitenkin uhka tälle myönteiselle kehitykselle. Ammattitaitoinen ihminen ei hänen mukaansa soveltunut mekaanisen järjestelmän osaksi ja siksi modernin tehtaanomistajan suurin tavoite olisikin vähentää tällaisten ihmisten tarvetta

yhdistämällä tiede ja pääoma toisiinsa valmistusprosessin kehittämiseksi. Monitaitoisista ammattimiehistä voitaisiin siirtyä samalla yhden asian osaaviin vaihetyöläisiin. Näille vaihetyöläisille ei tarvitsisi maksaa ammattimiehen korkeaa palkkaa ja jos mahdollista, myös miespuoliset vaihetyöläiset olisi kehityksen kulkiessa eteenpäin korvattava, jos mahdollista, naispuolisilla vaihetyöntekijöillä ja lapsilla. Kun voitiin tuottaa uusia ja enemmän tuotteita halvemmalla, oli mahdollista myydä niitä yhä suuremmalle joukolle ihmisiä. Tämän vuoksi sekä markkinat että talous kasvoivat nopeasti (Adam Smith oli esittänyt saman seuraussuhteen jo aiemmin omissa teoksissaan).

Höyrykoneiden kehitys merkitsi vapautumista ulkoisten voiman lähteiden asettamista rajoituksista. Hiilen ja muiden polttoaineiden tarve kasvoi höyrykoneiden yleistymisen myötä. Kanavien ja rautateiden rakentaminen lisääntyi, sillä höyryn avulla kulkevat uudenlaiset kulkuneuvot tarvitsivat puolestaan omat tiensä, joilla kulkea ja kuljettaa kuormiaan. Tällainen kehitys johti osaltaan uusien ammattikuntien syntymiseen: kaivostyöläiset, insinöörit, koneenkäyttäjät ja mekaanikot ilmestyivät uuden koneellisen ja mekaanisen ajan työn suorittajiksi. Kokonaisuutena arvioiden työvoiman tarve koneellistuvissa ja teollistuvissa yhteiskunnissa kasvoi nopeammin kuin koskaan aiemmin historiassa (Halsall 2006). Vastaavalla tavalla näytti tapahtuvan IT-alallakin aluksi. Vanhojen tehtävien ja toimintojen automatisointi vapautti ihmisiä uuden teknologian kehitys- ja ylläpitotyöhön. Aluksi näytti, että näitä uusia työtehtäviä riitti kaikille halukkaille, mutta nykyään ollaan jo tilanteessa, jossa voidaan avoimesti tunnustaa, että myös henkisen työn automatisointi synnyttää työttömyyttä.

3.4. Taylorin ”tieteellinen” työnjohto

Fredrick W. Taylorin teos ”The Principles of Scientific Management” julkaistiin vuonna 1911. Koko Scientific Management - liikkeen lähtökohtana oli Taylorin oma konservatiivinen käsitys työntekijöistä. Hän näki henkilöstön lähinnä vastahakoisena ja vaikeasti hallittavana tuotantotekijänä teollisessa tuotantoympäristössä. Taylor ei millään lailla tutkinut eikä selvittänyt syitä tai taustoja tähän tilanteeseen, vaan otti sen

mukaan ”luonnollisena” ja muuttumattomana alkuehtona tutkimukselleen samoin kuin hän otti myös työkalut ja teknologian tuotannon muuttumattomina tekijöinä. Taylor kokosi yhteen ja esitti selkeästi ne ajatukset, jotka olivat syntyneet ja kasvaneet 1800-luvun teollisuuspiireissä Yhdysvalloissa ja Isossa Britanniassa (Braverman 1977: 84). Hän kertoi itse teoksensa alussa ne kolme pääsyytä, miksi hän halusi kirjoittaa kirjansa:

1. osoittaa kuinka suuria tappioita koko maa kärsii, koska tehottomuus ja tuhlailu leimaavat lähes kaikkia ihmisten päivittäisiä tekemisiä.
2. vakuuttaa lukijansa siitä, että parannuskeino tähän tilanteeseen on pikemminkin järjestelmällinen johtaminen kuin epätavallisten ja poikkeuksellisten ihmisten etsiminen.
3. todistaa, että paras johtaminen on tiedettä, joka perustuu selkeästi määriteltyihin lakeihin, sääntöihin ja periaatteisiin.

Taylor tuli Yhdysvalloissa alkuaan tunnetuksi eräästä hänen ensimmäisistä työkokeiluistaan Bethlehem Steel Companyssa. Kokeilu tapahtui vuonna 1892, samaan aikaan kun Yhdysvallat oli aloittamassa sotaansa Espanjaa vastaan. Sodan puhkeamisen seurauksena oli metallin hintojen nopea nousu ja Bethlehem Steel Company myikin nopeasti koko 80000 t takkirautavarastonsa. Tämä takkirautamäärä piti myynnin jälkeen siirtää junanvaunuihin mahdollisimman nopeasti toimitettavaksi edelleen ostajille. Työ oli suunniteltu tehtäväksi 75 miehen voimin ja Taylor halusi nostaa silloisen päivänormin alkuarvosta 12,5 t/mies aina arvoon 47 t/mies. Taylor suunnitteli kaiken yksityiskohtaisesti etukäteen ja määräsi jokaiselle työvaiheelle ohjeajat, joiden toteutumista valvottiin tarkasti sekuntikellolla. Kokeessa mukana ollut työmies suoriutui päivän työstään suunnitellusti ja työtahtinsa tasaisena säilyttäen. Tämän kokeilun perusteella Taylor katsoi osoittaneensa ”tieteellisesti”, että päivänormi pitäisi muuttaa 47 tonniin. Työmiehen päiväpalkkaa nostettiin 60 % ja Taylorin kertomuksen mukaan hän oli tyytyväinen siihen. Tuottavuus nousi samaan aikaan kuitenkin yli 270 %, joten työnantajalle aiheutunut yksikkökustannus laski huomattavasti (Braverman 1977: 95–97).

Taylorin teoksen vaikutus länsimaiseen ajatteluun ja liikkeenjohtoon on ollut perustavaa laatua. Peter F. Drucker (Drucker 1981: 14) on sanonut, että vaikkakaan scientific

management ei ollut mikään yhtenäinen työn ja työn tekemisen filosofia tai malli, niin sen käytännön merkitys on kuitenkin muodostunut suureksi: se oli hänen mukaansa lähtökohta koko tuottavuuden ja tuotannon kehittämisajattelulle. George Soulen tunnetun väitteen mukaan scientific management liikkeenä hävisi 1930-luvun suuren laman myötä. Siihen mennessä scientific management tunnettiin kuitenkin jo kaikkialla Yhdysvalloissa ja sen osaaminen oli yleistynyt teollisuudessa niin laajasti, että sen menetelmiä ja käytäntöjä opetettiin jo monissa teknillisissä korkeakouluissa ja kauppakorkeakouluissa. Soulen sanoja mukaan siitä tuli samalla tavalla epämoderni ja unohdettu kuin pienestä uskonlahkosta tulee pieni ja unohdettu sen kasvaessa vähitellen koko suuren enemmistön uskonnoksi (Braverman 1977: 83).

Taylor ei huolimatta omista puheistaan koskaan tehnyt mitään varsinaista tieteellistä johtamistutkimusta, mutta kirjoitti selkeän oppaan toisten työn johtamisesta. Taylorin tausta teknikkona ja pitkä kokemus Midvale Steel Worksilla antoivat hänelle erittäin hyvät tiedot ja taidot verstastyön yksityiskohdista. Scientific managementin ruotsalainen käännös ”*rationell arbetsledning*” kuvaakin käsitettä hyvin. Kontrolli eli ohjaus ja hallinta on Taylorin mukaan pidettävä aina toimivan johdon vastuulla. Työntekijöiden valta päättää oman työnsä yksityiskohdista on otettava heiltä pois, jotta saavutetaan paras mahdollinen tulos heidän tekemästään työstä. Näin ollen työn johtaminen ja suorittaminen oli erotettava toisistaan. Vastaavaa ajattelua on ollut havaittavissa silloin, kun suomalaista ohjelmistotyötä ulkoistetaan halvemman työvoiman maihin. Työn suunnittelu ja toteuttaminen erotetaan tällöin toisistaan, ei vain organisatorisesti vaan myös maantieteellisesti.

Taylor esitti teoksessaan scientific managementille kolme periaatetta:

1. Taylorin 1. periaatteen mukaan yritysjohdon tulee koota kaikki ne perityt tiedot ja taidot, jotka tähän mennessä ovat olleet työntekijän omaisuutta ja luokitella ne sekä muodostaa niistä taulukko, jonka avulla sitten tämän tietämyksen perusteella voidaan määritellä säännöt, lainalaisuudet ja kaavat päivittäisen työn suorittamiselle.

2. Taylorin 2. periaatteen mukaan kaikki se ajattelutyö, joka koskee työtehtävien suorittamista, on niin pitkälti kuin mahdollista siirrettävä pois verstaasta keskitetylle suunnitteluosastolle.
3. Taylorin 3. periaate esittelee käsitteen etukäteen suunnitellusta ”pensumista”. Tällainen pensum (työsuunnitelma, työkortti) kertoo aina sen, mitä työvaiheessa tehdään, miten kyseinen työvaihe tehdään ja kuinka kauan sen tekeminen saa kestää.

Yritysjohdon rooli muuttui scientific managementin myötä selkeästi kaksijakoiseksi. Ensinnäkin yritysjohdon tuli tehdä uuden tuotantotavan tiedostamattomat piirteet ja tavoitteet tiedostetuiksi ja systemaattisiksi. Toiseksi sen tuli myös huolehtia siitä, että kun työntekijöiden ammattitaito ajan myötä pieneni, niin työntekijöistä saatiin muodostettua mahdollisimman helposti muokattavissa olevaa työvoimaa eli heistä tuli yksi hallittava tuotantotekijä. Johto yksinään piti hallussaan ja käytti niitä vallan välineitä, joita tieteen ja teknologian kehitys heille antoi (Braverman 1977: 111).

Taylorilla oli taipumus nostaa päivittäinen työmäärä niin korkeaksi, että ainoastaan pieni osa kaikista työntekijöistä pystyi sen saavuttamaan. Tavoitteekseen hän asetti saada irti maksimaalinen tai ”optimaalinen” tulos työntekijöiden päivän työstä. Suurin este sen saavuttamiselle oli Taylorin käsityksen mukaan laiskottelevien ja huijaavien työläisten hidas työtahti. Hän jakoi vilpin kahteen eri kategoriaan: luonnolliseen ja systemaattiseen. Luonnollinen vilppi aiheutuu Taylorin mukaan ihmisen myötäsyntyisestä laiskuudesta, pyrkimyksestä ottaa rauhallisesti työpaikallaan. Systemaattinen vilppi aiheutuu taas joidenkin ihmisten taipumuksesta pyrkiä päästä hyötymään muiden ihmisten kustannuksella. Jälkimmäinen vilppi syntyy tietoisen harkinnan seurauksena ja oli Taylorin mukaan huomattavasti tuomittavampaa ja vahingollisempaa (Braverman 1977: 91). Jos Taylor olisi tuntenut Shewhartin teorian prosessien vaihtelusta, hän olisi ymmärtänyt, että systemaattinen vilppi johtui erikoissyistä ja että niitä olisi ollut mahdollista vähentää.

Taloustieteilijä Leisersonin mukaan Taylor osoitti suurta kaksinaismoraalia moittiessaan työntekijöitä laiskoiksi heidän käyttäytyessään samojen periaatteiden mukaan kuin

työnantajat – eri säännöt olisivat näin ollen voimassa työnantajille ja työntekijöille! Sama järkevä menettely, joka oli hyväksyttävää ja taloudellista työnantajille, olikin moitittavaa ja itsekästä käyttäytymistä työntekijöiden ollessa kyseessä. Esimerkkinä tällaisesta Leiserson otti työnantajien normaalin menettelyn supistaa tuotantoaan hintojen ollessa alhaalla ja alentaa työntekijöille maksettavia palkkoja työvoiman tarjonnan ollessa suurta. Vastaavasti työntekijät voisivat tehdä työtään hitaammin silloin, kun työstä maksettu tuntipalkka oli alhaalla ja kieltäytyä työskentelemästä työnantajan tarjoamalla palkalla työvoimapulan vallitessa (Braverman 1977: 93).

Toinen viime vuosisadan alun merkittävä työn- ja liikkeenjohdon kehittäjä ja teknistieteellisen vallankumouksen puolestapuhuja, Henry Laurence Gantt toimi yhdessä Taylorin kanssa vuosina 1887–1893. Gantt kehitti myöhemmin kuuluisaksi tulleen ja myös ohjelmistotyössä toimivaksi koetun Gantt - kaavionsa osittain näiden kokemusten perusteella 1910–luvulla. (Henry Gantt).

3.5. Teollisen tuotannon organisoinnista

3.5.1. Vaihdettavat osat

Vuonna 1765 ranskalainen kenraaliluutnantti Jean Baptiste Vaquette de Gribeauval keksi, että tykkejä voitaisiin valmistaa nopeammin ja taloudellisesti kokoamalla ne vaihdettavista, vakioiduista osista. Tällainen osien vaihdettavuus tekisi myös tykkien korjaamisesta kenttäolosuhteissa helpompaa ja nopeampaa. Hänen ajatustensa pohjalta Honoré Blanc kehitti Système Gribeauvalin ja laajensi tätä ajattelua myös muihin aseisiin. (Honoré Blanc).

Britannian laivasto tarvitsi 1700 - ja 1800 -luvuilla aluksiensa köysiin puisia taljapyöriä suuria määriä. Näitä taljapyöriä valmistivat useat alihankkijat, joiden työntekijöiden työn laatu ja nopeus vaihteli suuresti. Tavallisessa laivassa oli noin 1000 erikokoista taljapyörää ja laivasto osti alihankkijoiltaan vuodessa 100000 uutta pyörää. Sir Samuel Bentham oli nimitetty Britannian laivaston ylitarkastajaksi vuonna 1795 ja eräs hänen

päätehtävänsä oli laivaston toiminnan ja tukikohtien uudenaikaistaminen. Benthamin johdolla laivasto ryhtyi hankkimaan höyrykoneita, joiden avulla laivaston telakoita ja verstaita ryhdyttiin koneistamaan ja muuttamaan enemmän tehdasmaisiksi tuotantolaitoksiksi.

Ensimmäisen kerran massatuotantoon päästiin Portsmouthin telakoilla vuonna 1803. Marc Isambard Brunel toimitti Portsmouthin taljapyörätehtaalle 3 erillistä tuotantolinjaa (jokaiselle suuruusluokalle omansa), joissa käytettiin hänen patentoimiaan työstökoneita. Ensimmäinen linja, jolla valmistettiin keskikokoisia taljapyöriä, asennettiin tammikuussa 1803. Vuoteen 1808 mennessä Portsmouthin tehtaan vuotuinen tuotanto oli jo noussut 130000 taljapyörään. Tämä valmistustapa ei kuitenkaan levinnyt muualle Britanniassa ennen, kun se palasi takaisin muutaman kymmenen vuoden kuluttua Yhdysvalloista ja tuli silloin laajemmin tunnetuksi ”amerikkalaisena valmistustapana” Samuel Coltin perustettua Britannian hallituksen pyynnöstä asetehtaan Lontooseen 1850–luvulla (Clark 2005).

3.5.2. Aseiden valmistus ja standardointi

Eli Whitney teki vuonna 1798 Yhdysvaltain sotaministeriön kanssa sopimuksen 10000 musketin valmistamisesta. Valtionvarainministeri Wolcott lähetti muutama kuukausi sopimuksen allekirjoittamisen jälkeen Whitneylle teoksen, jossa kuvattiin varsin tarkasti aseenvalmistuksen menetelmiä ja tekniikoita (kyseessä oli luultavasti jokin Blancin monista kirjoituksista). Tästä kirjoituksesta Whitney omaksui ajatuksen vaihdettavista osista ja niiden käytöstä asetuotannossa. Vuonna 1801 hän piti sotaministeriön edustajille demonstraation ja havainnollisti, miten vaihdettavista osista koottu musketti oli mahdollista valmistaa nopeammin ja taloudellisemmin kuin aiemmin käsityönä yksilöllisesti valmistettu. Whitneyltä kesti kuitenkin kaikkiaan 9 vuotta toimittaa tilatut musketit armeijalle vaikka luvannut alkuaan toimittamaa tilatut aseet 1 vuoden aikana. Joidenkin Whitneyn arvostelijoiden mukaan hänen demonstraationsa oli taitavaa huijausta, jonka avulla Whitney sai ostettua itselleen lisää aikaa.

Colt sai vuonna 1836 patentin keksimälleen pyörivään sylinteriin perustuvalle mekanismille, jonka ansiosta käsiase voitiin laukaista useamman kerran lataamatta sitä laukausten välillä. Hankalan alun jälkeen asiat muuttuivat, kun Yhdysvaltain hallitus tilasi Coltilta 1000 revolveria Meksikon sotaa varten. Hän solmi Whitneyn kanssa alihankintasopimuksen, jonka perusteella hallituksen tilatut revolverit valmistettiin Whitneyn tehtaassa. Vuosien 1846–1848 aikana käydyn sodan jälkeen Coltin maine tehokkaana ja luotettavana aseena olikin jo levinnyt laajalle. Colt esitteli revolveriaan ja sen toimintaa vuoden 1851 maailmannäyttelyssä Lontoossa.

Vuonna 1855 Colt avasi maailman suurimman yksityisen asetehtaan Hartfordiin Connecticutiin. Hän oli tutkinut tarkasti Whitneyn tehtaan valmistustekniikoita ja työtapoja ja hyödyntänyt niitä jo Lontoon tehtaassaan. Tämän tutkimuksen pohjalta Colt kehitti niitä edelleen ja hän ottikin käyttöön uusia valmistustekniikoita ja tuotantotapoja (esimerkkeinä aseiden kokoonpano vaihtokelpoisista osista ja virtaviivaistettu revolverien tuotantolinja). Seuraavana vuonna Coltin asetehdas pystyi valmistamaan jo 150 asetta päivässä. Suuri kunnia tästä menestyksestä kuului Coltin tehtaanjohtajalle Elisha K. Rootille, joka oli lahjakas ja toimelias tuotannon kehittäjä. Colt ymmärsi markkinoinnin ja mainonnan merkityksen ja Yhdysvaltojen sisällissodan syttyessä vuonna 1861 Colt olikin jo maailman tunnetuin ampuma-ase. Hänen kuolemansa jälkeen Root nousi koko yhtiön johtoon. Hänen johdollaan Colt-yhtiö koulutti 1800-luvun loppupuoliskolla suuren määrän mekaanikkoja ja keksijöitä, jotka edelleen kehittivät uusia työstökoneita ja valmistustekniikoita mekaanisen teollisuuden käyttöön.

Aseiden valmistukseen kehitetty valmistustapa ja työstökoneiden keksiminen olikin ratkaiseva edellytys mekaanisen teollisuuden kehittymiselle. Aseteollisuuden tuotantotavat ja työstökoneet levisivätkin nopeasti muille teollisen tuotannon aloille. *Standardointi* tuli mahdolliseksi mekaanisten työstökoneiden avulla tapahtuvan valmistuksen myötä. Tarkat kellot ja taskukellot, polkupyörät, autot ja muut kulkuneuvot, kodin tehtävissä käytetyt laitteet, toimistolaitteet ja muissa ympäristöissä hyödynnettävät tuotteet tulivat tavallisten ihmisten ulottuville, koska työstökoneet tekivät suurimman osan niiden valmistustyöstä tarkemmin, nopeammin ja halvemmalla kuin entisajan ammattimiehet käsityön aikakaudella (Samuel Colt).

Ohjelmistoteollisuudessa koodin uudelleenkäyttö ja komponenttiajattelu edustavat selkeästi samaa ajattelua, mitä aseteollisuudessa ensimmäisen kerran toteutettiin Coltin tehtaassa.

