Ohjelmistotuotantomenetelmien kehittyminen 1950-luvulta nykypäivään

Lauri Suomalainen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 20. joulukuuta 2013

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Lauri Suomalainen

Työn nimi — Arbetets titel — Title

Ohjelmistotuotantomenetelmien kehittyminen1950-luvulta nykypäivään

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	20. joulukuuta 2013	8

Tiivistelmä — Referat — Abstract

Tiivistelmä

Avainsanat — Nyckelord — Keywords
avainsana 1, avainsana 2, avainsana 3

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Tämä kandidaatintutkielma tarkastelee ohjelmistotuotantomenetelmien kehittymistä ohjelmistokehittämisen alkuajoista nykypäivään ja lähitulevaisuuteen. Se käsittelee erilaisia ohjelmistotuotantomenetelmiä kronologisesti. Jokaisen menetelmän kohdalla pyrin vastaamaan seuraaviin kysymyksiin:

- Mistä ohjelmistotuotantomenetelmässä on kyse?
- Miksi sitä käytetään/käytettiin ja mitä hyötyä siitä on/oli?
- Mitkä olivat sen heikkoudet?

Ohjelmistotuotannon eri osa-alueita tarkastellaan tutkielmassa ohjelmistotuotantomenetelmiä määrittävinä piirteinä. Tämä tarkoittaa sitä, että tarkasteltaessa esimerkiksi miten vaatimusmäärittely toteutetaan jossain tietyssä ohjelmistotuotantomenetelmässä, keskitytään prosessin konkreettisen toteutuksen sijasta sen asemaan ja erityispiirteisiin menetelmän kontekstissa.

Ensimmäiset luvut toimivat pohjustuksena tutkielmalle. Luku 2 esittelee lyhyesti keskeisimmät ohjelmistotuotantoon ja tuotantomenetelmiin liittyvät käsitteet ja Luku 3 käsittelee ohjelmistotuotannon alkuaikoja, jolloin tietokoneet itsessään olivat vielä verrattain uusi ilmiö ja suuret ohjelmistot ja niiden tuottaminen oli vähäistä. Alun alkaen laitteiston rajallisuus sitoi myös ohjelmistojen mahdollisuuksia, mutta laitteiston kehittyessä syntyi tarve organisoidummalle ohjelmistojen kehittämiselle.

Luku 4 käsittelee perinteistä ohjelmistotuotantomallia eli niin sanottua "vesiputousmallia". Tarkastelen mallin perusperiaatteita, sen nousemista aikansa keskeisimmäksi ohjelmistotuotantomenetelmäksi, sen onnistumisia ja epäonnistumisia sekä sen heikkouksia ja siihen kohdistettua kritiikkiä.

Luku 5 tarkastelee inkrementaalisia ja iteratiivisia ohjelmistotuotantomenetelmiä sekä prototyyppaamista. Tyyppiesimerkkeinä käsittelen RUP:ia eli Rational Unified Processia sekä spiraalimallia, jota voidaan pitää vesiputousmallin kehittyneempänä versiona.

Luvussa 6 otetaan tarkasteluun ketterät ohjelmistotuotantomenetelmät. Vaikka ketterä ohjelmistokehitys on kattokäsitteenä monelle erilaiselle mene-

telmälle kuten XP ja Kanban, käsittelen Scrumia tyyppiesimerkkinä, sillä se on kaikista ketteristä menetelmistä suosituin ja tunnetuin.

Viimeisessä luvussa käsittelen lyhyesti yhteenvetona tämän hetken vallitsevia ohjelmistotuotantomenetelmiä. Tarkastelen myös mahdollista lähitulevaisuuden trendejä ja mahdollisuuksia

2 Peruskäsitteistöä

Tietojenkäsittelytieteeseen ja ohjelmistotuotantoon liittyy monia alakohtaisia käsitteitä. Kaikkien termien merkitykset eivät ole suoraan johdettavissa itse termistä, sillä osa termistöstä on verrattain vapaita käännöksiä ulkomaisista termeistä ja osa konsepteista on laajoja. Lisäksi joillain termeillä viitataan puhekielessä vapaammin eri asioihin. Esimerkiksi softwarella voidaan puhekielessä viitata niin yksittäiseen tietokoneohjelmaan kuin laajaan ohjelmistoonkin. Tämän takia lyhyt käsitteenmäärittely tullee tarpeeseen.

