Ohjelmistotuotantomenetelmät

Jarl-Erik Malmström

Aine
Helsingin Yliopisto
Tietojenkäsittelytieteen laitos

Helsinki, 1. maaliskuuta 2013

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Jarl-Erik Malmström

Työn nimi — Arbetets titel — Title

Ohjelmistotuotantomenetelmät

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Aine	1. maaliskuuta 2013	6

Tiivistelmä — Referat — Abstract

Tiivistelmä.

Avainsanat — Nyckelord — Keywords
agile, ketterä, open source, avoin

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Tämän kirjoitelman tarkoituksena on tarkastella ohjelmistotuotannon menetelmiä, niiden historiaa, lähestymistapaa ohjelmistotuotantoon ja menetelmien heikkouksia sekä vahvuuksia. Ohjelmistotuotantomentelmät ovat muodostuneet eri aikoina erilaisista lähtökohdista ja eri menetelmät sopivat erilaisiin ohjelmistotuotantoprojekteihin.

Kirjoitelma käy lyhyesti läpi erilaisia lähestymistapoja ohjelmistotuotantoon ja miten erilaisten ohjelmistotuotantoprojektien erityispiirteet vaikuttavat menetelmän valintaan.

Ohjelmistotuotannossa on kaksi perustavanlaatuista vaihetta: analysointivaihe ja rakennusvaihe. Nämä kaksi vaihetta riittävät ohjelmiston totetuttamiseen, jos ohjelmisto on pieni ja tuotettavan ohjelmiston käyttäjät ovat itse toteuttajia.

Tällaisesta ohjelmistokehityksestä myös asiakkaat ovat valmiita maksamaan, sillä vaiheet pitävät sisällään aidosti luovaa työtä, joka suoraan edistää tuotettavan ohjelmiston käytettävyyttä.

Suuremman ohjelmistotuotantoprojektin täytäntöönpano vaatii lisäksi muita vaiheita, jotka eivät suoraan edistä tuotettavaa ohjelmistoa ja lisäksi kasvattavat ohjelmistotuotannon kustannuksia.[2]

Ohjelmistotuotannon alkuaikoina käytetty ”ohjelmoi ja korjaa” -mallin sisältää kaksi vaihetta. Ohjelmoidaan ensin ja mietitään vaatimuksia, rakennetta sekä testausta myöhemmin. Mallilla oli useita heikkouksia. Usean korjausvaiheen jälkeen ohjelmakoodi oli niin vaikeasti rakennettu, että oli hyvin kallista muuttaa koodia. Tämä korosti tarvetta suunnitteluaiheelle ennen ohjelmointia.

Usein hyvin suunniteltu ohjelmisto ei vastannut käyttäjien toiveita. Joten syntyi tarve vaatimusmäärittelylle ennen suunnitteluvaihetta.

Ohjelmistot olivat usein kalliita korjata koska muutoksiin ja testaamiseen oli valmistauduttu huonosti. Tämä osoitti tarpeen eri vaiheiden tunnistamiselle, sekä tarpeen huomioida testaus ja ohjelmiston muuttuminen jo hyvin varhaisessa vaiheessa.[1]

2 Ohjelmistotuotantomenetelmät

2.1 Vesiputousmalli

Vesiputousmalli lähestymistapa auttoi poistaa monia ongelmia aiemmin törmänneet ohjelmistoprojekteissa. vesiputousmalli on tullut perusta useimmat ohjelmistot hankinta standardeja hallituksen ja teollisuuden. Jotkut sen ini- OLEVIIN ongelmia on käsitelty lisäämällä laajennuksia koskemaan vähitellen kehitys