3.5.3. Liukuhihnan esiinmarssi

Henry Fordin perustaessa yhtiönsä vuonna 1903 oli autojen valmistaminen tyypillisesti ammattimiehille varattua työtä. Nämä ammattimiehet olivat saaneet koulutuksensa monasti Michiganin ja Ohion lukuisissa vaunu- ja polkupyöräverstaissa, ja osasivat hoitaa kaikki vaativat ja monimutkaisetkin työt, joita autojen valmistukseen kuului. Vähitellen työt organisoitiin uudelleen, ammattimiehen avuksi palkattiin halvempia apumiehiä tai –poikia tekemään vähemmän vaativia työvaiheita ja pian yksi mies ei enää koonnutkaan autoa alusta loppuun asti. Uuden työtavan mukaan saman auton parissa työskenteli samanaikaisesti useita ammatti- ja apumiehiä, joille jokaisella oli suoritettavana oma osuutensa työn kokonaisuudesta (Braverman 1977: 130–131).

Kun vuonna 1908 julkistetun T-mallin kysyntä kasvoi nopeasti, oli Fordin pakko ryhtyä kehittämään uusia tuotantotekniikoita ja -menetelmiä voidakseen valmistaa kaikki markkinoiden haluamat autot. Fordin ratkaisun lähtökohtina oli neljä asiaa: osien vaihdettavuus, jatkuva vuo, työn jakaminen ja turhien liikkeiden eliminointi. Ratkaisuksi kehitettiin ”liukuhihna”, jota aluksi käytettiin erilaisten autonosien ja komponenttien kokoonpanossa. Vuoden 1914 tammikuussa liukuhihna otettiin viimein käyttöön autojen loppukokoonpanossa Highland Parkin tehtaalla Detroitissa (joidenkin sähköisten lähteiden mukaan tämä tapahtui jo 7.lokakuuta 1913). Kolmen kuukauden kuluessa oli T-mallin kokoonpanoaika pudonnut yhteen kymmenesosaan entisestä ja sen tuotantokapasiteetti moninkertaistunut vastaavasti (Braverman 1977: 131). Malli Fordin liukuhihnaan oli saatu Chicagon suurista teurastamoista, joissa jo 1860-luvulla oli otettu käyttöön katossa kulkevat raiteet, joita pitkin niissä roikkuvia eläinten ruhoja siirrettiin eri käsittely-asemien läpi. Näiden raiteiden avulla pystyttiin teurastamojen kapasiteetti kasvattamaan moninkertaiseksi verrattuna aiempaan (Assembly Line, Inventions).

Työntekijöiden reaktio ei suinkaan ollut myönteinen tähän muutokseen, joka pakotti heidät työskentelemään ilman taukoja pakkotahtisen hihnan tahdissa. Ihmiset lähtivät suurin joukoin muiden työnantajien palvelukseen ja Ford Company olikin joutumassa tilanteeseen, jossa sillä oli omistuksessaan suuria tehtaita, mutta ei ihmisiä näihin tehtaisiin autoja valmistamaan. Kriisi oli pahentunut kesällä 1913 pidetyn Toisen internationaalin jälkeen ja pahimmillaan henkilöstön vaihtuvuus oli jo yli 380 % vuositasolla. Ratkaisuna näihin henkilöstön kanssa koettuihin ongelmiin Ford kertoi julkisuudessa päätöksestään nostaa kerralla kaikkien työntekijöiden päiväpalkka 5 dollariin. Yhtiö sai tämän jälkeen tehtailleen niin paljon työväkeä, että se pystyi valitsemaan parhaat päältä ja pitämään loput reservissä (Braverman 1977: 132–133).

Fordin suuren menestyksen ja voittojen takana olevista tekijöistä tärkeimpänä pidetään yleensä tuotannon tehokkuutta (tehokkuudella tarkoitetaan tällöin ensisijaisesti valmistamisen matalaa yksikkökustannusta). Fordin tuotannon johtavana periaatteena oli valmistaa ainoastaan yhtä tuoteversiota, mustaa T- mallia, tehtaan täydellä kapasiteetilla ilman keskeytyksiä kunnes markkinoiden kysyntä olisi tyydytetty. Tuotanto-ohjelma pakotti työntekijät ja koneet suorittamaan toistuvasti yksiä ja samoja työvaiheita niin nopeasti kuin mahdollista. Liukuhihna pakotti heidät kaikki samaan tahtiin ja tästä aiheutui usein ongelmia, koska ihmiset eivät luonnostaan olleet mitään mekaanisesti toimivia koneiston rattaita. Tällaisessa massatuotannossa tuotannon virtaa ohjataan prosessin ulkopuolelta ilman takaisinkytkentää materiaalista tai työntekijöiltä, joita liukuhihna kirjaimellisesti pakottaa liikkumaan eteenpäin hihnan nopeudella. Jatkuva, keskeytymätön homogeenisten yksikköjen virta mahdollisimman suurella nopeudella takasi alhaisimman mahdollisen yksikkökustannuksen Fordin ajattelun mukaan. Kun hihnan nopeutta voitiin myös vähitellen nostaa, putosi yksikkökustannus tietysti entisestään. Fordin tehtaat valmistivat kaikkiaan yli 15 miljoonaa T-Fordia vuoteen 1927 mennessä. Siihen mennessä asiakkaiden maut ja halut olivat kuitenkin ratkaisevasti muuttuneet: valmistetut tuotevariantit eivät enää vastanneetkaan niitä tuotteita, joita asiakkaat halusivat ostaa (Braverman 1977: 131).

4. LAATUAJATTELUN HISTORIA

Tämä luku käsittelee nykyaikaisen laatuajattelun historiaa, sen suurimpia edelläkävijöitä ja heidän työlleen rakentuneita teorioita, malleja ja menetelmiä.

4.1. Tilastollisen laadunohjauksen synty

Walther A Shewhart oli se henkilö, joka loi tilastollisen laadunohjauksen perusteorian ja joka täten loi perustan koko nykyaikaiselle laatutoiminnalle. Vuosina 1918–1924 hän työskenteli Western Electricsillä kehittäen siellä teorioitaan ja kokeitaan laadun ja tuottavuuden parantamiseksi tilastollisten menetelmien avulla.

Shewhart yhdisti tutkimus- ja kehitystyössään menestyksellisesti tilastotieteen, talousajattelun ja insinööritaidon opit yhdeksi yleiseksi tilastollisen laadunohjauksen metodologiaksi, jonka avulla voidaan ohjata ja valvoa lähes mitä prosessia tahansa. Hänen tunnetuin keksintönsä on tarkastuskortti, yksinkertainen mutta tehokas työkalu, joka oli ensimmäinen askel sillä polulla, joka Shewhartin sanojen mukaan johti ”taloudellisen ohjauksen tieteellisen perustan rakentamiseen”.

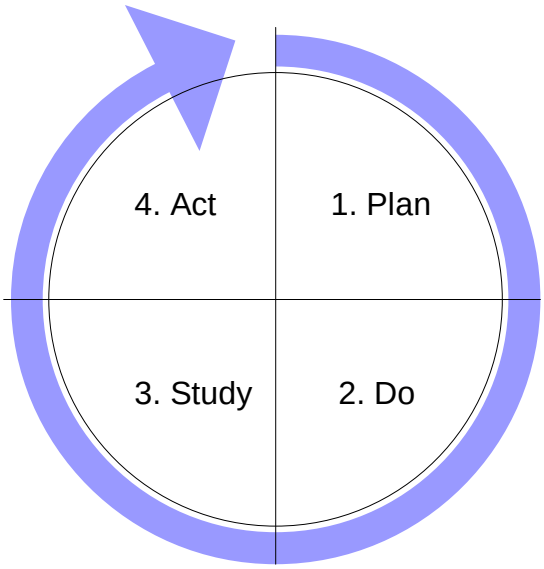
Shewhart tutki Hawthornen tehtaalla puhelinten valmistusprosessia. Hän teki havaintoja eri ongelmien syistä ja tutki keräämäänsä aineistoa tilastomatematiikan avulla. Hän havaitsi, että prosessista mitatut arvot vaihtelivat suuresti. Vaihtelu johtui eri syistä, jotka hän jakoi kahteen ryhmään (Deming 1994b: 172–174):

1. *Luonnolliset syyt* johtuivat valmistusprosessin ja –järjestelmän ominaisuuksista ja rakenteesta. Näihin syihin ei työntekijä toiminnallaan voinut millään lailla vaikuttaa. Tällaisia syitä ovat esimerkiksi tehdasympäristön fysikaaliset olosuhteet, raaka-aineen ominaisuudet ja työkalujen normaali tarkkuus.
2. *Erikoissyyt* voitiin aina johtaa johonkin erikoiseen, yksittäiseen tapahtumaan, joka ei normaalioloissa esiintynyt prosessissa. Erikoissyy voi olla esimerkiksi työkalun tai koneen rikkoutuminen, epäkurantti materiaalierä tai virheellinen asetusarvo.

Shewhart uskoi, että tällaisesta tilastomatemaattisesta lähestymistavasta olisi suurta hyötyä teollisuudelle. Hän uskoi, että on parempi tutkia tilannetta ensiksi eikä rynnätä suinpäin tekemään muutoksia, joiden vaikutuksia ei voida ennakoida. Tiedemiehenä Shewhart oli aikaansa edellä: nykyisin tiedekirjallisuudessa keskustellaan biologisten ja teknisten järjestelmien stokastisesta käyttäytymisestä ja pohditaan mahdollisuuksia soveltaa tilastollisia menetelmiä niihin. Shewhart toteutti tätä ajatusta jo 1920-luvulla ja sen vuoksi häntä voidaankin siksi pitää koko nykyaikaisen laatuajattelun isänä. Hänen pääteoksensa, "*Economic Control of Quality of Manufactured Product* " vuodelta 1931 on vielä tänäkin päivänä koko tilastollisen laadunohjauksen perusteos. Shewhartia, Demingiä ja Jurania yhdistää kaikkia Western Electricsin Hawthornen tehdas Chicagossa: kaikki ovat vuorollaan työskennelleet siellä 1920-luvulla tuotannon laatuongelmien parissa. Shewhart toimi selkeästi sinä edelläkävijänä, jonka tilastollisten perusteorioiden varaan sekä Deming että Juran jälkeenpäin rakensivat oman tutkimuksensa ja teoriansa (Creech 1994: 201–202).

4.2. Demingin opit: PDCA-ympyrä ja osaamisperusta

Shewhartin kehittämä PDSA –ympyrä (*Plan, Do, Study and Act*) on tullut paremmin tunnetuksi Demingin ympyränä ja PDCA -ympyränä (*C=Check*). Se havainnollistaa hyvin niitä toiminnan vaiheita, mitkä tuotetta tai prosessia kehitettäessä läpikäydään, mutta kuvaa samalla oppimista ja kehitystä laajemmaltikin.



Kuva 3. *Shewhartin PDSA - ympyrä (Deming 1994b: 131).*

Ympyrän ensimmäinen osa suunnittelu (*1. Plan*) on lähtökohta kaikelle kehittämiselle. Suunnittelussa määritellään haluttu parannus/muutos ja se työ mitä aiotaan tehdä sen aikaansaamiseksi. Usein ihmisen kiirehtivät suunnitteluvaiheen läpi toteutukseen hätäisesti. Suunnitteluvaiheessa on tavallisesti mukana useita vaihtoehtoja, joista toteutettavaksi valitaan se, jonka vaikuttaa lupaavimmalta tuottavuudeltaan tai uuden tietämyksen perusteella arvioiden. Toteutusvaiheessa (*2. Do*) suoritetaan koe, vertailu tai testi suunnitteluvaiheen päätösten mukaisesti. Testit kannattaa pitää pienimuotoisena ja pyrkiä minimoimaan hallinnon osuus. Shewhart kehitti tämän mallinsa tukemaan yksilön tai pienen koeryhmän toimintaa, joten sitä ei yleensä yritetä soveltaa suuriin, formaaleihin kehityshankkeisiin. Ajatusmallina se on tietenkin käyttökelpoinen monissa erilaisissa kehitystilanteissa. Tutkimus- tai tarkasteluvaiheessa (*3. Study alt. Check*) toteutusvaiheen tuloksia arvioidaan mahdollisimman monipuolisesti suunniteltujen kriteerien mukaan. Tässä vaiheessa on arvioitava vastasivatko tulokset ja havainnot ennakko-odotuksia ja toiveita vai ilmenikö niissä jotain odottamatonta. Usein tulosten perusteella huomataan selviä parannusmahdollisuuksia myös itse koejärjestelyihin. Viimeisessä vaiheessa (*4. Act*) suunniteltu muutos joko otetaan käyttöön tai se hylätään.

Joskus on tarpeen aloittaa ympyrä alusta ja lähteä jatkokehittämään alkuperäistä suunnitelmaa. Deming näkee, että lopullinen päätös muutoksesta on aina perusteltava selkeästi tutkimusvaiheen tulosten ja arvioiden avulla (Deming 1994b: 131–132).

Demingin mukaan maailmalla yleisesti vallassa olevan johtamistyylin on muututtava. Järjestelmä ei pysty ymmärtämään itseään ja muutoksen aikaansaamiseen vaaditaan ulkopuolista näkökulmaa. Hän kutsuu tällaista järjestelmän, prosessin tai yhteisön ulkopuolelta tulevaa lähestymistapaa tietämysperustaksi (*profound knowledge*). Demingin määrittelemä tietämysperusta (*Profound Knowledge*) koostuu seuraavista neljästä tietämyksen alueesta, jotka ovat kiinteästi sidoksissa toisiinsa (Deming 1994b: 93):

1. Järjestelmien ymmärtäminen
2. Prosessien vaihtelun teoria
3. Tietämyksen ja tietämisen teoria
4. Psykologia

Yksilön ei tarvitse olla minkään alueen saati sitten kaikkien neljän asiantuntija pystyäkseen ymmärtämään sekä hyödyntämään niiden periaatteita ja käytäntöjä omassa työssään. Tietämysperustan neljä aluetta ovat riippuvaisia toisistaan sekä toimivat aina vuorovaikutuksessa toistensa kanssa. Näin ollen esimerkiksi psykologian tiedot ilman prosessien vaihtelun ymmärtämistä ovat riittämättömiä ja saattavat helposti osoittautua käyttökelvottomiksi ohjelmistoprosessin parantamisessa.

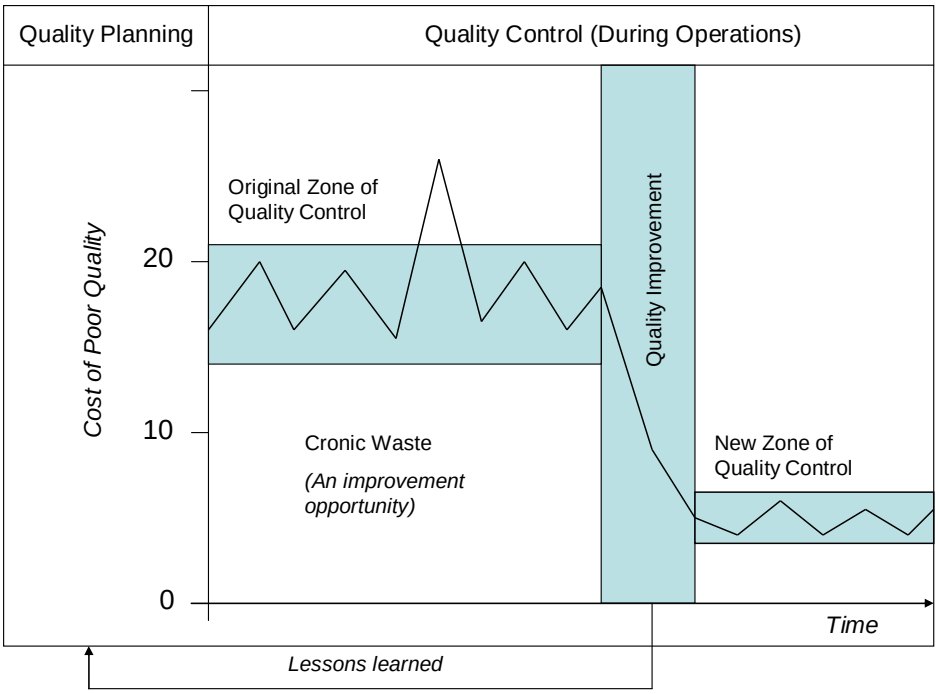
Demingin opit prosessien vaihtelusta voidaan kiteyttää seuraaviin väittämiin:

1. Prosessien tilastollisuuden ymmärtäminen on kaiken kehitystoiminnan perusta. Tuottavuutta sekä tuotteita ja palveluita parannetaan tehokkaimmin vähentämällä prosessien vaihtelua ja epävarmuutta (Deming, 1994a: 3).
2. Prosessien vaihtelu johtuu joko luonnollisista (yleiset) tai ulkoisista (erityiset) syistä. Ensiksi mainitut muodostavat 94 % kokonaisvaihteluista (Deming 1994b: 33).
3. Luonnollisten syiden poistaminen eli prosessin parantaminen on aina yritysjohdon vastuulla. Työntekijät voivat vain tunnistaa ja korjata jälkimmäiset eli erityiset syyt. (Deming 1994b: 33–34).

Näin ollen onkin Demingin mukaan varsin järjetöntä väittää jokaisen työntekijän olevan vastuussa oman työnsä laadusta, kun valtaosa vaihtelusta (virheistä) johtuu prosessin luonnollisista ominaisuuksista. Prosessin muuttamisesta ja parantamisesta on vastuussa aina organisaation ylin johto. On järkevää että se kenellä on valta päättää asioista kantaa myös vastuun.

4.3. Juranin trilogia ja Pareto-periaate

Joseph M. Juran muodostaa W. Edwards Demingin kanssa sen kivijalan, jonka varaan koko nykyinen laatuajattelu ja -toiminta on 1900-luvun jälkipuoliskolla rakennettu. Juran toi johtamisen näkökohdat mukaan laatutyöhön jo 1940-luvulla kuuluisassa teoksessaan ”Quality Control Handbook”. Hän näki yrityksen johdon perustehtävänä olevan joko muutoksen aikaansaamisen (*breakthrough*) tai sen estämisen (*control*) (Juran 1995: 5).



Kuva 4. *Juranin trilogia (Juran 1995: 429)*

Juranin mukaan laatutyö jakautuu periaatteessa kolmeen eri osaan: laadun suunnitteluun, laadun ohjaukseen ja laadun parantamiseen. Laadun suunnittelun (*Quality Planning*) avulla valmistaudutaan kohtaamaan laadunohjauksen haasteet ja vaatimukset. Laadun suunnittelun tavoitteena on tunnistaa ne alueet, joissa vaaditaan parannustoimenpiteitä. Laadun parantamisen (*Quality Improvement*) tarkoitus on tietenkin parantaa prosessien suorituskykyä ja nostaa toiminnan laatu yhä paremmaksi. Laadun parantaminen toteuttaa niitä toimenpiteitä, joita laadun suunnitteluvaiheessa on määritelty. Laadun ohjaus (*Quality Control*) on jokapäiväistä, operatiivista laatutyötä, jolla varmistetaan yrityksen prosessien toimivuutta. Tätä mallia kutsutaan yleisesti Juranin trilogiaksi (*Juran's Trilogy*) (Juran 1995: 429).

Vilfredo Federico Damaso Pareto (1848 - 1923) tutki tulojen jakautumista eri yhteisöissä sekä taloudellisten toimijoiden valintoja. Hän oli ensimmäisiä tutkijoita, jotka analysoivat talouden ongelmia käyttämällä matemaattisia välineitä apunaan. Pareto tutki Britannian kansantalouden tilastoja ja kehitti niiden perusteella lakinsa tulonjakaumasta, jonka mukaan jokaisen tulotason ja sitä enemmän ansaitsevien ihmisten määrän välillä on olemassa lineaarinen riippuvuus. Pian tätä havaintoa ryhdyttiin kutsumaan Pareton periaatteeksi tai 80–20 -säännöksi. Pareto - jakauma onkin sen laajempi yleistys. Silloin kun keskitytään olennaiseen eli vain tärkeimpiin asioihin, kun eliminoidaan tai hoidetaan tärkeimmät 20 % ongelmien syistä, niin kaikista ongelmista saadaan 80 % ratkaistua ja poistettua. Juran kehitti Pareton työn pohjalta oman teoriansa, jolle hän antoi nimen "vital few and useful many" (Juran 1995: 47). Seuraavat ajatukset ovat tyyppiesimerkkejä Pareton periaatteen soveltamisesta:

1. 20 % käyttämästäsi ajastasi vastaa 80 % tuloksistasi.
2. Jokaisen prosessin osista vain muutamat (20 %) ovat elintärkeitä (*vital*) ja loput (80%) ovat jokseenkin harmittomia (*trivial*).
3. Yleensä 20 % jokaisen kehitysprojektin tehtävistä aiheuttaa 80 % sen kustannuksista.
4. Varaston kokonaisarvosta suurin osa (80%) oli sitoutunut muutamaan tuotekategoriaan (20%).

