Unified Process for EDUcation [UPEDU]

Sami Pietinen
Joensuun yliopisto
Tietojenkäsittelytieteen ja tilastotieteen laitos
30.8.2007

Sisällys

1. Johdanto....3
2. Konseptuaalinen malli taustalla....4
3. UPEDU-prosessi....5
3.1 Prosessin vaiheista....7
3.2 Disipliineistä....8
3.3 Roolit....9
3.5 Aktiviteeteista....19
3.6 Artefakteista....19
3.5 Miten UPEDU eroaa RUPista?....21
4. Prosessin vaiheet....22
4.1 Alkuvaihe....22
4.2 Suunnittelu-/Tarkennusvaihe....24
4.3 Rakennusvaihe....26
4.4 Siirtymävaihe....28
5. Disipliinit....30
5.1 Vaatimustenhallinta....30
5.2 Analysointi ja suunnittelu....34
5.3 Implementaatio....37
5.4 Testaus....39
5.5 Konfiguraation ja muutostenhallinta....42
5.6 Projektinhallinta....44
9. Laadusta....49
10. Mittaamisesta....49
11. Prosessin arvioinnista ja parantamisesta....50
12. Käytetyt työkaluohjelmat....51
Viitteet....52
Liite1: Dokumenttiin kohdistuneet muutokset....52

1. Johdanto

Tämä UPEDU-prosessimallia esittelevä tuotos kertoo keskeisimmät asiat itse prosessista ja ottaa hieman kantaa myös sen soveltamiseen Ohjelmistoprojektityö -kurssilla. Kurssin aikataulu ja arvioinnin perusteet ovat esitetty erillisessä dokumentissa, joka ei siis kuulu tähän tuotokseen. Tämä kirjoitelma perustuu pääosin www-sivuun www.upedu.org/upedu (viite [1]) ja joitain keskeisimpiä osia on käytetty myös kirjasta Software Engineering Process with the UPEDU (viite [2]).

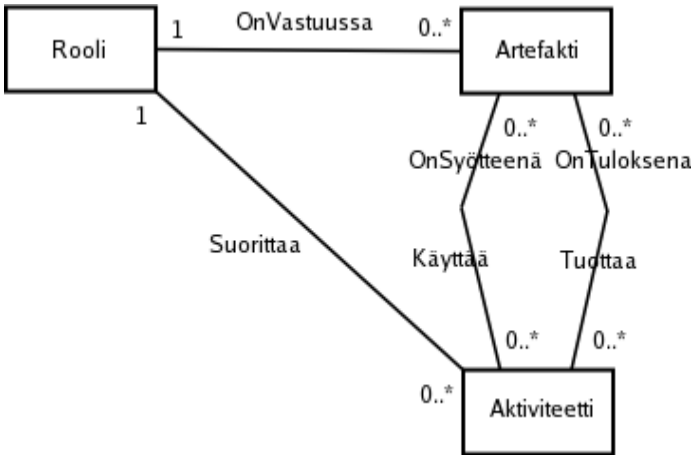
Lähdemme liikkeelle ohjelmistoprosessin käsitteestä. Ohjelmistoprosessi (software process) antaa ohjeita ja suuntaviivoja tehokkaalle ohjelmistojen kehitykselle ja evoluutiolle, perustan ohjelmistojen peräkkäisille sukupolville, vähentää riskien mahdollisuuksia ja vaikutuksia, parantaa tuottavuutta, laatua ja ennakoitavuutta, vangitsee ja dokumentoi parhaat käytännöt, joita saadaan aikaisempien kokemisten perusteella, edistää ja ylläpitää yhteistä visiota ja kulttuuria sekä tarjoaa suunnitelman tehokkaiden työkalujen käyttöön. Kokonaisuudessaan ohjelmistoprosessi sisältää kaikki ne aktiviteetit, joita tarvitaan käyttäjien tarpeiden muuttamiseksi toimivaksi ohjelmisto-tuotteeksi. Ohjelmistoprosessi määrittelee tavan, jolla ihmiset toimeenpanevat ja suorittavat ohjelmiston elinkaaren eri vaiheet. [2]

UPEDU on yksinkertaistettu prosessi RUP (Rational Unified Process) ohjelmistoprosessista. Se on tarkoitettu käytettäväksi opetus- tai koulutusympäristössä ja sen tavoitteena on varmistaa laadullisesti korkeatasoisen ohjelmistotuotteen kehitys, joka vastaa loppukäyttäjän tarpeita projektin kuitenkin pysyessä suunnitellussa aikataulussa ja budjetissa. UPEDU ei sovellu teollisuuden tuotantokäyttöön, mutta tarjoaa systemaattisen (disciplined) lähestymistavan tehtävien ja vastuiden jakamiseen kehitysorganisaatiossa onnistuneiden ohjelmistojen kehittämiseksi. Prosessia on yksinkertaistettu, jotta oppija voi keskittyä ohjelmistoprosessin keskeisiin asioihin, ydinasioihin, ja nämä ydinasiat painottuvat prosessin kognitiivisiin toimintoihin. Tällaiseen rakenteelliseen ratkaisuun liittyy oletus, että oppija saavuttaa sen avulla perusymmärryksen, josta hän saa vankan pohjan laajennettujen (extended) ohjelmistoprosessin aktiviteettien suorittamiseen tulevaisuudessa [2].

UPEDU (Unified Process for EDUcation) on iteratiivinen ohjelmistoprosessi ohjelmien inkrementaaliseen kehittämiseen. *Iteratiivisuuden* ideana on, että projektin vaiheet voidaan jakaa pienemmiksi hallintarakenteiksi, *iteraatioiksi*. *Inkrementaalisuus* puolestaan tarkoittaa ohjelman kehittämistä vähitellen niin, että jokainen valmis *inkrementti*, eli lisäys kehitettävään ohjelmaan, tuo ohjelmaan jotain uusia piirteitä. Inkrementaalisuuden hyötyjä ovat esimerkiksi uusien tuoteversioiden nopea julkaiseminen ja testauksen helpottuminen. Testaus on helpompaa, koska sitä ei jätetä kehitysprosessin loppuvaiheeseen, vaan testausta suoritetaan samanaikaisesti kehityksen kanssa mahdollistaen näin keskittymisen pienempiin kokonaisuuksiin. Inkrementaalinen kehitys myös vähentää virheiden kumuloitumista. Inkrementaalisuus vastaa osaltaan myös ongelmaan, joka syntyy kun kaikkia kehitettävän ohjelman vaatimuksia ei voida selvittää ohjelman kehityksen alkuvaiheessa. Inkrementaalisuus antaa tilaa vaatimuksien kehittymiselle ohjelmiston kehitysprosessin aikana. Jonkinlainen keskeneräinen versio kehitettävästä ohjelmasta auttaa myös asiakasta hahmottamaan millaisia uusia toimintoja ja ominaisuuksia ohjelman tulisi sisältää sekä tarjoaa kontekstin, jonka avulla esittää kehitystiimille kyseiset kehitysideat.

2. Konseptuaalinen malli taustalla

UPEDU perustuu konseptiin, jossa ohjelmistoprosessi nähdään abstraktien aktiivisten entiteettien, roolien kollaboraationa. Roolit suorittavat tehtäviä, joita kutsutaan aktiviteeteiksi. Aktiviteetit kohdistuvat konkreettisiin ja todellisiin entiteetteihin, joita kutsutaan artefakteiksi. *Rooli* määrittelee henkilön vastuut ohjelmistoprosessissa. *Aktiviteetit* ovat työn pienimpiä osia. Työn osittaminen aktiviteetteihin jakaa työn pienempiin helpommin hallittaviin osiin, jolloin työn suunnittelu ja edistymisen seuranta helpottuvat. *Artefaktit* ovat fyysisiä entiteettejä, jotka syntyvät ja kehittyvät aktiviteettien tuloksena. Niitä ovat mm. erilaiset dokumentit, tiedostot, mallit ja mallien elementit. UPEDUn taustalla olevan konseptuaalisen mallin entiteettien, eli roolien, artefaktien ja aktiviteettien, relaatioita esittää kuva 1.



Kuva 1. Konseptuaalinen malli [3].

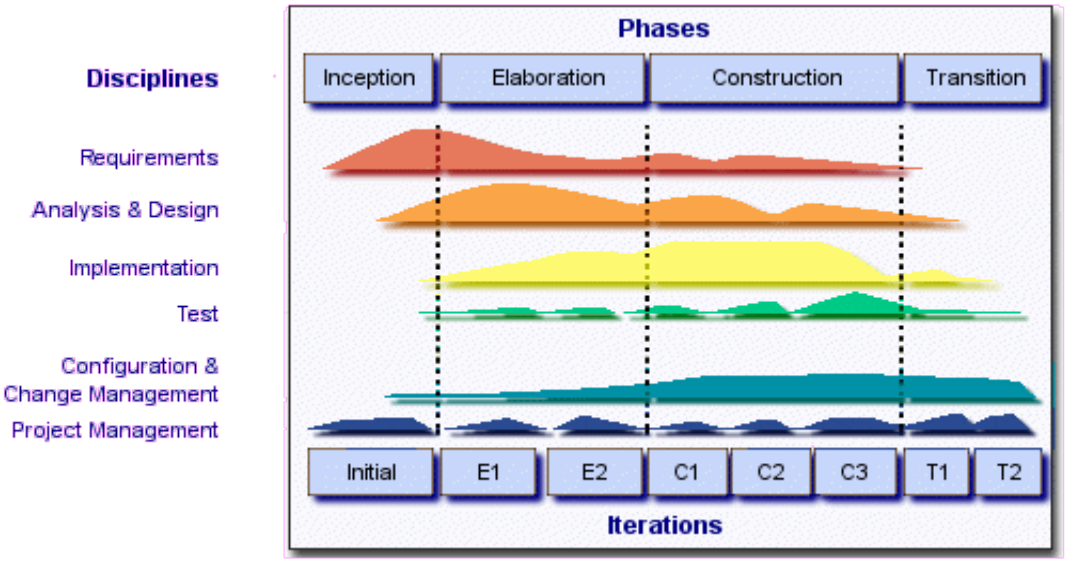
Mallin periaatteena on että rooli, joka on määrätty kehitystiimin jäsenelle, voi olla vastuussa yhdestä tai useammasta artefaktista. Jos rooli kuuluu kehitysorganisaation ulkopuoliselle yksilölle, saattaa olla että rooliin ei silloin kuulu varsinaisia vastuita artefakteista, mutta roolin omaavalla yksilöllä voi olla oikeus esimerkiksi lukea ja tarkastaa tiettyjä artefakteja tai luoda uusiakin artefakteja, esimerkiksi muutospyyntöjä. Rooli suorittaa aktiviteetteja, joiden tuloksena voi olla yksi tai useampi artefakti. Aina aktiviteetti ei kuitenkaan luo uutta artefaktia tai muuta sitä. Aloitettua aktiviteettia ei tarvitse suorittaa kerralla loppuun asti, vaan aktiviteetin pariin voidaan palata useampaan kertaan prosessin eri vaiheissa, jolloin osittainen aktiviteetin suorittaminen aiheuttaa vain lisäyksen artefaktiin, mutta ei rakenna sitä valmiiksi. Tätä kutsutaan kehityssyklin läpi kulkevaksi artefaktien kehittymiseksi. Artefakti voi toimia syötteenä aktiviteetille, joka taas voi käyttää artefaktia.

3. UPEDU-prosessi

Aktiviteettien ja artefaktien joukkoa, jota todella käytetään organisaatiossa, kutsutaan organisaation *prosessimalliksi* (process model). Aktiviteettien ja artefaktien joukkoa, joka on määritelty yleisellä tasolla ei millekään tietylle organisaatiolle, ja joka sisältää kaikki disipliinien käytännöt, kutsutaan *referenssiprosessimalliksi* (reference process model). UPEDU-prosessimalli vastaa tasoja 2 ja 3 SEIn (Software Engineering Institute) kehittämässä SW-CMMn (Software Capability Maturity Model) referenssiprosessimallissa [2] .

UPEDU-prosessimalli sisältää kaksi ulottuvuutta. Horisontaalinen ulottuvuus sisältää neljä ajallisesti toisiaan seuraavaa vaihetta (phase): alkuvaihe (inception), suunnittelu-/tarkennusvaihe (elaboration), rakennusvaihe (construction) sekä siirtymävaihe (transition). Vertikaalinen ulottuvuus sisältää projektissa suoritettavat disipliinit (disciplines), jotka ryhmittävät projektissa suoritettavat aktiviteetit loogisiin ryhmiin.

Disipliinejä ovat vaatimustenhallinta (requirements), analysointi ja suunnittelu (analyses and design), implementointi (implementation), testaus (test), konfiguraation ja muutosten hallinta (configuration and change management) sekä projektin hallinta (project management). Prosessin elinkaaren vaiheet ja disipliinit linkittyvät toisiinsa iteraatioiden avulla ja yksittäinen vaihe voi sisältää yhden tai useamman iteraation disipliinejä riippuen projektikohtaisista ominaispiirteistä. Iteraatiot ovat hallinnallisia rakenteita, jotka mahdollistavat elinkaaren vaiheiden loppuun saattamisen kontrolloinnin käyttäen disipliineille tarkoituksenmukaisia aktiviteetteja ja artefakteja. Iteraatio on tavallaan miniprojekti. UPEDU-prosessi on esitetty kuvassa 2.

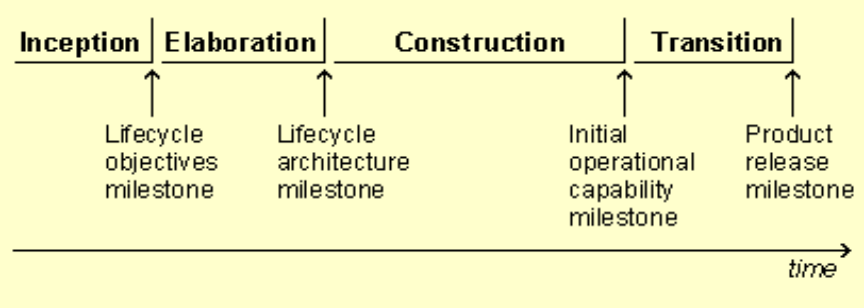


Kuva 2. UPEDU-prosessi [1].

Kuvassa 2 olevasta tyypillisestä disipliinien painotuksesta nähdään, että yleensä alkuvaiheessa aikaa käytetään eniten vaatimustenhallintaan (requirements), josta painopiste siirtyy analysoinnin ja suunnittelun kautta implementaatioon ja sen jälkeen testaukseen. Kaikki disipliinit ovat yleensä läsnä jokaisessa vaiheessa, vaikka painopiste työmäärien suhteen vaihteleekin.

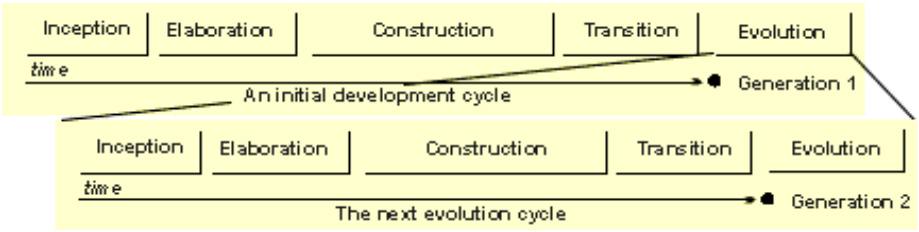
Kuvasta 3 nähdään, että jokainen prosessin vaihe päättyy nimettyyn merkkipaaluun tai virstanpylvääseen. Horisontaalinen ulottuvuus on käytäntöön panonsa suhteen (vaiheet, iteraatiot,

virstanpylväät) luonteeltaan dynaaminen kun taas vertikaalinen ulottuvuus on staattinen (prosessin komponentit, aktiviteetit, työnkulut, artefaktit ja roolit). Jokaisen virstanpylvään kohdalla suoritetaan arviointi siitä, onko vaiheen tavoitteet ja päämäärät saavutettu ja näin asian ollessa voidaan siirtyä seuraavaan vaiheen suorittamiseen. Jos tavoitteita ei ole saavutettu, voidaan tilanne ratkaista esimerkiksi vaiheen suoritusaikaa pidentämällä. Virheiden ja puutteiden korjauksia ei kannata ajoittaa myöhempään ajankohtaan vaan ne kannattaa korjata välittömästi löytymisen jälkeen.



Kuva 3. Prosessin vaiheet ja virstanpylväät [1].

UPEDU mahdollistaa kaikkien kehitysvaiheiden läpikäynnin tarvittaessa useampaan kertaan. Yksittäistä kaikkien vaiheiden läpikäyntiä kutsutaan *kehityssykliksi* (development cycle) ja se tuottaa ohjelmistosta uuden sukupolven (generation). Kehityssykleillä on tavallisesti toisistaan eriävät painotukset disipliinien työmäärissä. Kehityssykliä seuraavia vaiheiden läpikäyntejä kutsutaan *evoluutiosykleiksi* (evolution cycles). Evoluutiosyklin alkuvaihe sekä sitä seuraava suunnittelu-/tarkennusvaihe ovat tavallisesti kehityssyklin vastaaviin vaiheisiin verrattuna pienempiä. Syklit voivat olla puhtaasti peräkkäisiä, mutta yleisempää on vaiheiden osittainen päällekkäisyys. *Ohjelmiston elinkaari* (software lifecycle) koostuu kehityssyklistä ja sitä mahdollisesti seuraavista evoluutiosykleistä. Syklejä ja sukupolvia on esitetty kuvassa 4.



Kuva 4. Syklit ja sukupolvet [1].

UPEDU-prosessille on olemassa web-pohjainen *prosessityökalu*, joka sisältää joukon aktiviteetteja ja artefakteja, jotka on kuvattu yleisellä tasolla niin, että se sisältää kaikki tietyn disipliinin käytännöt ei millekään tietylle organisaatiolle. Kyseinen prosessityökalu sisältää siis UPEDUn referenssiprosessimallin. Se löytyy osoitteesta http://www.yoopeedoo.org/ (kohta UPEDU) tai suoraan osoitteesta http://www.upedu.org/upedu/.

3.1 Prosessin vaiheista

Kehityssykliin kuuluu neljä vaihetta: alkuvaihe (konsepti), suunnittelu-/tarkennusvaihe (suunnittelu), rakennusvaihe (toteutus) sekä siirtymävaihe (tuotteen toimitus).

Alkuvaiheessa tarkoituksena on tehdä business case eli oikeutus kehitettävälle ohjelmistolle. Alkuvaihetta voidaan kutsua myös nimellä konseptivaihe. Vaiheen keskeisiä tehtäviä ovat alustava vaatimusten laatiminen, alustava riskien arviointi, operationaalisen vision luominen, projektia tukevan ympäristön valmisteleminen ja tarvittaessa konseptuaalisen prototyypin rakentaminen. Lisäksi erotellaan kriittiset käyttötapaukset ja primääriset skenaariot, joiden vaikutuksesta saatetaan joutua tekemään suunnittelussa kompromisseja. Vaihe määrittelee projektin laajuuden ja tätä laajuutta tulee hallita, jotta projektin aikana tulevat muutokset eivät paisuttaisi aikataulua ja budjettia hallitsemattomasti. Alkuvaiheen virstanpylvästä kutsutaan nimellä tavoite-virstanpylväs (lifecycle objectives milestone). Alkuvaiheen jälkeen seuraava vaihe on suunnittelu-/tarkennusvaihe.