Software eli ohjelmisto käsittää tietokoneohjelman tai -ohjelmia sekä kaiken niihin liittyvän informaation ja materiaalin kuten tietokannat ja dokumentaation.
Tietokonelaitteisto eli hardware käsittää tietokoneen fyysiset osat kuten prosessorin ja kovalevyn. Laitteistoa tarvitaan ohjelmistojen suorittamiseen, ja laitteisto tarvitsee toimiakseen toimintaohjeet matalan tason tietokoneohjelmina. Käytännössä tietokoneohjelmistot ja -ohjelmat sekä tietokonelaitteisto eivät ole käyttökelpoisia yksinään, vaan kumpaakin tarvitaan toisen järkevään käyttöön.

Termi software engineering, suomeksi ohjelmistotuotanto, alkoi esiintyä kirjallisuudessa 1960-luvun puolivälissä. Termi itsessään on ollut usein keskustelun ja väittelyn kohteena ja ohjelmistotuotannon kuulumista insinööritieteisiin on kyseenalaistettu. [2, 3, 5] Watts S. Humphrey on määritellyt ohjelmistotuotannon tarkoittavan kurinalaista laadukkaiden ohjelmistojen tuottamista hyödyntäen niin luonnontieteellisiä, matemaattisia kuin insinööritieteidenkin periaatteita ja käytänteitä[4]. IEEE Computer Society määrittelee termin viittaavan kurinalaiseen, systemaattiseen ja arvioitavissa olevaan lähestymistapaan ohjelmistojen tuotannossa, käytössä ja ylläpidossa[1]. Ilkka Haikala

ja Jukka Märijärvi tulkitsevat määrittelyjen tarkoittavan ohjelmistotyötä, jonka tuloksena syntyvät järjestelmät täyttävät käyttäjiensä kohtuulliset toiveet ja odotukset ja tämän lisäksi valmistuvat laadittujen aikataulujen ja kustannusarvioiden puitteissa[3].

Ohjelmistotuotantoon kuuluvat kaikki ohjelmistotuotantoprosessin osa-alueet. Haikala ja Märijärvi [3] määrittelevät ne seuraavasti:

Määrittely sisältää asiakasvaatimusten analyysin ja niistä johdetaan ohjelmistovaatimukset.

Suunnittelu pitää sisällään ohjelmiston määrittelyssä jäsenneltyjen toiminnallisuuksien ja ominaisuuksien suunnittelun

Toteutus tarkoittaa ohjelmiston ohjelmointia sekä testauksen toteutusta

Testaus pyrkii karsimaan ohjelmistosta ohjelmointivirheitä ja muita vikoja. Tyypillisiä testaustapoja ovat yksikkö- ja integraatiotestaus.

Dokumentointi käsittää ohjelmistoprojektin aikana tuotettavan kirjallisen materiaalin, kuten projektisuunnitelmat, testaussuunnitelmat ja jopa ohjelmakoodin kommentoinnin.

Käyttöönotto ja **ylläpito** ovat asiakkaan ongelmien ratkomista, virheiden korjaamista ja tarvittaessa uusien ominaisuuksien lisäämistä.

Laatujärjestelmällä ja **laadunvarmistuksella** on tarkoitus taata, että ohjelmisto täyttää käyttäjän ja asiakkaan toiveet ja odotukset.

Projektinhallinta on työkalu ohjelmistotuotantoprojektin organisointiin. Suuret ohjelmistoprojektit koostuvat usein useasta rinnakkain tai peräkkäin etenevistä osaprojekteista ja tällöin niiden järjestelmällinen hallinta voi olla keskeistä koko projektin onnistumisen kannalta.

Tuotteenhallinta: Usein kaupallisella ohjelmistolla on useita eri konfiguraatioita jolloin se voidaan aina räätälöidä yksilöllisesti kullekin asiakkaalle sopivaksi. Tuotteenhallinnan tarkoitus on varmistaa, että asiakkaalla on tarvitsemansa toimiva versio ohjelmistosta.

Ohjelmistotuotantomenetelmä on koko ohjelmistotuotantoprosessin kattava viitekehys, joka ohjaa prosessin osa-alueitten käytännön toteutusta.