Pareton periaatetta voidaan hyödyntää luokiteltaessa ongelmien syitä tärkeisiin ja toisarvoisiin. Juranin mukaan vasta tällaisen alustavan jaon (kategorisoinnin) jälkeen on mahdollista ryhtyä järkeviin toimenpiteisiin laadun parantamiseksi. Pareto -analyysi antaa myös alustavan kuvan niistä asenteista ja ennakkoluuloista, joita organisaatiossa saattaa olla kehitys- ja muutoshanketta kohtaan (Juran 1995: 56).

4.4. Laatu on ilmaista

Phil Crosby oli yhdysvaltalainen liikkeenjohtaja ja konsultti, joka julkaisi teoksensa "*Quality Is Free*" vuonna 1979. Crosby joka popularisoi laadun parantamisen ja johtamisen Yhdysvalloissa ja hänen kehittämänsä nollavirhekampanjat levisivät 80-luvulla läpi koko länsimaisen liike-elämän. Crosbyn mukaan laadun parantaminen sinällään ei ollut vaikeaa: kunhan ihmiset vain tunnistavat ongelman ja pääsevät yhteisymmärrykseen korjaustoimenpiteistä, sujuu parantaminenkin hyvin. Hänen käsityksensä mukaan laadun parantamisessa on yhtä paljon kyse toiminnassa mukanaolevien ihmisten käännyttämisestä kuin ongelmien ratkaisemisesta. Crosby uskoi, että se mikä toimii laadun parantamisessa yhdellä alalla, toimii myös muillakin - samat periaatteet pätevät liiketoiminnassa yleisesti. Crosbyn mukaan ”totuus” laadusta muodostuu neljästä periaatteesta (Crosby 1990: 60–62):

1. Laatu tarkoittaa vaatimustenmukaisuutta eikä suinkaan sitä, että joku tuote on ”hyvä” Vaatimusten on oltava realistisia ja selkeästi ilmaistuja.
2. Jokaisessa liiketoiminnassa on aina halvempaa tehdä asiat oikein ensimmäisellä kerralla. Laatu syntyy estämällä virheiden syntyminen eikä tarkastamalla tuotteita jälkeenpäin.
3. Useimmat laatuongelmista syntyvät jo suunnittelu- ja luomisvaiheessa. Mitä aikaisemmin virheet saadaan kiinni sitä pienemmät laatukustannukset.
4. Laatutyön standardin on oltava virheettömyys eikä suinkaan mikään hyväksyttävä virhetaso.

Nollavirhetavoite on Crosbyn mukaan ainoa oikea normi yrityksen toiminnalle. Deming arvosteli kuitenkin sitä ja yleensäkin numeeristen tavoitteiden asettamista ja väitti

kivenkovaan, että tuollainen lähestymistapa kertoi aina prosessien perusasioiden ja tilastollisen laadunohjauksen täydellisestä osaamattomuudesta (Deming 1994b: 31–33).

4.5. Lean Thinking

Lean ajattelun lähtökohtana toimii Womackin ja Jonesin mukaan aina arvo, ja arvo siinä muodossa kuin se tarkasteltuna loppuasiakkaan perspektiivistä käsitetään (Womack ja Jones 1996: 16). He esittävät kirjassaan ”Lean Thinking” viisi vaihetta, joiden kautta organisaatiot voivat siirtyä lean ajatteluun:

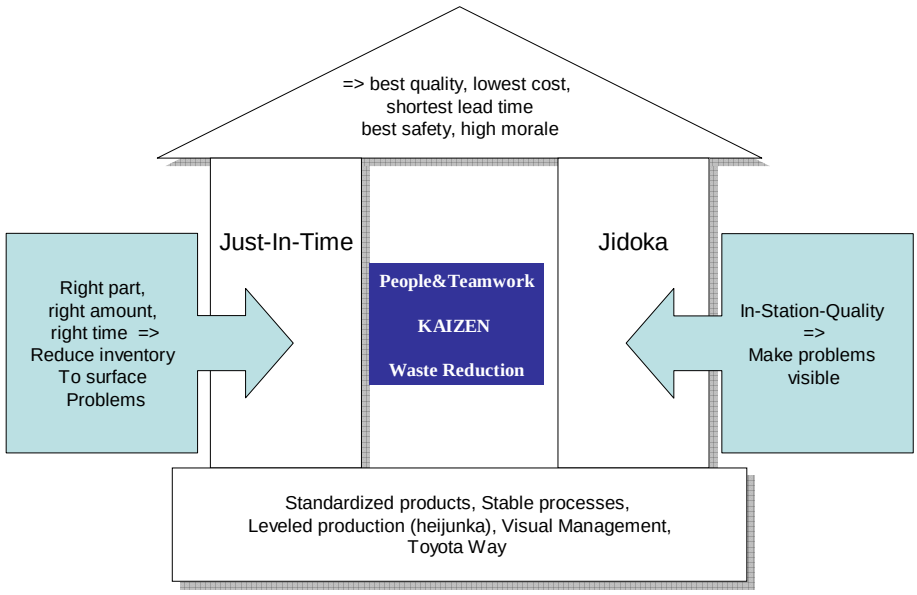
1. Arvon määrittäminen (*Specify Value*). Vain loppuasiakas voi määrittää tuotteen todellisen arvon (Womack ja Jones 1996: 16).
2. Arvovirran tunnistaminen (*Identify the Value Stream*). Tämä koostuu kaikista vaiheista, joita tarvitaan tuotteen saamiseksi asiakkaan hyödynnettäväksi (Womack ja Jones 1996: 19).
3. Virtaus (*Flow*). Arvovirran pitää kulkea ilman esteitä ja hidasteita (Womack ja Jones 1996: 21).
4. Veto (*Pull*). Tuotteet pitää valmistaa asiakkaan tilauksesta eikä varastoon (Womack ja Jones 1996: 24).
5. Pyri täydellisyyteen (*Pursue Perfection*). Jatkuvan parantamisen kulttuuria on edistettävä jatkuvasti ja kaikin voimin (Womack ja Jones 1996: 25).

Japanilainen sana ”*muda*” tarkoittaa jätettä, jotain tarpeetonta, joka kuluttaa resursseja mutta ei tuota lisäarvoa. Muda voi ilmetä ihmisten tekemänä turhana työnä, virheinä tai vääränlaisina tuotteina, joita kukaan ei halua ostaa. Womackin ja Jonesin mukaan mudaan on olemassa tehokas lääke: lean ajattelu. Sen avulla on mahdollista tehdä enemmän vähemmällä (vähemmällä ihmistyöllä, lyhyemmässä ajassa, pienemmällä materiaalinkulutuksella ja tilankäytöllä). Womackin ja Jonesin teos käsittelee sitä muutosta, mitä japanilaisessa teollisuudessa tapahtui toisen maailmansodan jälkeen kun japanilaiset yritykset alkoivat soveltaa Demingin ja Juranin oppeja toiminnassaan. (Womack ja Jones 1996: 15).

Taichi Ohno (1912–1990) oli se Toyotan johtaja, joka eniten vaikutti Toyotan tuotantojärjestelmän (Toyota Poduction System = TPS) syntymiseen (Womack ja Jones 1996: 15). Ohnon mukaan oli olemassa 7 erilaista mudaa:

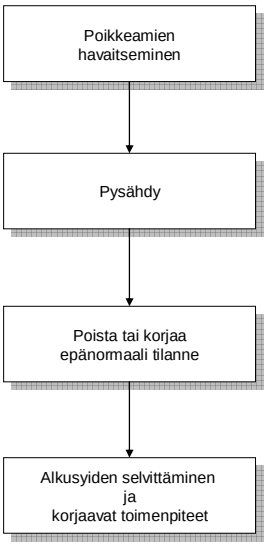
1. Kuljetus (*Transportation*)
2. Varasto (*Inventory*)
3. Liike (*Motion*)
4. Odottaminen (*Waiting*)
5. Ylityöstäminen (*Over-processing*)
6. Ylituotanto (*Over-production)*
7. Viat (*Defects*)

Nämä seitsemän mudaa eivät ole toisistaan riippumattomia, joten niiden vähentäminen on järkevää ratkaista holistisesti kaikkia samanaikaisesti tarkastellen ja muuttaen. Womack ja Jones ovat lisänneet vielä luetteloon 8. mudan: tuotteet ja tavarat, jotka eivät ole asiakkaiden vaatimusten mukaisia ja muut tutkijat vielä 9. mudan: ihmisten alihyödyntämisen.



Kuva 5. *Toyota Production System (Liker and Morgan 2006:7)*

TPS) on tuotantojärjestelmä, jonka johtavana periaatteena on kaiken turhan ja jätteen eliminoiminen tuotteiden valmistuksessa. Sitä on myös kutsuttu nimillä ” ”lean manufacturing system” ja ”Just-in-Time (JIT) system,”. TPS rakentuu paljolti kahden konseptin (*pilarin*), jidokan ja Just-In-Timen, varaan. Näistä ensimmäinen, jidoka, käännetään joskus vapaasti ”automaatioksi inhimillisen kosketuksen kanssa”. Tämä tarkoittaa sitä, että ongelman syntyessä valmistuslaitteisto pysähtyy välittömästi ja estää näin viallisten tuotteiden valmistamisen. Just-in-Time -konseptin mukaan jokaisessa prosessissa tuotetaan vain sen verran kuin seuraavassa prosessissa tarvitaan jatkuvan vuon ylläpitämiseksi.



Kuva 6. *Jidokan eri vaiheet (Liker and Morgan 2006:7).*

Yllä olevassa kaaviossa esitetään jidokan mukainen toiminta poikkeamien hallinnassa. Kun havaitaan poikkeama, on välittömästi pysähdyttävä ja pysäytettävä työnteko (prosessi, kone, tms.). Poikkeama on aina epänormaali tilanne, joka pitää mahdollisimman nopeasti korjata. Tutkimalla tätä epänormaalia tilannetta selvitetään sen alkusyyt. Alkusyiden selvittyä voidaan sitten määritellä korjaavat toimenpiteet ja palata normaaliin tilanteeseen. Jidokan tehtävä on tuoda pienetkin ongelmat esiin (Liker & Morgan 2006:7).

”*Kaizen*” on japanilainen sana, josta on vakiintunut tarkoittamaan jatkuvaa parantamista. Taguchi kyseenalaisti aikoinaan koko idean työstandardeista ja valmistuksen toleransseista. Hän osoitti kokeillaan, että on parempi asettaa haluttu nimellisarvo mahdollisimman tarkasti ja jatkuvasti pyrkiä pienentämään vaihtelua sen ympäriltä. Tämä kehitystyö johtaa väistämättä parempaan laatuun ja pienempiin kustannuksiin. Jos organisaatio ymmärtää ja huolehtii niistä asioista, joilla on todellista merkitystä, niin asiakkaat kokevat saavansa hyvää palvelua ja organisaatiot voivat tuottaa palvelunsa mahdollisimman taloudellisesti – keskittymällä olennaiseen päästään parempaan lopputulokseen. (Deming 1994a: 141)

Liker ja Morgan ovat artikkelissaan ”The Toyota Way in Services: The Case on Lean Product Development” määritelleet seuraavat neljä periaatetta TPS-mallin mukaiselle tuotekehitysprosessille (Liker&Morgan 2006: 9–10):

1. *On määriteltävä asiakaslähtöinen arvo*, jonka perusteella voidaan eliminoida kaikki turha (muda) prosessista.
2. *On panostettava tuotekehitysprosessin alkuun*, jotta voidaan kartoittaa kaikki mahdolliset ratkaisut ja minimoida virheiden lähteet mahdollisimman aikaisin.
3. *On luotava kehitysprosessin vuo*, jonka avulla tuotekehityksen kuorma on hallinnassa. Kuormituksen vaihtelua voidaan ennakoida ja suunnitella joustavan työvoiman avulla.
4. *On harjoitettava ankaraa standardointia* vaihtelun pienentämiseksi ja joustavuuden lisäämiseksi, sillä standardointi on jatkuvan parantamisen edellytys ja perusta muiden prosessiperiaatteiden noudattamiselle.

Likerin ja Morganin mukaan juuri stabiili ja standardoitu prosessi muodostaa perustan koko muulle kehitystoiminnalle. Japaninkielinen sana ”*heijunka*” tarkoittaa juuri tällaista tilastollisesti ohjattavaa, stabiilia prosessia, jonka tuotokset ovat ennakoitavissa. Toyota on pyrkinyt standardoimaan mahdollisimman pitkälle tuotteidensa suunnittelun ja konfiguroinnin sekä prosessinsa ja henkilöstönsä osaamisen. (Liker & Morgan 2006: 11–12).

4.6. Business Process Reengineering

Michael Hammerin (Hammer 1996: xii) mukaan business process reengineering (BPR) on “Liiketoimintaprosessien perusteellista uudelleenajattelua ja radikaalia uudelleensuunnittelua, joiden avulla saavutetaan dramaattisesti parantunut suorituskyky”. Hammer painottaa prosessin kokonaisuuden merkitystä; ”prosessi koostuu toistettavista aktiviteeteista, jotka vain yhdessä luovat asiakkaalle arvoa” (Hammer 1996:xii).

Aiemmin oli keskitytty eri ihmisten työtehtäviin ja liiketoiminta oli organisoitu näiden työtehtävien perusteella. Prosessipohjainen organisaatio pakottaa hänen mukaansa yritykset keksimään uudelleen kaikki järjestelmänsä sekä johtamisensa periaatteet ja välineet. Muutos alkaa Hammerin mukaan siitä, että huomion keskipiste kohdistetaan prosesseihin ja asiakkaisiin. Ilman asiakkaita, heidän tarpeitaan ja näitä tarpeita varten rakennettuja prosesseja ei yrityksen olemassaololle ole mitään edellytyksiä (Hammer 1996: xiii).

Hammer sanoo tällaiseen muutokseen kuuluvan seuraavat neljä askelta (Hammer 1996:14–17):

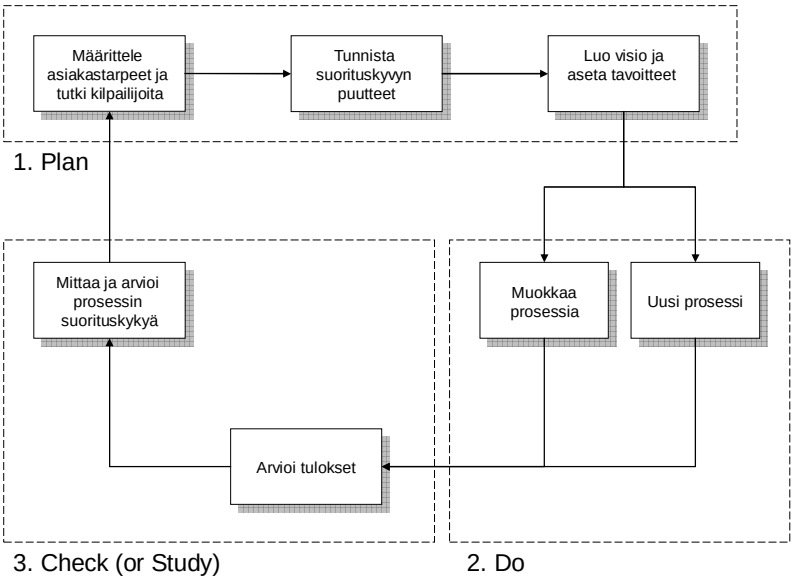
1. prosessien tunnistaminen ja nimeäminen
2. prosessien tiedostaminen ja organisaation perehdyttäminen
3. prosessien mittaaminen
4. prosessien johtaminen

Yrityksen on valittava ne kriittiset suureet, joita se haluaa mitata. Asiakkaan julkisesti esittämät vaatimukset eivät yksinään riitä, vaan hänen toimintaansa pitää tutkia ja selvittää myös ne hiljaiset vaatimukset, jotka ovat tärkeitä asiakkaan toiminnalle. Yhtä tärkeää kuin avainsuureiden mittaaminen on kehittää oikeat säätöalgoritmit, joiden syötteinä prosessista saatuja mitta-arvoja hyödynnetään (Hammer 1996: 15–16).

Hammer luokittelee prosesseissa tehtävän työn kolmeen kategoriaan sen tuottaman lisäarvon mukaan: arvoa lisäävään eli hyvään, nolla-arvoiseen mutta välttämättömään ja arvoa vähentävään työhön eli pahaan (Hammer 1996: 33–34). Teollisuudessa erityisesti

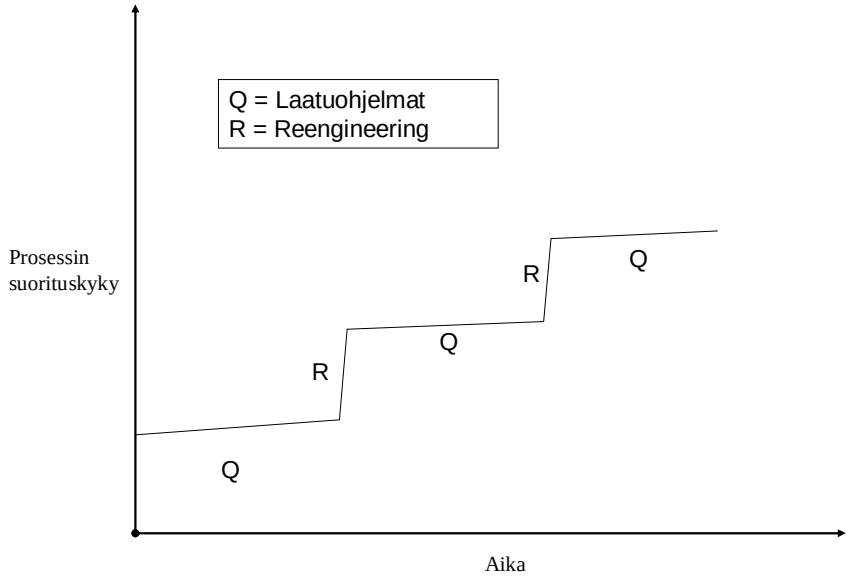
Scientific Managementin myötä tuli tavaksi pilkkoa työ mahdollisimman pieniin suoritettaviin osiin (työvaiheisiin). Tällaisten pienten osien yhdessä pitäminen vaatii Hammerin käyttämän metaforan mukaan turhan paljon ”liimaa” ja kun osat on vihdoin saatu liimattua ja pysymään yhdessä, onkin lopputulos tavallisesti ruma ja huonosti toimiva. Jokainen liimattu sauma eri osien välillä muodostaa tällöin potentiaalisen ongelmien lähteen. Ratkaisu tällaisen tilanteen välttämiseksi on laajentaa näitä sirpaloituneita työn osia suuremmiksi ja vähentää näin ollen tarvittavan ”liiman” määrää. Kun yritys keskittyy ajattelemaan toimintaansa enemmän prosessilähtöisesti, niin Hammerin mukaan perinteiset työtehtävät laajenevat ja monipuolistuvat (Hammer 1996: 35).