Suunnittelu-/tarkennusvaiheessa suunnitellaan projekti, määritellään ohjelmiston ominaisuudet ja toiminnot sekä luodaan systeemin perustaso (system baseline). *Perustaso* tarkoittaa katselmoitua ja hyväksyttyä julkaisuversiota (release) artefaktien joukosta, jota voidaan muuttaa vain formaalin muutostenhallinnan tai konfiguraationhallinnan kautta. Se vastaa sovittua pohjaa jatkokehitykselle ja evoluutiolle. *Julkaisuversio* on lopputuotteen osajoukko, joka on arvioinnin kohteena suurimmissa virstanpylväissä. Se on vakaa (stable), suoritettavissa oleva versio tuotteesta, johon kuuluu myös kaikki artefaktit, joita tarvitaan tuotteen käyttöön, kuten julkaisutiedote (release notes) tai asennusohjeet. Julkaisuversio voi olla sisäinen tai ulkoinen. *Sisäistä julkaisuversiota* käyttää kehitysorganisaatio osana virstanpylvästä tai demonstroidakseen sitä käyttäjille tai asiakkaille. *Ulkoinen julkaisuversio* taas toimitetaan loppukäyttäjille. Julkaisuversioiden avulla voidaan paremmin välttyä ”90% tehty, 90% jäljellä” syndroomalta. Tässä vaiheessa varmistetaan myös projektin rahoitus, valitaan ulkoiset toimittajat jos sellaisia tarvitaan. Vaiheen tarkoituksena on analysoida sovellusalueen ratkaisu, jolle ohjelmiston arkkitehtuuri muodostetaan ja kehittää kokonaisvaltainen suunnitelma, jossa eri riskielementit otetaan huomioon. Vaiheen tuloksena syntyy malli systeemin toiminnasta, johon sisältyy systeemin konteksti, skenaariot, sovellusalueen malli (domain model), perustason tuotevisio, joka perustuu sovellusalueen malliin, kehityssuunnitelma (development plan), arviointikriteerit sekä tuotteen julkaisun kuvaus (product release description). Tässä vaiheessa voidaan tarvittaessa kirjoittaa myös alustava käyttöohje ja testaussuunnitelma. Suunnitteluvaiheen virstanpylvästä kutsutaan nimellä arkkitehtuuri-virstanpylväs (lifecycle architecture milestone). Suunnitteluvaihetta seuraa rakennusvaihe.

Rakennusvaihe sisältää ohjelmistotuotteen ohjelmoimisen. Vaiheen tuloksena syntyy suoritettava koodi, systeemi- ja käyttäjädokumentaatio, käyttöönottosuunnitelma sekä joitain laadunvarmistustuloksia. Vaiheen tuloksena voi syntyä myös toiminnallinen prototyyppi (behavioral prototype). Rakennusvaiheen virstanpylväänä on toimiva ohjelma. Vaiheen virstanpylvään nimeksi on annettu alustavan operationaalisen kyvykkyyden virstanpylväs (initial operational capability milestone).

Viimeisenä kehityssyklin vaiheena on *siirtymävaihe*, joka sisältää ohjelmistotuotteen luovutuksen käyttäjälle. Tässä vaiheessa voidaan tehdä myös pientä loppuviimeistelyä esimerkiksi ohjelman mukana tuleviin tukimateriaaleihin tai itse ohjelmaan ja sen asetuksiin sekä testata tuotteen toimivuus asiakkaan ympäristössä tai sitä vastaavassa ympäristössä. Vaiheen tarvittaviin artefakteihin kuuluvat kaikki projektin valmiit artefaktit. Tämän vaiheen virstanpylväänä toimii tuotteen julkaisu virstanpylväs (product release milestone).

3.2 Disipliineistä

UPEDU käyttää kuvassa 1 horisontaalisella ulottuvuudella olevista toiminnoista nimitystä disipliini. *Disipliini* on roolien, aktiviteettien ja artefaktien integraatio, joka johtaa mainittuja osia suurempaan entiteettiin. Prosessi taas muodostuu disipliinien joukosta. Disipliini voidaan kuvailla myös aktiviteettien joukkona, missä yhden aktiviteetin tulos (output) on toisen aktiviteetin syöte (input). Disipliinit voidaan suorittaa roolin toimesta missä järjestyksessä tahansa, edellyttäen kuitenkin, että disipliinin käyttämät syöteartefaktit ovat saatavilla.

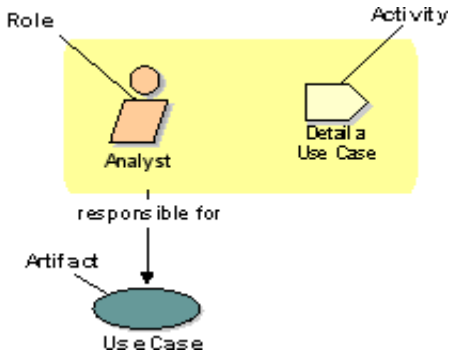
Disipliinejä on kuusi kappaletta: vaatimustenhallinta, analysointi ja suunnittelu, implementointi, testaus, konfiguraation ja muutosten hallinta sekä projektin hallinta. Ne voidaan jakaa kahteen ryhmää: kehitys (engineering) ja hallinta (management) -disipliineihin. Kehitys-disipliineihin kuuluvat vaatimustenhallinta, analysointi ja suunnittelu, implementointi sekä testaus. Hallinta-disipliineihin taas konfiguraation ja muutosten hallinta sekä projektin hallinta.

Kehitys-disipliineihin kuuluvat ohjelmistoprosessin tekniset aspektit ja ne keskittyvät artefaktien ja kehitettävään tuotteeseen liittyvien toimitettavien (deliverables) luomiseen. Asiakkaan tarpeet muutetaan ohjelmiston vaatimuksiksi, vaatimukset ohjelmiston malliksi (design), mallin implementoidaan kooditasolla ja lopuksi ohjelmisto testataan. Vaatimus-disipliini määrittelee käyttötapaus-kaavioiden avulla systeemin käyttäytymisen. Analyysi ja suunnittelu-disipliini muuttaa vaatimukset spesifikaatioksi, joka kertoo miten systeemi tulee implementoida eli tuloksena on suunnittelumalli (design model) toteutettavalle järjestelmälle. Implementaatio-disipliini määrittelee koodin organisoinnin, toteuttaa luokat ja objektit, testaa kehitetyt komponentit ja integroi ne suoritettavaksi systeemiksi. Testaus-disipliini arvioi tuotteessa saavutetun laadun tasoa ja raportoi tulokset.

Hallinta-disipliineihin kuuluva konfiguraation ja muutosten hallinta identifioi, määrittelee ja asettaa perustason (baseline) tuotteeseen kuuluville osille. Perustaso antaa virallisen standardin, johon peräkkäinen työskentely perustuu ja johon voidaan tehdä muutoksia vain ennalta määritellyn muutostenhallintamenettelyn avulla. Tämä disipliini kontrolloi siis tuotteen muutoksia ja tuotteen perustasoon kuuluvien osien julkaisuversioita. Disipliiniin kuuluu myös tuotteen osien ja muutospyyntöjen tilan kirjanpito sekä raportointi, tuotteen osien täydellisyyden, yhdenmukaisuuden, oikeellisuuden varmistaminen sekä tallennuksen, käsittelyn ja toimituksen kontrollointi ja hallinta. Muutosten- ja konfiguraationhallinta on keskeinen osa projektin laajuuden hallintaa.

3.3 Roolit

Rooli on abstrakti määritelmä aktiviteettien suorittajalle. Rooli on yleensä vastuussa joistakin artefakteista. Kehitystiimin jäsen voi ottaa roolin, joka voi kuulua yksilölle tai ryhmälle. Roolin ajureina toimivat aktiviteetit, jotka on suoritettava tiettynä ajankohtana. Yksittäisellä henkilöllä voi olla samanaikaisesti useita eri rooleja. Yhtä lailla usealla eri henkilöllä voi olla sama rooli ja he suorittavat samaa aktiviteettia, jolloin kyseessä on ryhmälle kuuluva rooli. Roolit eivät ole yksilöitä, vaan ne kuvaavat yksilöiden vastuita. Roolien ei ole tarkoitus varsinaisesti rajoittaa aktiviteetteja, joita yksilö voi suorittaa, vaan painottaa disipliinien kognitiivisia aktiviteetteja, vaikka varsinkin suuremmissa projekteissa voi olla perusteltua, että joidenkin roolien haltijat olisivat eri henkilöitä. Suurin osa rooleista realisoituu kehitysorganisaation sisällä, mutta myös kehitysorganisaation ulkopuolisilla rooleilla on tärkeä merkitys. Tällainen kehitysorganisaation ulkopuolinen rooli on esimerkiksi osakas (stakeholder). Roolin, aktiviteetin ja artefaktin suhdetta selventää kuva 6.

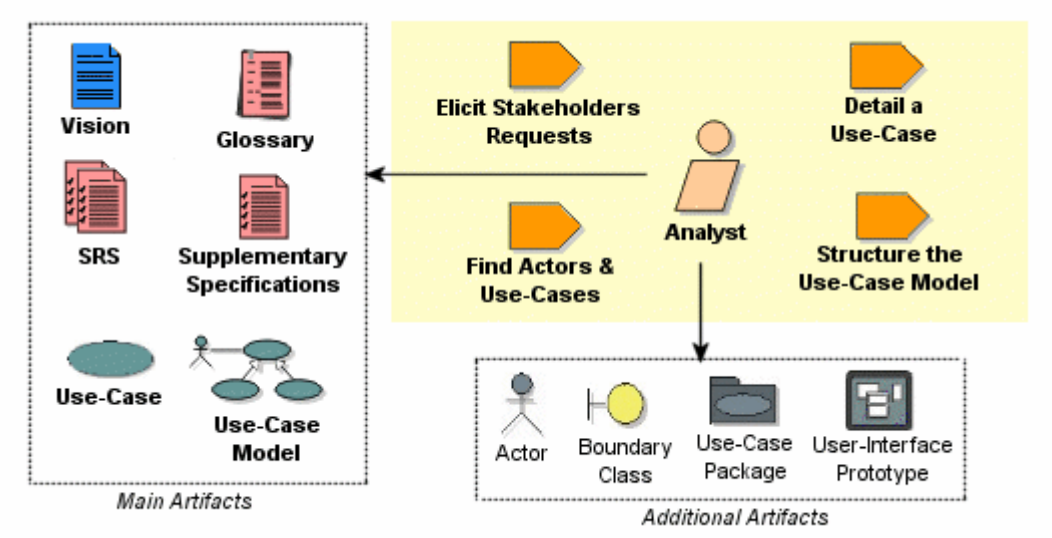


Kuva 6. Esimerkki roolista, aktiviteetista ja artefaktista [1].

Rooleilla on joukko yhteneväisiä aktiviteetteja, joita ne suorittavat. Nämä aktiviteetit liittyvät läheisesti toisiinsa ja ovat funktionaalisesti kytkettyjä. Yleensä tällaisen joukon aktiviteetteja suorittaa parhaiten sama yksittäinen henkilö. Aktiviteetit liittyvät läheisesti artefakteihin niin, että artefaktit ovat aktiviteettien syötteitä ja tulosteita. Artefaktit toimivat myös aktiviteettien välisenä informaation välityskeinona. UPEDU-prosessi tunnistaa ainakin seuraavat roolit: analysoija (Analyst), suunnitteluja (Designer), implementoija (Implementer), integroija (Integrator), testaaja (Tester), muutostenhallintavastaava (Change Control Manager), konfiguraatiovastaava (Configuration Manager), projektipäällikkö (Project Manager), katselmoija (Reviewer) ja osakas (Stakeholder). Myös muita rooleja on mahdollista käyttää.

3.3.1 Analysoija

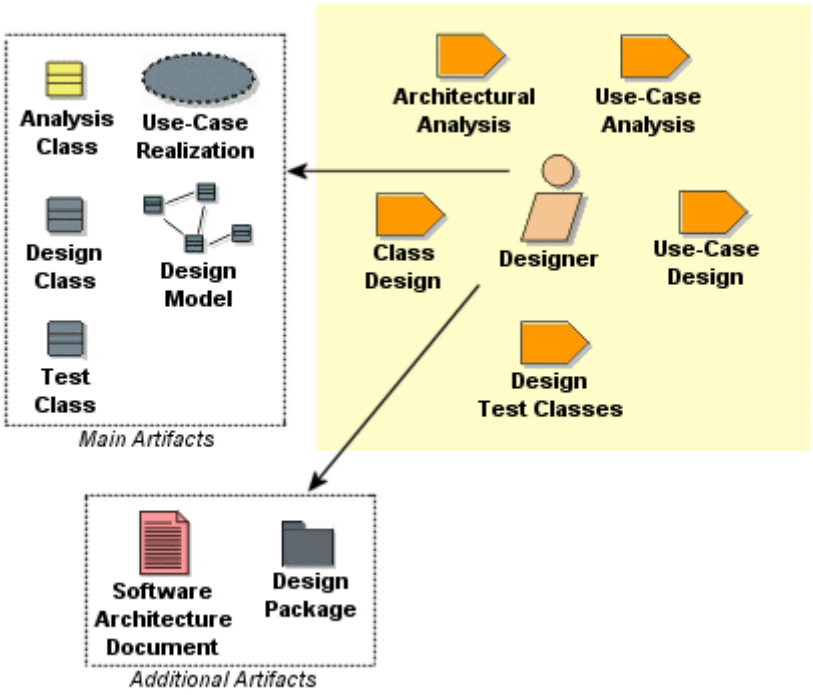
Systeemin analysoijan (Analyst) rooliin kuuluu johtaa ja koordinoida vaatimusten kehittämistä sekä käyttötapausten mallintamista rajaamalla systeemin toimintoja sekä itse systeemin. Analysoijalta vaaditaan hyvää kommunikointitaitoa sekä liiketoiminnan ja teknologian tietoutta. Analysoijan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 7.



Kuva 7. Analysoijan aktiviteetit ja artefaktit [1].

3.3.2 Suunnittelija

Systeemin suunnittelijan (Designer) rooliin kuuluu komponenttien vastuiden, operaatioiden, attribuuttien ja komponenttien välisten riippuvuuksien määrittäminen. Suunnittelija myös ratkaisee miten komponentteja säädetään sopimaan implementaatioympäristöön. Suunnittelijan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 8.

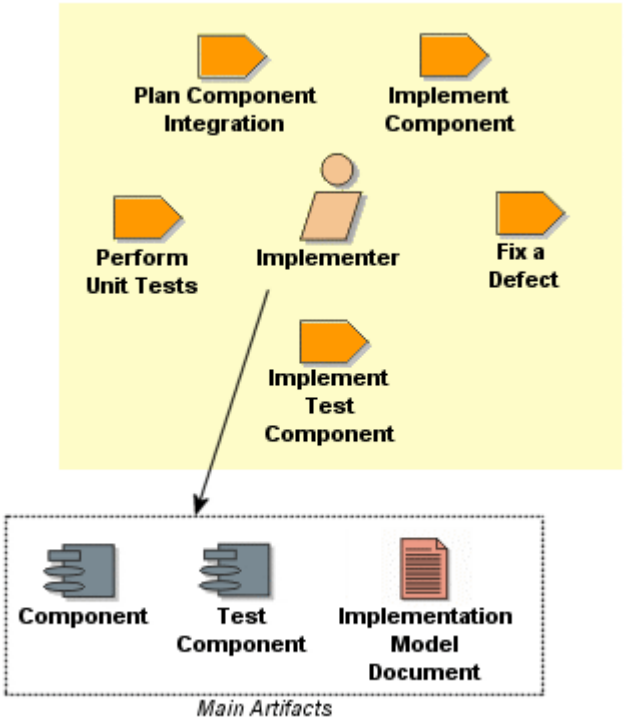


Kuva 8. Suunnittelijan aktiviteetit ja artefaktit [1].

3.3.3 Implementoija

Systeemin implementoijan (Implementer), toisin sanoen toteuttajan, rooliin kuuluu vastuu komponenttien kehittämisestä ja testaamisesta käyttäen projektin standardeja. Nämä komponentit integroidaan myöhemmin suuremmiksi kokonaisuuksiksi, esimerkiksi alijärjestelmiksi. Jos komponenttien testaamisessa joudutaan luomaan ajureita (drivers) tai tynkiä (stubs), kuuluu implementoijan vastuuseen luoda ja testata kyseiset testauskomponentit ja niitä vastaavat alijärjestelmät.

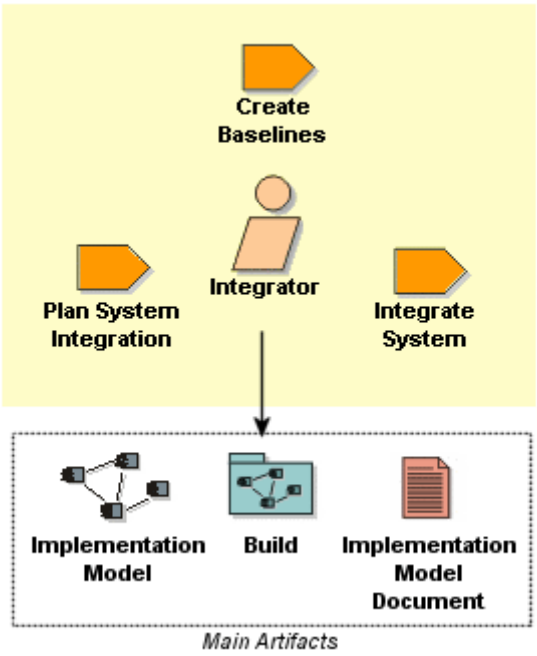
Implementoijalla tulee olla tarvittavat tiedot ja taidot systeemistä tai sovelluksesta jota testataan. Hänen tulee olla perehtynyt testaamiseen ja testaamisessa tarvittaviin työkaluihin sekä testaamisen automatisoiviin työkaluihin. Hänellä tulee olla lisäksi tarvittavat ohjelmointitaidot. On suositeltavaa että implementoijan rooli, joka on vastuussa alijärjestelmän toteuttamisesta, on vastuussa myös alijärjestelmien sisältämistä komponenteista. Implementoijan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 9.



Kuva 9. Implementoijan aktiviteetit ja artefaktit [1].

3.3.4 Integroija

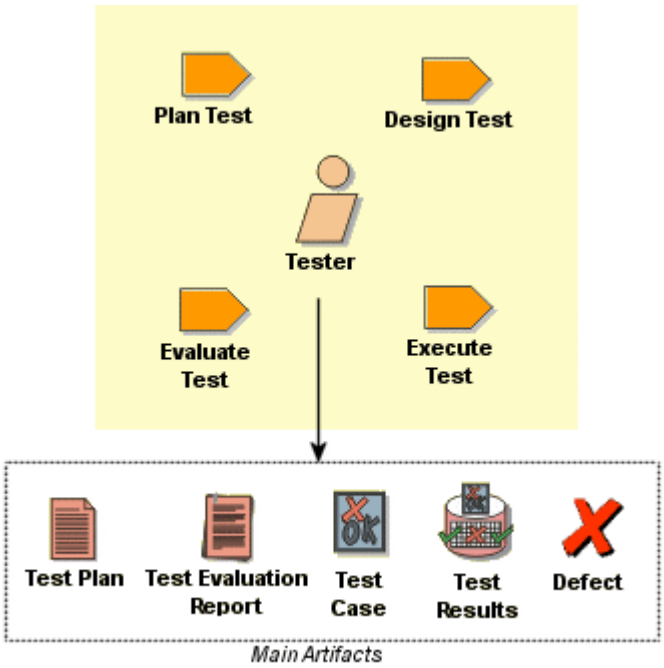
Integroijan (Integrator) vastuuseen kuuluu komponenttien yhdistäminen käännöksen (build) tuottamiseksi. *Käännös* on operationaalinen versio systeemistä tai sen osasta, joka demonstroi lopulliseen tuotteeseen tulevien kyvykkyyksien (capabilities) osajoukkoa. Integroitavat komponentit saadaan integrointi-työtilaan (integration workspace) implementoijilta testattuina. Integroija on vastuussa myös integroinnin suunnittelemisesta, joka tehdään järjestelmä tai alijärjestelmätasolla, joilla jokaisella on oma integrointi-työtilansa. Testatut komponentit toimitetaan implementoijan omasta työtilasta alijärjestelmän integrointi-työtilaan. Integroidut alijärjestelmät toimitetaan alijärjestelmien integrointi-työtilasta järjestelmän integrointi-työtilaan. Integrointi tapahtuu siis systemaattisesti. Erityisesti pienissä projekteissa tai alijärjestelmien integroinnissa voi olla tarpeellisesta että integroija toimii myös testaajan roolissa. Hänellä voi olla vielä lisäksi myös implementoijan rooli. Tällä voidaan joskus saavuttaa henkilöresurssien tehokkaampi käyttö, mutta varsinkin järjestelmätason integroinnissa on yleensä parempi, että integrointi ja testaus suoritetaan eri tiimeissä. Integroijan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 10.