2.1.1 Vaiheet

1970-luvulla vesiputousmalli vaikutti suuresti eri vaiheisiin perustuviin ohjelmistotuotannon malleihin. Vesiputousmallin lähestymistapa auttoi poistamaan monia aiemmin ohjelmistotuotantoa vaivanneita ongelmia. Vesiputousmallista tuli perusta monille teollisuuden ja hallituksen ohjelmistohankintojen standardeille. [1] Tarkemmin Winston W. Roycen malli sisältää seuraavat vaiheet: järjestelmä- ja ohjelmistovaatimusmäärittely, analyysi, ohjelmistonrakenteen suunnittelu, rakennus, testaus ja ohjelmiston käyttäminen. Perättäisten ohjelmistotuotantovaiheiden välillä on iteraatiota järjestelmän rakenteen tarkentuessa yksityiskohtaisemmaksi tuotannon edetessä. Iteraatioiden tarkoituksena on suunnitelman edetessä pitää muutosvauhti käsiteltävän kokoisena.[2]

2.1.2 Ohjelmiston suunnittelu

Lineearinen ohjelmistotuotantoprosessi sisältää huomattavan riskin. Vasta testivaiheessa, menetelmän loppupuolella, saattaa tulla esille ilmiöitä, joita ei ollut mahdollista tarkalleen analysoida aikaisemmassa vaiheessa. Ellei pieni muutos koodissa korjaa ohjelmistoa vastaamaan oletettua käytöstä, vaadittavat muutokset ohjelmiston rakenteeseen saattavat olla niin häiritseviä, että muutokset rikkovat ohjelmistolle asetettuja vaatimuksia. Tällöin joko vaatimuksia tai suunnitelmaa on muutettava. Tässä tapauksessa tuotantoprosessi on palannut alkuun ja kustannusten voidaan olettaa nousevan jopa 100%.[2]

Ongelman korjaamiseksi vaatimusmäärittelyn jälkeen - ennen analyysia - on tehtävä alustava rakenteen suunnittelu. Näin ohjelmistosuunnittelija välttää talletamiseen tai aika -ja tilavaatimuuksiin liittyvät virheet. Analyysin edetessä ohjelmistosuunnittelijan on välitettävä aika- ja tilavaatimukset sekä operatiivisiset rajoitteet analyysin tekijälle.

Näin voidaan tunnistaa projektille varatut alimitoitetut kokonaisresurssit tai virheellinen operatiivinen suunnitelma aikaisessa vaiheessa. Vaatimukset ja alustava suunnitelma voidaan iteroida ennen lopullista suunnitelmaa, ohjelmointia ja testausvaihetta.[2]

2.1.3 Dokumentointi

On laadittava ymmärrettävä, valaiseva ja ajantasainen dokumentti, jonka jokaisen työntekijän on sisäistettävä. Vähintään yhden työntekijällä on oltava syvällinen ymmärrys koko järjestelmästä, mikä on osaltaan saavutettavissa dokumentin laadinnalla. Ohjelmistuotannon hyvin tärkeä sääntö on erittäin kattava dokumentointi. Ohjelmistosuunnittelijoiden on kommunikoitava rajapintojen(interface) suunnittelijoiden, ja johdon kanssa. Dokumentti antaa ymmärrettävän perustan rajapintojen suunnitteluun ja hallinnollisiin ratkaisuihin. Kirjallinen kuvaus pakottaa ohjelmistosuunnittelijan yksiselitteiseen ratkaisuun ja tarjoaa konkreettisen todistuksen työn valmistumisesta.[2]

Hyvän dokumentoinnin todellinen arvo ilmenee tuotannossa myöhemmin testausvaiheessa, ohjelmistoa käytettäessä sekä uudelleen suunniteltaessa. Hyvän dokumentin avulla esimies voi keskittää henkilöstön ohjelmistossa ilmenneisiin virheisiin. Ilman hyvää dokumenttia, ainoastaan ohjelmistovirheen alkuperäinen tekijä kykenee analysoimaan kyseessä olevan virheen.[2]

Dokumentti helpottaa ohjelmiston käyttöönottoa operatiivinen henkilöstön kanssa. Käyttöönotossa ilmenneiden mahdollisten ohjelmistovirheiden korjaamisessa selkeä dokumentti on välttämätön.[2]

2.1.4 Toinen versio

Dokumentoinnin jälkeen toinen ohjelmistoprojektin onnistumiseen vaikuttava tärkein tekijä on sen alkuperäisyys. Jos kyseessä olevaa ohjelmistoa kehitetään ensimmäistä kertaa, on asiakkaalle toimitettava käyttöönotettava versio oltava toinen versio, mikäli kriittiset rakenteelliset ja operatiiviset vaatimukset on huomioitu.