Hammer tuo esiin työntekijän roolin muuttumisen pelkästä suorittajasta itsenäisesti toimivaksi ammattilaiseksi. Asiakkaan ja ammattilaisen vuorovaikutus on monesti suorempaa ja tiheämpää kuin ammattilaisen ja hänen esimiehensä. Ammattilaiseksi ei suinkaan synnytä vaan sellaiseksi tuleminen vaatii koulutusta ja harjoittelua ja todellinen ammattilaisuus syntyy kokemuksen ja oppimisen myötä. Ammattilainen myös reflektoi omaa työtään koko ajan – tämän vuoksi ammattilaisuus vaatiikin holistista lähestymistapaa työhön (Hammer 1996: 47–49). Myös johdon roolin on muututtava eli managerin on vaihduttava leaderiksi. Hammer käyttää teoksissaan varsin paljon metaforia urheilun maailmasta ja tuo esiin esimiehen eräänlaisena valmentajana, joka harjoittaa joukkuetta ja määrää myös sen pelitaktiikan. Hammer painottaa prosessin omistajan ja valmentajan eroa: valmentaja työskentelee aina ihmisten kanssa. Pelitaktiikka ja pelikirja eivät yksinään riitä vaan tarvitaan osaavat ja motivoituneet ihmiset niiden toteuttamiseen (Hammer 1996: 118–119).



Kuva 7. *Hammerin Business Process Reengineeringin osat (Hammer 1996: 81)*

BPR-hanke lähtee aina asiakastarpeiden määrittelystä ja kilpailijoiden tutkimisesta (=nykyisen tilanteen kartoittamisesta). Seuraavaksi on tunnistettava organisaation tämän hetken puutteet (=eräänlainen gap-analyysi). Tämän analyysin perusteella asetetaan hankkeelle tavoitteet. Näiden tavoitteiden pohjalta lähdetään muokkaamaan nykyistä tai luomaan aivan uutta prosessia. Muutostyön jälkeen tapahtuu työn tulosten arviointi. Tämän jälkeen mitataan ja arvioidaan prosessin suorituskykyä. Hammerin malli pohjautuu lähes täysin Demingin PDCA-ympyrään, missä asiat on esitelty mielestäni loogisemmin. Hammerin mallissa varsinaista päätöstä uuden prosessin käyttöönotosta ei ole näkyvissä (=Act-vaihe).



Kuva 8. *Laatuohjelmien ja Reengineeringin vuorottelu (Hammer 1996: 83).*

Hammerin kuvio Total Quality Managementin ja Business Process Reengineeringin yhteensovittamisesta on oikeastaan vain muunnos Juranin trilogiasta. Hammerin mukaan tavalliset laadunparannusohjelmat tuottavat vain pieniä parannuksia prosessien suorituskykyyn. Jos yritys haluaa tehdä toimintaansa (=prosesseihinsa) suuria parannuksia, tarvitaan liiketoimintaprosessien radikaalia uudelleen rakentamista (=*business process reengineering*). Juran on esittänyt asian jo paljon selkeämmin ja perustellummin. Hammerin kuviossa Q vastaa Juranin trilogian Quality Control-vaihetta ja R vastaa Juranin Breakthrough-vaihetta (= Quality Improvement). Juranin trilogiassa käytetään usein y-akselin dimensiona prosessin virheiden määrää, jonka yleisesti määritellään olevan käänteinen prosessiin suorituskykyyn nähden (Juran 1995: 6).

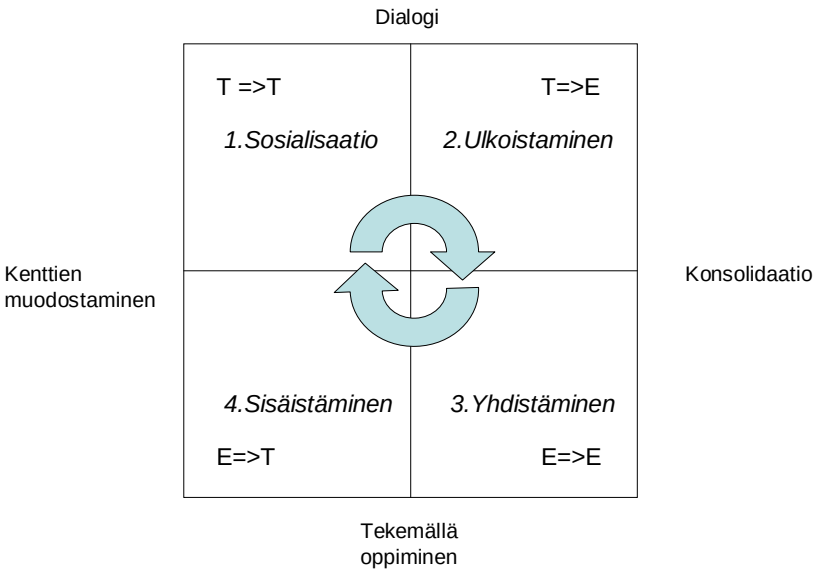
4.7. Tietämyksen hallinta ja oppivat organisaatiot

Yritykset ovat aina (joko enemmän tai vähemmän tietoisesti) ”hallinneet” tietämystään. Vaikkakaan tiedon koodaamisen, tallentamisen, siirtämisen, vaihtamisen ja

hyödyntämisen konsepti ei ole mitenkään uutta organisaatioille, ovat hallinnan käytännöt, mukaan lukien ohjelmistokehitys ja projektien hallinta, entistä enemmän keskittymässä juuri tietämykseen.

4.7.1. Tietämyksen muuntuminen

Nonaka ja Takeuchi sanovat, että organisatorinen tietämys syntyy uuden tuotekehityksen myötä (*New Product Development=NPD*) ja onkin eräällä tavalla sen suora johdannainen. Näin ollen se, miten hyvin NPD- prosessia hallitaan ja johdetaan, määrää kuinka hyvin yritys pystyy luomaan organisatorista tietämystä. Tätä organisatorista tietämystä yritys sitten hyödyntää kaikissa liiketoiminnoissaan. Polanyin mukaan kaikki explisiitti tieto on syntynyt alkuaan tacit tiedosta ja hänen mielestään explisiittiä ja tacit tietoa ei voidakaan erottaa täysin toisistaan. Ero näiden kahden välillä on siinä, että tacit tiedon ilmaiseminen kielen avulla ei ole mahdollista (Nonaka & von Krogh, 2009: 636–637).



Kuva 9. *Tiedon/tietämyksen muuntumisprosessit (Rekola 2006: 98)*

Ensimmäinen askel on **sosialisaatio** (*tacitista tacitiin*). Suurin osa tiedosta on ihmisten aivoissa. Tietotyöläisen tavoitteena on löytää tapoja, joilla tällainen hiljainen tacit-tieto saadaan kerättyä. Sosialisaatio onkin tiedon jakamista erilaisten sosiaalisten vuorovaikutustapojen avulla. Tällaisia tapoja ovat esimerkiksi kasvokkain tapahtuva viestintä ja yhdessä työn tekeminen, jolloin muodostuu tyypillisesti kaksi roolia: ohjaajan ja ohjattavan. Toinen askel on **ulkoistaminen** (*tacitista explisiittiin*). Ulkoistamisen prosessi antaa hiljaiselle tiedolle näkyvän muodon ja muuttaa sen näin explisiitiksi. Nonaka ja Takeuchi määrittelevät sen ”*tietämyksen luomisen prosessiksi, jossa hiljainen tieto muuttuu näkyväksi ja saa metaforien, analogioiden, konseptien tai mallien muodon*”. Tässä muodossa tietoa ihmiset voivat yhdessä käsitellä sitä. Tämän prosessin käynnistäjäksi tarvitaan usein välittäjää, esimerkkinä journalisti, joka haastattelee tiedemiehiä ja näiden haastattelujen perusteella muokkaa heidän erikoistietämystään yleisempään ja ymmärrettävämpään muotoon. Kolmas askel on **yhdistäminen** (*explisiitistä explisiittiin*), joka yhdistää ulkoisen tiedon palasia uusiksi kokonaisuuksiksi. Mitään uutta tietoa ei varsinaisesti synny tässä vaiheessa. Yksittäiset tiedonjyväset on vain koottu yhteen varastoon loogisesti uuteen järjestykseen. Näin ollen niiden muodostama kokonaisuus avautuu ehkä järkevämmin ja yksiselitteisemmin. Neljäs askel on **sisäistäminen** (*explisiitistä tacitiin*), joka tapahtuu uuden tietämyksen sekoittuessa ja sulautuessa vanhaan. Joissain tapauksissa sisäistäminen liittyy vahvasti käytännön työssä tapahtuvaan oppimiseen. Sisäistäminen muokkaa tai kokoaa yhteisiä tai yksilöllisiä kokemuksia ja tietoja yksilöllisiksi ajatusmalleiksi. Nämä ajatusmallit vaikuttavat taas puolestaan yksilöiden ajatteluun ja toimintaan. Kun kaikki muutosvaiheet on käyty läpi, ovat myös organisaation tavat muuttuneet (Rekola 2006: 98–99).

4.7.2. The Fifth Discipline

Peter M. Senge esittää kirjassaan ”The Fifth Discipline” viisi tiedon- ja tietämyksenalaa (”*komponenttiteknologiaa*”), joiden kehittäminen on hänen mielestään tärkeää organisaatiolle, joka aikoo menestyä dynaamisessa ja haasteellisessa ympäristössä (Senge 1992: 6–10):

1. Systeemiajattelu (*Systems Thinking*) : Yksi kokonaisuus, joka on enemmän kuin osiensa summa ja jonka käyttäytymistä ei voi ymmärtää ja hallita vain tarkastelemalla ja ohjaamalla sen osia analyyttisesti (= tarvitaan holistinen lähestymistapa).
2. Henkilökohtainen hallinta, kompetenssi (*Personal Mastery*): Ihmisen tietojen, taitojen ja asenteiden kehittäminen ja hyödyntäminen. Organisaation oppimiskyky ja – potentiaali ei koskaan ylitä sen jäsenten oppimiskykyä ja – potentiaalia. Ihmisten halu kehittyä työssään perustuu siihen, että he kokevat työllään olevan merkitystä ja arvoa.
3. Mentaalimallit (*Mental Models*): Omien ajattelumallien tunnistaminen ja omien mielikuvien hahmottaminen ovat olennaisia asioita. Tällaisista malleista ja kuvista käytävä keskustelu mahdollistaa niiden muuttamisen ja kehittämisen
4. Yhteisen näkemyksen rakentaminen (*Building Shared Vision*): Kun organisaatio omaa yhteisen näkemyksen tulevaisuuden tavoitetilasta, on sillä myös mahdollisuus järkevästi toimimalla saavuttaa tuo tila.
5. Tiimioppiminen (*Team Learning*): Tiimioppiminen alkaa aina vuoropuhelusta (*dialogos*). Tiimin jäsenten oma kehittyminen on Sengen mukaan nopeampaa tiiminä kuin yksin toimittaessa. Organisaatio ei ole koskaan ”valmis” vaan se oppii ja muuttuu koko ajan joko huonompaan tai parempaan suuntaan.

Kehitys kulkee usein eteenpäin erilaisten kriisien kautta. Poliittisten ja maltillisten ratkaisujen kautta vaimennetaan vain oireita, mutta sairauden syyt jäävät jäljelle. Vasta tilanteen mentyä riittävän pahaksi hyväksytään se tosiseikka, että ilman suuria (joskus jopa vallankumouksellisia) muutoksia ei mitään todellista ja pysyvää parannusta saada aikaan. Kompleksisissa systeemeissä vaikutussuhteet eivät ole aina selkeästi havaittavissa, mutta ihmiset usein kuitenkin olettavat, että toisiaan ajallisesti lähellä olevat tapahtumat ovat riippuvuussuhteessa toisiinsa. Kyse voi kuitenkin olla pelkästä sattumasta, syyn ja seurauksen välillä voi olla suurikin ajallinen ja paikallinen ero. Jos ratkaisu on jo alkuaankin ongelmaan sopimaton, saadaan tilanne muuttumaan vain pahemmaksi tällaista ratkaisua käyttämällä (Senge 1992: 57–61).

4.7.3. Quality Improvement Paradigm ja Experience Factory

Victor R. Basili on Marylandin yliopiston emeritusprofessori, joka on tutkinut yli 40 vuotta ohjelmistonkehitystä, erityisesti sen hallintaan, arviointiin ja johtamiseen käytettyjä menetelmiä ja malleja. Quality Improvement Paradigm (QIP) on Basilin organisaatioiden oppimiseen ja ohjelmistoprosessin parantamiseen kehittämä malli. Basili näkee ohjelmistoyritysten suurena ongelmana olevan niiden sekavan ja epäselvän strategian, jota yrityksen henkilöstön on vaikea ymmärtää ja sisäistää.

Kuva 10. *Quality Improvement Paradigm (Basili b:70).*

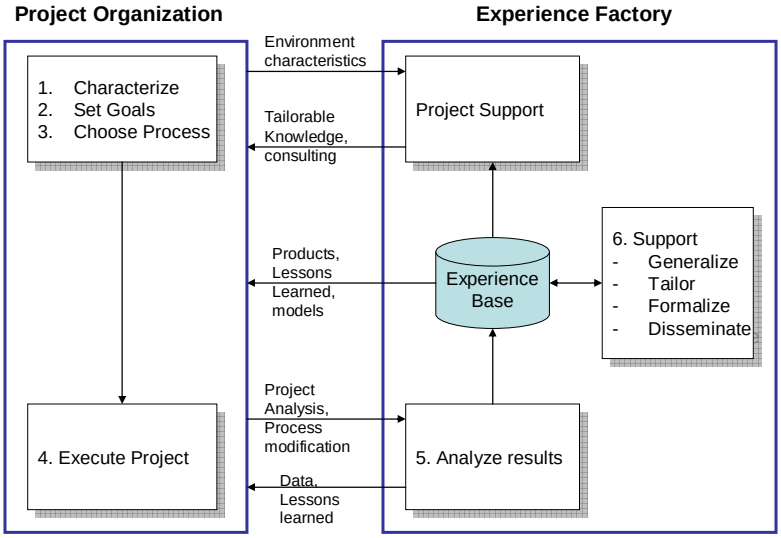
Ohjelmistoliiketoiminnassa yrityksen strateginen kyvykkyys rakentuu aina olemassa olevien tuotteiden ja organisaation osaamisen varaan. Basilin laadun parantamisen paradigma (QIP) on kuusivaiheinen:

1. Nykytilan mallintaminen
2. Tavoitteiden asettaminen
3. Prosessien, menetelmien, tekniikoiden ja työkalujen valinta
4. Projektin suorittaminen

5. Tulosten analysointi
6. Kokemusten paketointi ja tallentaminen

Jos verrataan toisiinsa QIP-mallia ja Demingin tunnetuksi tekemää PDCA-ympyrää, voidaan helposti havaita, että vaiheet 1. ja 2 vastaavat Plan-osaa, vaiheet 3. ja 4 vastaavat puolestaan Do-osaa, vaihe 5. on Check-osa ja vaihe 6. taas sama kuin Act-osa (Basili b: 69–70).

Experience Factory (EF) on kokeelliseen oppimiseen perustuva parantamismalli. Se on rakennettu ohjelmistokehityksen tietämyksen ja toiminnassa kerääntyneiden kokemusten uudelleenkäytölle. Organisaatio, joka hallinnoi Experience Factorya voi olla joko looginen tai fyysinen, mutta on kuitenkin aina pidettävä erossa varsinaisesta projektiorganisaatiosta. Experience Base (=tietämyskanta) on EF:n keskeinen osa. Tietämyksen hallinnan näkökulmasta katsottuna EF perustuu ensisijaisesti tiedon koodaamiseen ja tämän vuoksi se käsitteleekin pääasiassa eksplisiittiä tietoa. (Basili b: 75)



Kuva 11. *Experience Factory Organizations (Basili b:74).*

EF tallentaa, muokkaa ja uudelleen käyttää kaikenlaista ohjelmistokehityksen tietämystä ja kokemusta (esim. resurssimalleja, aikatauluja, muutoksia ja virheitä, prosessien määritelmiä, tuoterakenteita ja osia sekä laatutietoa). Tietämys ja kokemukset pyritään paketoimaan aina helpommin hyödynnettävään muotoon. Pakettien muoto voi vaihdella suuresti, ne voivat olla esimerkiksi muuttujien välisiä yhtälöitä, analysoituun dataan perustuvia histogrammeja tai muita kuvaajia sekä määrättyyn projektityyppiin, projektivaiheisiin ja/tai aktiviteetteihin liittyneitä opetuksia tai käytännön ohjeita. Basilin mukaan jokaista kehitysprojektia tulee ajatella kokeena, jonka tuloksena organisaation tieto lisääntyy. Tämä tieto tulee varastoida yrityksen EB- tietokantaan, josta se on mahdollista paremmin jakaa ja hyödyntää. Laadun parantamista pitääkin hänen mielestään mitata tuotteissa tapahtuneista näkyvistä parannuksista eikä vain prosessissa tapahtuneista muutoksista (Basili b: 69–74).

4.7.4. The Capability Maturity Model Integration

Alkuperäinen Capability Maturity Model (CMM) pohjautui Software Engineering Instituten ohjelmistonkehitystä varten tehtyyn prosessin kypsyysmalliin, joka auttaa kehittämään luotettavia, ennustettavia ja itseään korjaavia ohjelmistoprosesseja, joiden kautta puolestaan voidaan kehittää korkealaatuisia ohjelmistoja. Capability Maturity Model Integration (CMMI) on CMM:stä edelleen kehitetty prosessimalli, joka koostuu 22 eri prosessialueesta (*Process Area*). Mallia voidaan soveltaa kahdella eri lähestymistavalla, joko jatkuvalla (*Continuous Representation*) tai portaittaisella (*Staged Representation*). Portaittainen lähestymistapa käyttää kypsyystasoja (*maturity levels*) esittääkseen organisaation prosessien tilan kokonaisuudessaan arvioiden. Jatkuva lähestymistapa puolestaan hyödyntää kyvykkyystasoja (*capability levels*) esittääkseen organisaation yksittäisten prosessien tilan CMMI:n prosessialueiden kriteereillä arvioiden.

Taulukko 2. CMMI-mallin kyvykkyys- ja kypsyystasot (CMMI for Development, Version 1.3: 23)

Taso	Jatkuva lähestymistytapa	Portaittainen lähestymistapa
0	Keskeneräinen (*incomplete*)	-
1	Suoritettu (*performed*)	Pohjataso (*initial*)
2	Johdettu (*managed*)	Johdettu (*managed*)
3	Määritelty (*defined*)	Määritelty (*defined*)
4	-	Kvantitatiivisesti johdettu (*Quantitavely Managed*)
5	-	Optimoitu (*Optimized*)

Kyvykkyystasot eri prosessialueille ovat:

1. Keskeneräinen (taso 0). Prosessia ei ole olemassa tai se ei toimi kokonaisuudessaan.
2. Suoritettu (taso 1). Prosessi on olemassa ja toimii.
3. Johdettu (taso 2). Prosessia valvotaan ja ohjataan.
4. Määritelty (taso 3). Prosessin vaiheet ja aktiviteetit on tarkasti määritelty ja sen syötteet ja vasteet tunnetaan.

Kypsyystasot ovat vastaavasti seuraavat:

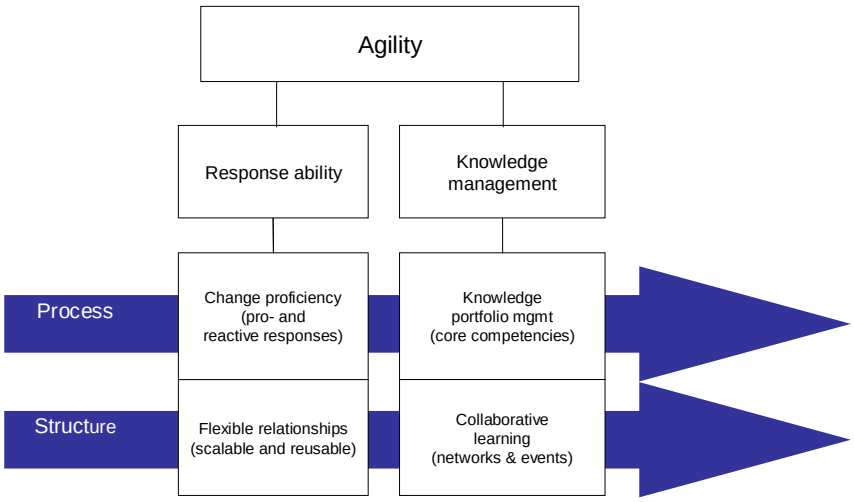
1. Pohjataso (*Initial*). Työntekijät toimivat ilman minkäänlaista analyysiä toimiensa seurauksista. Menestys perustuu paljolti sankarillisiin yksilösuorituksiin.
2. Johdettu (*Managed*). Päälliköt ottavat vastuun toiminnasta ja sen kehittämisestä. Menestys on joskus jopa tietoisen yhteistyön ansiota.
3. Määritelty (*Defined*). Työntekijöiden osaamista kehitetään sovittaen se yhteen liiketoiminnan strategian kanssa. Menestys syntyy suunnitelmien ja niiden toteutuksen johdosta.
4. Kvantitatiivisesti johdettu (*Quantitavely Managed*). Henkilöstöä valtuutetaan ja sen osaamista integroidaan. Ohjaus tapahtuu kvantitatiivisesti. Toiminnan tulokset ovat hyvin ennustettavissa.