Kuva 10. Integroijan aktiviteetit ja artefaktit [1].

3.3.5 Testaaja

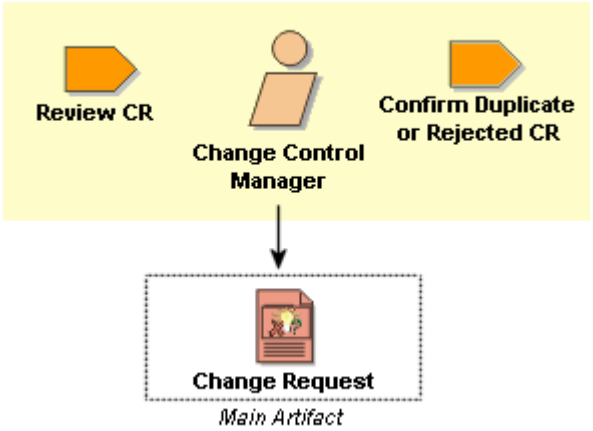
Testaaja (Tester) on vastuussa testauksen ydinaktiviteeteista. Niihin kuuluu tarvittavien testien suoritus ja testauksen tuloksen kirjaus. Testaajan tulee identifioida sopivin implementointitapa testille, implementoida yksittäiset testit, valmistella testausympäristö testien suorittamista varten ja suorittaa testit, kirjata tulokset ja verifioida testauksen suoritus. Myös suorituksessa tapahtuvien virheiden analysointi ja virheistä toipuminen kuuluu testaajan vastuulle. Testaajalla tulee olla tietämystä erilaisista testaamisen lähestymistavoista ja tekniikoista, diagnostisia taitoja sekä ongelmanratkaisutaitoja. On toivottavaa, että testaajalla on tietämystä systeemistä tai sovelluksesta jota testataan sekä tietoverkoista ja järjestelmäarkkitehtuureista. Tarvittavat tiedot ja taidot riippuvat pitkälti siitä, millaisia testejä suoritetaan ja missä vaiheessa projektin elinkaarta. Jos testauksessa käytetään testaamisen automatisoivia työkaluja, tulee testaaja olla lisäksi koulutettu käyttämään kyseistä työkalua, kokemusta testauksen automatisointityökaluista, ohjelmointitaitoja sekä diagnostisia ja virheiden etsintä ja poisto (debugging) -taitoja. Testaajan roolia annettaessa tulee ottaa huomioon ainakin mahdollisten testaajakandidaattien taidot ja kokemus testaamisesta sekä tiimin koko. Esimerkiksi suurissa tiimeissä voi olla tehokkainta antaa yhdelle tai useammalle tiimin jäsenelle ainoastaan testaajan rooli. Testaajan rooliin kuuluvia aktiviteetteja ja artefakteja on esitetty kuvassa 11.



Kuva 11. Testaajan aktiviteetit ja artefaktit [1].

3.3.6 Muutostenhallintavastaava

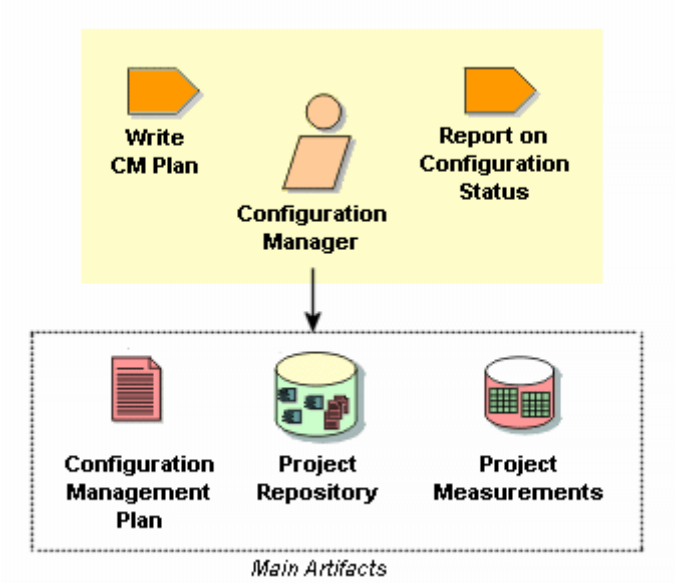
Muutostenhallintavastaava (Change Control Manager) valvoo muutostenhallintaprosessia. Tässä roolissa toimii yleensä konfiguraation (tai muutosten) hallinta neuvosto (Configuration – tai Change – Control Board, CCB). Neuvostoon kuuluu kaikkien eri intressiryhmien edustajia, kuten asiakkaita, kehittäjiä, käyttäjiä. Pienessä projektissa roolin voi täyttää vain yksittäinen henkilökin, kuten esimerkiksi projektipäällikkö tai ohjelmistoarkkitehti. Muutostenhallintavastaava määrittelee myös muutospyyntöjen hallintaprosessin, joka dokumentoidaan konfiguraationhallintasuunnitelmaan (Configuration Management Plan). Muutostenhallintavastaavan tulee osata arvioida muutospyyntöjen vaikutuksia projektin kustannuksiin ja aikatauluun. Hänen tulee osata kommunikoida tehokkaasti neuvotellakseen projektin laajuuden muutoksista ja päätelläkseen miten jokainen muutospyyntö tulee käsitellä ja kenen toimesta. Muutostenhallintavastaavan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 12.



Kuva 12. Muutostenhallintavastaavan aktiviteetit ja artefaktit [1].

3.3.7 Konfiguraatiovastaava

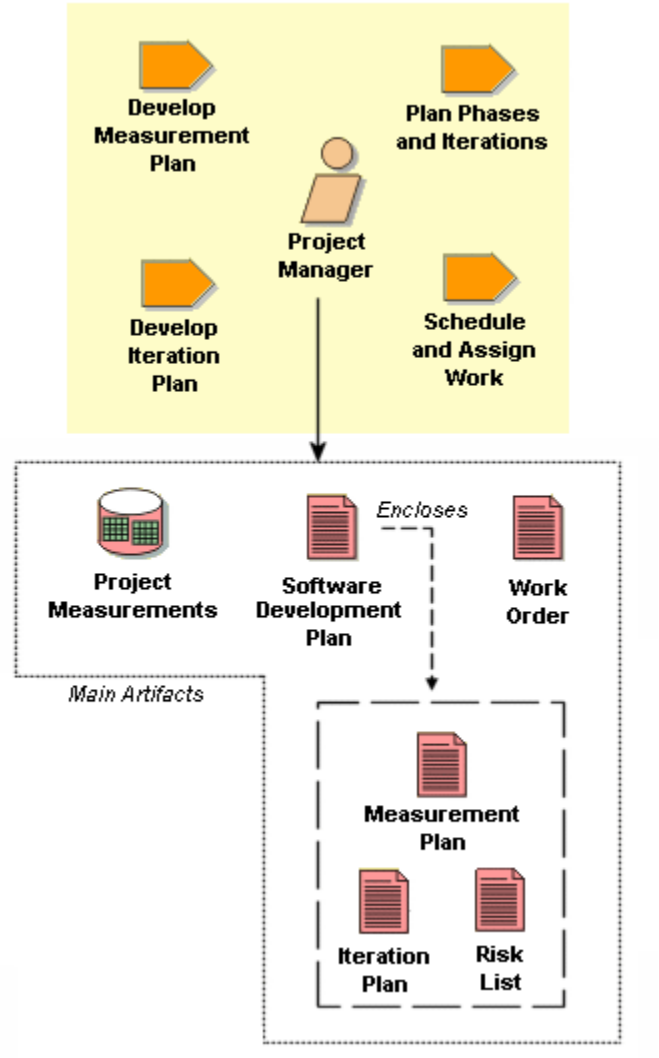
Konfiguraatiovastaava (Configuration Manager) tarjoaa yleisen konfiguraationhallinta-infrastruktuurin ja ympäristön kehitystiimille. Konfiguraationhallinta tukee tuotteen kehitysaktiviteettia tarjoamalla kehittäjille ja integroijille sopivat työtilat, joissa rakentaa ja testata työtään niin, että kaikki artefaktit ovat tarvittaessa saatavilla kyseiseen kehitysversioon. Konfiguraatiovastaavan on myös pidettävä huolta siitä, että kokoonpanoympäristö helpottaa tuotteen tarkastus, muutos ja vianjäljitys aktiviteetteja. Hän kirjoittaa konfiguraationhallinnasta suunnitelman ja raportoi edistymisen statistiikoista, jotka perustuvat muutospyyntöihin. Konfiguraatiovastaavan tulee ymmärtää konfiguraationhallintaan liittyvät periaatteet ja hänellä tulee olla koulutusta tai kokemusta kokoonpanonhallintatyökaluista. Hän kiinnittää huomiota yksityiskohtiin ja on vakuuttava ja jämäkkä, jotta kehittäjät eivät sivuuta kokoonpanonhallinnan menettelytapoja ja proseduureja. Kokoonpanonhallintavastaavan rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 13.



Kuva 13. Kokoonpanovastaavan aktiviteetit ja artefaktit [1].

3.3.8 Projektipäällikkö

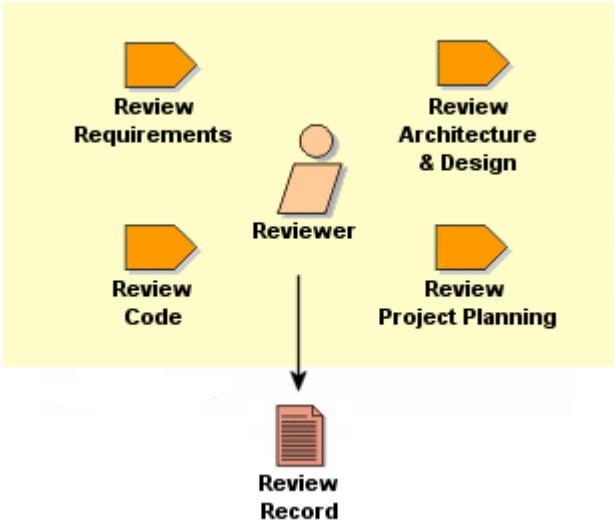
Projektipäällikkö (Project Manager) koordinoi vuorovaikutusta asiakkaan ja käyttäjien kanssa, allokoi resursseja ja priorisoi. Hän pitää projektin tiimin fokuksen oikeissa asioissa. Projektipäällikön tehtäviin kuuluu myös vakiinnuttaa käytännöt, joiden avulla projektin artefaktien integriteetti ja laatu voidaan varmistaa. Projektipäälliköltä vaadittuihin tietoihin ja taitoihin vaikuttaa mm. projektin koko sekä tekninen ja hallinnollinen monimutkaisuus. Hänellä on hyvä olla kokemusta sekä sovellusalueesta että ohjelmistojen kehittämisestä. Häneltä vaadittavia taitoja ovat esimerkiksi riskien analysointi ja hallinta, estimointi, suunnittelu, päätösanalysointi. Hän on fokusoitunut tuottamaan asiakkaalle arvoa, osaa esiintyä, kommunikoida ja neuvotella, osaa objektiivisesti ja totuudenmukaisesti arvioida tuotoksia, on käytännönläheinen projektin laajuutta arvioidessaan sekä implementoidessaan suunnitelmia. Projektipäällikön rooliin kuuluvia aktiviteetteja ja artefakteja esittää kuva 14.



Kuva 14. Projektipäällikön aktiviteetit ja artefaktit [1].

3.3.9 Katselmoija

Katselmoijan (Reviewer) tehtäviin kuuluu formaalien katselmuksien suunnittelu ja pitäminen eri disipliineissä. Katselmoinnilla voidaan tarkoittaa tässä yhteydessä siis epäformaalia katselmusta, tarkastusta tai läpikäyntiä. Katselmoijalta vaaditaan yksityiskohtaisia tietoja käytetyistä fasilitaatio ja mallinnustekniikoista sekä jonkin verran liiketoiminnan ja teknologian tietämystä. Katselmoitavia artefakteja voivat olla mm. ohjelmiston vaatimukset, lähdekoodi, arkkitehtuuri ja suunnitteludokumentit sekä projektisuunnitelma. Katselmoijan aktiviteetteja ja artefakteja on esitetty kuvassa 15.



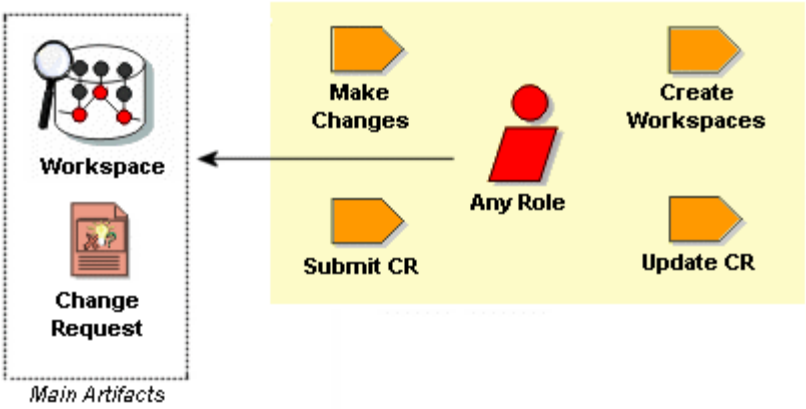
Kuva 15. Katselmoijan aktiviteetit ja artefaktit [1].

3.3.10 Osakas

Osakkaat (Stakeholders) ovat yksilöitä tai organisaatioita, joilla on joko suora tai epäsuora vaikutus systeemille asetettuihin vaatimuksiin liittyviin artefakteihin ja joihin kehityksen tuloksena syntyvä järjestelmä vaikuttaa. Tähän ryhmään kuuluu esimerkiksi järjestelmän loppukäyttäjät, ohjelmiston kehitysprojektin projektipäälliköt, ihmiset jotka ovat tekemisissä sellaisten organisaation prosessien kanssa, joihin kehityksen tuloksena syntyvä järjestelmä vaikuttaa, kehittäjät jotka ovat vastuussa järjestelmän kehittämisestä ja ylläpidosta, organisaatio joka käyttää järjestelmää tarjotakseen asiakkailleen palveluja sekä ulkoiset henkilöt kuten lain sääntelijät (regulators) ja sertifiointeja antavat asiantuntijat. Yleensä osakkailla on toisistaan eriävät näkökulmat järjestelmän ratkaisemaan ongelmaan ja myös erilaiset tarpeet, jotka ratkaisun tulee ottaa huomioon. Toimivan ratkaisun kehittämiseksi on keskeistä tunnistaa keitä nämä osakkaat ovat ja mitä vaatimuksia heillä on. Erilaisia osakkaita ovat esimerkiksi asiakas tai asiakkaan edustaja, käyttäjä tai käyttäjän edustaja, sijoittaja, osakkeenomistaja, tuotantojohtaja, hankkija, suunnittelija, testaaja sekä dokumentoija jne.

3.3.11 Kaikille rooleille yhteistä

Jokainen UPEDU-prosessin tunnistama rooli, jolle on annettu tarvittavat pääsyoikeudet, voi käyttää kokoonpanonhallintajärjestelmää hakeakseen (check-out) sieltä tai lisätäkseen (check-in) sinne tuotteeseen liittyviä artefakteja. Jokaisella roolilla on myös oikeus lisätä omistamiaan muutospyyntöjä ja päivittää niitä. Kaikille rooleille yhteisiä aktiviteetteja ja artefakteja esittää kuva 16.



Kuva 16. Kaikille rooleille yhteiset aktiviteetit ja artefaktit [1].

3.5 Aktiviteeteista

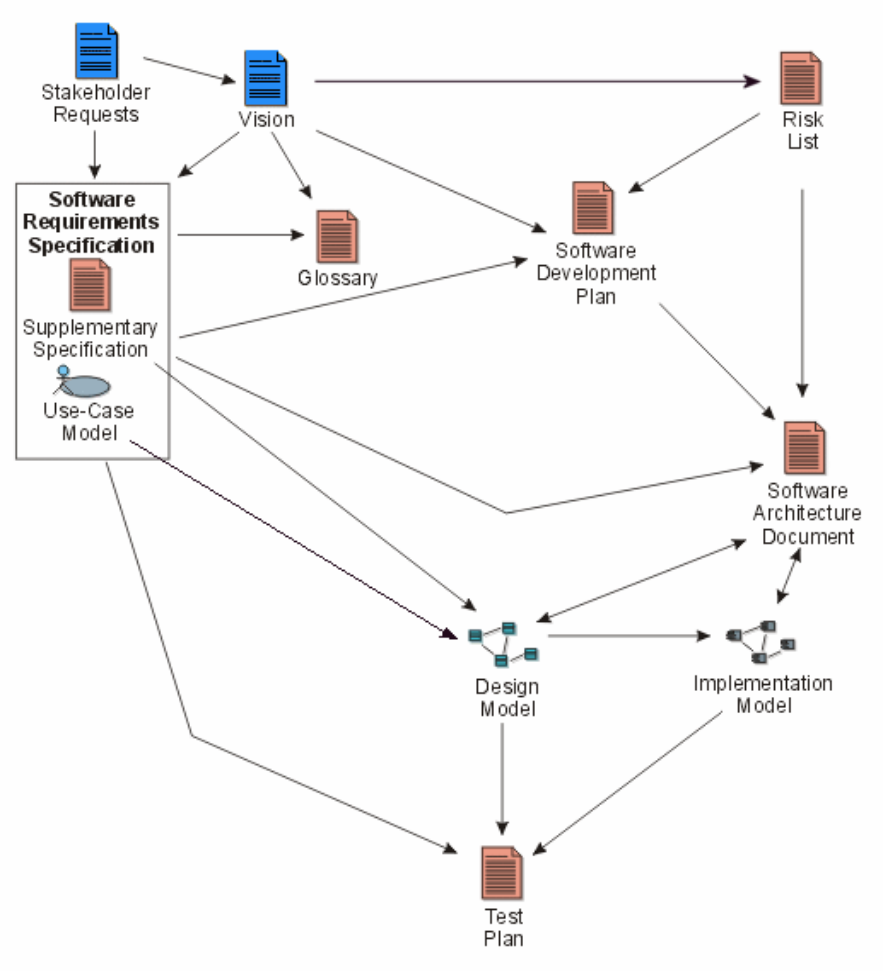
Aktiviteetti tarkoittaa tehtävän luonnetta. Aktiviteetteja ovat esimerkiksi analysointi, koodaus, suunnittelu, testaus ja niin edelleen. Jokaiseen disipliiniin kuulu joukko määriteltyjä aktiviteetteja. Aktiviteetti on työn pienin osa, jota voidaan mitata. Mittaus voi kohdistua esimerkiksi käytettyyn työmäärään. Aktiviteetti voidaan ajatella myös toiminnaksi, jonka suorittaa henkilö, jolla on jokin tietty rooli. Aktiviteetit asetetaan siis jonkin tietyn roolin suoritettavaksi. Aktiviteetilla täytyy olla selvä tarkoitus, kuten esimerkiksi artefaktin luominen tai päivittäminen.