3 Ohjelmistotuotannon alkutaival

1940-luvun puolivälistä 1950-luvun lopun ensimmäiset ohjelmoitavat elektronisesti tallentavat tietokoneet olivat nykystandardeilla mitattuna valtavan kokoisia, hitaita ja muistikapasiteetiltaan mitättömiä. Rajoitustensa vuoksi tietokoneohjelmointia pidettiin toissijaisena ja yksinkertaisena toimena verrattuna itse tietokonelaitteiston suunnitteluun ja rakentamiseen. [2] Lähemmin ohjelmoinnin kanssa työskenteleville alkoi selvitä ohjelmoinnin monimutkaisuus ja ongelmat, kuten rajoitetun muistin hallinta ja ohjelmien välinen interaktio. Tarve järjestelmälliselle ohjelmien tuotannolle oli syntymässä.

Edsger Dijkstran mukaan alkukantaisilla tietokoneilla ohjelmointi nähtiin tapana venyttää tietokonelaitteiston rajoja ja kehittyneempien tietokoneitten uskottiin tekevän ohjelmoinnista lähes triviaalia kun rajoja ei enää tarvitsisi venyttää. [**?**] Niin sanottujen kolmannen sukupolven tietokoneiden tultua markkinoille vuosien 1963 ja 1965 välisenä aikana ohjelmoijat löysivät itsensä kuitenkin uusien ongelmien keskeltä, sillä uusi laitteisto oli myös paljon monimutkaisempi kuin aikaisemmat laitteet. Dijkstra itse summaa suurimpien ongelmien johtuneen kuitenkin tietokonelaitteiston tehokkuuden valtavasta tehokkuuden kasvusta:

> Niin kauan kun ei ollut koneita, ohjelmointi ei ollut mikään ongelma; kun meillä oli muutama heikko tietokone, ohjelmointi oli vähäinen ongelma. Nyt kun meillä on giganttisia tietokoneita, on ohjelmoinnista tullut yhtä giganttinen ongelma! [**?**]

Tietotekniikkayritysten hankkiessa uutta tehokkaampaa laitteistoa seurasi luonnollisesti paine myös hyödyntää niiden kapasiteettia. Sen lisäksi, että ohjelmoijat joutuivat kohtaamaan haasteita ja toteuttamaan ratkaisuja joita oli aikaisemmin vain spekuloitu [**?**], huomattiin että koulutetusta tietotekniikka-alan työntekijöistä alkoi olla pulaa kysynnän vain kasvaessa. [2] Ohjelmistoprojektit alkoivat kallistua ja yhä useammin ne ylittivät reilusti budjettinsa

ja aikataulunsa mikäli valmistuivat koskaan.[**?**] Tätä vaihetta tietotekniikan historiassa on myöhemmin kutsuttu nimellä ohjelmistokriisi, software crisis. Dijkstran mukaan ilmiö tunnustettiin avoimesti vuoden 1968 Naton ohjelmistotuotantokonferenssissa Saksan Garmischissa [**?**]. Tapahtuma oli merkittävä myös siksi, että termi Software Engineering käytettiin tuolloin ensi kertaa suuressa virallisessa yhteydessä. Tapahtumassa käsiteltyyn myös ohjelmistojen tuotanto ja laaduvalvontasykliä, koodin uusiokäyttöä sekä insinööritieteiden soveltamista ohjelmistojen tuotantoon. [2]

4 Perinteiset ohjelmistotuotantomenetelmät

Perinteisillä ohjelmistotuotantomenetelmillä viitataan suunnitelmavetoisiin tuotantomenetelmiin, joita määrittää projektin alussa suoritettava laaja vaatimusmäärittely ja yksityiskohtainen suunnittelu ja niitä sarjallisesti seuraavat toteutus-, testaus- ja käyttöönottovaiheet. Menetelmien lineaarisen etenemisen ja 'soljumisen' vaiheesta toiseen vuoksi niitä kutsutaan myös kattotermillä 'vesiputousmalli'. Ensimmäisen kerran mallin dokumentoi H.D. Benington vuonna 1956 artikkelissaan *Production of large computer programs.* [**?**, **?**] Tohtori Winston Royce julkaisi vuonna 1970 paljon keskustelua herättäneen tekstinsä *Managing the Development of Large Software Systems.* Artikkelissa Royce esittelee yksinkertaisen vesiputousmallin, mutta esittää sitä kohtaan kritiikkiä esittäen siitä iteratiivisempia versioita. Siitä huolimatta, että Roycen oma suositus oli tehdä vesiputousmallin vaiheet kahdesti aluksi tuottaen prototyypin ja myöhemmin varsinaisen tuotteen, artikkeliin on viitattu vesiputousmallin keskeisenä hahmotelmana.