5. Optimoitu (*Optimized*). Jatkuvan kehittämisen käytännöt ovat mukana kaikessa toiminnassa.

Parantamalla eri prosessialueiden suorituskykyä ja ratkomalla ilmitulleita ongelmia pyritään saavuttamaan yhä parempia tuloksia ja kehittymään alati korkeammalle tasolle. Tällä hetkellä CMMI-mallia on olemassa tällä hetkellä kolmeen erilaiseen liiketoimintaan muokattuna: hankintaan (*for acquisition*), kehitykseen (*for development*) ja palveluihin (*for services*) (CMMI Overview 2010: 8).

4.7.5. Muutoshalukkuus ja oppimiskyky

Se, miten hyvin organisaatiot pystyvät parantamaan ohjelmistoprosessiaan riippuu suuresti niiden toimintaympäristöstä ja henkilöstön muutoshalukkuudesta. Tekijät, jotka ovat vaikuttaneet myönteisesti joissain oloissa, saattavat osoittautua lähes tehottomiksi joissain toisissa. Innovaatioiden luominen ja tuotteistaminen ohjelmistoissa riippuu suuresti organisaation oppimiskyvystä, erityisesti silloin kun kyse on monimutkaisista dynaamisista toimintaympäristöistä. Jotkut ovat käsitelleet tutkimuksissaan organisaatioiden ketteryyttä yleisesti, kun taas toiset ovat keskittyneet uusien informaatioteknologiarakenteiden hallintaan ja niiden vaikutukseen organisaatioiden ketteryyden kehittämisessä. Ketteryydellä ajatellaan yleisesti olevan tekemistä enemmän liiketaloudellisen strategisen suuntautumisen kuin yrityksen koon kanssa.



Kuva 12. *Ketteryyden perustekijät Doven mukaan (Börjesson 2006: 12)*

Ketteryys edellyttää "*kykyä hallita ja soveltaa tietoa tehokkaasti niin, että organisaatio voi jatkuvasti menestyä muuttuvassa ja ennalta arvaamattomassa toimintaympäristössä*". Dove esittää organisaation ketteryyden kahdeksi tärkeäksi perustekijäksi kyvyn vastata erilaisiin muutoksiin (*response ability*) sekä tietämyksen hallinnan (*knowledge management*) ja sen tehokkuuden. Organisaation pitää kehittää näitä molempia perustekijöitä tasapainoisesti, jotta se voisi kehittyä ketterämmäksi ja menestyä paremmin nopeasti muuttuvassa toimintaympäristössä.

Kykyä vastata erilaisiin muutoksiin voidaan kehittää parantamalla muutoksen hyödyntämistä (tämä tarkoittaa prosessin kehittämistä) ja luomalla joustavia suhteita (tämä on puolestaan rakenteiden muuttamista). Tietämyksen hallintaa parannetaan vastaavasti kahden alueen kautta: tietämysportfolion hallinnan (prosessin kehittämisen) ja yhteisoppimisen (rakenteiden muuttamisen). Tietämysportfolion hallinta käsittää relevantin tiedon tunnistamisen, keräämisen, jakelun ja uudistamisen (Börjesson 2006: 11-12).

5. OHJELMISTOKEHITYKSEN HISTORIA

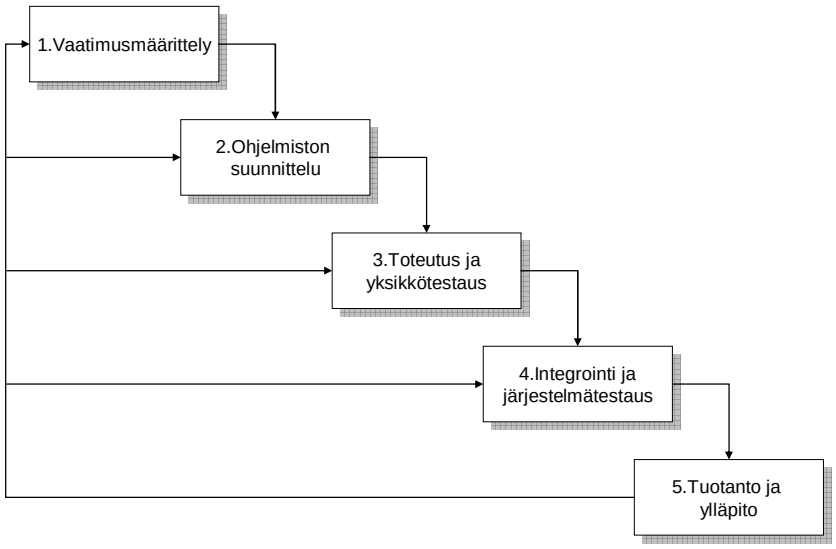
Tämä luku käsittelee ohjelmistokehityksen menetelmiä ja malleja 1970-luvun alusta 2000-luvulle asti.

5.1. Perinteiset kehitysmallit

On olemassa monia eri malleja ohjelmistojen kehittämiseen ja valmistukseen, mutta periaatteessa kaikista niistä voidaan tunnistaa seuraavat vaiheet: määrittely, suunnittelu ja toteutus, verifiointi ja jatkokehitys. Jokaisen ohjelmiston toiminnallisuus ja sen käytön rajoitukset on määriteltävä, ohjelmiston on suunniteltava ja toteutettava määritysten mukaisesti, toteutetun ohjelmiston vastaavuus määriteltyyn ja suunniteltuun on tietenkin verifioitava. Kun uuden ohjelmiston kehityshanke on ohi, voidaan alkaa suunnitella sen jatkokehitystä (Sommerville 2009: 28).

5.1.1. Vesiputousmalli

Winston W. Royce kirjoitti vuonna 1970 artikkelin (”Managing the Development of Large Software Systems”) jossa käsiteltiin ensimmäisen kerran suurten ohjelmistojen kehittämistä hallitusti yhtenä kokonaisuutena. Royce oli johtanut esittelemänsä prosessimallin yleisemmistä järjestelmien kehittämisessä käytetyistä prosesseista. Hänen alkuperäisessä mallissaan oli alkuaan seitsemän vaihetta: järjestelmävaatimukset, ohjelmistovaatimukset, analyysi, ohjelmien suunnittelu, koodaus, testaus ja käyttö. Roycen malli tuli myöhemmin tunnetuksi vesiputousmallin nimellä, vaikka Royce itse ei kyseistä nimeä koskaan käyttänytkään. Alkuperäisestä mallista on myöhemmin kehitetty lukuisia uusia versioita.



Kuva 13. *Vesiputousmalli (Sommerville 2009: 30).*

Tyypillisen vesiputousmallin vaiheet ovat seuraavat (Sommerville 2009: 30–31):

1. *Vaatimusmäärittelyssä* järjestelmän (ohjelmiston) palvelut, rajoitukset ja tavoitteet tunnistetaan ja kirjataan käyttäjiltä saatujen tietojen pohjalta. Tyypillisesti tämän vaiheen tuloksena syntyy dokumentti, järjestelmäkuvaus, joka toimii kaikkien seuraavien vaiheiden ohjausvälineenä.
2. *Suunnitteluvaiheessa* määritellyt vaatimukset kohdistetaan ohjelmiston eri osille. Näin saadaan luotua ohjelmiston ylätason arkkitehtuuri, joka määrittää eri osien toiminnot ja niiden väliset suhteet.
3. Määriteltyjen ohjelmiston osien *toteutus* on konkreettista ohjelmointia eli koodausta. Toteutuksen tuloksena syntyvä koodi testataan aluksi *yksikkö-testauksessa*, jolloin varmistetaan, että jokainen ohjelmiston osa vastaa määrittelyä.
4. *Integraatiovaiheessa* eri ohjelmaosat tai yksiköt yhdistetään toimivaksi kokonaisuudeksi, lopulliseksi ohjelmistoksi. Tämä kokonaisuus testataan *järjestelmätestissä*, jossa varmistetaan että toteutettu ohjelmisto ja sen toiminta vastaavat määriteltyjä vaatimuksia.

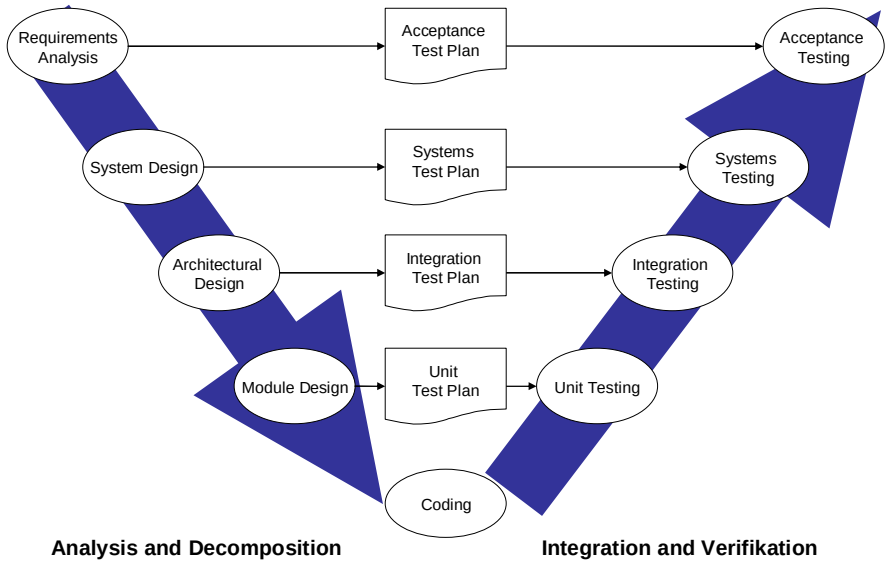
5. Hyväksytyn järjestelmätestin jälkeen ohjelmisto toimitetaan asiakkaalle otettavaksi *tuotantokäyttöön* ja *ylläpitovaiheen* katsotaan alkaneen. Tuotanto- ja ylläpitovaiheessa asiakas hyödyntää ohjelmistoa normaalissa toiminnassaan.

Ohjelmiston käyttöönoton jälkeen löytyy lähes aina jotain virheitä, poikkeamia tai piirteitä, jotka eivät vastaa asiakkaan odotuksia ja toiveita. Ylläpitovaiheen aikana ohjelmistoa pyritäänkin jatkokehittämään asiakkailta ja markkinoilta saadun informaation mukaisesti. Periaatteessa jokaisen vaiheen tuloksena syntyy yksi tai useampia dokumentteja, jotka hyväksytään formaaleissa hyväksymiskokouksissa. Vasta tällaisen hyväksymisen jälkeen voidaan seuraava vaihe aloittaa (Sommerville 2009: 31).

Royce ehdotti alkuperäisessä artikkelissaan, että suuremmissa hankkeissa noudatettaisiin pilotointia. Esimerkkinä tästä hän mainitsi 30 kuukauden hankkeeseen sisältyvän 10 kuukauden pilottijakson, jonka jälkeen voitaisiin tehdä vielä tarvittavia muutoksia. Hän suositteli myös, että ohjelmiston kriittisiä osia olisi jo koekäytetty asiakkaan ympäristössä mahdollisimman todellisissa olosuhteissa. Roycen alkuperäiseen malliin sisältyi täten jo ajatus iteratiivisesta kehittämisestä (Royce 1970: 334).

5.1.2. V-malli

V-malli on oikeastaan suora jatkokehitelmä vesiputousmallista. V-malli on laajalti käytössä Saksan liittovaltion eri hallintoviranomaisilla ja puolustusvoimissa. Mallin konseptia kehitettiin samanaikaisesti 1980-luvulla sekä Saksassa että Yhdysvalloissa ja sen julkaisi ensimmäisenä Paul E. Brook vuonna 1986. V-mallissa jokaista analyysi- ja dekompositiovaihetta vastaa aina yksi integraatio- ja testausvaihe. Samaan aikaan kun ohjelmistoa määritellään ja suunnitellaan, on luotava verifikaatiosuunnitelmat ja proseduurit, joiden avulla pyritään varmentamaan kehitystyön tuotosten toiminta ja virheettömyys. V-mallissa oletetaan, että jokainen kehitysvaihe tarkastetaan ja hyväksytään, ennen kun edetään seuraavaan vaiheeseen. Edellisen vaiheen tuotokset toimivat aina syötteinä seuraavalle vaiheelle (V-Model).



Kuva 14. *V-mallin eri vaiheet (V-Model)*

V-mallin analyysi- ja dekompositiovaiheet

1. Vaatimusten määrittely (Requirement Analysis) Kehitysprosessin ensimmäisessä vaiheessa tunnistetaan ja päätetään ne vaatimukset, joita tuotteelle asetetaan. On tärkeää, että vaatimusmäärittelyn aikana keskitytään siihen, mitä tuotteen tai ohjelman halutaan tekevän eikä siihen miten se tullaan rakentamaan. Keskustelu , käyttäjien kanssa on käytävä yleisellä tasolla, sillä ratkaisujen tarkka suunnittelu tapahtuu myöhemmissä vaiheissa. Tämän vaiheen aikana laadittavassa määrittelydokumentissa on tyypillisesti informaatiota kehitettävän järjestelmän toiminnoista, suorituskyvystä, turvallisuudesta, tietovarastoista ja liittymistä. Määrittelydokumentti ohjaa vahvasti hyväksymistestauksen (*acceptance test*) suunnittelua.

2. Järjestelmän suunnittelu (System Design) Järjestelmätason toimintojen suunnittelu tehdään seuraavaksi. Syötteenä toimii pääasiassa edellisen vaiheen tuloksena syntynyt määrittelydokumentti. Järjestelmän toimintaa hahmotetaan erilaisilla kaavioilla ja tietovarastojen kuvauksilla. Jos jokin aiemmin määritellyistä vaatimuksista päätetään jättää pois järjestelmästä, on myös määrittelydokumentti ja hyväksymistestaussuunnitelma päivitettävä tältä osin.

3. Arkkitehtuurin suunnittelu (Architecture Design) Järjestelmän tai ohjelmiston arkkitehtuurin suunnitteluvaiheessa lyödään lukkoon toteutuksen rakenne, loogiset yksiköt ja niiden päätehtävät. Loogisten yksiöiden väliset yhteydet ja niiden rajapintojen toiminta määritellään.

4. Moduulisuunnittelu (Module Design) Tässä vaiheessa edellisessä arkkitehtuurisuunnittelussa nimettyjen yksiköiden (*moduulien*) sisäinen toiminta kuvataan ja päätetään. Jokainen moduuli on tämän vaiheen jälkeen valmis koodattavaksi sellaisenaan.

V-mallin integraatio- ja verifikaatiovaiheet

1. Yksikkötestaus (Unit Testing) Ohjelmistojärjestelmässä yksikkö on sen pienin osa, joka voidaan testata erillään muusta järjestelmästä. Yksikkötestausvaiheessa kaikki järjestelmän yksiköt pyritään testaamaan moduulitestaussuunnitelman mukaisesti.

2. Integraatiotestaus (Integration Testing) Tässä verifikaatiovaiheessa testataan erilliset yksiköt yhdessä. Tavoitteena on havaita niiden liittymien ja rajapintojen viat ja puutteet ja varmistaa, että erilliset yksiköt voivat toimia yhtenä kokonaisuutena.

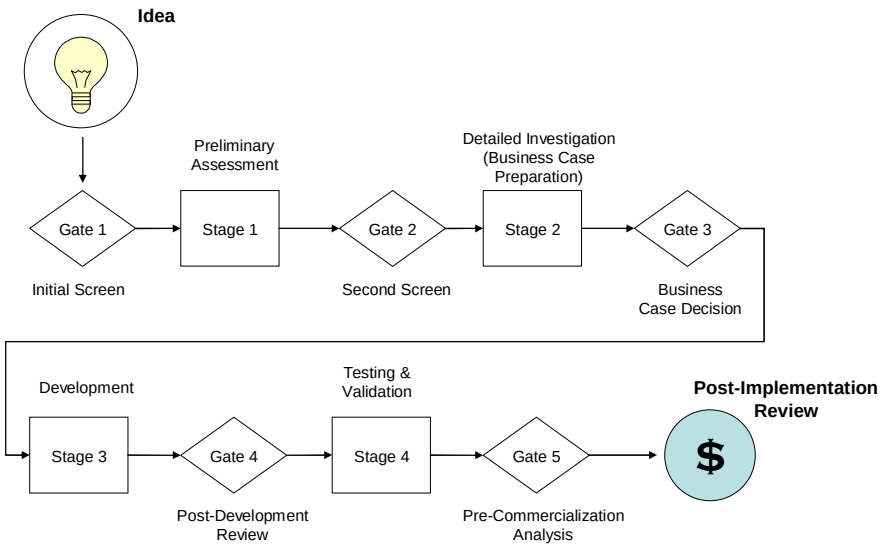
3. Järjestelmätestaus (System Testing) Kun integraatiotesti on saatu tehtyä hyväksytysti, on varmistettava, että toteutettu järjestelmä täyttää määritellyt vaatimukset. Järjestelmätestaus suoritetaan testisuunnitelman mukaisesti, yleensä järjestelmän toimittajan tiloissa.

4. Hyväksymistestaus (Acceptance Testing) Hyväksymistestauksessa valmista tuotetta ja sen toimintaa verrataan vaatimusdokumentteihin. Havaitut poikkeamat

dokumentoidaan ja luokitellaan. Hyväksymistestin tarkoituksena on varmistaa, että järjestelmä on valmis ja sopiva käyttöönotettavaksi. V-malli on vesiputousmallia kehittyneempi siinä suhteessa, että jokaista kehitysvaihetta vastaa aina verifikaatiovaihe, jolla pyritään varmistamaan tuotosten toiminta ja kelvollisuus (V-Model).

5.1.3. Stage-Gate-malli

Stage-Gate-malli on Robert G. Cooperin luoma uuden tuotekehityksen (*New Product Development = NPD*) prosessimalli. Uusien tuotteiden julkaiseminen, nopeasti kehittyvien teknologioiden hallinta ja innovaatioiden tuotteistaminen vaativat yrityksiltä paljon. Cooperin Stage-Gate-mallissa pyritään nopeuttamaan tuotteiden kehitysprosessia samalla kun tavoitteena on virheellisten päätösten ja huonon laadun välttäminen. Mitä aiemmin virhe havaitaan sitä pienemmät sen aiheuttamat virhekustannukset.