Useimpiin UPEDUn aktiviteetteihin on olemassa ohjeistuksia, joiden avulla saa paremman kuvan aktiviteetista ja sen erilaisista suoritustavoista. Ohjeistuksia löytyy esimerkiksi UPEDU-prosessin prosessityökalusta, joka on osoitteessa http://www.upedu.org/upedu/ (kohta: Overview -> Guidelines). Ohjeita aktiviteetin suorittamiseksi löytyy myös prosessityökalusta avaamalla ensin tietystä disipliinistä kertovan sivun, valitsemalla kohdan aktiviteetit (aktivities) sivun ylälaidasta ja klikkaamalla avautuvan sivun sisältämää kuvaa halutun aktiviteetin kohdalla, jolloin kyseiseen aktiviteettiin liittyvät ohjeet näkyvät avautuvalla sivulla. Yksittäiseen disipliiniin liittyy yleensä useita erilaisia aktiviteetteja ja aktiviteeteilla voi olla enemmän tai vähemmän kiinnitetty suoritusjärjestys. Prosessityökalusta ja tästä dokumentista kannattaakin katsoa työnkulkukaaviot, jotka esittävät aktiviteettien tavallisimman suoritusjärjestyksen. Työnkulkukaaviot löytyvät sekä prosessityökalusta että tästä dokumentista disipliinikohtaisten osuuksien alta.

3.6 Artefakteista

Artefaktit tallentavat ja välittävät projektissa tarvittavaa informaatiota. Ohjelmiston kehitysprosessi määrittelee tarvittavat artefaktit. Artefakteja ovat esimerkiksi visio-dokumentti, vaatimusmäärittely (Software Requirement Specification, SRS), arkkitehtuuridiagrammi, UML-diagrammit, lähdekoodi, testiskriptit, suoritettava koodi, tiedostot ja käyttöopas. Artefaktit ovat aktiviteettien tuotoksia ja niitä käytetään myös aktiviteettien syötteinä. Ne voivat muodostua toisista artefakteista,

esimerkiksi suunnittelumalli (design model) sisältää tavallisesti useita aftefakteja, esimerkiksi luokkia tai alijärjestelmiä. Tärkeimpiä UPEDU-prosessin artefakteja on esitetty kuvassa 17.



Kuva 17. UPEDU-prosessin artefakteja [1].

UPEDU käyttää laajasti hyväkseen UML-kuvauskieltä (Unified Modeling Language), joka soveltuu erityisesti komponentti- ja olioperustaisen ohjelmiston kehitykseen. UPEDUssa malleihin ja mallien elementteihin liittyy raportteja, jotka keräävät työkalu-ohjelmissa olevien tuotoksien informaation. Raportti voi olla artefakti tai joukko artefakteja. Suurimmalle osalle artefakteja on olemassa ohje, joka kuvailee artefaktin tarkemmalla tasolla.

Useimpiin artefakteihin on olemassa mallipohja ja Case-tapaus, joiden avulla saa paremman kuvan artefaktista ja sen käytöstä. Näitä löytyy esimerkiksi UPEDU-prosessin prosessityökalusta osoitteesta http://www.upedu.org/upedu/ (kohta: Artifact Sets -> Templates...). Ohjelmistoprojektityö -kurssilla tullaan käyttämään kuitenkin juuri kyseistä kurssia varten luotuja dokumenttipohjia.

UPEDUn prosessityökalu ei sisällä dokumenttipohjaa Visio-dokumentille. Kyseistä dokumenttia vastaavana tietokokoelmana voidaan kuitenkin pitää yleisemmin tunnettua tuotevisio -dokumenttia (Product Vision). Pienissä projekteissa se riittää perustaksi projektin lopussa tapahtuvaan tuotoksien arviointiin visiosta saatavien evaluointikriteerien perusteella.

Käyttöohjeelle ei myöskään löydy dokumenttipohjaa prosessityökalusta. Käyttökelpoinen standardi kevyehkölle käyttöohjeelle on IEEE Std 1063-2001, jonka mukaan käyttöohjeen tulee sisältää mm. seuraavat osat [4]:

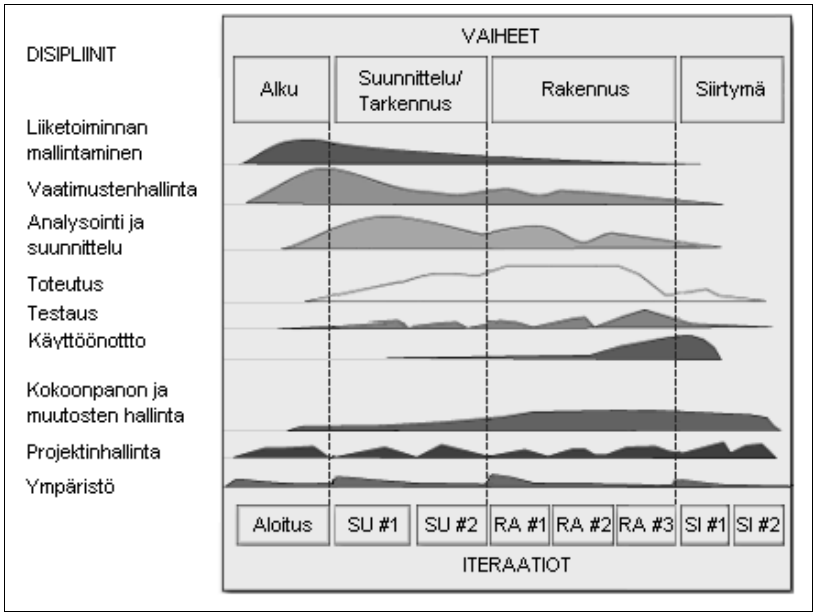
- identifikaatio (etusivu tai vastaava)

  - a) dokumentin nimi
  - b) dokumentin versio ja julkaisupäivämäärä
  - c) ohjelmistotuote ja versio
  - d) kyseessä oleva organisaatio
- sisällysluettelo
- johdanto
- ohjeistus dokumentin käyttöön
- konseptuaalinen tausta ohjelman käytölle
- proseduurit (ohjeistus-esitysmuoto)
- tietoa ohjelmiston komennoista (referenssi-esitysmuoto)
- virheilmoitukset ja ongelmista selviytyminen

Tämän lisäksi täytyy mainita standardiin kuulumaton kohta: tiedossa olevat puutteet ohjelmassa. Puutteet voivat olla mainittuna joko käyttöohjeessa tai julkaisukohtaisissa muistiinpanoissa (release notes).

3.5 Miten UPEDU eroaa RUPista?

UPEDU-prosessi sisältää kaikki samat prosessin vaiheet kuin RUP-prosessikin, mutta joitakin disipliinejä on jätetty pois. UPEDU ei sisällä RUP-prosessin kehitysdisipliineihin kuuluvia liiketoiminnan mallintamis- ja käyttöönottodisipliiniä ja hallintadisipliineissä ei ole erillistä ympäristödisipliiniä. Syntyvien artefaktien määrä on huomattavasti pienempi UPEDU-prosessissa. UPEDU sisältää vain 37 prosenttia alkuperäisen mallin artefakteista ja vain 10 prosenttia ohjeista (guidelines) [3]. RUP-prosessi on esitetty kuvassa 5.



Kuva 5. RUP-prosessi (vrt. Kuva 2).

4. Prosessin vaiheet

Tässä luvussa kerrotaan UPEDU-prosessin neljästä vaiheesta, jotka ovat alkuvaihe (konsepti), suunnittelu-/tarkennusvaihe (suunnittelu), rakennusvaihe (toteutus) sekä siirtymävaihe (tuotteen toimitus). Jokaisesta vaiheesta on kuvattu vaiheen tavoitteet, aktiviteetit, virstanpylväs sekä virstanpylvääseen liittyen vaiheen evaluointikriteerit ja artefaktien tila vaiheen lopussa. Tämän kohdan sisältämät luettelot ja listat on tuotu tähän tuotokseen suoraan prosessityökalusta ([1]), tosin hieman lyhennettyinä.

4.1 Alkuvaihe

Alkuvaiheen (Inception) ensisijainen päämäärä on saavuttaa yhteisymmärrys eri sidosryhmien kanssa projektin elinkaaren tavoitteista. Alkuvaiheella on merkitystä vain kun aloitetaan kehittämään jotain uutta, johon liittyy liiketoimintaa ja vaatimuksia koskevia riskejä. Näihin riskeihin on otettava kantaa ennen projektin jatkamista eteenpäin. Tarkoitus on siis varmistaa, että projekti on sekä kannattava että kaikin puolin mahdollinen. Vaiheen aktiviteetteihin kuuluu mm. prosessin käyttöönotto, tiimin rakentaminen, uuden sovellusalan ymmärtäminen, uusiin työkaluihin tutustuminen. Alkuvaiheessa vaatimustenhallinnan ja projektinhallinnan työmäärät ovat tavallisesti suurimpia verrattuna muihin disipliineihin.

4.1.1 Vaiheen tavoitteet

Alkuvaiheen tavoitteisiin kuuluu [1]:

- määritellä ja vakiinnuttaa ohjelmiston laajuus ja rajaus, rajoittavat tekijät, operationaalinen visio (johdetaan määritellystä ongelmasta), hyväksymiskriteerit, mitä tuotteeseen kuuluu ja mitä ei
- erotella kriittiset käyttötapaukset, primääriset skenaariot jotka vaikuttavat eniten systeemin suunnitelmaan
- vähintään yhden kandidaattiarkkitehtuurin kokeilu ja mahdollisesti demonstrointi (proto)
- potentiaalisten riskien arviointi
- kokonaiskustannusten ja aikataulun arviointi (tarkemmat estimaatit elaboraatio-vaiheelle)

4.1.2 Vaiheen aktiviteetit

Alkuvaiheen aktiviteetteihin kuuluu [1]:

- projektin laajuuden määrittäminen
    - konteksti, tärkeimmät vaatimukset ja rajoitteet niin, että projektin hyväksymiskriteerit voidaan muodostaa
- business casen suunnittelu ja valmisteleminen
    - vaihtoehtojen arviointi liittyen riskien hallintaan, henkilöstön hankintaan

    - projektin suunnitelma, kustannukset/aikataulu/kannattavuus kompromissit

- kandidaattiarkkitehtuurin syntetisointi

    - valmista/osta/uudelleenkäytä (make/buy/reuse) kompromissien arviointi ja systeemin suunnitelmaan liittyvien kompromissien arviointi, jotta kustannukset, aikataulu ja resurssit voidaan arvioida

    - konseptin todistus esim. mallilla (joka simuloi vaatimuksia) tai prototyypillä (jolla voidaan tutkia suuria riskitekijöitä, lähinnä teknologiaan liittyviä)

- projektin ympäristön valmisteleminen

    - projektin organisaatio, työkalut, prosessin parannus

4.1.3 Virstanpylväs

Alkuvaiheen päättävää virstanpylvästä kutsutaan elinkaaren tavoitteet -virstanpylvääksi (Lifecycle Objectives Milestone). Virstanpylväs toimii merkkinä projektin evaluoinnille, jossa tulee päättää mm. onko projektin jatkaminen kannattavaa ja vaiheen tavoitteet saavutettu.

4.1.3.1 Evaluointikriteerit

Elinkaaren tavoitteet virstanpylvään evaluointikriteereihin kuuluu [1]:

- yksimielisyys laajuuden määrityksestä ja kustannukset/aikataulu-estimaateista

- yksimielisyys siitä, että juuri oikea joukko vaatimuksia on kerätty ja että kaikki osapuolet ymmärtävät vaatimukset samalla tavalla

- yksimielisyys kustannus/aikataulu –estimaateista, prioriteeteista, riskeistä, kehitysprosessista

- kaikki riskit on identifioitu ja jokaiselle niille on olemassa mitigaatiostrategia (mitä projektissa tehdään riskin vaikutusten pienentämiseksi)

4.1.3.2 Artefaktien tila vaiheen lopussa

Alla oleva taulukko 1 kuvaa keskeisten artefaktien tilan kun elinkaaren tavoitteet -virstanpylväs on saavutettu. Lista on tärkeysjärjestyksessä.

Visio (Vision)	Projektin ydinvaatimukset, avain- ominaisuudet, keskeiset rajoitteet on dokumentoitu.
Riskilista (Risk List)	Alustavat projektin riskit identifioitu.
Iteraatiosuunnitelma (Iteration Plan)	Iteraatiosuunnitelma ensimmäiselle elaboraatio-vaiheen iteraatiolle valmis ja katselmoitu.
Sanasto (Glossary)	Tärkeät termit määritelty, sanasto katselmoitu.
Käyttötapausmalli (Use-Case Model)	Tärkeät käyttötapaukset ja toimijat identifioitu, tapahtumienkulut hahmoteltu kriittisille käyttö-tapauksille.
Projektivarasto (Project repository), Muutospyyntö (Change Request)	Konfiguraationhallinnan ympäristö pystytetty.

Taulukko 1. Keskeisten artefaktien tilat elinkaaren tavoitteet -virstanpylväässä.

Muita artefakteja, joita voi aloittaa tekemään jo alkuvaiheessa, ovat vaatimusmäärittely, täydentävä määrittely (Supplementary Specification), käyttöliittymän prototyyppi ja ohjelmiston kehityssuunnitelma sekä artefaktit, joista edellä mainitut artefaktit koostuvat.

4.2 Suunnittelu-/Tarkennusvaihe

Suunnittelu-/Tarkennusvaiheen (Elaboration), josta voidaan käyttää myös nimitystä elaboraatiovaihe, päämääränä on systeemin arkkitehtuurin perustason luominen, joka tarjoaa vakaan perustan rakennusvaiheen suunnittelu- ja implementaatiotyölle. Arkkitehtuuri kehitetään arkkitehtuurisesti merkittävimpien käyttötapausten perusteella ottaen myös riskit huomioon. Arkkitehtuurin stabiliteetti voidaan arvioida yhden tai useamman arkkitehtuurisen prototyypin avulla. Tämän vaiheen toimitettaviin artefakteihin kuuluu malli systeemin käyttäytymisestä, mukaan lukien systeemin konteksti, skenaariot, sekä sovellusalueen malli (domain model). Toimitettaviin artefakteihin kuuluvat myös perustason (baseline) tuotevisio (product vision), joka perustuu sovellusmalliin (domain model), kehityssuunnitelma, evaluointikriteerit ja tuotteen julkaisuversion kuvaus (product release description).

4.2.1 Vaiheen tavoitteet

Elaboraatiovaiheen tavoitteisiin kuuluu [1]:

- varmistaa että arkkitehtuuri, vaatimukset ja suunnitelmat ovat tarpeeksi vakaita, ja että riskien vaikutuksia on riittävästi pienennetty, jotta projektin kustannukset ja aikataulu voidaan ennustaa riittävän tarkasti.
- osoittaa ja ottaa käsittelyyn arkkitehtuurisesti merkittävät riskit
- perustaa perustasoinen (baselined) arkkitehtuuri, joka johdetaan arkkitehtuurisesti merkittävien skenaarioiden avulla, jotka tuovat yleensä projektin tekniset riskit esille.
- kehittää evolutionäärinen prototyyppi sekä yksi tai useampi pois heitettävä prototyyppi, joiden avulla voidaan pienentää tiettyjä riskejä kuten esimerkiksi:
  - suunnitteluun/vaatimuksiin liittyvät kompromissit
  - komponenttien uudelleenkäyttö
  - tuotteen toteutettavuus ja soveltuvuus, tai demonstraatiot investoijille, asiakkaille ja loppukäyttäjille
- demonstroida se, että perustason (baselined) arkkitehtuuri tukee systeemin vaatimuksia kohtuulliseen hintaan ja kohtuullisessa ajassa
- perustaa projektia tukeva ympäristö (mm. mallipohjat, ohjeet, työkaluohjelmat jne.)

4.2.2 Vaiheen aktiviteetit

Elaboraatiovaiheen aktiviteetteihin kuuluu [1]:

- määritellä, validoida ja perustasoittaa (baselining) arkkitehtuuri niin nopeasti kuin on vain käytännöllistä
- hioa visiota
- luoda ja perustasoittaa (baselining) yksityiskohtaiset iteraatiosuunnitelmat rakennusvaiheelle
- hioa kehitystapausta (development case) ja laittaa paikalle kehitysympäristö (mm. prosessi, työkaluohjelmat, automatisoinnin tuki, päämääränä tukea rakennustiimiä)
- hioa arkkitehtuuria ja valita komponentit
- potentiaaliset komponentit arvioidaan ja tehdään valmista/osta/uudelleenkäytä (make/buy/reuse) päätökset ymmärretään sillä tasolla, että rakennusvaiheen kustannukset ja aikataulu voidaan määrittää
- valitut arkkitehtuuriset komponentit integroidaan ja arvioidaan suhteessa primäärisiin skenaarioihin. Tarvittaessa suunnitellaan arkkitehtuuri uudelleen, ottaen huomioon vaihtoehtoiset mallit tai vaatimuksien uudelleenharkinnan.

4.2.3 Virstanpylväs

Elaboraatiovaiheen virstanpylväs on nimeltään arkkitehtuuri-virstanpylväs (lifecycle architecture milestone). Tämä virstanpylväs perustaa hallitun perustason systeemin arkkitehtuurille ja mahdollistaa projektin tiimin skaalautumisen rakennusvaiheen aikana.

4.2.3.1 Evaluointikriteerit

Arkitehtuuri-virstanpylvään evaluointikriteereihin kuuluu [1]:

- tuotevisio (product vision) ja vaatimukset ovat vakaita
- arkkitehtuuri on vakaa
- avainlähestymistavat, joita käytetään testaamisessa ja arvioimisessa ovat taattuja
- ajettavien/suoritettavien prototyyppien testaus ja arviointi ovat osoittaneet, että suurimpiin riskielementteihin on otettu kantaa ja ne on uskottavasti ratkaistu
- rakennusvaiheen iteraatiosuunnitelmat ovat tarpeeksi yksityiskohtaisia ja täsmällisiä työn jatkamiseksi
- rakennusvaiheen iteraatiosuunnitelmia tukee uskottavat estimaatit
- kaikki asianosaiset ovat yhtä mieltä siitä, että visio voidaan saavuttaa, jos nykyiset suunnitelmat toteutetaan koko systeemin kehittämiseksi, nykyisen arkkitehtuurin kontekstissa
- todellinen resurssien käyttö/kustannukset versus suunniteltu resurssien käyttö/kustannukset on hyväksyttävissä

4.2.3.2 Artefaktien tila vaiheen lopussa

Alla oleva taulukko 2 kuvaa keskeisten artefaktien tilan kun arkkitehtuuri-virstanpylväs on saavutettu.

Riskilista (Risk List)	Päivitetty ja katselmoitu. Uudet riskit ovat todennäköisesti arkkitehtuuriin liittyviä, lähinnä ei-funktionaalisten vaatimusten käsittelyyn.
Ohjelmiston arkkitehtuuri (Software Architecture Document)	Luotu ja perustasot määritetty (baselined), sisältäen arkkitehtuurisesti merkittävät käyttötapaukset (käyttötapaus perspektiivi, use-case view) ja avainmekanismien sekä mallielementtien identifikaation (looginen perspektiivi, logical view)
Suunnittelumalli (Design Model)	Määritelty ja perustasot määritetty, Arkkitehtuurisesti merkittävien käyttötapausten realisaatiot määritelty ja vaadittu käyttäytyminen on allokoitu sopiville suunnitteluelementeille (design elements). Komponentit on identifioitu. Arkkitehtuuriset komponentit on integroitu ja arvioitu primäärisiä skenaarioita vastaan.
Toteutusmalli (Implementation Model), sekä siihen liittyvät artefaktit, mukaan lukien komponentit.	Alustava rakenne luotu, suurimmat komponentit identifioitu ja protoiltu.
Visio (Vision)	Hiottu perustuen informaation jota on saatu vaiheen aikana, muodostaen vankan ymmärryksen kriittisimmistä käyttötapauksista, jotka ohjaavat arkkitehtuuriin ja suunnitteluun liittyviä päätöksiä.
Iteraatiosuunnitelma (Iteration Plan)	Rakennusvaiheen suunnitelma valmis ja katselmoitu.
Käyttötapausmalli (Use-Case Model)	Käyttötapausmalli noin 80%:sesti valmis, kaikki käyttötapaukset on identifioitu käyttötapausmallin selvityksessä (survey), kaikki toimijat on identifioitu ja suurin osa käyttötapauskuvauksista (use-case descriptions) on kehitetty.
Täydentävät määritykset (Supplementary Specification)	Täydentävät vaatimukset, jotka määrittävät ei-funktionaaliset vaatimukset, on dokumentoitu ja katselmoitu.