Lyhyessä ajassa suhteessa varsinaiseen aikatauluun suunnitellaan ja rakennetaan prototyyppiversio ennen varsinaista rakennettavaa ohjelmistoa. Jos suunniteltu aikataulu on 30 kuukautta, niin pilottiversion aikataulu on esimerkiksi 10 kuukautta. Ensimmäinen versio tarjoaa aikaisen vaiheen simulaation varsinaisesta tuotteesta.[2]

2.1.5 Testaus

Testaus on projektin resursseja vaativin vaihe. Testausvaiheessa vallitsee suurin riski taloudellisesti ja ajallisesti. Loppuvaiheessa aikataulua on vähän varasuunnitelmia tai vaihtoehtoja. Alustava suunnitelma ennen analysointia ja ohjelmointia sekä prototyypin valmistaminen ovat ratkaisuja ongelmien löytämiseen ja ratkaisemiseen ennen varsinaiseen testivaiheeseen siirtymistä.

Testivaiheen tulee pääasiallisesti suorittaa siihen erikoistunut asiantuntija, joka ei välttämättä osallistunut varsinaiseen ohjelmointiin. Väite, että ohjelmistosuunnittelija on paras henkilö testaamaan suunnitelemansa ohjelmiston, koska ymmärtää aihealueen parhaiten, on merkki siitä että dokumentointi ei ole ollut riittävää.

Useimmat virheet ovat luonteeltaan ilmiselviä, jotka voidaan löytää visuaalisella tarkastelulla. Jokaisen analyysin ja ohjelmakoodin tulee tarkastaa toinen henkilö, joka ei osallistunut varsinaiseen työhön. Jokainen tietokoneohjelman looginen polku on testattava ainakin kerran.[2]

2.1.6 Asiakas

Jostain syystä ohjelmiston suunnitelmaan ja aiottuun toimintaan sovelletaan laajaa tulkintaa, jopa aikasemman yhteisymmärryksen jälkeen. On

tärkeää sitouttaa asiakas formaalilla tavalla mahdollisimman aikaisessa vaiheessa projektia, näin asiakkaan näkemys, harkinta ja sitoumus vahvistaa kehitystyötä.[2]

2.2 Itratiivinen menetelmä

Lineaarisesti vaiheesta toiseen etenevän ohjelmistotuotantomalli ei sopinut erityisesti interaktiivisiin loppukäyttäjien sovelluksiin. Suunnitelmaperustaiset standardit pakottivat dokumentoimaan yksityiskohtaisesti heikosti ymmärretyistä käyttöliittymän vaatimuksista. Tätä seurasi suuri määrä käyttökelvottoman koodin suunnittelu ja ohjelmointia.

2.3 Ketterät kehitysmenetelmät

3 Ohjelmistojen laadun varmistus

3.1 Laadun varmistus vesiputousmallissa

3.2 Laadun varmistus ketterissä menetelmissä

4 Ohjelmistotuotantomenetelmän valinta

4.1 Ohjelmistotuotannon riskit

4.2 Ennustettavuus

4.3 Muuttuva liiketoimintaympäristö

4.4 Organisaation koko

4.5 Kehittäjien taidot

4.6

5

6

7 Lähteet

[1] Boehm, B. W.: *A spiral model of software development and enhancement.* Computer, 21:61–72, may 1988, ISSN 0018-9162.

[2] Royce, Winston W: *Managing the development of large software systems.* Teoksessa *proceedings of IEEE WESCON*, nide 26. Los Angeles, 1970.