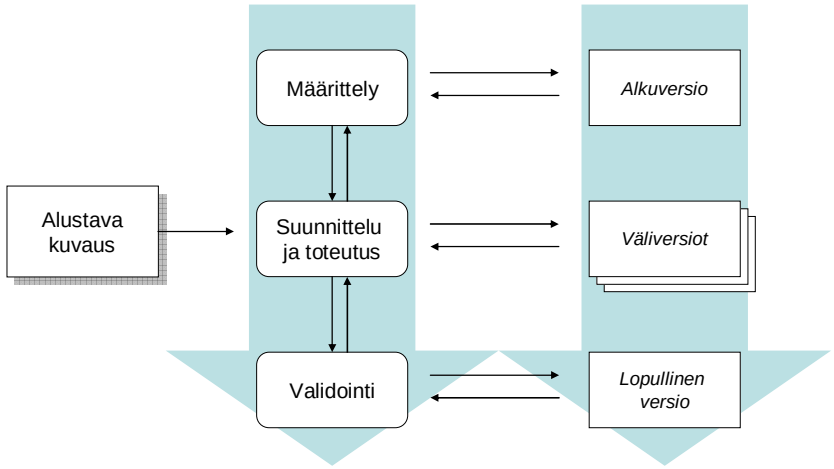
Kuva 15. *Stage-Gate-malli (Cooper 1990: 46)*

Tavallisesti mallissa on 4–7 stage-gate -paria, niiden määrä riippuu paljolti kehitettävän tuotteen ja kehitysorganisaation kompleksisuudesta. Jokainen portti (*gate*) toimii lähtöpaikkana seuraavalle etapille (*stage*). Päästäkseen portista läpi on hankkeen täytettävä asetetut vaatimukset (*criteria*). Jokaiselle portille on määritelty omat vaatimuksensa ja tuotoksensa (*deliverables*). Hanketta arvioidaan aina ensisijaisesti liiketoiminnan perspektiivistä: vaikka hanke olisi laadullisesti ja teknisesti erinomainen, mutta ei täytä kuitenkaan liiketoiminnallisia kriteerejä, ei sitä saa päästää eteenpäin prosessissa. Varsinainen kehitystyö tehdään aina etapeilla, portit toimivat vain tarkastuspisteinä (*checkpoints*) joilla arvioidaan hankkeen tilannetta ja laatua (Cooper 1990: 45–46).

Porttikokouksen (*gate meeting*) pohjalta voidaan hankkeen suhteen tehdä neljä erilaista päätöstä: hanke päästetään eteenpäin (GO), hanke lopetetaan (KILL), hanke pysäytetään (HOLD) tai hanke palautetaan takaisin (RECYCLE). Porttikokousten byrokratia pitäisi pyrkiä pitämään minimissään, sillä muuten näistä tarkastuspisteistä ja niiden valmisteluista tulee itse kehitystyötä tärkeämpi ja suurempi prosessin osa. On aina muistettava, että tuotteen tai hankkeen tarkastaminen ei tuota mitään lisäarvoa asiakkaan näkökulmasta katsottuna (Cooper 1990: 46).

5.2. Inkrementaalinen ja iteratiivinen kehittäminen

Inkrementaalinen ja iteratiivinen kehitystapa on olennainen osa kaikissa ketterissä kehitysmenetelmissä. Tällaisella syklisellä kehitysmallilla pyritään korjaamaan perinteisen vesiputousmallin puutteita. Jokaiseen kehitysvaiheeseen liittyy aina paluureitti edelliseen vaiheeseen. Tällä mahdollistetaan useampien kierrosten l. iteraatioiden hyödyntäminen ohjelmistokehitystyössä. Näin on mahdollista suunnitella ja rakentaa ohjelmistoja pienemmissä osissa ja siten hallita kehitystyönriskejä paremmin. Jokainen iteraatio vie aina tuotetta vähän eteenpäin ja jokainen lisäys (*inkrementti*) tuo siihen uutta haluttua toiminnallisuutta (Sommerville 2009: 32).



Kuva 16. *Inkrementaalinen kehitysmalli (Sommerville 2009: 33).*

Inkrementaalinen kehitysmalli vastaa paremmin kuin perinteinen vesiputous- tai V-malli sitä tapaa, miten tavalliset ihmiset ratkaisevat päivittäisessä toiminnassaan syntyviä ongelmia. On harvinaista, että ongelman kokonaisratkaisu löytyy heti ja että kehitystyössä voidaan edetä jatkuvasti vaiheesta seuraavaan. Tyypillisesti ongelman ratkaisua rakennetaan pala palalta kokeillen ja korjaillen, tarvittaessa palataan takaisin ja tehdään tarvittava muutos ja tällä lailla ratkaisua parannetaan kierros kierrokselta. Muutosten tekeminen asiakasvaatimuksiin on halvempaa kuin jäykemmissä malleissa, sillä tarvitaan vähemmän dokumentointia ja hallintoa vaatimuksien muuttuessa. Asiakkaiden on myös helpompi antaa palautetta ja kehitysehdotuksia toimivan demo-ohjelmiston perusteella kuin vain tyytyä kommentoimaan määrittely- ja suunnitteludokumentteja. Kehitystyö on nopeampaa ja myös asiakas pystyy hyödyntämään ohjelmistoa vaikka se ei olisikaan toiminnallisuudeltaan täysin valmis (Sommerville 2009: 33).

5.3. Agile Manifesto

Ryhmä IT-alan vaikuttajia antoi vuonna 2001 manifestin, jolla he halusivat julistaa uuden (ja paremman) ohjelmistokehityksen sanomaa ja esiinmarssia. Ryhmä painotti turhan byrokratian ja painolastin hylkäämistä ja nopeuden ja reagointikyvyn merkitystä sekä yhteistoiminnan merkitystä. Heidän julistuksensa on vaikuttanut suuresti ohjelmistoalan ammattilaisten ja opiskelijoiden ajatteluun. Jokainen, joka on ollut mukana suuremmassa järjestelmä- tai ohjelmistoprojektissa on tuskaillut samanlaisten ongelmien kanssa ja ihmetellyt, eikö todellakaan ole olemassa parempia tapoja ja välineitä saada työ tehdyksi.

Ryhmän tunnetuimman edustajan, Kent Beckin mukaan kaikki, joka ei suoraan edistä ohjelman määrittelyä, koodin tekemistä tai testausta, on turhaa ohjelmistotuotannossa. Hänen mukaansa ohjelmoijan kannattaakin kysyä aina itseltään, mitä muut ihmiset yrittävät oikein hänelle myydä, jos heidän ehdottamansa toimenpiteet eivät kuulu johonkin näistä kategorioista. Ketterän ohjelmistokehityksen manifesti kiteyttää selkeästi ja yksinkertaisesti nämä ajatukset: tuodaan esiin yksilöt ja vuorovaikutus prosessien ja työkalujen sijaan, keskitytään tekemään ohjelmistoja dokumenttien sijaan, korostetaan yhteistyötä asiakkaan kanssa sopimusneuvottelujen sijaan ja vastataan muutokseen suunnitelman noudattamisen sijaan.

> *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.*

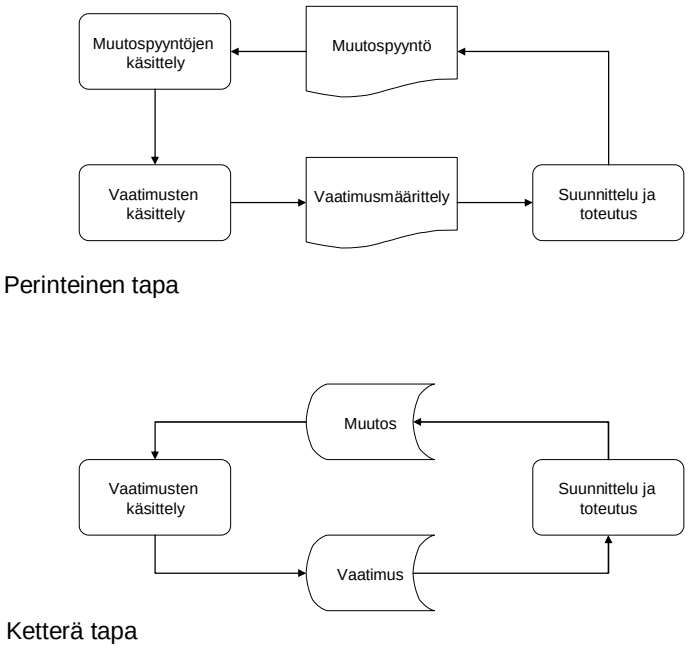
(Agile Manifesto)

5.4. Ketterät kehitysmenetelmät

Kaikki ketterät kehitysmenetelmät rakentavat samoille, yhteisille periaatteille ja kaikista niistä voidaan myös havaita ja tunnistaa yhteisiä tekijöitä, joita jo ketterässä manifestissa kuulutettiin. Ketterien kehitysmenetelmien yhteisiä periaatteita ovat seuraavat (Sommerville 2009: 60):

1. Asiakkaiden ja käyttäjien osallistuminen kehitystyöhön (*Customer involvement*)
2. Inkrementaalinen toimitustapa (*Incremental delivery*)
3. Huomio ihmisiin prosessin sijaan (*People not process*)
4. Muutokseen pyrkiminen sen vastustamisen sijaan (*Embrace change*)
5. Yksinkertaisuuden painottaminen (*Maintain simplicity*)

Käyttäjien osallistuminen muodostaa perustan tehokkaalle ja vaikuttavalle projektille. Projektiryhmän valtuuttaminen tekemään tarvittavat päätökset ilman että odotellaan projektin ohjausryhmän tai muiden korkeampien tahojen kokoontumista. Käyttäjien palaute ohjaa ja tarkentaa kaikkea kehitystyötä kohti tehokasta liiketoimintaratkaisua. Kaikkien projektin osallistuvien välisen viestinnän ja yhteistyön tulee olla tehokasta ja vaikuttavaa. Ihmisten asenteiden merkitys korostuu välttämättä silloin kun työtä tehdään tiiviisti ja nopeasti muuttuvassa ympäristössä. Ohjelmistotuotteiden toimituksen pitää tapahtua säännöllisin väliajoin. On kaikkien kannalta parempi toimittaa aina heti jotain riittävän hyvää kuin odotella sitä. että toimitetaan kaikki osat täydellisinä vasta aivan projektin lopussa. Tuotoksen hyväksymisen pääkriteerinä on oltava sen kyky tyydyttää ja vastata asiakastarpeisiin. Kaikkien kehityksen aikaisten muutosten tulee palautettavissa, joten versionhallinnan työkalujen ja käytäntöjen on oltava tällöin hyvällä tasolla. Korkean tason rajaukset ja vaatimukset tulee hyväksyä ja ”kiinnittää” ennen projektin alkua ja testausta tulee suorittaa koko projektin ajan sen sijaan, että testaus tapahtuisi vasta projektin lopussa (Sommerville 2009: 60–61).



Kuva 17. *Vaatimusten määrittely perinteisellä ja ketterällä tavalla.*

Ketterän vaatimusten määrittelyn ja perinteisen vaatimusten määrittelyn ero näkyy selvimmin siinä, että jokainen muutos käsitellään nopeasti ilman suurempaa byrokratiaa. Perinteinen muutospyyntömenettely hidastaa ja jäykistää kehitystä liikaa. Vaatimusten ja muutosten käsittely kokonaisina dokumentteina lisää työmäärää. Noudatettaessa ketterää tapaa pitää työkalujen pitää tukea valittua prosessia ja mahdollistaa nopeiden muutosten tekemisen yksittäisiin vaatimuksiin (Sommerville 2009: 63).

Scott Amblerin mukaan ketterät kehitysmenetelmät tuovat ohjelmistonkehityksen ammattilaisten päivittäiseen työhön seuraavat kolme muutosta:

1. Ketterä ohjelmistoprosessi vaatii vähemmän laadunvarmistusta. Parempi prosessin suorituskyky aiheuttaa vähemmän virheiden korjausta.
2. Suunnittelijoiden on totuttava epätäydellisiin artefakteihin. Päätöksiä on tehtävä yhä enemmän ja enemmän osittain epävarman tiedon pohjalta.

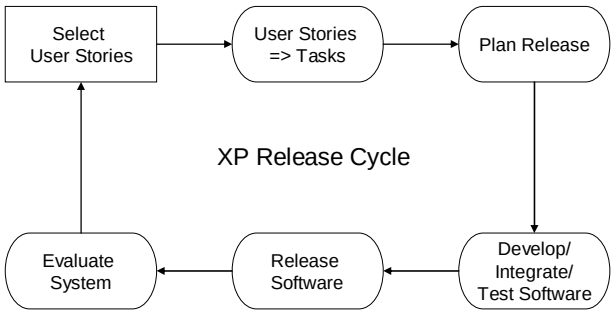
3. Kehitystiimin jäsenten on muututtava eli oltava sekä ohjelmistojen että toiminnan kehittämisen ammattilaisia (generalisteja ja spesialisteja samanaikaisesti), yhden ohjelmointikielen hyvä osaaminen ei enää riitäkään.

Ambler näkee laadun merkityksen ketterissä kehityshankkeissa elintärkeänä, vaikka dedikoidut laatuaktiviteetit ja - roolit vähenevät. Laatu on rakennettava kehitysprosessin sisään ja tämän kautta on mahdollista saada aikaan parempia ohjelmistotuotteita (Ambler 2005: 39).

5.4.1. Extreme Programming

Extreme Programming (XP) on tunnetuin ja laajalle levinnein ketteristä kehitysmenetelmistä. Nimi on Beckin vuonna 1999 lanseeraama ja kuvastaa sitä asennetta, jolle koko menetelmä rakentuu (”*pienen ruohonjuuritason ohjelmoijan kapina jäykkää byrokratiaa vastaan*”). Hyvien käytäntöjen ja periaatteiden avulla on tavoitteena viedä ohjelmointia uudelle äärimmäiselle tasolle. XP:n avulla on mahdollista nopeuttaa kehitystyötä monikertaisesti. Joissain tapauksissa on ollut mahdollista kehittää, integroida ja testata uusi ohjelmistoversio yhden päivän aikana (Sommerville 2009: 64).

XP:n kehityskierros (*Release Cycle*) lähtee liikkeelle vaatimusten määrittelystä, joka tapahtuu skenaarioiden (*User Stories*) avulla. Näistä käyttäjäkertomuksista muodostetaan ohjelmiston tehtävät (*tasks*). Julkistuksen kattavuuden ja aikataulun suunnittelun jälkeen siirrytään varsinaiseen ohjelmistonkehitykseen. Ohjelmiston kehittämisen, integroinnin ja testauksen jälkeen on vuorossa ohjelmiston julkistaminen. Julkistamisen jälkeen järjestelmän arvioidaan. Arvioinnista saadun palautteen pohjalta voidaan aloittaa sitten uusi kehityskierros (Sommerville 2009: 65).



Kuva 18. *XP Release Cycle (Sommerville 2009: 65).*

Monesti on sanottu, että XP ei ole oikeastaan mikään ”kehitysmenetelmä” vaan pikemminkin joukko käytäntöjä, joita ohjelmiston kehityksessä enemmän tai vähemmän tehokkaasti hyödynnetään. Tällaisia käytäntöjä ovat seuraavat (Sommerville 2009: 66):

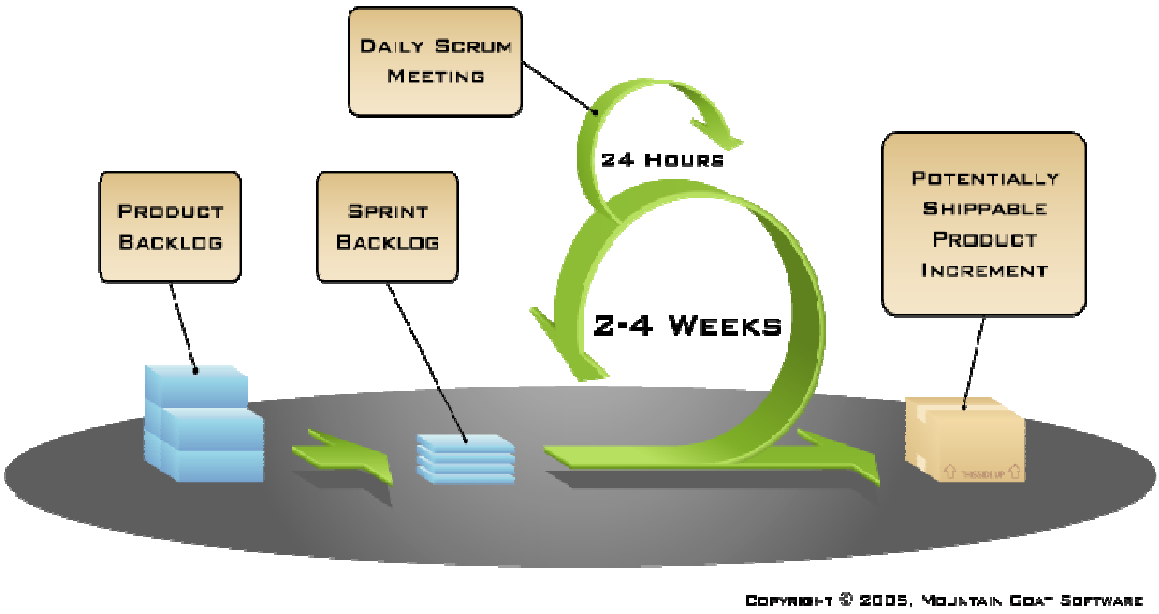
1. Inkrementaalinen suunnittelu (*Incremental planning*). Vaatimukset määritellään käyttäjäkertomusten (korttien) avulla. Jokaiseen julkaisuun valitaan toteutettavaksi osa korteista. Korttien ”tarinat” muunnetaan ”tehtäviksi”.
2. Pienet julkaisut (*Small Releases*). Pyritään tiheään julkaisufrekvenssiin, joka taas edellyttää sopivan pieniä muutoksia ja lisäyksiä ohjelmiston toiminnallisuuteen.
3. Yksinkertainen suunnittelu ja rakenne (*Simple Design*). Ainoastaan olennainen ja vaadittava toiminnallisuus toteutetaan – yksinkertainen on kaunista.
4. Testilähtöinen kehittäminen (*Test-first development*). Käytetään, jos mahdollista, aina automaattista testausta ja testit kirjoitetaan ennen toiminnallisuuden toteutusta (koodausta).

5. Refaktorointi (*Refactoring*). Kaikkien kehittäjien tulee refaktoroida koodiaan l. jatkuvasti hallitusti ja kurinalaisesti parantaa olemassa olevaa koodia muuttamalla sen sisäistä rakennetta koskematta sen ulkoiseen käyttäytymiseen.
6. Pariohjelmointi (*Pair programming*). Kehittäjät työskentelevät pareittain ja tukevat toisiaan, testaavat ja arvioivat toistensa työtä jatkuvasti.
7. Yhteinen omistus (*Collective ownership*). Kaikki parit tekevät työtä kaikkien ohjelmiston osien kanssa ja vastuu kaikesta koodista on kaikilla. Kenenkään ei sallita yksin ”omistavan” jotain koodin osaa. Jokainen voi tarvittaessa muuttaa mitä tahansa.
8. Jatkuva integrointi (*Continuous integration*). Kun tehtävä on toteutettu, se integroidaan järjestelmään. Integroinnin jälkeen kaikki järjestelmän yksikkötestit on läpäistävä.
9. Kestävä kehitystahti (*Sustainable pace*). Suuria ylityömääriä ei pidetä hyväksyttävinä vaan työ pyritään tekemään normaalina työaikana. Ylitöissä tehdyn koodin laatu on lähes poikkeuksetta huonompaa ja tuottavuuskin heikompaa kuin tavallisena työaikana.
10. Asiakas mukana kehityksessä (*On-site customer*). Järjestelmän loppukäyttäjien edustaja pyritään saamaan mukaan kehitystiimiin. Hänen vastuullaan on tällöin käyttäjien vaatimusten mukaan tuominen päivittäiseen kehitystyöhön.

5.4.2. Scrum – ketterää projektinhallintaa

Scrum on projektinhallinnan malli/viitekehys, jota käytetään yleisimmin ketterässä ohjelmistokehityksessä. Sitä voidaan soveltaa kuitenkin myös muidenkin projektien hallinnassa. Ensimmäisinä ajatuksen scrumin kaltaisesta kehitysprosessista esittivät Nonaka ja Takeuchi vuonna 1986 ilmestyneessä artikkelissaan ”The New Product Development Game”. He kuvasivat uudentyyppisen lähestymistavan tuotekehitykseen, jossa yksi ainoa useammasta toiminnon työntekijöistä koottu (*cross-functional*) ryhmä suorittaa koko kehitysprosessin alusta loppuun. Tällainen holistinen lähestymistapa edellytti projektin toteuttamista vahvasti toisiinsa lomittuneiden vaiheiden kautta. Toiminnan ajatusmallina toimi rugby-joukkue, joka ryhmänä pyrkii etenemään ja

toimimaan tiiviisti yhtenä kokonaisuutena. Joukkue on tällöin vahvasti itseohjautuva ja – organisoituva sekä pystyy nopeasti mukautumaan toiminnan ja ympäristön muutoksiin (Nonaka and Takeuchi 1986: 2-3).