Taulukko 2. Keskeisten artefaktien tilat arkkitehtuuri -virstanpylväässä.

4.3 Rakennusvaihe

Rakennusvaihe (Construction) sisältää ohjelmistotuotteen ohjelmoimisen. Vaiheen tuloksena syntyy suoritettava koodi, systeemi- ja käyttäjädokumentaatio, käyttöönottosuunnitelma sekä joitain laadunvarmistustuloksia. Rakennusvaiheen jokaisen iteraation suunnitellaan toteutettavaksi tietyt komponentit, joiden valinta riippuu käytetystä integrointitavasta ja testausstrategista.

4.3.1 Vaiheen Tavoitteet

Rakennusvaiheen tavoitteisiin kuuluu [1]:

- kehityskustannusten minimointi optimoimalla resurssien käyttöä ja välttämällä tarpeettomia parannuksia ja muokkauksia. Älä korjaa sitä mikä toimii.
- saavuttaa riittävä laatu ja versiot (alpha, beta) niin nopeasti kuin vain käytännöllistä
- saattaa päätökseen kaikkien vaadittujen vaatimusten analysointi, suunnittelu, kehitys ja testaus
- kehittää iteratiivisesti ja inkrementaalisesti kokonainen tuote, joka on valmis siirrettäväksi käyttäjille
- päättää, onko ohjelmisto ja käyttäjät valmiita sovelluksen käyttöönottoon
- saavuttaa jonkin asteista rinnakkaisuutta kehitystiimien toiminnassa

4.3.2 Vaiheen Aktiviteetit

Rakennusvaiheen aktiviteetteihin kuuluu [1]:

- resurssienhallinta, kontrollointi ja prosessin optimisointi
- komponenttien kehittämisen valmiiksi saaminen ja testaus määriteltyjä evaluointikriteerejä vastaan
- tuotteen julkaisujen (release) arviointi vision hyväksymiskriteerejä vastaan

4.3.3 Virstanpylväs

Rakennusvaiheen virstanpylväänä on toimiva ohjelma. Vaiheen virstanpylvään nimeksi on annettu alustavan operationaalisen kyvykkyyden virstanpylväs (initial operational capability milestone). Tämä virstanpylväs määrittää, onko tuote valmis otettavaksi käyttöön betatestausympäristössä.

4.3.3.1 Evaluointikriteerit

Alustavan operationaalisen kyvykkyyden virstanpylvään evaluointikriteereihin kuuluu [1]:

- tuotteen julkaisuversio on vakaa ja tarpeeksi kypsä otettavaksi käyttöön käyttäjäyhteisössä
- kaikki osakkaat/sidosryhmät ovat valmiina tuotteen siirtämiseen käyttäjäyhteisöön
- todelliset resurssikustannukset versus suunnitellut resurssikustannukset ovat hyväksyttäviä

4.3.3.2 Artefaktien tila vaiheen lopussa

Alla oleva taulukko 3 kuvaa keskeisten artefaktien tilan kun alustavan operationaalisen kyvykkyyden virstanpylväs on saavutettu.

Systeemi	Suoritettava systeemi valmis beta-testaukseen.
Implementaatiomalli	Kaikki komponentit luotuja rakennusvaiheen lopussa.
Iteraatiosuunnitelma	Suunnitelma siirtymävaiheelle valmis ja katselmoitu.
Suunnittelumalli	Päivitetty uusilla elementeillä, jotka on identifioitu vaatimusten selvityksen täyttyessä.
Valinnaiset artefaktit	
Täydentävät määritykset	Päivitetty rakennusvaiheen aikana esiin tulleilla uusilla vaatumuksilla.
Käyttötapausmalli	Päivitetty rakennusvaiheen aikana esiin tulleilla uusilla käyttötapauksilla.

Taulukko 3. Keskeisten artefaktien tilat operationaalisen kyvykkyyden virstanpylväässä.

Muita artefakteja, joita tehdään rakennusvaiheessa, ovat käyttöohje ja testaukseen liittyvät dokumentit. Käyttötapauksien realisaatiomäärittely (Use-Case Realization) tulee olla myös valmiina rakennusvaiheen lopussa.

4.4 Siirtymävaihe

Ohjelmiston lopullinen versio luovutetaan käyttäjälle siirtymävaiheessa (transition). Tässä vaiheessa voidaan tehdä myös pientä loppuviimeistelyä esimerkiksi ohjelman mukana tuleviin tukimateriaaleihin tai itse ohjelmaan ja sen asetuksiin sekä testata tuotteen toimivuus asiakkaan ympäristössä tai sitä vastaavassa ympäristössä. Vaiheen tarvittaviin artefakteihin kuuluvat kaikki projektin valmiit artefaktit.

4.4.1 Vaiheen tavoitteet

Siirtymävaiheen tavoitteisiin kuuluu [1]:

- betatestaus systeemin validoimiseksi käyttäjän odotuksia vasten
- systeemin käyttö rinnakkain vanhan järjestelmän kanssa
- operationaalisten tietokantojen konvertointi
- käyttäjien ja ylläpitäjien koulutus
- markkinointi-, levitys- ja myyntihenkilöstön tuominen kuvaan ja muut tuotteistukseen liittyvät aktiviteetit
- yhteisymmärryksen saavuttaminen sidosryhmien kanssa siitä, että käyttöön otettavat perustasot (baselines) ovat valmiita
- viimeiset hienosäädöt kuten bugien korjaus, käytettävyyden ja tehokkuuden parannukset

- käyttöön tulevien perustasojen arviointi tuotteen viimeistä visiota ja hyväksymiskriteereitä vastaan

4.4.2 Vaiheen aktiviteetit

Siirtymävaiheen aktiviteetteihin kuuluu [1]:

- käyttöönottosuunnitelmien suorittaminen
- loppukäyttäjien tukimateriaalin viimeistely
- toimitettavan tuotteen testaus tuotantoympäristössä
- tuotteen julkaisun luominen
- palautteen saaminen käyttäjiltä ja ja sen perusteella tehtävä tuotteen hienosäätö
- tuotteen laittaminen loppukäyttäjien saataville

4.4.3 Virstanpylväs

Siirtymävaiheen virstanpylväänä toimii tuotteen julkaisu virstanpylväs (product release milestone). Tämän virstanpylvään saavutettua arvioidaan onko kaikki tavoitteet saavutettu ja onko tarvetta aloittaa uusi kehityssykli tai esimerkiksi julkistuksen jälkeinen ylläpitosykli. Rinnakkaisten kehityssyklien tapauksessa tuotteen julkaisu virstanpylväs voi sijaita rinnakkain esimerkiksi toisen kehityssyklin alkuvaiheen loppuosan kanssa. Tuotteen julkaisu virstanpylväs saavutetaan kun tuotteen hyväksymistestaus ja projektin hyväksymiskatselmus (Project Acceptance Review) on suoritettu onnistuneesti.

4.4.3.1 Evaluointikriteerit

Tuotteen julkaisu virstanpylvään evaluointikriteereihin kuuluu [1]:

- käyttäjän tulee olla tyytyväinen
- todelliset resurssikustannukset versus suunnitellut resurssikustannukset ovat hyväksyttäviä

4.4.3.2 Artefaktien tila vaiheen lopussa

Alla oleva taulukko 4 kuvaa keskeisten artefaktien tilan kun tuotteen julkaisu virstanpylväs on saavutettu.

Tuotteen käännös (The Product Build)	Vastaa tuotteen vaatimuksia ja on valmis asiakkaan käytettäväksi.

Taulukko 4. Keskeisten artefaktien tilat tuotteen julkaisu virstanpylväässä.

Käytännössä kaikki tuotteeseen liittyvät artefaktit tulee olla valmiit siirtymävaiheen päättyessä, jolloin koko projekti päätetään tai sitten projektia jatketaan uudella kehityssyklillä tai ylläpitosyklillä.

5. Disipliinit

Tässä luvussa esitellään UPEDU-prosessimalliin kuuluvat disipliinit, joita ovat vaatimusten hallinta-, analyysi ja suunnittelu-, implementointi-, testaus-, konfiguraation ja asetustenhallinta- sekä projektinhallinta -disipliini. Jokaisesta disipliinistä kuvataan disipliinin tarkoitus, roolit ja aktiviteetit, artefaktit, työnkulku sekä suhde muihin disipliineihin.

5.1 Vaatimustenhallinta

Vaatimustenhallinta-disipliini koostuu joukosta aktiviteetteja, joiden tarkoituksena on johtaa mm. Visio-dokumenttia apuna käyttäen järjestelmän vaatimukset sekä validoida ja ylläpitää näitä vaatimuksia. Vaatimuksia voidaan pitää sopimuksen perustana asiakkaan ja kehitysorganisaation välillä. Vaatimustenhallintaprosessia kehitettäessä tulee ottaa huomioon kehitettävän tuotteen tyyppi sekä organisaation kulttuuri ja kokemuksen taso. Vaatimustenhallinta-disipliinin tarkoitus on mm. varmistaa perusta ohjelmistoprosessiin kuuluvien tahojen väliselle kommunikaatiolle. Vaatimustenhallinnan aktiviteetteja on kuvattu mm. SEIn (Software Engineering Institute) kehittämässä ohjelmistoprosessin kypsyysmallissa (Software Capability Maturity Model, SW-CMM).

5.1.1 Tarkoitus

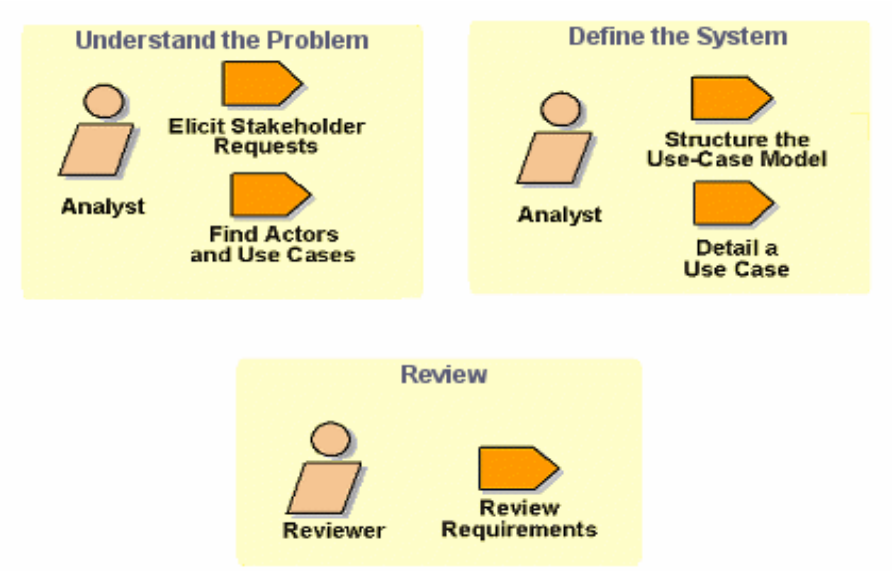
Vaatimustenhallinta-disipliinin tarkoitus on [1]:

- Saavuttaa ja ylläpitää asiakkaiden ja muiden sidosryhmien kanssa yhteisymmärrys siitä, mitä systeemin tulisi tehdä
- Tarjota järjestelmän kehittäjille parempi ymmärrys systeemin vaatimuksista
- Määritellä systeemin rajat
- Tarjota pohja iteraatioiden teknisen sisällön suunnitteluun
- Tarjota pohja systeemiin liittyvien kustannusten ja ajankäytön arviointiin
- Määritellä systeemille käyttöliittymä, keskittyen käyttäjien tarpeisiin ja päämääriin

Vaatimustenhallinta -disipliiniin kuuluu mm. seuraavien asioiden määrittäminen: dokumentointiformaatti, mallinnuskielen formaliteetti, vaatimusten relatiivinen prioriteetti, tarvittavan työn määrä per vaatimus, tekniset ja johtamiseen liittyvät riskit sekä alustava arvio ohjelmiston kehitysprojektin laajuudesta. Suurissa organisaatioissa voidaan tarvita vielä mm. konseptin tutkimusta, toteutettavuustutkimusta, tarpeiden selostusta, vision kehittämistä jne.

5.1.2 Roolit ja aktiviteetit

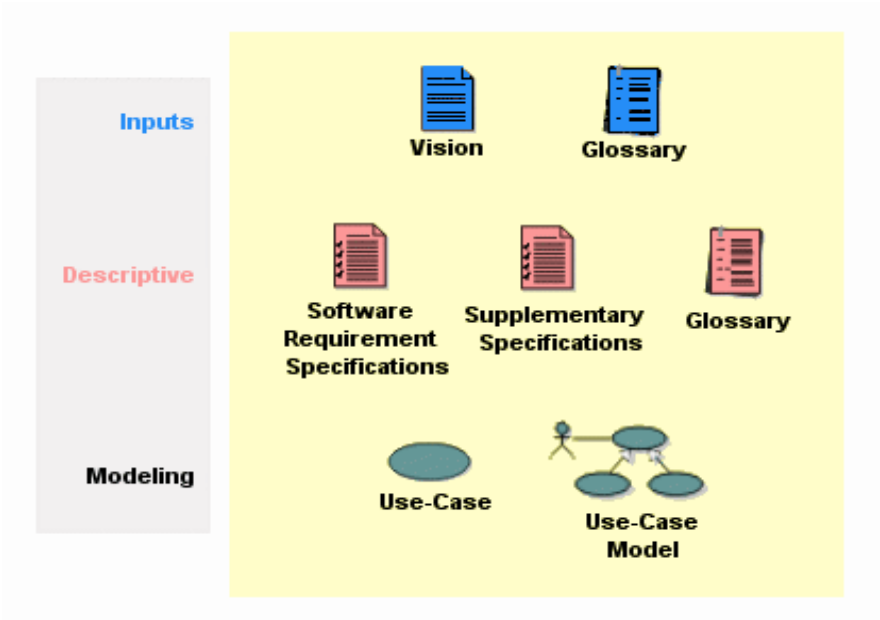
Vaatimustenhallinta-disipliini koostuu viidestä aktiviteetista, joista ensimmäiset neljä suorittaa analysoija ja viimeisen katselmoija. Analysoijan tehtäviin kuuluu osakkaan vaatimuksen selvittäminen, toimijoiden ja käyttötapausten etsiminen, käyttötapausten strukturointi ja käyttötapausten tarkentaminen. Katselmoija taas katselmoi vaatimukset. Vaatimustenhallinta -disipliinin roolit ja aktiviteetit on esitetty kuvassa 18.



Kuva 18. Vaatimustenhallinta -disipliiniin kuuluvat roolit ja aktiviteetit [1].

5.1.3 Artefaktit

Vaatimustenhallinta-disipliinin artefaktit jakautuvat kolmeen ryhmään: syöte-, deskriptiiviset ja mallinnus-artefaktit. Syötteenä toimiviin artefakteihin kuuluu visio-dokumentti (Vision) ja sanasto (Glossary). Visio-dokumentti on itse asiassa koko UPEDU-prosessin syöteartefakti. Sanasto on osaksi myös deskriptiivinen, koska tämä artefakti valmistuu järjestelmän spesifikaation edistyessä. Deskriptiivisiin artefakteihin sisältyy ohjelmiston vaatimusmäärittely, täydentävä spesifikaatio sekä sanasto. Mallinnusartefakteihin sisältyvät sekä käyttötapaukset (Use case) että käyttötapausmalli (Use Case Model). Vaatimustenhallinta-disipliinin artefaktit on esitetty kuvassa 19.



Kuva 19. Vaatimustenhallinta-disipliinin artefaktit [1].

Visio-dokumentti on tarkoitettu antamaan korkean tason kuvaus osakkaille kehitettävästä tuotteesta ja sen laatii tavallisesti ei-tekniset ihmiset. Vaatimusmäärittelyn tavoin myös visio-dokumenttia voidaan käyttää sopimuksena projektin aloittamiselle. Visio-dokumentti sisältää suurimman osan

funktionaalisista vaatimuksista narratiivisessa muodossa. Visio-dokumentti ei kuitenkaan ole osa UPEDU-prosessia, koska se oletetaan olevan saatavilla ja sitä käytetään ainoastaan syöte-artefaktina. Dokumentti keskittyy kuvaamaan käyttäjien tarpeita, päämääriä ja tavoitteita, kohdemarkkinoita, käyttäjäympäristöä ja alustaa sekä tuotteen ominaisuuksia. Se on tärkeä syöte projektin hyväksymisprosessissa ja vastaa kysymyksiin mitä ja miksi.

Sanasto sisältää tärkeät termit, joita käytetään tietyn ongelma-alueen terminologian määrittelyssä. Sanasto-artefakteja on vain yksi kappale koko järjestelmälle.

Järjestelmän vaatimuksia tulee käsitellä pakettina, jonka osia voi löytyä eri artefakteista ja ohjelmista, joita käytetään apuna informaation luomisessa ja kokoamisessa. *Vaatimusmäärittely* (Software Requirement Specification, SRS) sisältää täydelliset vaatimukset koko järjestelmälle tai sen osalle. Ei-funktionaaliset vaatimukset tallennetaan täydentävät määritykset -dokumenttiin (Supplementary Specification).

Vaatimusmäärittely noudattaa IEEE 830-1993 standardia, jonka mukaan hyvän vaatimusmäärittelyn tulee sisältää mm. seuraavat osiot [5]:

- vaatimusmäärittelyn tarkoitus ja laajuus
- tekijät, jotka vaikuttavat tavoiteltuun järjestelmään ja sen vaatimuksiin
- ohjelmiston vaatimukset

Vaatimusmäärittelyn sisältö voi vaihdella riippuen käytetystä tavasta, jolla vaatimukset kehitetään. Hyvin rakennetun vaatimusmäärittelyn tunnusmerkkejä voidaan käyttää apuna dokumentin rakennetta suunniteltaessa ja osana ohjeistusta dokumentin kirjoittajille sekä lisäksi vielä arviointiperusteina katselmuksissa.

Täydentävä spesifikaatio sisältää järjestelmän ei-funktionaaliset vaatimukset. Ne asettavat ulkoisia rajoitteita siihen, miten systeemi voidaan kehittää. Spesifikaatio voi sisältää myös rajoitteita, jotka liittyvät systeemin ylläpitoon. Ei-funktionaalisia vaatimuksia voivat olla esimerkiksi, käytettävyys, luotettavuus, tehokkuus, tuettavuus ja suunnittelun rajoitteet.