4.1 Vesiputousmallin rakenne

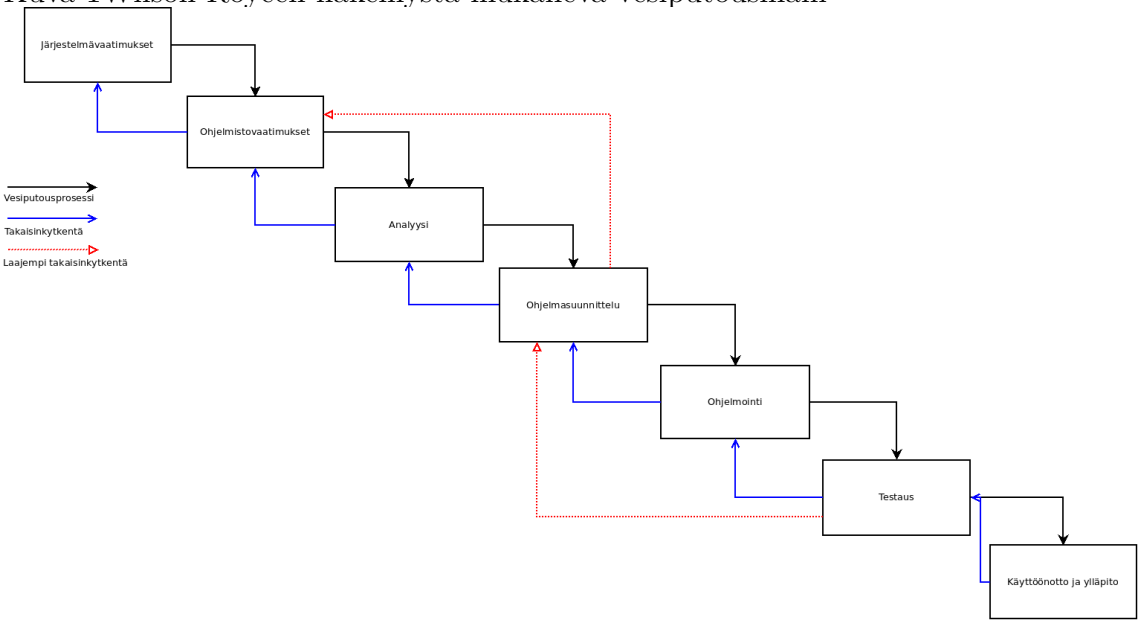
Vesiputousmalli koostuu toisiaan seuraavista työvaiheista, joista seuraava aloitetaan aina edellisen valmistuttua. Ajalleen tyypillisesti malli muistuttaa insinööritieteiden suunnitteluprosessia, jossa ennen projektin konkreettista toteuttamista tehdään perinpohjaiset suunnitelmat dokumentointeineen. Niin Benningtonin kuin Roycenkin mallin ensimmäiset askeleet liittyvät projek-

tin vaatimusten määrittelyyn ja kartoitukseen. Roycen mallin ensimmäiset askeleet ovat järjestelmävaatimukset ja ohjelmistovaatimukset[?]. Benington esittää vaiheet vähän laajemmin aloittaen korkeatasoisella ja yleisluontoisella toimintasuunnitelmavaiheella. Vaiheeseen kuuluu asiakasvaatimukset sekä projektin ajalliset ja budjetilliset vaatimukset. Toimintasuunnitelmavaihetta seuraa toiminnalliset määrittelyt ja laitteistospesifikaatiot. [?] Näiden pohjalta määritellään itse ohjelmiston toiminnalliset-, tehokkuus-, käyttö- ja laatuvaatimukset. Royce painotti dokumentaation tärkeyttä projektin onnistumisen kannalta ja alkuvaiheiden valmistuttua tuloksena tulisi olla määrittelydokumentti ja alustava suunnitteludokumentti. Kun vaatimukset on määritelty siirrytään analyysivaiheeseen, jossa selvitetään vaatimusten tekniset yksityiskohdat. Analyysin perusteella suunnitellaan lopulta itse ohjelmiston yksityiskohdat, joihin kuuluvat muun muassa ohjelmiston arkkitehtuuri, rajapinnat, käyttöliittymä ja testaus. Roycen mukaan suunnitteluvaiheen lopuksi dokumentoituna tulisi olla ohjelmiston lopullinen suunnitelma, käyttöliittymäsuunnitelma sekä ohjelmiston testaussuunnitelma. Ohjelmointivaihe toteutetaan suunnitelmien pohjalta ja niitä seuraten. Ohjelmointivaiheen ollessa ohitse siirrytään testausvaiheeseen, jossa ohjelman eri moduulit testataan yhdessä ja erikseen testaussuunnitelman pohjalta. Lopputuotoksena saadaan dokumentti testauksen tuloksista. Viimeinen vaihe on ohjelmiston käyttöönotto ja ylläpito. Tähän vaiheeseen kuuluu mahdollisten käytössä löydettyjen virheiden korjaaminen ja ohjelmiston muuttaminen tai siihen ominaisuuksien lisääminen mikäli tarve sille syntyy.