Kuva 19. *Scrumin yleinen prosessikaavio (Scrum Overview 2006).*

Scrum-menetelmässä kuvataan toteutettava työ ominaisuuslistaan l. tuotteen työlistaan (*product backlog*). Siihen on koottu kaikki tarvittavat työt tuotteen tavoitellun lopputilan saavuttamiseksi. Tätä työlistaa lähdetään purkamaan niin kutsuttujen sprinttien avulla. Jokaisen sprintin alussa pidetään sprintin suunnittelupalaveri (*sprint planning meeting*), jossa asiakas / tuotteen omistaja (*product owner*) määrittelee tärkeysjärjestyksen työlistassa luetelluille töille. Tämän jälkeen Scrum-tiimi valitsee tulevan sprintin aikana tehtävissä olevat työt. Työt siirretään tuotteen työlistasta sprintin työlistaan (*sprint backlog*). Sprintit kestävät yleensä kahdesta neljään viikkoa (Schwaber and Sutherland 2011: 8).

Scrum-mallin mukaisia toimijoita ovat Scrum Master, tuotteen omistaja ja kehitystiimi. Scrum Masterin vastuulla on varmistaa, että tiimi noudattaa ennalta sovittuja arvoja, käytäntöjä ja sääntöjä. Scrum Master myös auttaa tiimiä tekemään parhaansa ympäristössä, joka ei välttämättä ole optimoitu kompleksisten tuotteiden kehittämiseen.

Hän myös edustaa Scrum – tiimiä ulospäin. Scrum Master ei ole projektipäällikkö vaan enemmänkin tiimin valmentaja ja fasilitaattori. Tuotteen omistaja on se henkilö, joka vastaa tuotteen kehitysjonosta ja kehitystiimin työn arvon maksimoimisesta. Tämä henkilö ylläpitää tuotteen kehitysjonoa ja varmistaa, että se on kaikkien nähtävissä. Tuotteen omistajan on aina oltava yksi henkilö (Schwaber and Sutherland 2011: 6–7).

Scrum-hankkeissa on kehitystiimien oltava varsin itseohjautuvia. Kukaan esimies ei tule kertomaan kehitystiimille, miten tuotteen työlista tulisi muuttaa julkaisukelpoiseksi ohjelmaksi. Kehitystiimi ratkaisee asiat itsenäisesti ja jokainen kehitystiimin jäsen soveltaa omaa osaamistaan esiin nouseviin haasteisiin. Kehitystiimin optimaalinen koko on 7 henkilöä (+- 2 henkilöä). Alle 5 hengen kehitystiimissä on koettu tapahtuvan vähemmän yhteistyötä ja sen tuloksena saadaan vähemmän tuottavuushyötyjä. Lisäksi pieni kehitystiimi saattaa törmätä osaamispulaan jossain sprintin osassa eikä silloin pysty tuottamaan julkaisukelpoista ohjelmaa. Jos taas kehitystiimissä on yli 9 henkilöä, kasvaa tarvittavan koordinoinnin määrä liikaa saavutettuun hyötyyn nähden (Schwaber and Sutherland 2011: 6).

5.4.3. Lean Software Development

Mary ja Tom Poppendieckin kirja ”Lean Software Development – An Agile Toolkit” yhdistää ketterän ohjelmistokehityksen ja lean ajattelun. He määrittelevät seuraavat Lean Software Developmentin periaatteet (Poppendieck 2003):

1. Poista jäte eli kaikki turha (*Eliminate waste*)
2. Edistä oppimista (*Amplify Learning*)
3. Päätä asioista mahdollisimman myöhään (*Decide as late as possible*)
4. Toimita nopeasti (Deliver as fast as possible)
5. Valtuuta tiimi (*Empower the team*)
6. Rakenna laatua (Build Integrity in)
7. Näe (ja ajattele) kokonaisuus (*See the whole*)

1. Poista jäte (kaikki turha)

Lean ajattelun mukaiset ohjelmistokehityshankkeet alkavat aina nykyisen tilanteen arvioinnilla: jäte on tunnistettava ja ryhdyttävä välittömästi viestimään ongelmasta. Yksinkertaisesti ilmaistuna: kaikki mikä ei lisää tuotteen ja ohjelmiston arvoa asiakkaan näkökulmasta katsottuna on jätettä. Poppendieck määrittelee seuraavat ohjelmistokehityksen seitsemän jätettä, mudaa:

1. Osittain tehty työ (kehitysprosessin “varasto”)
2. Ylimääräiset prosessit (nämä voidaan helposti löytää, kun ollaan tekemisissä dokumentaatiokeskeisen kehityksen kanssa)
3. Ylimääräiset piirteet, ”featuret” (kehitetään vain ja ainoastaan sitä, mitä asiakas haluaa juuri nyt)
4. Tehtävien vaihtelu (jokaisen tulisi tehdä vain yhtä asiaa kerrallaan).
5. Odottaminen (ohjeiden, informaation vuoksi)
6. Luovutukset (paljon hiljaista tietämystä katoaa)
7. Viat (varsinkin ne viat, joita testit eivät saa kiinni nopeasti)

On tärkeää seurata sitä, miten arvo virtaa läpi kehitysprosessin asiakasvaatimuksesta toimitukseen asti. Jos asiakasvaatimus odottaa yhdessä jonossa käsittelyä ja hyväksymistä, toisessa jonossa suunnittelua, kolmannessa toteutusta, neljännessä testausta, jne., ei kai ole mitenkään yllättävää että eteneminen kehitysprosessissa on hidasta. Parempi tapa olisi, että yksi tiimi (tai solu) ottaa hoitaakseen aina yhden vaatimuksen alusta loppuun asti (kehdosta hautaan) nopeasti ja ilman turhia keskeytyksiä.

2. Edistä oppimista

Ohjelmistonkehitys on oppimispeliä: kuvittele (eli tee oletus) sitä mikä voisi toimia, kokeile toimiiko se, opi kokeen tuloksista, tee se uudestaan. Ihmiset jotka suunnittelevat kokeita tietävät, että osittain epäonnistuneista kokeista oppii parhaiten. Kehitysympäristöön ei pitäisi tuoda lennokkaita iskulauseita, jotka muka parantavat tiimihenkeä tai antavat työntekijöille luvan kukoistaa työssään. Ne kertovat vain siitä, että niiden luoja ei ole koskaan tehnyt käytännön kehitystyötä haastavissa olosuhteissa. Oppiminen tapahtuu parhaiten nopean takaisinkytkennän kautta. Mitä lyhyempi viive,

sitä paremmin ja vähemmällä jätteellä prosessi toimii. Vaihtelua ei voida eliminoida ja nollavirhetavoitteet eivät toteudu normaalissa dynaamisessa ympäristössä. Ohjelmistonkehityksessä tehdäänkin sen vuoksi iteraatioita, joiden avulla pyritään pienentämään eroa asiakkaan vaatimusten ja ohjelmiston toiminnallisuuden välillä. Näiden iteraatioiden kesto pyritään pitämään mahdollisimman lyhyenä (käytännössä enintään viikkoina).

3. Päätä asioista mahdollisimman myöhään

Peruuttamattomien päätösten tekeminen tulisi jättää mahdollisimman myöhään, jolloin ne voidaan tehdä toteutuneiden asioiden eikä ennusteiden perusteella. Tavoitteena tulee näin olla tehdä päätökset ja valinnat mahdollisimman myöhäisessä. Kun päätös tai valinta on tehty, siirrytään sen toteuttamiseen sujuvasti ja nopeasti. Hyvin toimivassa organisaatiossa viiveet ovat vähäisiä ja toiminta tapahtuu ilman esteitä eteenpäin virraten. Nykyisin on olemassa useita tapoja, joiden avulla voidaan vaihtoehdot avoimina mahdollisimman myöhään:

1. Epätäydellisen suunnitteluinformaation hyödyntäminen
2. Suora, tekijöiden välinen yhteistyö
3. Vaihtelun mahdollistaminen, vaihtoehtojen salliminen
4. Joustava muutosten hallinta
5. Refaktoroinnin ja uudelleenkäytettävän koodin suosiminen
6. Automaattinen testaus

4. Toimita mahdollisimman nopeasti

Ne, joiden mielestä nopea ohjelmistonkehitys on lähinnä hakkerointia, eivät näe mitään mieltä nopeissa toimituksissa vaan uskovat että kaikki pitää suunnitella ja tehdä varovasti ja rauhallisesti minimoimalla kaikki riskit. Vastaavasti länsimaiset autonvalmistajatkaan eivät voineet käsittää japanilaisten autotehtaiden Just-In-Time – konsepteja eivätkä ymmärtäneet niiden tuottavuusvaikutuksia. Uskottiin että on hyvä, kun varastohyllyt ovat täynnä ja että paras tapa tehdä voittoa oli hankkia uusia suuria koneita ja ajaa niitä täysillä ympäri vuorokauden. Tämä ”viisaus” on sittemmin osoittautunut vääräksi. Nopeat toimitukset luovat myös paremman mahdollisuuden nopeaan asiakaspalautteeseen ja jatkuvaan parantamiseen.

5. Valtuuta tiimi

Ihmisten, jotka tekevät työn on tehtävä myös työtä koskevat päätökset. Ketterässä organisaatiossa työnkulku perustuu paikalliseen, nopeaan viestintään ja tiimin jäsenten väliseen yhteistoimintaan eikä suinkaan johdon antamiin direktiiveihin. Tiimi muokkaa itse omat prosessinsa, kerää tarvittavan informaation päätöksenteolle sekä sitoutuu omiin päätöksiinsä ja tavoitteisiinsa. Työskenneltäessä lähellä asiakasta tapahtuu väistämättä oppimista ja kehitystiimin ja asiakkaiden välinen yhteistyö nopeutuu. Johdon tehtävänä on varmistaa, että tiimillä on edellytykset tehdä oikeita päätöksiä, muuttaa kurssia tarvittaessa välittömästi ja saada aikaan hyviä tuloksia ja ohjelmistoja.

6. Rakenna laatua

Käyttäjäkokemus, neuvova ja intuitiivinen käyttöliittymä ovat tärkeitä – käyttäjät ja käyttäjien edustajat ovat mukana kehitystyössä ja päivittäin on tehtävä valinnat mitä kehitetään ja mitä jätetään pois. Ohjelmiston teknisen toiminnallisuuden on oltava korkealla tasolla ja kaikkien osien on toimittava hyvin yhdessä. Kaikkien työntekijöiden on oltava mukana suunnittelussa, joten tiimiltä vaaditaan jatkuvaa viestintää ja yhteistyötä. Virheet tulee pyrkiä havaitsemaan ja korjaamaan mahdollisimman aikaisessa vaiheessa ja viipymättä.

7. Näe ja ymmärrä kokonaisuus

Lähes kaikki teoriat siitä, miten ohjelmistonkehitystä pitäisi johtaa, perustuvat uskomukseen että pilkkomalla kokonaisuus osiin ja parantamalla näitä osia, muutetaan myös kokonaisuutta paremmaksi. Osien toiminnan yksittäinen optimointi johtaa poikkeuksetta kokonaisuuden suboptimointiin. Paras tapa välttää suboptimointi on tehdä ihmiset vastuullisiksi siitä, mihin he voivat vaikuttaa sen sijaan, että he ovat vastuullisia siitä mitä valvovat. Käytännössä tämä tarkoittaa sitä, että laadun mittaamisen tulee tapahtua suurempien kokonaisuuksien eikä vain yksittäisten osien suhteen. Sama koskee myös yksilökohtaisia palkitsemisjärjestelmiä: kukaan ihminen ei voi tehdä hyvää tulosta yksin, joten siitä palkitsemistakaan ei tulisi tehdä yksilökohtaisesti.

6. ANALYYSI JA YHTEENVETO

Tutkimusotteeni on ollut pääosiltaan kriittisen rationalismin mukainen. Teorian osuus on hallitseva koko tutkimukseni kannalta ja ymmärrän myös, että ennakkokäsitykseni asioista ja tutkimuksen kohteista ohjaa vahvasti havaintojen tekoa. Tutkielmani tieteellisfilosofisena viitekehyksenä olen käyttänyt Kuhnin teoriaa paradigmojen muutoksesta, Popperin kolmen maailman teoriaa ja Habermasin teoriaa tiedonintresseistä. Olen kartoittanut tutkimuksessani ohjelmistokehityksen laatuajattelua käyttäen välineenä kolmea toisiinsa liittyvää kertomusta: teollisen valmistuksen historiaa, laatutoiminnan historiaa ja ohjelmistokehityksen historiaa. Näiden kolmen historian kautta on ohjelmistojen ja ohjelmistotyön laatu muodostunut siihen muotoon ja kattavuuteen kuin se tänä päivänä ymmärretään.

Tutkimukseni aikana olen tunnistanut seuraavat paradigmojen muutokset:

1. Ammattimiehen kokonaisvaltaisesta työn hallinnasta yksinkertaiseen vaihetyön suorittamiseen (scientific managementin nykyinen asema työelämässä). Sama kehityskulku, joka tapahtui Taylorin oppien avulla teollisuudessa, on tapahtunut ja tapahtumassa ohjelmistokehityksessä.
2. Laadun tarkastamisesta ja tuotteen korjaamisesta on siirrytty prosessien ja strategiseen laadun suunnitteluun, joka perustuu loppujen tilastolliselle laadun ohjaukselle). Ketterä kehitysmalli edesauttaa laadun suunnittelun tuomista ohjelmistoprosessin määrittely- ja suunnitteluvaiheiden sisään.
3. Ohjelmistokehityksen muuttuminen pelkästä teknologian rakentamisesta sosiaalisen vuorovaikutuksen suuntaan (ohjelmoijien vastaisku). Kovasta systeemiajattelusta on siirrytty pehmeämmän systeemiajattelun suuntaan. Se onko kyseessä koko paradigman vaihtuminen vai alkuperäisen paradigman laajentuminen, onkin eräs tämänhetkisen tieteenfilosofiseen keskustelun peruskysymyksiä.

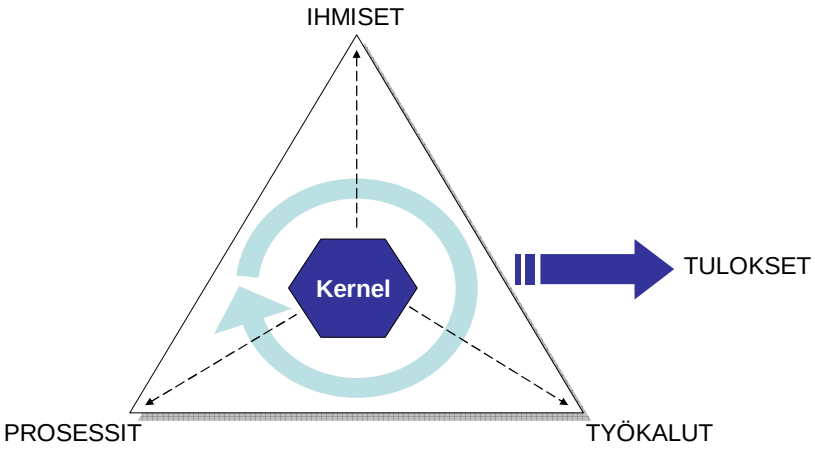
Ohjelmistonkehitystyön rationalisoinnissa esiintyvät käytännöt (moduuli- ja komponenttiajattelu sekä koodin uudelleenkäyttö) ovat suoria jälkeläisiä Gribeauvalin

ja de Pronyn periaatteiden mukaan toteutetuille teollisen valmistuksen suunnittelu- ja tuotantotavoille (esimerkkeinä näistä tavoista voidaan mainita tuotteiden vaihdettavat osat, tavaroiden sarjatuotanto, suurten tuotteiden valmistaminen liukuhihnalla). Ketterä kehitys on mielestäni nähtävä terveenä reaktiona tälle kehitykselle. Agile Manifesto painottaa periaatteissaan tekijöiden suoraa dialogia ja kestävää kehitystä byrokratian ja työntekijöiden ylikuormittamisen sijaan. Lean ajattelu Toyotan Wayn mukaan painottaa sen sijaan prosessia ihmisten sijaan. Siinä missä ketterä lähestymistapa toivottaa muutoksen ja vaihtelun tervetulleiksi jokapäiväiseen työhön, pyrkii TPS standardoimaan prosessit mahdollisimman pitkälle ja siirtämään joustot lähinnä henkilöstön vastuulle. TPS-mallin suora kopioiminen ohjelmistokehitykseen on tuskin onnistunut hyvin missään länsimaisessa yrityksessä.

Ohjelmistojen laatuajattelun on mielestäni perustuttava Shewhartin, Demingin ja Juranin opetuksiin. Shewhartin tilastollinen laadunohjaus, Demingin tietämysperusta ja Juranin trilogia toimivat moitteetta nykyisessäkin tilanteessa ja muodostavat hyvän viitekehyksen ohjelmistoprosessin parantamiselle. Asiantuntijoita nimitetään mediassa ja tavallisessa keskustelussa usein liiankin kevyin perustein oman alansa ”guruiksi”. Näiden kolmen miehen kohdalla tuo nimitys on paikallaan, sillä heidän tietämyksensä on ollut syvällistä ja universaalia. Joskus tuntuu kuin mitään todella merkittävää ei laadusta olisikaan edes kirjoitettu heidän jälkeensä - monet uudet termit ja metodit ovat jälkeenpäin osoittautuneet vain vanhaksi viiniksi uudessa mukissa tarjoiltuna. Kun muki on tyhjä, niin jäljelle on jäänyt vain kaipaus entiseen.

Ivar Jacobsonin Kernel-mallin avulla on mahdollista hyödyntää ketterän kehityksen menetelmiä ja työtapoja suuremmissakin hankkeissa ilman että tällaisten hankkeiden hallinta ja valvonta heikkenee. Kaiken toiminnan perustan muodostaa tällöin ydinkokonaisuus (*kernel*), joka sisältää kaiken sen mitä ilman toiminta ei onnistu, mutta ei kuitenkaan yhtään enempää. Jacobsonin mukaan toimintaan on joka tilanteessa aina löydettävä ihmisistä, prosessista ja työkaluista rakennettu minimikokoonpano, jonka avulla halutut tulokset saavutetaan. Tällainen minimalismi on läheistä sukua konseptille, joka tunnetaan yleisesti ”Occamin partaveitsen” nimellä. Sen mukaan on valittava

kaikista mahdollisista selityksistä aina yksinkertaisin, sillä se on reaalimaailmassa kaikkein todennäköisin.



Kuva 20. *Jacobsonin Kernel-mallin osat.*

Kernel –malli edustaa samaa suuntausta Occamin partaveitsen kanssa ja Pareton periaatteen kanssa. Perinteisesti ajatellen on ollut selvää, että mitä enemmän resursseja ja henkilöitä projektille on annettu, sitä nopeammin se pystyy saavuttamaan tavoitteensa (eli tuottamaan halutut tuloksensa). Resurssien ylitarjonta johtaa valitettavan usein tuhlailuun ja vähemmän tehokkaisiin työmenetelmiin ja on tilanteita ja tapauksia, joissa uusien ihmisten ja resurssien lisäys on vain hidastanut ohjelmistoprojektin etenemistä. Occamin partaveistä soveltaen voidaan tällöin päätellä, että projektiin on varattava juuri niin monta henkilöä kuin sen suorittaminen menestyksellisesti vaatii, mutta ei yhtään enempää. Kaikenlaisten marginaalien ja puskurien rakentaminen on näin ollen kielletty. Jacobsonin mukaan onkin järkevää tietoisesti lievästi aliresurssoida ohjelmistojen kehityshankkeita.

Ihmiset

Ohjelmistonkehitys vaatii ihmisiltä ammattimaisuutta ja osaamista. Työelämä tuntuu valitettavasti suosivan yhä kapeampialaista teknologioiden detaljiosaamista laajempien, yleisten ongelmanratkaisu- ja vuorovaikutustaitojen sijaan. Tätä suuntausta on osittain vahvistanut 90-luvulla alkanut asiantuntijoiden sertifiointiliiketoiminta, joka on sekoittanut monien työnantajien käsitystä siitä, mikä ohjelmistokehitystyössä oikeastaan on tärkeintä: asiakkaiden tarpeiden ja käyttäytymisen ymmärtäminen ja tämän tietämyksen muuntaminen toimivaksi ohjelmistoksi.