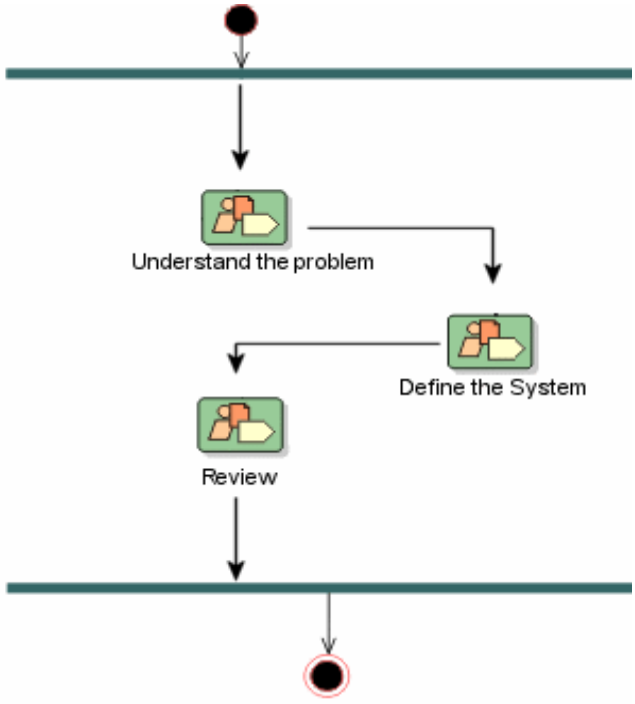
Vaatimusten kehitykseen kuuluu lisäksi systeemin rajojen määritys. Vaatimuksien selvittämiseen on olemassa useita metodeja, joita ovat esimerkiksi [1]:

- henkilökohtainen haastattelu
- työpajan pitäminen, jossa voidaan käyttää esimerkiksi seuraavia metodeja:
  - aivoriihi
  - kuvakäsikirjoitus
  - roolipeli
  - olemassa olevien vaatimusten katselmus
  - selvittämättömien asioiden lista, jossa jokaiselle merkinnälle annetaan vastuuhenkilö sen selvittämisestä
- prototyyppien käyttö, joita ovat esimerkiksi:
  - pois heitettävä proto
  - evolutionäärinen proto
  - operationaalinen proto

5.1.4 Työnkulku

Jokainen työnkulun kohta edustaa avaintaitoa, jota pitää soveltaa tehokkaan vaatimustenhallinnan suorittamiseksi. Ongelman analysointi ja ymmärtäminen kuuluu kehityssyklin alkuvaiheeseen (inception phase), kun taas systeemin määrittely ja määrityksen hiominen kuuluu suunnitteli-/tarkennusvaiheeseen (elaboration phase). Järjestelmän laajuutta ja muuttuvia vaatimuksia hallitaan koko projektin ajan. Kuvassa 20 oleva työnkulku on esimerkki

todennäköisestä tehtävien suoritusjärjestyksestä ensimmäisen iteraation aikana, mutta tehtävien suoritusjärjestys voi olla vaihteleva varsinkin palattaessa esitettyihin tehtäviin uudelleen.



Kuva 20. Vaatimustenhallinta-disipliinin työnkulku [1].

5.1.5 Suhde muihin disipliineihin

Analysointi ja suunnittelu-disipliini saa primäärisen syötteensä vaatimuksista. Se käyttää syötteenään vaatimustenhallinta-disipliinin aktiviteettien avulla tuotettua käyttötapausmallia (use case model) ja sanastoa. Analysointi ja suunnittelu-vaiheessa voidaan löytää virheitä käyttötapausmallista, jolloin virheestä tehdään muutospyyntö ja lopuksi muutos toteutetaan käyttötapausmalliin. Testaus-disipliini validoi systeemin käyttäen apunaan esimerkiksi käyttötapausmallia. Käyttötapaukset ja täydentävä spesifikaatio toimivat syötteenä vaatimuksille, joiden avulla testaus-disipliiniin kuuluva arviointitehtävä määritellään, sekä ne toimivat syötteinä myös peräkkäisille testeille ja arviointiaktiviteeteille. Konfiguraation ja muutostenhallinta-disipliini tarjoaa muutostenhallintamekanismin vaatimuksille. Muutoksen ehdotus tapahtuu jättämällä muutospyyntö, jonka arvioi muutostenhallintaneuvosto. Projektinhallinta-disipliinin aktiviteettien avulla suunnitellaan projekti ja siihen kuuluvat iteraatiot. Käyttötapausmalli ja ohjelmiston kehityssuunnitelma ovat tärkeitä syötteitä iteraatioiden suunnitteluaktiviteeteille.

5.2 Analysointi ja suunnittelu

Analysoinnin tarkoituksena parantaa ymmärrystä asiakkaalle kehitettävästä ratkaisusta. Se on kognitiivinen prosessi, jossa informaatiota lisätään olemassa olevaan tietoon. Analogiana tälle prosessille on kristallisaatioprosessi, jossa informaatioelementtejä määritellään progressiivisesti ja sidotaan yhteen, aivan kuten molekyylejä kristalliin. Suunnittelu on myös kognitiivinen prosessi ja sen tarkoituksena on kehittää informaatiorakenne. Informaatiorakenne syntyy perustamalla linkkejä olemassa olevien informaatio-osasten välille. Suunnitteluaktiviteetit voidaan aloittaa vasta kun analysointi on tuottanut sopivan konsentraation informaatiota. Analysointi ja suunnittelu voidaan kuitenkin yhdistää monellakin eri tavalla. Esimerkiksi analysointi voi olla olennainen osa

suunnitteluaktiviteetteja kun molemmat aktiviteetit on yhdistetty toisiinsa, mutta ne voidaan suorittaa myös erillisinä aktiviteetteina.

Analysointi ja suunnittelu -disipliinin tarkoituksena on analysoida ja suunnitella systeemi, joka toimii suunnitellussa implementaatio-ympäristössä ja suorittaa ne tehtävät ja toiminnot, jotka on määritelty käyttötapauskuvauksissa. Tuloksena olevan systeemin on täytettävä kaikki sille asetetut vaatimukset ja systeemin rakenne on oltava tarpeeksi joustava, jotta se tukee funktionaalisten vaatimusten ja ympäristölle asetettujen rajoitusten muuttumista ja päivitystä. UPEDU-prosessi näkee analysointi ja suunnittelu -disipliinin keskeiseksi laadun määrääjäksi, koska kyseinen disipliini on vastuussa tuotteen luomisesta [2].

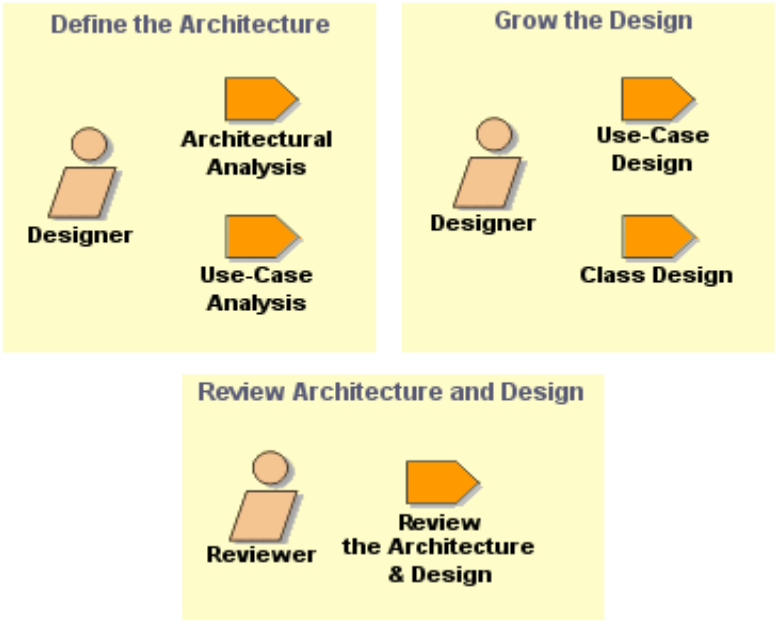
5.2.1 Tarkoitus

Analysointi ja suunnittelu-disipliinin tarkoitus on [1]:

- muuttaa vaatimukset suunnitelmaksi kehitettävästä järjestelmästä
- kehittää vakaa ja joustava arkkitehtuuri järjestelmälle
- mukauttaa järjestelmän suunnitelma vastaamaan implementaatioympäristöä ja suunnitella se tehokkuus mielessä pitäen

5.2.2 Roolit ja aktiviteetit

Analysointi ja suunnittelu-disipliinissä suunnittelija määrittelee järjestelmän arkkitehtuurin arkkitehtuurin analysointi-aktiviteetin avulla tehden samalla myös käyttötapausten analysointia. Käyttötapauksia analysoidessa saattaa samalla joutua päivittämään arkkitehtuuria, jotta se vastaisi systeemin muuttunutta käyttäytymistä. Suunnittelija kasvattaa systeemin suunnitelmaa (design) lisäämällä ja tarkentamalla käyttötapauksia ja luokkia. Katselmoijan tehtäviin kuuluu arkkitehtuurin ja systeemin suunnitelman tarkastus. Analysointi ja suunnittelu-disipliinin roolit ja aktiviteetit on esitetty kuvassa 21.



Kuva 21. Analysointi ja suunnittelu-disipliinin roolit ja aktiviteetit [1].

5.2.3 Artefaktit

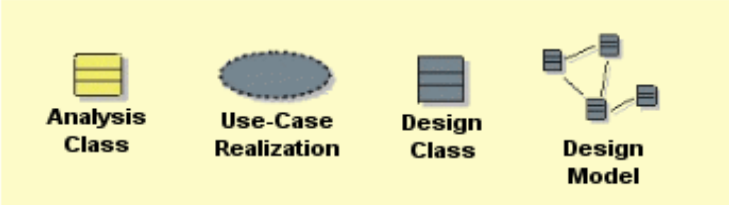
Tämän disipliinin artefakteja ovat analyysiluokka (Analyses Class), käyttötapausrealisaatio (Use-Case Realization), suunnitteluluokka (Design Class) ja suunnittelumalli (Design Model). Kolme viimeistä artefaktia - käyttötapausrealisaatio, suunnitteluluokka ja suunnittelumalli – yhdistetään joskus ryhmäksi, jota kutsutaan *ohjelmiston arkkitehtuuri artefaktiksi* (software architecture artifact), koska niitä käytetään ohjelmiston arkkitehtuurin määrittämisessä [2].

Analyysiluokat ovat systeemin prototyyppimäisiä luokkia. Niiden tärkein sisältö on luokan stereotyyppien määritelmät, jotka kuvaavat esimerkiksi raja-, ohjaus- tai entiteettiluokkia. *Rajaluokat* (Boundary classes) helpottavat systeemin ymmärtämistä kuvaamalla sellaiset systeemin osat, joiden on oltava vuorovaikutuksessa systeemin ulkopuolisten entiteettien kanssa. *Ohjausluokat* (Control classes) koteloivat ohjauskäyttäytymisen, joka on spesifistä yhdelle tai muutamalla käyttötapaukselle. Ne siis mallintavat systeemin dynamiikkaa. *Entiteettiluokat* (Entity classes) mallintavat informaatiota ja siihen assosioitua käyttäytymistä, joka pitää tallentaa. Analyysiluokkien joukko yhdessä muodostaa systeemistä varhaisen konseptuaalinen mallin.

Käyttötapausrealisaatio kuvaa yhteistyössä olevien objektien avulla, miten tietty käyttötapaus on realisoitu järjestelmän suunnittelumallissa. Käyttötapausrealisaatio on käyttötapauksen suunnitteluperspektiivi. Käyttötapauksen realisaatio voidaan tehdä käyttötapauksesta usealla eri tavalla esimerkiksi saman tuoteperheen sisällä. Käyttötapausrealisaation tarkoituksena on eriyttää systeemin spesifikaatioon ja systeemin suunnitelmaan liittyvät kysymykset toisistaan.

Suunnitteluluokka on sellaisen objektijoukon kuvaus, joilla on samat vastuut, suhteet, attribuutit ja merkitykset. Luokat määrittelevät objekteja jotka realisoiva, toisin sanoen implementoivat, käyttötapauksia. Arkkitehtuurisesti merkittävät luokat identifioidaan ja määritellään elaboraatiovaiheen aikana ja jäljelle jäävät luokat rakennusvaiheen aikana.

Suunnittelumalli on objektimalli, joka kuvaa käyttötapausten realisaation ja toimii abstraktiona implementaatiomallille ja sen lähdekoodille. Se on luonnos, joka kuvaa miten lähdekoodi strukturoidaan ja kirjoitetaan. Suunnittelumallia käytetään keskeisenä syötteenä implementaatio- ja testaus-disipliinin aktiviteeteille. Se koostuu pakettien hierarkiasta, jonka ”lehdet” esittävät luokkia tai käyttötapausrealisaatioita. Hyvä suunnittelumalli on yhdenmukainen suhteessa järjestelmän vaatimuksiin, adaptiivinen ja valmis toteutettavaksi. Tärkeimmät analyysi ja suunnittelu-disipliinin artefaktit on esitetty kuvassa 22.



Kuva 22. Tärkeimmät analysointi ja suunnittelu-disipliinin artefaktit [1].

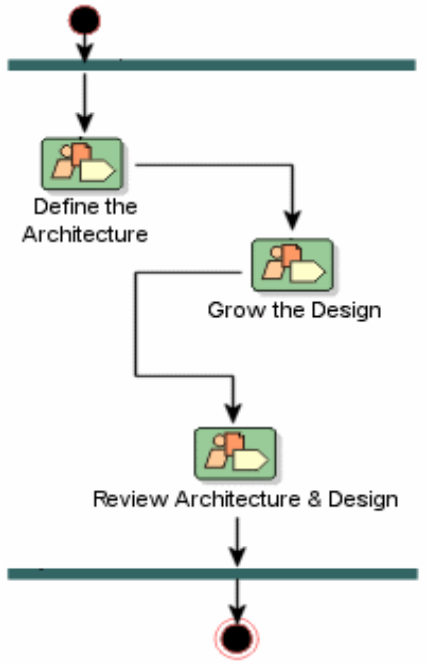
UPEDU-prosessi ehdottaa systeemin mallin esittämiseen niin kutsuttua *2+1 täydentävää perspektiiviä* (complementary views), joka koostuu siis yhteensä kolmesta perspektiivistä, joita ovat: looginen-, käyttötapaus- ja implementaatioperspektiivi. Numeraalinen osuus nimeämisessä perustuu siihen, että looginen- ja implementaatioperspektiivi on johdettu käyttötapausperspektiivistä. Suunnittelijat vangitsevat suunnittelupäätöksensä käyttämällä loogista- ja implementaatioperspektiiviä. Käyttötapausperspektiiviä käytetään sitten kahden muun mainitun

perspektiivin valaisemisessa ja validoimisessa. *Käyttötapausperspektiivi* sisältää muutaman avainskenaarion tai käyttötapauksen, joita käytetään arkkitehtuurin suurimman osan johtamiseen ja validoimiseen. Kyseisen perspektiivin graafisena representaationa toimii käyttötapauskaaviot toimijoineen ja käyttötapauksineen. *Looginen perspektiivi* käsittelee systeemin funktionaalisia vaatimuksia. Sen graafinen representaatio koostuu luokkakaavioista, vuorovaikutuskaavioista ja tilakaavioista. Siihen kuuluu mm. pakettirakenne, kiinnostavat luokat sekä käyttötapausrealisaatioita. *Implementaatioperspektiivi* kuvaa staattisten ohjelmamoduulien organisaation kehitysympäristössä. Sen graafinen representaatio taas koostuu komponenttikaavioista, joiden avulla voidaan kuvata mm. koodikirjastojen rakenne, lähdekoodi ja muut toimitettavat systeemitiedostot. [2]

5.2.4 Työnkulku

Alkuvaiheessa analyysi- ja suunnittelu-disipliinissä tarkastellaan onko visioitu systeemi toteuttamiskelpoinen ja minkälaisia potentiaalisia teknologiaratkaisuja voitaisiin käyttää. Jos kehitykseen liittyvät riskit tuntuvat pieniltä esimerkiksi sovellusalueen hyvän tuntemuksen vuoksi, voidaan tämä alkutarkastelu jättää pois.

Varhainen suunnittelu-/tarkennusvaihe keskittyy alustavan arkkitehtuurin luomiseen systeemille. Kun kaikki alustavat elementit on identifioitu, niitä tarkennetaan kasvattamalla systeemin mallia. Tuloksena on alustava joukko luokkia, joita vielä tarkennetaan implementaatio-disipliinissä. Analyysi- ja suunnitteludisipliini työkulku on esitetty kuvassa 23.



Kuva 23. Analyysi ja suunnittelu-disipliinin työnkulku [1].

5.2.5 Suhde muihin disipliineihin

Vaatimustenhallinta-disipliini tarjoaa primäärisen syötteen analysointi ja suunnittelu-disipliinille. Implementaatio-disipliini tarjoaa ohjelmiston komponenttirakenteen, jota käytetään kehitysympäristössä. Projektinhallinta-disipliini suunnittelee projektin ja siihen kuuluvat iteraatiot.

5.3 Implementaatio

Implementaatio yhdistetään usein ohjelmointitehtäviin ohjelmiston kehitysprosessissa. Tämä disipliini merkkaa kristallisaatioprosessin (katso kohta 7.2) valmistumista. Suunnitteluartefaktien muuttaminen ohjelmakoodiksi tulee suunnitella hyvin ja tulos validoida yksikkötestauksen avulla.

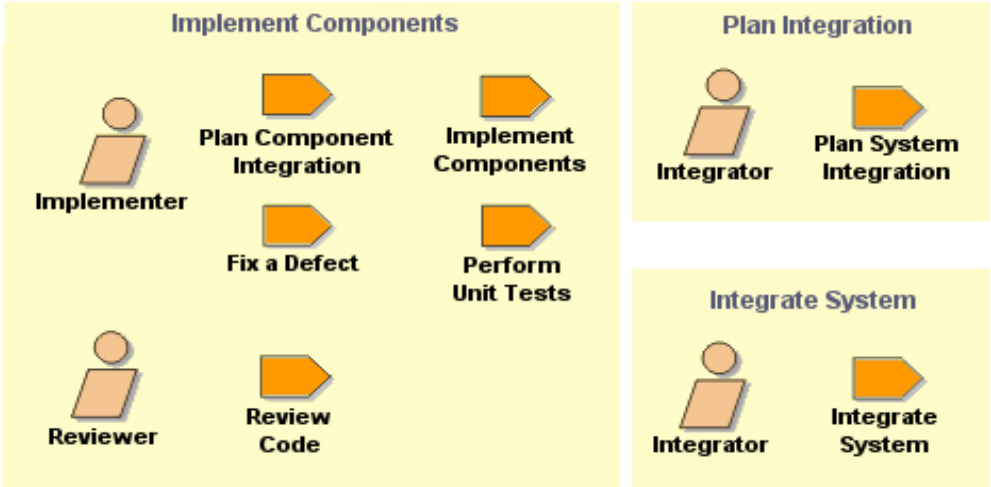
5.3.1 Tarkoitus

Implementaatio-disipliinin tarkoitus on [1]:

- määritellä koodin organisaatio alijärjestelmien avulla, jotka organisoidaan tasoiksi
- implementoida luokat ja objektit komponenttien (lähdekoodi-tiedostot, binääri-tiedostot, suoritustiedostot ja muut) tasolla
- testata kehitetyt komponentit yksikköinä
- integroida eri implementoijien (tai tiimien) tuotokset ajettavaksi järjestelmäksi

5.3.2 Roolit ja aktiviteetit

Implementaatio-disipliinissä implementoija suunnittelee komponenttien integraation, implementoi komponentit, tekee yksikkötestausta sekä korjaa löytyneitä virheitä. Implementoijien tulee pitää huoli siitä, että dokumentaatio pysyy ajan tasalla koko rakennusvaiheen ajan. Katselmoija tarkastaa koodin ja integroija suunnittelee systeemitason integroinnin ja toteuttaa sen. Katselmoijan rooli on avain koodin hyvään laatuun ja koodin yhdenmukaisuuteen suhteessa koodaussuosituksiin. Pienissä projekteissa integroijan rooli saattaa olla triviaali. Implementaatio-disipliinin roolit ja aktiviteetit on esitetty kuvassa 24.

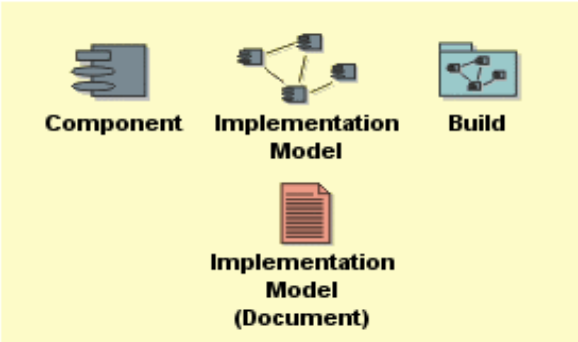


Kuva 24. Implementaatio-disipliinin roolit ja aktiviteetit [1].