4.2 Vesiputousmallin ongelmia

Koska vesiputousmallille on haettu vaikutteita insinööritieteistä, muistuttaa se paljolti esimerkiksi rakennussuunnitteluprosessia: Siltaa rakentaessa ennen itse rakentamisen aloittamista suunnitellaan sillan arkkitehtuuri, tehdään lujuuslaskelmat, allokoidaan tarvittavat resurssit ja dokumentoidaan suunnitelmat tarkasti. Tarkasti dokumentoiduilla suunnitelmilla itse rakentamisvaihe voitaisiinkin siirtää ulkopuolisen tahon vastuulle ja näin rakennusalalla usein tehdäänkin. [?] Rakentaminen nähdään triviaalina ja mekaanisena ohjeiden

Kuva 1Wilson Roycen näkemystä mukaileva vesiputousmalli



seuraamisena. Martin Fowlerin mukaan menetelmän ongelmana ohjelmistotuotannossa on se, että suunnitelmien virheet ja heikkoudet ilmenevätkin juuri ohjelmointi- ja testausvaiheissa jolloin ohjelmiston toteutus ei olekaan enää kovin suoraviivaista. Juuri suunnittelun korostamisessa piilee myös yksi vesiputousmallin kritisoiduimmista piirteistä. Vesiputousmalli nojaa kokonaisvaltaiseen suunnitteluun ja suunnitelman noudattamiseen, mutta ohjelmistojen kehittäminen tapahtuu usein nopeasti muuttuvassa liiketoimintaympäristössä. Käytännössä on usein mahdotonta selvittää kaiken kattavia ja ulkoisille muutospaineille immuuneja järjestelmä- ja ohjelmistovaatimuksia. [**?**]

Joskus todelliset vaatimuksetkin selviävät vasta ohjelmistoa oikeassa ympäristössä käytettäessä. Teoriassa hyvä suunnitelma voi osoittautua huonoksi myöhemmissä konkreettemmissa työvaiheissa, mutta vesiputousmallin rakenne vaikeuttaa muutoksien tekemistä suunnitelmiin. Managing the Development of Large Software Systems -artikkelissa Royce esittääkin, että vaiheiden välillä tulisi olla takaisinkytkentöjä, jolloin uutta tietoa saadessa voitaisiin palata korjaamaan aikaisempien työvaiheiden virheitä. [**?**]

5 Inkremetaaliset ja iteratiiviset menetelmät sekä prototyyppaus

6 Ketterät menetelmät

7 Ohjelmistojen kehitys nykyään ja tulevaisuudessa

Lähteet

[1] Abran, Alain, Moore, James W., Bourque, Pierre, Dupuis, Robert ja Tripp, Leonard L.: *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE, 2004. `http://www.swebok.org/`, ISO Technical Report ISO/IEC TR 19759.

[2] Grier, David Alan: *Software Engineering: History*. Teoksessa *Encyclopedia of Software Engineering*, sivut 1119–1126. 2010.

[3] Haikala, I. ja Märijärvi, J.: *Ohjelmistotuotanto*. Korkeakoulu-sarja. Satku, 2003, ISBN 9789521404863. `http://books.google.fi/books?id=xIVaAAAACAAJ`.

[4] Humphrey, W. S.: *The software engineering process: definition and scope*. Teoksessa *Proceedings of the 4th international software process workshop on Representing and enacting the software process*, ISPW '88, sivut 82–83, New York, NY, USA, 1988. ACM, ISBN 0-89791-314-0. `http://doi.acm.org/10.1145/75110.75122`.

[5] Mahoney, Michael S.: *Finding a History for Software Engineering*. IEEE Annals of the History of Computing, (1):8–19, ISSN 1058-6180. `http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1278847`.