Esimerkki tällaisesta sertifiointiin johtavasta koulutustoiminnasta on erään alan yhtiön lanseeraama Certified Scrum Developer – koulutus. Pitääkö tavallisen ohjelmistokehittäjän todellakin käydä tällainen kurssi pystyäkseen toimimaan Scrum-projektissa? Mieleen nousee Scrumin perusidean mukainen kuva rugbyjoukkueen kaltaisesta ryhmästä, jonka pelaavien jäsenten sertifiointi tarkastetaan aina ennen jokaista aloitusta.

Prosessit

Ketterissä kehitysmalleissa itse ohjelmistoprosessi on määritelty varsin joustavaksi. Jokainen projekti soveltaa yleistä mallia sitten omien tarpeidensa ja tapojensa mukaan. On kuitenkin tärkeää että kaikilla kehittäjillä on sama näkemys sovelletusta prosessista sekä sen vaiheista ja tarkastuspisteistä. Useimmiten prosessin kriittisin vaihe on sen alussa eli juuri asiakasvaatimusten määrittelyssä. Ellei määrittelyyn panosteta riittävästi, eivät myöskään myöhemmät vaiheet voi onnistua hyvin.

Esimerkkinä huonosta prosessimallin soveltamisesta ja sen aiheuttamista vahingoista voidaan pitää erään kotimaisen tietotekniikkayhtiön Helsingin vesilaitokselle vuonna 1992 toimittama asiakastietojärjestelmää. Tietotekniikkayhtiön ohjelmistoprosessi oli ylimmällä tasolla määritelty vesiputousmallin (Valtion Tietokonekeskuksen vaihejakomallin) mukaiseksi, mutta alempien tasojen toiminnoissa oli ymmärretty väärin yksi kriittinen tekijä. Hankkeen projektisuunnitelmasta ja aikataulusta unohtui pois täysin testauksen esiintuomien virheiden korjaaminen. Itse testauksen jälkeen olisi pitänyt varata kohtuullisesti aikaa ja riittävästi tekijöitä havaittujen virheiden korjaukseen. Asia huomattiin vasta, kun itse testaus oli jo aloitettu, vaikka

projektisuunnitelmaa ja aikataulua oli esitelty laajasti molempien osapuolten johtoryhmissä sekä hankkeen ohjausryhmässä.

Työkalut

Työkalujen merkitys on tuntunut kasvaneen viimeisten vuosien aikana. Tämä ei koske ainoastaan itse ohjelmointivälineitä vaan myös kehitysympäristön hallintajärjestelmiä (erityisesti vaatimusten, muutosten ja konfiguraatioiden hallinnan). Riskinä on se, että liiallinen työkalukeskeisyys saattaa rajoittaa ratkaisuja: ihminen, jonka ainoa työkalu on vasara, näkee helposti kaikissa ongelmissa vain nauloja. Työkalujen on luonnollisesti tuettava organisaation olemassa olevia prosesseja ja niiden muokkaamisen tulisi helppoa ja nopeaa. Työkalujen käytettävyyden merkitystä ei voi yliarvioida.

Esimerkki ohjelmistokehityksen hallintajärjestelmähankkeesta, jossa käytettävyyttä ja käyttäjien tarpeita ei riittävästi otettu huomioon, on erään suuren kansainvälisen energia-alan yhtiön SCM-ohjelmiston käyttöönotto 2000-luvun alussa. Konsernissa oli tuotekehityksen käytössä ihan kelvolliset versionhallinnan työkalut, mutta nämä toimivat järjestelmät haluttiin korvata yhdellä integroidulla ratkaisulla. Henkilöstön mielestä vaatimusten määrittely oli selvästi huonommalla tasolla, koska erilaisten MS Office-dokumenttien kirjoittaminen ja levittäminen teetti sille paljon ylimääräistä työtä. Konsernin tuotekehityksen ja tietohallinnon johto hankki uuden integroidun SCM-ratkaisun kehittäjien vastustuksesta huolimatta. Parin vuoden kalliiden kokeilujen ja epäonnistumisten jälkeen koko hanke haudattiin hiljaisuudessa, sillä toiminnoiltaan laajemman, mutta työnkulultaan vanhoja työkaluja jäykemmän SCM-järjestelmän suorituskyky osoittautui todellisessa tuotantokäytössä heikoksi ja kehittäjien ajasta suuri osa kului odotellessa hitaan järjestelmän suoritteita.

Tulokset

Agile Manifeston mukaan ohjelmistokehityksen ainoa oikea mittari on hyvin toimiva ohjelma. Lähes jokaisella yrityksellä on kuitenkin oma tulkintansa siitä, miten tätä tuotosta todellisuudessa pitäisi mitata. Monet yhtiöt tekevät asian itselleen helpoksi ja mittaavat vain työtuntien kulumista/kulutusta eivätkä suinkaan näiden tuntien aikana saavutettuja tuloksia. Kun ennalta suunniteltu tuntimäärä on täynnä, todetaan vain mihin on tultu tai jouduttu.

Esimerkki. Verohallinnon tietojärjestelmien uudistaminen 1990-luvulla oli hyvä osoitus tällaisesta lähestymistavasta: hankkeen aikataulut pitivät aluksi nimellisesti, mutta laatu ja toiminnallisuus joustivat alusta alkaen. Lopulta koko hanke päättyi suureen, tiedotusvälineissäkin laajalti uutisoituun katastrofiin sekä hankejohdon ja verohallinnon ylimmän johdon välisiin riitoihin.

Uuden talouden loistavat lupaukset 2000-luvun alussa osoittautuivat harhakuviksi ja turhiksi toiveiksi – hyvinvointi ei jakaantunutkaan tasaisesti vaan erot ovat päinvastoin kasvaneet. Ketterien ohjelmistonkehitysmenetelmien esiinmarssista on paljon puhuttu tutkijoiden ja asiantuntijoiden muodostamissa yhteisöissä viimeiset kymmenen vuotta, mutta onko mitään suurta läpimurtoa tapahtunut käytännön kehitystyössä? Paradigman muutos kestää joskus jopa vuosikymmeniä kuten esimerkiksi Kuhnin omalla kohdalla kävi. Kuluikin lähes kaksikymmentä vuotta ennen kun hänen teoriansa paradigmojen muutoksesta nousi vallitsevaksi paradigmaksi tieteenfilosofian alueella.

Mihin ihmisen toiminnan kategoriaan ohjelmistokehitys kuuluu? Onko se taidetta vai tiedettä, vai sittenkin pelkästään käsityötä? Jos ohjelmistokehitys on tekninen käsityötaito, voidaan aiheellisesti kysyä, kuinka paljon ammattitaitoa sitten tavallisessa ohjelmointityössä vaaditaan? Ohjelmointi eli koodaus on useimmiten suoraviivaista toteuttamista ja ei sellaisenaan vaadi suurtakaan ammattitaitoa (näin edellyttäen tietenkin että ohjelman suunnittelu on tehty kunnolla). Asiakkaan liiketoiminta-prosessien ymmärtäminen ja määrittely, käyttäjien tarpeiden tunnistaminen ja sopivan ohjelmistoarkkitehtuurin sekä liittymien määrittely edellyttävät huomattavasti laajempaa osaamista.

Ohjelmistonkehityksessä käytetään paljon teollisesta valmistamisesta lainattuja menetelmiä ja käytäntöjä. Toimivatko nämä menetelmät ja käytännöt edes tyydyttävästi ohjelmistokehityksessä, koska toiminnan ja tekemisen kohteet (objektit) kuuluvat pääasiassa maailmoihin 2 ja 3 eivätkä fyysiseen maailmaan 1 kuten tavaroita teollisesti valmistettaessa? Tiedonintressien perspektiivistä arvioiden nämä menetelmät ja käytännöt palvelevat pikemminkin teknistä tiedonintressiä ja sopivat siten paremmin fyysisen maailman tuotteiden kehittämiseen. Hallintahan on tällöin toiminnan

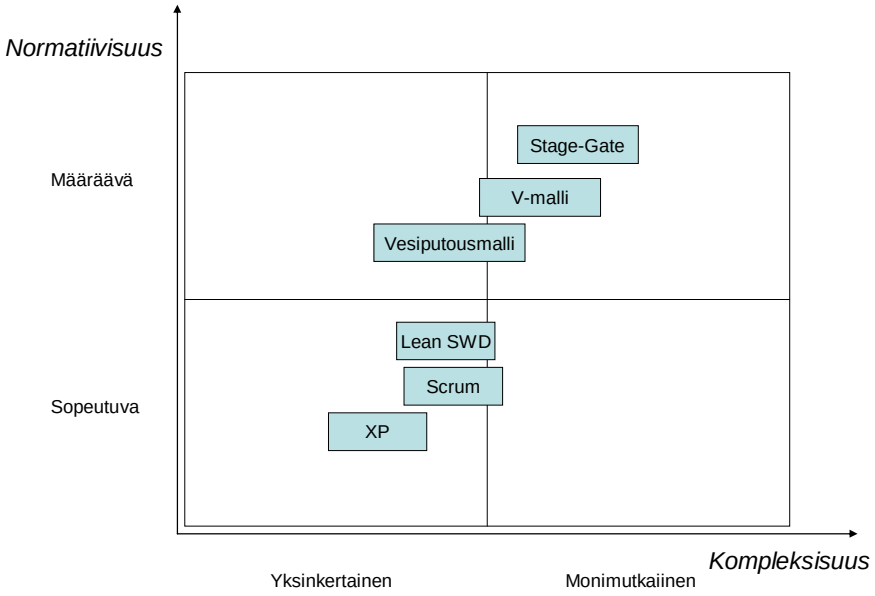
ensisijainen tavoite ja ymmärtäminen sekä vääristä uskomuksista vapautuminen ovat vastaavasti vähempiarvoisia.

Organisaation tulisi tunnistaa kaikissa rakenteissaan ja toiminnassaan todelliset periaatteensa ja arvonsa. Zen-buddhismin opetuksia mukaillen: meidän on nähtävä, tajuttava asioiden ja olentojen todellinen luonto eikä pitäytyä tuijottamaan niiden ulkoista kuorta (=*pointing direct to reality*). Kehittyneet ulkoiset tavat ja käytännöt voivat kyllä heijastaa yhteisön sisäistä kulttuuria ja tilaa, mutta yhtä hyvin ne voivat johtaa meidät harhaan.

"We cannot solve the problems we have created with the thinking that created them."
- Albert Einstein

Uusia aiheita ja tutkimuskysymyksiä jatkotutkimusta varten

Tutkimusaihe 1. Millaisia riippuvuuksia voidaan löytää ja tunnistaa yrityksen liiketoimintaympäristön dynamiikan ja sen oman kehitystoiminnan ketteryyden välillä. Millaisia tuloksia ketterillä kehitysmenetelmillä saavutetaan verrattuna perinteisiin kehitysmenetelmiin? Kuinka paljon on riittävästi (ihmisiä, työkaluja ja prosesseja) hyvien kehittämistulosten aikaansaamiseksi?



Kuva 21. *Kehitysmalleja normatiivisuus-kompleksisuus kentässä.*

Tutkimusaihe 2. Yllä olevan kuvaan olen hahmottanut eri kehitysmallien sijoittumista normatiivisuuden ja kompleksisuuden muodostamalle alueelle. Tähän liittyen voidaan tunnistaa seuraavia tutkimusaiheita:

1. Miten ohjelmistokehittäjät kokevat eri menetelmien kompleksisuuden ja normatiivisuuden?
2. Hypoteesina voidaan ajatella ketterien kehitysmenetelmien sopivan paremmin pieniin ja vähemmän kriittisiin ohjelmistohankkeisiin. Onko näin myös todellisuudessa?
3. Arvioidaan menetelmiä ja malleja seuraavan näiden kahden ulottuvuuden perusteella: Normatiivisuus (määräävä – sopeutuva) ja kompleksisuus (yksinkertainen – monimutkainen).

LÄHTEET

Kirjat

Braverman, Harry 1977. *Arbete och monopolkapital*. Stockholm: Raben&Sjögren.

Börjesson, Anna 2006. *Making Software Process Improvement Happen, doctoral dissertation*. Gothenburg: IT University of Gothenburg.

Campbell, Tom 1986. *Sju teorier om samhället*. Stockholm: Akademilitteratur.

Crosby, Phil 1990. *Frågor och svar om kvalitet*. Lund: Studentlitteratur.

Creech, Bill 1994. *The Five Pillars of TQM*. New York: Truman Talley Books.

Dahlbom, Bo and Mathiassen, Lennart 1993. *Computers in Context*. Cambridge MA: NCC Blackwell.

Deming, W. Edwards 1994. *Out of the Crisis*. Nineteenth printing. Cambridge MA: Cambridge University Press.

Deming, W. Edwards 1994. *The New Economics*. Second edition. Cambridge MA: Massachusetts Institute of Technology, CAES.

Drucker, Peter 1981. *Ledarskap i brytningstider*. Malmö: Liber.

Fuller, Steve 2003. *Kuhn vs Popper*. Cambridge UK: Icon Books.

Hammer, Michael 1996. *Beyond Reengineering*. New York: HarperCollins

Hart-Davis, Adam 2000. *Nero & Leimaus*. Hämeenlinna: Karisto.

Juran, Joseph M. 1995. *Managerial Breakthrough*. Second edition. New York: McGraw-Hill Inc.

Juran, Joseph M. and Blanton Godfrey, A. 1998. *Juran's Quality Handbook*. Fifth edition. New York: McGraw-Hill Inc.

Kragh, Helge 1989. *An Introduction to the Histography of Science*. Cambridge MA: Cambridge University Press.

Kuhn, Thomas S. 1994. *Tieteellisten vallankumousten rakenne*. Helsinki: Art House.

Niiniluoto, Ilkka 1996. *Informaatio, tieto ja yhteiskunta*. Viides, täydennetty painos.

Helsinki: Edita Oy.

Niiniluoto, Ilkka 1990. *Maailma, minä ja kulttuuri*. Helsinki: Otava.

Nonaka, Ikijuro and Takeuchi, Hirotaka 1995. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics*. New York: Oxford University Press.

Rekola, Katri 2006. *Product-centric Service Development: The Development Process Tools and Methods, doctoral dissertation*. Vaasa: Universitas Wasaensis.

Senge, Peter M. 1992. *The Fifth Discipline*. London: Century Business.

Sommerville, Ian 2009. *Software Engineering*. Ninth edition. Boston MA: Pearson Education Inc.

Watts, Alan W. 1997. *Zen*. Toinen painos. Helsinki: Otava.

Womack, James P. and Jones, Daniel T. 1996. *Lean Thinking : Banish waste and create wealth in your organization*. New York: Simon & Schuster.

Artikkelit

Ambler, Scott (2005). Quality in an Agile World, *Software Quality Professional*, 2005 September.

Cooper, Robert G. (1990). Stage-Gate Systems: A New Tool for Managing New Products, *Business Horizons*, 1990 May-June.

Copeland, Lee (2001). Extreme Programming. *Computerworld*, 2001 December.

Dahlbom, Bo and Lennart Mathiassen (1997). The Future of Our Profession. *Communications of the ACM*. 1997 June.

Gibson (2006), from Product-focused Software Process Improvement: 8th International conference, SEI 2005.

Lairman, Greg and Victor R. Basili (2003). Iterative and Incremental Development: A Brief History, *Computer*, 2003 June.

Liker, Jeffrey K. and James P. Morgan (2006). The Toyota Way in Services: The Case on Lean Product Development. *Academy of Management Perspectives*, 2006 May.

Nonaka, Ikujiro and Hirotaka Takeuchi (1986). The New Product Development Game, *Harvard Business Review*, 1986 January-February.

Nonaka, Ikujiro and Georg von Krogh (2009). Tacit Knowledge and Knowledge Conversion: Controversy and Advancement in Organizational Knowledge Creation Theory. *Organization Science*, May-June 2009, pp 635-652.

Poppendieck, Mary (2003). Lean Software Development. *C++ Magazine*, 2003 Fall Issue.

Elektroniset lähteet

Basili, Victor R. *Experience Factory and its Relationship to other Quality Approaches.* College Park: University of Maryland. Saatavana World Wide Webistä <URL: http://www.cs.umd.edu/~basili/publications/chapters/C21.pdf>

Basili, Victor R. *Learning through Application: The maturing of the QIP in SEL.* College Park: University of Maryland. Saatavana World Wide Webistä <URL:http://www.cs.umd.edu/~basili/publications/chapters/C29.pdf>.

Basili, Victor R., Frank E. McGarry, Rose Pierski and Marvin V. Zelkovitz. *Lessons learned from 25 years of process improvement: The Rise and Fall of NASA Software Engineering Laboratory.* College Park: University of Maryland. Saatavana World Wide Webistä <URL: http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/83.88.pdf >.

Bohman, James and William Rehg (2011). Jürgen Habermas, revision Tue Sep 6, 2011. Teoksessa: *The Stanford University Encyklopedia of Philosophy*. Stanford CA: Center for the Study of Language and Information, Stanford University. Saatavana World Wide Webistä <URL: http://plato.stanford.edu/entries/habermas/>.

Carnegie Mellon Software Engineering Institute (2010). *CMMI for Development, Version 1.3.* Saatavana World Wide Webistä <URL: http://www.sei.cmu.edu/reports/10tr033.pdf>.

Carnegie Mellon Software Engineering Institute (2010). *CMMI Overview*. Saatavana World Wide Webistä <URL: http://www.sei.cmu.edu/library/assets/cmmi-overview07.pdf>.

Clark, Celia (2005). Block Mills - Top historic site and Building at Risk. *The Portsmouth Society – News*. 2005 May. Saatavana World Wide Webistä <URL: http://www.portsmouthsociety.org.uk/nl2005/nlmay05doc07.htm>.

Eclipse Foundation (2012). *Scrum Overview*. Saatavana World Wide Webistä <URL: http://epf.eclipse.org/wikis/scrum/Scrum/guidances/supportingmaterials/scrum_overview_610E45C2.html>.

Halsall, Paul (editor) (2006). Andrew Ure. Teoksessa: *The Internet History Sourcebooks Project.* The Fordham University. Saatavana World Wide Webistä <URL: http://www.fordham.edu/halsall/mod/1835ure.asp>.

History Channel Website (2012). *Samuel Colt*. A&E Television Networks. Saatavana World Wide Webistä <URL: http://www.history.com/topics/samuel-colt.>.

Manifesto for Agile Development. Saatavana World Wide Webistä <URL: http://agilemanifesto.org/>.

Royce, Winton (1970). *Managing the Development of Large Software Systems*. Saatavana World Wide Webistä <URL: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>.

Schwaber, Ken and Sutherland, Jeff (2011). *The Definite Guide to the Scrum: The Rules of the Game*. Saatavana World Wide Webistä <URL: http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf>.

Thornton, Stephen (2009). Karl Popper, revision Mon Feb 9, 2009. Teoksessa: *The Stanford University Encyklopedia of Philosophy.* Stanford CA: Center for the Study of Language and Information, Stanford University. Saatavana World Wide Webistä <URL: http://plato.stanford.edu/entries/popper/>.

Waterfall-model.com (2011). *V-Model.* Saatavana World Wide Webistä <URL: http://www.waterfall-model.com/v-model-waterfall-model/>.

Wikipedia Foundation (2012). *Henry Gantt*. Saatavana World Wide Webistä <URL: http://en.wikipedia.org/wiki/Henry_Gantt>.

Wikipedia Foundation (2012). *Honoré Blanc*. Saatavana World Wide Webistä <URL:http://en.wikipedia.org/wiki/Honoré_Blanc>.

Wikipedia Foundation (2012). *Francois Quesnay*. Saatavana World Wide Webistä <URL: http://en.wikipedia.org/wiki/Francois_Quesnay>.

What-when-how.com (2012). *Assembly Line (Inventions)*. The-Crankshaft Publishing. Saatavana World Wide Webistä <URL:http://what-when-how.com/inventions/assembly-line-inventions/>.