5.3.3 Artefaktit

Implementaatio-disipliinin artefakteja ovat komponentit, implementaatiomalli sekä käännös (build). Lisäksi koodin tarkastuksesta syntyy tarkastuspöytäkirja. Pientä systeemiä kehitettäessä saattaa olla perusteltua vain yhden käännöksen toteuttaminen. *Käännös* on operationaalinen versio täydellisestä järjestelmästä tai sen osasta. Implementaatio-disipliinin lopullisia tuotteita ovat suoritettavat komponentit, jotka on johdettu paketeista ja niiden sisältämistä komponenteista. Paketit ja niiden komponentit pohjautuvat luokkiin ja luokkakaavioihin, jotka taas on johdettu käyttötapauskaaviosta.

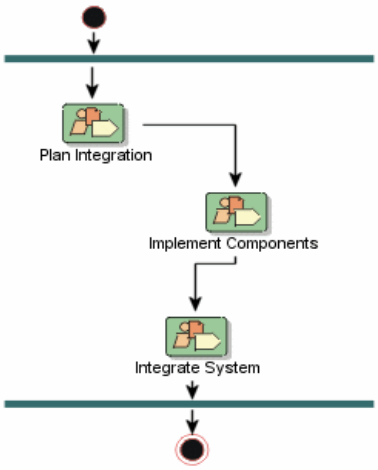
Joitain muita kaavioita, kuten tila- tai yhteistyökaavioita, saatetaan tarvita luokkien kuvaamisessa. Jos kehitettävä systeemi on pieni, systeemin implementaatiomalli voi olla hyvinkin samanlainen kuin sen suunnittelumalli. Implementaatiomallia voidaan käyttää apuna työn jakamisessa itse implementoijille. Implementaatio-disipliinin tärkeimmät artefaktit on esitetty kuvassa 25.



Kuva 25. Implementaatio-disipliinin tärkeimmät artefaktit [1].

5.3.4 Työnkulku

Implementaatio-disipliini rajaa laajuuden, jolla yksittäiset luokat yksikkötestataan. Systeemi- ja integrointitestit kuuluvat testaus-disipliiniin. Jokaisessa iteraatiossa, alkaen suunnittelu/tarkennus-vaiheesta, kehitetään integrointisuunnitelma (eng. Integration Plan), implementoidaan komponentit sekä integroidaan ne systeemiin. Implementaatio-disipliinin työnkulku on esitetty kuvassa 26.



Kuva 26. Implementaatio-disipliinin työnkulku [1].

5.3.5 Suhde muihin disipliineihin

Vaatimustenhallinta-disipliini kuvaa kuinka kerätä käyttötapausmalleiksi vaatimukset, jotka implementaation tulisi täyttää. Analysointi ja suunnittelu-disipliini kuvaa miten suunnittelumalli (eng. Design model) tulisi kehittää. Suunnittelumalli kuvaa implementoinnin tarkoitusta ja toimii primäärisenä syötteenä implementaatio-disipliinille. Testaus-disipliini kuvaa kuinka integrointitestata jokainen käännös (build) systeemin integroinnin aikana ja kuinka testaamisella verifioidaan kaikkien vaatimusten toteutuminen. Projektinhallinta-disipliini kuvaa miten projekti voidaan parhaiten suunnitella.

5.4 Testaus

Tämä disipliini toimii tavallaan palvelun tarjoajana toisille disipliineille. Testaus keskittyy pääosin tuotteen laadun arvioimiseen erilaisten testausmenetelmien avulla. Testaus ymmärretään usein tuotteen laadun parantamiseksi, mutta testaus tulisi ymmärtää saavutetun laadun mittaamisena. Tavallisimpia testausmenetelmiä ovat:

- yksikkötestaus
- integrointitestaus
- regressiotestaus
- systeemitestaus
- hyväksymistestaus

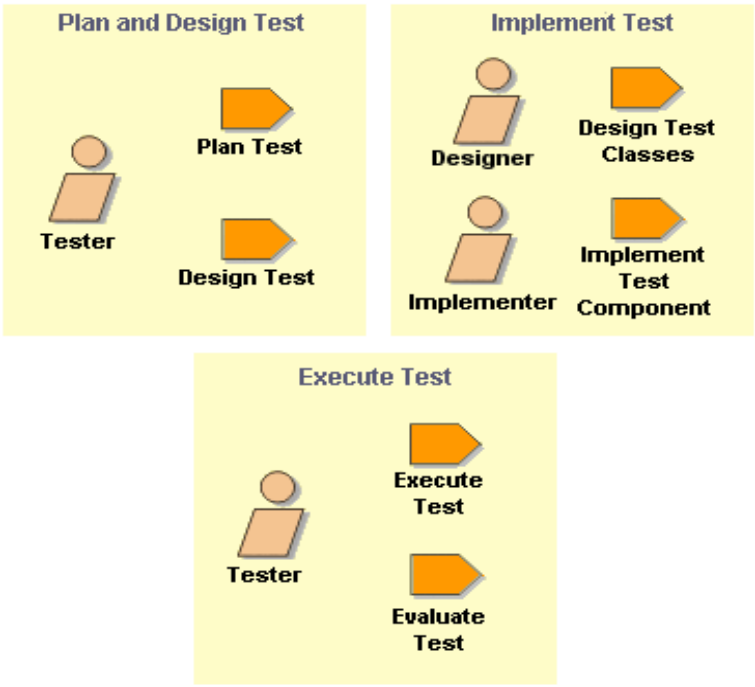
5.4.1 Tarkoitus

Testaus-disipliinin tarkoitus on [1]:

- löytää ja dokumentoida viat ohjelmiston laadussa
- antaa neuvoja liittyen koettuun ohjelmiston laatuun
- todistaa vaatimusten määrittelyn ja suunnittelun aikaan tehtyjen olettamuksien validiteetti konkreettisilla demonstraatioilla
- validoida että ohjelmistotuote toimii kuten on suunniteltu
- validoida että vaatimukset on implementoitu asianmukaisesti

5.4.2 Roolit ja aktiviteetit

Testaajan kuuluu suunnitella testaus ja sen suorittaminen sekä lopulta testitapauksien ajaminen ja tulosten evaluointi. Jotta testaus olisi mahdollista, sitä varten saattaa joutua tekemään uusia vain testauksessa käytettäviä luokkia ja niiden suunnitteleminen kuuluu suunnittelijan tehtäviin. Implementoija toteuttaa suunnitellut testiluokat ja komponentit. Testaaja ajaa testit ja suorittaa testitulosten evaluoinnin. Testaus-disipliinin roolit ja aktiviteetit on esitetty kuvassa 27.



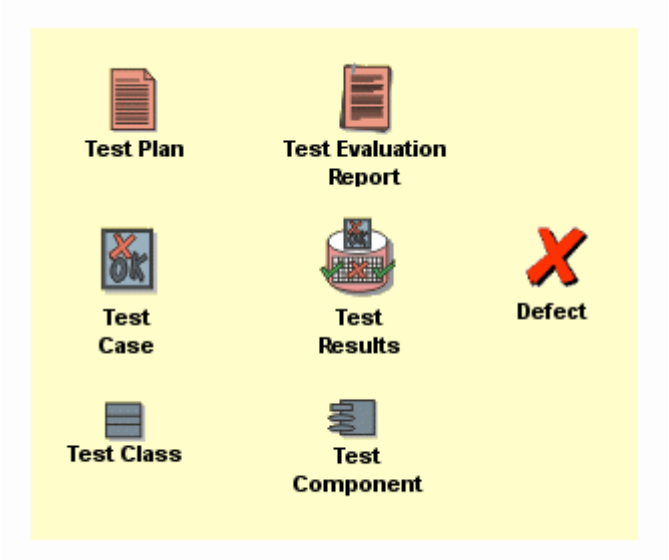
Kuva 27. Testaus-disipliinin roolit ja aktiviteetit [1].

Yksikkötestauksen suorittaa implementoija ja muut testausaktiviteetit jäävät suunnittelijoille ja testaajille.

5.4.3 Artefaktit

Testaus-disipliinin artefakteja ovat testaussuunnitelma, testitapaukset ja niihin liittyvät testiluokat ja testikomponentit, testitulokset ja testauksen evaluointiraportti sekä löydetyt defektit. *Testaussuunnitelmassa* identifioidaan olemassa oleva projektin informaatio ja ohjelmistokomponentit, jotka tulisi testata, listataan korkealla tasolla testattavat vaatimukset, kuvataan käytetyt testausmenetelmät, identifioidaan testauksessa tarvittavat resurssit ja arvioidaan tarvittava työmäärä testauksen suorittamiseen sekä listataan testauksen tuottamat ja toimitettavat elementit.

Testitapaukset dokumentoidaan *testitapaukset-dokumenttiin*, joka tulee olla tarpeeksi yksityiskohtainen, jotta muutkin kuin dokumentin kirjoittaja pystyvät testitapaukset suorittamaan. *Testitapaus* sisältää mm. kuvauksen testattavasta kokonaisuudesta, testauksessa käytettävät syötteet ja odotetut tulosteet, testausympäristön kuvauksen sekä testitapausten väliset riippuvuudet. Jos testitapaus epäonnistuu, tulee siitä kirjoittaa *defektiraportti*, joka kuvaa mm. pääpiirteissään tilanteen, jossa defekti löydettiin sekä jatkotoimenpiteet sen korjaamiseksi. On olemassa defektien kuvaamista käsitteleviä standardeja kuten IEEE-10444. Se sisältää taksonomian anomalioiden kuvaamiseksi. Testitapauksista saatavat tulokset dokumentoidaan *testaustulokset-dokumenttiin*, jossa kerrotaan lyhyesti mikä testi onnistui ja mikä epäonnistui. *Testauksen evaluointiraportissa* tehdään yhteenveto testaustuloksista ja määritellään vaatimus- ja lähdekoodiperustainen testauskattavuus. Testaustulosten ja avainmittareiden perusteella määritellään vielä ehdotus testauksen jälkeisille jatkotoimenpiteille. Testaus-disipliinin tärkeimmät artefaktit on esitetty kuvassa 28.

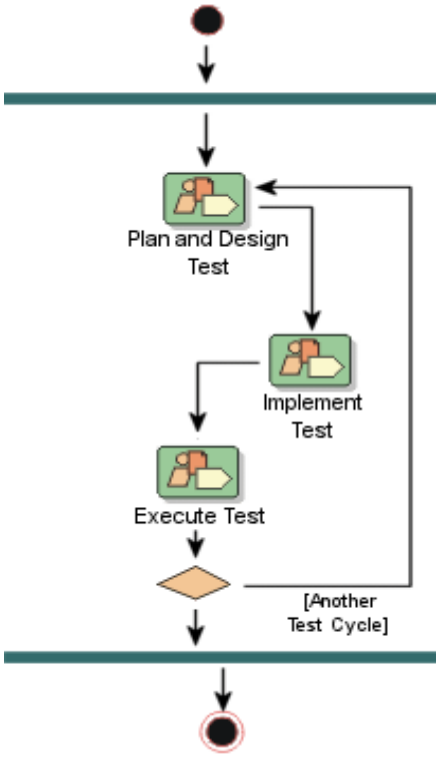


Kuva 28. Tärkeimmät testaus-disipliinin artefaktit [1].

5.4.4 Työnkulku

Testaus suunnitellaan ja sen yleispiirteet kuvataan testaussuunnitelmassa. Hyvän testaus-suunnitelman lisäksi tarvitaan testiluokkien ja -komponenttien suunnittelua ja toteutusta. Lopuksi

testitapaukset suoritetaan ja evaluoidaan. Tavallisesti koko testaus-disipliinin työnkulku suoritetaan useaan otteeseen eli testaussyklejä tarvitaan useita. Syynä useisiin testaussykleihin on tavallisesti se, että tuotteeseen on joko lisätty tai siinä on muutettu jotain ominaisuutta, jolloin se pitää testata uudestaan, tai jotain testiä ei voida suorittaa tai edes suunnitella ennen kuin systeemistä on jotkut tietyt osat jo toteutettu. Testausta ei tule kuitenkaan jättää myöhempään vaiheeseen vaan testaus tulee suorittaa samaan aikaan kehityksen kanssa. Testaus-disipliinin työnkulku on esitetty kuvassa 29.



Kuva 29. Testaus-disipliinin työnkulku [1].

5.4.5 Suhde muihin disipliineihin

Testaus-disipliinin avulla voidaan varmistaa mittaamisen avulla rakennus-disipliinin tuotosten laadukkuus. Toisaalta testaus-disipliinissä muodostaa useita dokumentteja ja tiedostoja, joiden oikea versio on olennaisen tärkeää tietää ja säilyttää ja tässä auttaa konfiguraation ja muutostenhallinta-disipliini. Testaus-disipliinissä on mukana useita henkilöitä, joten testaus vaatii organisointia projektinhallinta-disipliinin aktiviteettien avulla.

5.5 Konfiguraation ja muutostenhallinta

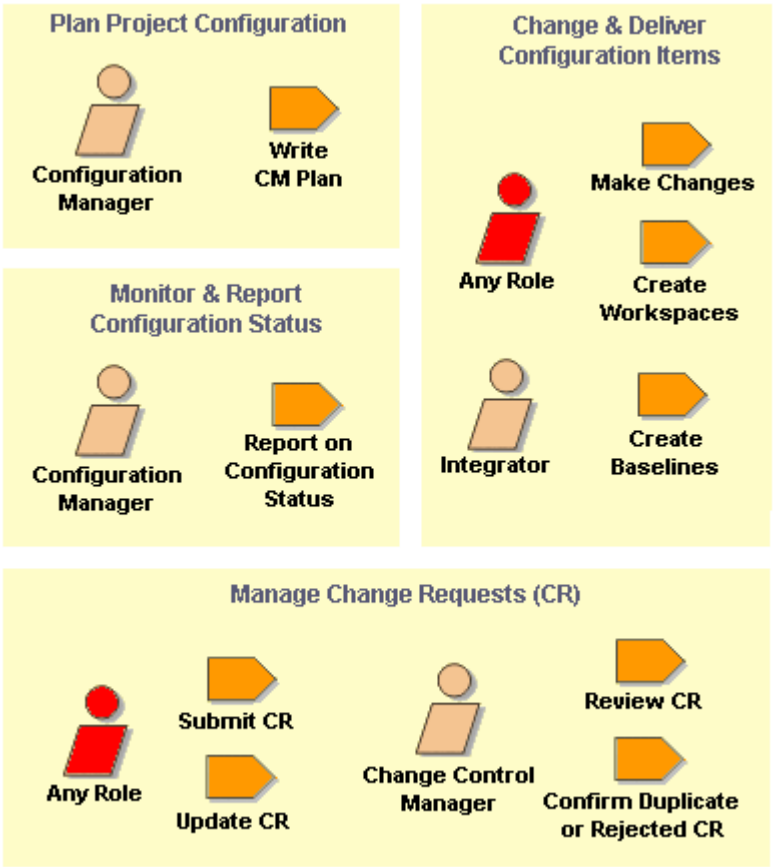
Konfiguraation ja muutostenhallinta (Configuration and Change Control Management, CM and CCM) kontrolloi muutoksia projektin artefakteihin sekä ylläpitää niiden integriteettiä. Tämän disipliinin tärkeimmät osat ovat muutospyyntöjen hallinta (Change Reguest Management, CRM), konfiguraation hallinta (Configuration Management, CM) sekä mittaus (Measurement).

5.5.1 Tarkoitus

Organisaation konfiguraation ja versionhallintasysteemillä (Configuration and Change Request Management System, CM System) hallitaan useita artefakteja, joita syntyy yhteisen projektin sisällä. Kontrolloinnin avulla voidaan välttää kalliiksi tulevia sekaannuksia ja varmistaa, että muokkauksen tuloksena olevat artefaktit eivät ole konfliktissa. Konflikteja voivat aiheuttaa esim. samanaikainen artefaktin päivitys tai useat kehitettävän ohjelman eri versiot, tai tilanne, jossa kaikille ei välity tietoa artefaktin päivityksestä. CM systeemin avulla voidaan myös paremmin välttyä artefaktien tahattomalta tuhoamiselta.

5.5.2 Roolit ja aktiviteetit

Konfiguraation ja muutostenhallintaan kuuluu sellaisten kohteiden identifiointi, jotka kuuluvat konfiguroinnin alaisuuteen. Näin voidaan rajoittaa niihin tehtäviä muutoksia, auditoida muutoksia sekä määritellä ja hallita niiden konfiguraatioita. Konfiguraation ja muutostenhallinta-disipliinin rooleja ja aktiviteetteja on esitetty kuvassa 30.

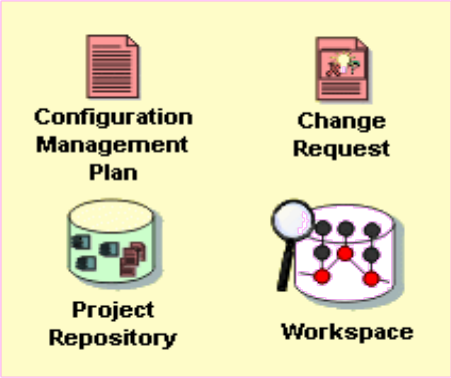


Kuva 30. Konfiguraation ja muutostenhallinta-disipliinin roolit ja aktiviteetit [1].

5.5.3 Artefaktit

Konfiguraation ja muutostenhallintaan kuuluvia artefakteja ovat konfiguraationhallintasuunnitelma, muutospyynnöt (change requests), projektin artefaktivarasto (project repository) sekä työtilat (workspaces). *Konfiguraationhallintasuunnitelmassa* määritellään mm. vastuuhenkilöt

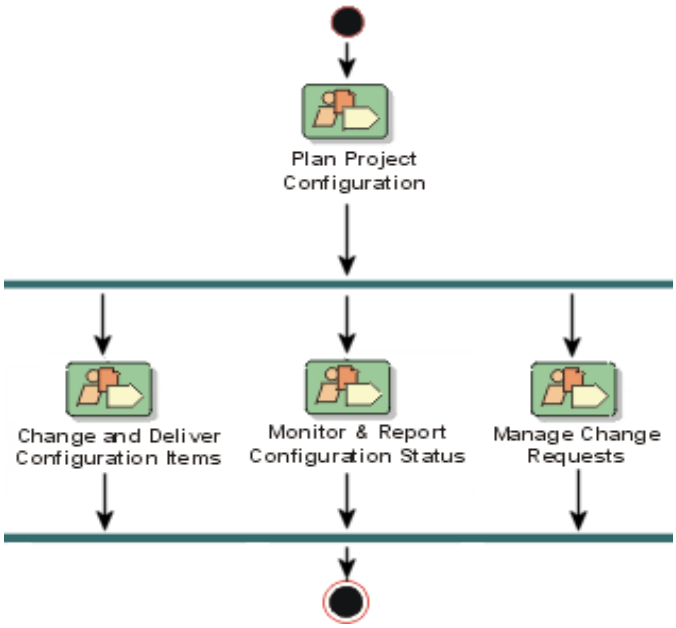
konfiguraation- ja muutostenhallinnalle, konfiguraationhallintaympäristö, konfiguraationhallinta-ohjelma sekä mahdollinen projektiympäristön ulkopuolinen kehitysympäristö. Konfiguraation-hallintaohjelma kuvaa mm. konfiguraation identifiointitavan, muutostenhallintaprosessin sekä konfiguraation statuksen kirjanpidon. Konfiguraation ja muutostenhallintaan kuuluvia artefakteja esittää kuva 31.



Kuva 31: Konfiguraation ja muutostenhallinta-disipliinin artefaktit [1].

5.5.4 Työnkulku

Konfiguraation ja muutostenhallinnan suunnittelu sekä konfiguraatioympäristönluominen kuuluu tehdä projektin alussa. Kuvassa 32 olevan työnkulun muita tehtäviä suoritetaan ajoittain koko projektin elinkaaren läpi.



Kuva 32. Konfiguraation ja muutostenhallinta-disipliinin työnkulku [1].

5.5.5 Suhde muihin disipliineihin

Organisaation CM systeemiä käytetään koko koko tuotteen elinkaaren ajan. CM systeemiin kuuluu mm. tuotteen hakemistorakenne ja sitä kautta konfiguraation ja muutostenhallinta-disipliini on yhteydessä kaikkiin muihin disipliineihin.

5.6 Projektinhallinta

Ohjelmistoprojektin hallinta on taidetta. Se sisältää kilpailevien päämäärien tasapainottamista, riskien hallintaa ja monenlaisten esteiden ylittämistä asiakkaan ja käyttäjän tarpeiden tyydyttämiseksi. Projektinhallinta on ilmeisen vaikea tehtävä, sillä vain harvat projektit todella onnistuvat kaikkien arviointikriteerien mukaan. Tosin hyvä projektinhallintakaan ei vielä takaa projektin onnistumista. Projektipäällikön vastuuseen kuuluvia teknisluontoisia asioita ovat perinteisesti vastuu budjetista, tehtävien jakamisesta, aikataulusta ja niin edelleen, mutta täytyy muistaa että tehtävä on myös ihmisten eikä vain asioiden johtamista. Tällaisiin ei-teknisiin tehtäviin kuuluu mm. ihmisten odotuksien hallinta.

Projektipäällikkö varmistaa kontrollinhallinta-aktiviteettien avulla, että projekti sujuu suunnitelmien mukaan. Hän käyttää mitattavia päämääriä voidakseen arvioida miten suunnitelmat ja todellisuus vastaavat toisiaan esimerkiksi valmistumisen, laadun ja budjetin suhteen. Mitattavien päämäärien avulla voidaan arvioida myös muutospyyntöjen aiheuttamia vaikutuksia projektin laajuuteen. Projektipäällikkö mittaa kehitystiimin tehokkuutta sekä projektin tuloksia ja kirjaa ylös mahdolliset poikkeavuudet suunnitelmista.

Iteratiivinen kehitys vie aikaa ensimmäisellä kerralla enemmän kuin traditionaaliset lähestymistavat. Yleisen sudenkuopan välttämiseksi tulee iteraatioiden liian suurta määrää varoa, sillä jokaisen uuden iteraation suunnitteluun, monitorointiin ja kontrollointiin tarvitaan jonkin verran lisätyötä.

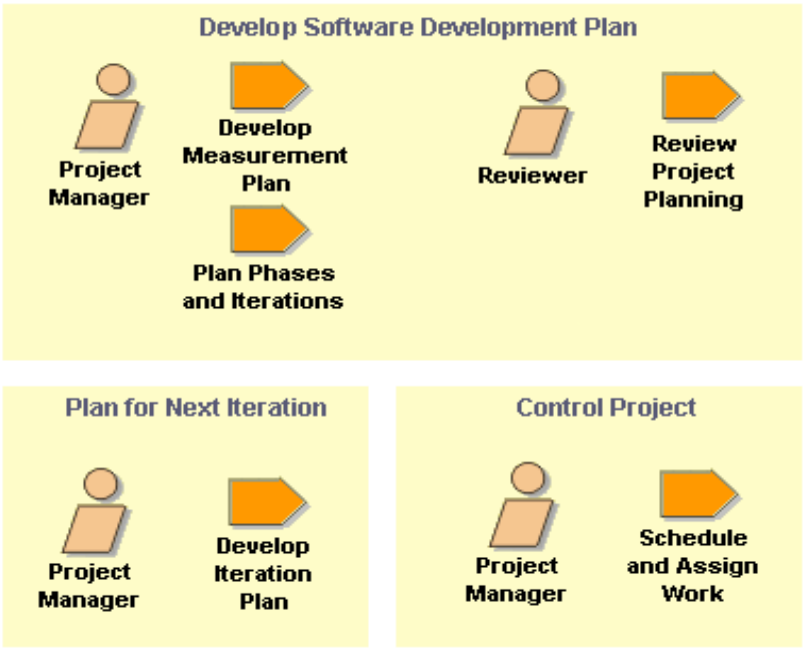
5.6.1 Tarkoitus

Projektinhallinta-disipliinin tarkoitus on [1]:

- tarjota kehys ohjelmisto-intensiivisten projektien hallintaan.
- tarjota käytännönläheisiä ohjeita projektien suunnitteluun, henkilöstön hankintaan, suoritukseen ja monitorointiin.
- tarjota kehys riskienhallintaan.

5.6.2 Roolit ja aktiviteetit

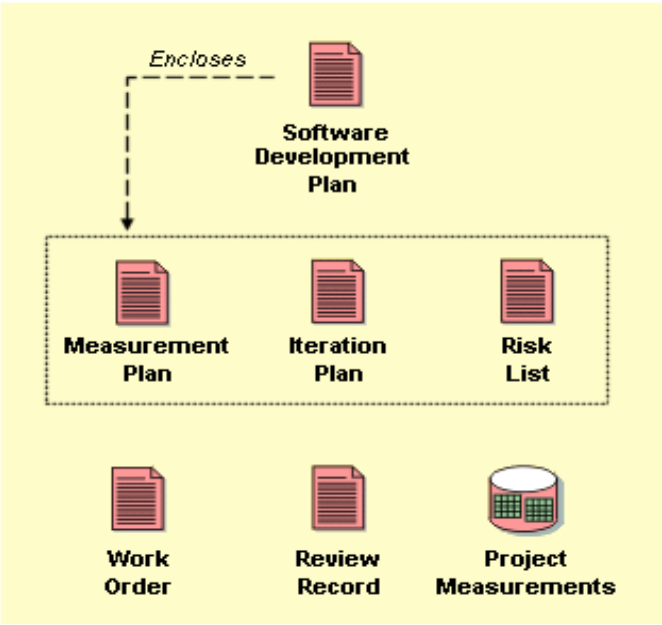
Projektinhallinta-disipliiniin kuulu projektipäällikön ja katselmoijan rooli. Projektipäällikkö aktiviteetteihin kuuluu ohjelmiston kehityssuunnitelman laatiminen, johon sisältyy mm. vaiheiden ja niiden sisältämien iteraatioiden suunnittelu. Projektipäällikkö kontrolloi projektia mm. aikatauluttamalla ja jakamalla töitä sekä kirjoittaa iteraatiosuunnitelman seuraavalle iteraatiolle ennen sen aloittamista. Katselmoijan tehtävänä on katselmoida projektin suunnittelu. Projektinhallinta-disipliinin roolit ja aktiviteetit on esitetty kuvassa 33.



Kuva 33. Projektinhallinta-disipliinin roolit ja aktiviteetit [1].

5.6.3 Artefaktit

Projektinhallinta-disipliinin artefaktit liittyvät prosessin ja projektin suunnitteluun ja suorittamiseen. Tämän disipliinin artefakteihin kuuluu kehityssuunnitelma, johon sisältyy mittaussuunnitelma, iteraatiosuunnitelma sekä riskilista. Alustava ohjelmiston kehityssuunnitelma katselmoidaan alkuvaiheen loppupuolella ja päivitetään jokaisen suuremman virstanpylvään kohdalla. Kyseinen suunnitelma katselmoidaan myös kun vastaan tulee ongelmia, jotka aiheuttavat muutoksia suunnitelmaan. Tärkeimpiä projektinhallinta-disipliinin artefakteja esittää kuva 34.



Kuva 34. Tärkeimmät projektinhallinta-disipliinin artefaktit[1].

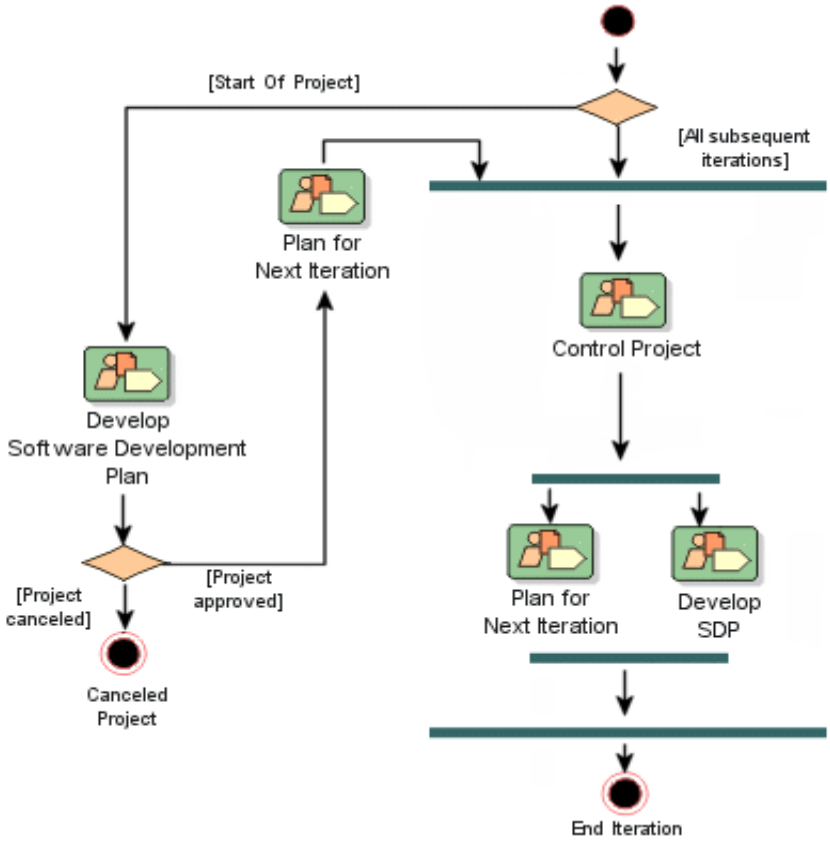
Mittaussuunnitelma määrittelee mitä primitiivisiä metriikoita tulisi kerätä ja mitä metriikoita tulisi laskea projektin aikana. Tarkoituksena on monitoroida projektin edistymistä vertaamalla sitä projektin tavoitteisiin. Mittaus-suunnitelma kirjoitetaan alkuvaiheen aikana tai joskus osana kehitystapauksen (Development Case) prosessin konfigurointia. Suunnitelmaa voidaan kuitenkin tarkistaa ja päivittää projektin aikana. Projektin minimaalinen joukko metriikoita mittaa ansaittua arvoa (Earned Value), defektitrendejä (Defect Trends) ja testauksen edistymistrendiä. Ansaittua arvoa käytetään aikataulun ja budjetin uudelleen estimoinnissa sekä projektin laajuuden muutostarpeen identifioinnissa. Defektitrendejä seuraamalla nähdään defektien määrän kehitys, jonka avulla voidaan arvioida työmäärää, jonka defektien korjaamiseen vielä tarvitaan. Testauksen edistymistrendistä voidaan päätellä, kuinka suuri osa systeemin toiminnallisuudesta on todellisuudessa valmis.

5.6.4 Työnkulku

Ensimmäisessä iteraatiossa, eli aloitusiteraatiossa, mikä ajoittuu alkuvaiheeseen, kehitetään alustava visio-dokumentti sekä riskilista ja ne katselmoidaan. Tarkoituksena on tässä vaiheessa hankkia riittävä rahoitus, jonka jälkeen voidaan aloittaa tarkempi projektin suunnittelu ja laajuuden määrittely. Alustava ohjelmiston kehityssuunnitelma luodaan ja projekti käynnistetään alustavan iteraatiosuunnitelman mukaisesti. Tämän jälkeen visio-dokumenttia ja riskilistaa kannattaa täydentää, jotta ne antavat vahvan pohjan ohjelmiston kehityssuunnitelmalle.

Ohjelmiston kehityssuunnitelman perusteella voidaan nyt päättää kannattaako alkuvaihetta jatkaa vai hylätä koko projekti. Seuraavaksi alustavaa iteraatiosuunnitelmaa tarkennetaan kuvaamaan koko aloitusiteraatio. Seuraavan iteraation suunnitteluaktiviteetin avulla projektipäällikkö, analysoija sekä suunnittelija päättävät, mitkä vaatimukset tutkitaan, tarkennetaan ja realisoidaan. Ennen rakennusvaihetta huomio keskittyy vaatimuksien suhteen lähinnä niiden keruuseen ja tarkennukseen. Iteraatioiden suunnittelua tarkennetaan kokemaan koko alkuvaihetta. Kun iteraatio suorituksessa saavutaan sen loppuun, iteraatio katselmoidaan, jotta saadaan selvitetyksi onko iteraation päämäärät saavutettu. Projektin kontrollointia voidaan suorittaa tarvittaessa esimerkiksi päivittäin. Seuraavan iteraation suunnitteluaktiviteetin kanssa rinnakkain päivitetään ohjelmiston kehityssuunnitelmaa, jotta ne vastaavat toisiaan.

Jokaisen vaiheen viimeisessä iteraatiossa saavutetaan virstanpylväs, jolloin on syytä suorittaa kattavampi katselmus. Koko projektin päätösvaiheessa suoritetaan projektin hyväksymiskatselmus, jossa selvitetään onko kehitettävä tuote hyväksyttävä vai tarvitaanko vielä uusia iteraatioita. Projektinhallinta-disipliinin työnkulku on esitetty kuvassa 35.



Kuva 35. Projektinhallinta-disipliinin työnkulku [1].

5.6.5 Suhde muihin disipliineihin

Projektinhallinta-disipliini tarjoaa kehyksen projektin luomiselle ja hallinnalle, joten tämä disipliini liittyy kiinteästi kaikkiin muihin disipliineihin.

9. Laadusta

Ohjelmistotuotteen laadunhallinnan tavoitteena on saavuttaa vaadittu tuotteen laatu. Tämän saavuttamiseksi tulisi ensiksi määritellä laatuvaatimukset ja niiden implementointi. Tuotteen laatuun liittyy standardi ISO/IEC 1926 – Software Engineering – Product Quality sekä standardi ISO/IEC 14598 – Evaluation of software products. ISO/IEC 1926 määrittelee laatukriteerit sekä metriikat, joiden avulla mitata laatua. ISO/IEC 14598 standardi antaa ohjeita ja vaatimuksia itse evaluointiprosessille. Molemmat laatustandardit perustuvat malliin, jossa ohjelmiston kehitysprosessi vaikuttaa ohjelmistotuotteen sisäiseen laatuun, joka taas vaikuttaa sen ulkoiseen laatuun, ja tämä taas tuotteen käytönaikaiseen laatuun. Sisäinen ja ulkoinen laatu liittyvät suoraan ohjelmistotuotteeseen, kun taas käytönaikainen laatu liittyy vaikutukseen, joka tuotteen käytöllä on sen erilaisissa käyttökonteksteissa [7]. ISO/IEC 1926 standardi sisältää laadun kriteereitä, jotka kohdistuvat mm. ohjelman toiminnallisuuteen, luotettavuuteen, käytettävyyteen, tehokkuuteen, ylläpidettävyyteen sekä siirrettävyyteen [6]. Aivan standardin mukaiseen laadun mittaamiseen ja arviointiin ei pienessä nopeasti läpi vietävässä projektissa ole välttämättä aikaa ja resursseja. Tuotelaadun lisäksi on olemassa esimerkiksi resurssilaatu ja prosessilaatu. Laatu voidaan jaotella myös objektiiviseksi tai subjektiiviseksi.

Kuten jo aikaisemmin suunnittelu-disipliinistä kertovan kappaleen yhteydessä mainittiin, UPEDUssa laadukas ohjelma nähdään saavutettavan ensisijaisesti hyvällä ohjelman suunnittelulla. Testaus koetaan usein virheellisesti tavaksi parantaa laatua. Testaus tulisi kuitenkin nähdä saavutetun laadun mittaamisena eikä se itsessään luo laatua [2]. Laadunhallintaan (quality control) liittyy lisäksi erilaiset katselmointi- ja tarkastusmenettelyt. Se mitä laadulla tarkoitetaan, riippuu pitkälti näkökulmasta, joka tarkastelussa otetaan. Laadun määrittelee asiakas ja toimittaja sekä käytössä olevat standardit.

10. Mittaamisesta

Mittaamisen avulla voidaan pienentää riskejä seuraamalla projektin evoluutiota ja tuotteen laatua. Työkalujen pitäisi olla tarkasti suunniteltuja mittaamisen tukemiseen. Jos prosessien aktiviteettien mittaamista ei automatisoida tai jos kehittäjät tuntevat metriikat hyökkääviksi, he saattavat helposti vältellä näiden työkalujen käyttöä. Kokonaisen projektin arvioiminen ei ole helppoa. Harvoissa organisaatioissa on riittävästi aikaa tehdä projektin post-mortem analyysejä ja projektitiimi on usein innokas aloittamaan jo seuraavan projektin. Yksi mahdollisuus on arvioida projektia jokaisen iteraation lopussa. Iteraation aikana projektipäällikön tulee koko ajan seurata projektin edistymistä ja riskejä, jotta voidaan varmistaa että esiin nousevat vaikeudet eivät ohjaa projektia sivuraiteelle. Ennen jokaisen iteraation loppua, projektipäällikkö voi suunnitella palaverin iteraation arvioimiseksi. Tämän palaverin tarkoituksena olisi arvioida tuotteen edistymistä kyseisen iteraation aikana suhteessa iteraatiosuunnitelmaan kirjattuihin evaluointikriteereihin. Iteraatio voidaan nähdä onnistuneeksi kun kaikki riskit on redusoitu suunnitellulle tasolle, kaikki suunnitellut toiminnallisuudet on toteutettu sekä laatutavoitteet saavutettu.[2]

Mittaamissuunnitelma sisältää mittaamiset ja metriikat, joiden avulla projektipäällikön pitää seurata projektin edistymistä. Projektia arvioidaan kolmen perustavanlaatuisen alueen avulla: tekninen edistyminen, taloudellinen status sekä henkilöstöhankinnan edistyminen. *Tekniset metriikat* (technical measurements) perustuvat erilaisiin aktiviteetteihin, jotka suoritetaan iteratiivisessa kehitysprojektissa. Esimerkiksi projektipäällikkö hahmottelee suoritettavat tehtävät (tasks) ja sitten monitoroi niiden valmistumista. *Taloudelliset metriikat* (financial metrics) saadaan mittaamalla kustannuksia, joita syntyy henkilö- ja materiaaliresursseista kehitysprojektin aikana. Taloudellisen kehityksen seuraaminen suoritetaan yleensä ansaitun arvon systeemiä käyttäen, joka tarjoaa yksityiskohtaisen näkemyksen kustannuksista ja aikataulusta. Tutkimalla näitä metriikoita voidaan päätellä noudattaako projekti suunniteltua budjettia ja aikataulua vai tarvitaanko arvioihin

korjauksia. Ohjelmistoprojektityön kurssilla keskitytään ensisijaisesti käytetyn työmäärän (effort) mittaamiseen. [2]

11. Prosessin arvioinnista ja parantamisesta

Organisaatiot käyttävät ohjelmistoprosessejamalleja määrittääkseen ohjelmiston kehittämiseen liittyvät aktiviteetit, aktiviteettien väliset suhteet ja niiden tuottamat artefaktit. UPEDU on siis myös tällainen prosessimalli. Referenssimalleja taas käytetään ohjelmistoprosessin arvioimisessa, esim. CMM (Capability Maturity Model). CMM perustuu joukkoon ohjelmiston kehityksen kyvykkyyksiä, joita organisaatioilla tulisi olla saavuttaessaan prosessin eri kypsyystasoja. Kuinka hyvin organisaation prosessi vastaa CMM referenssiprosessimallia voidaan ilmaista viisiportaisen luokittelun avulla: 1. perusprosessi (initial), 2. toistettava (repreatable), 3. määritelty (defined), 4. hallittu (managed) ja 5. optimoiva (optimizing). CMMn lisäksi muita sertifiointiin johtavia referenssiprosessimalleja ovat esim. ISO 9001 sekä SPICE (ISO/EIC 15504). Prosessin arviointi tapahtuu vertaamalla organisaation malliprosessia referenssiprosessimalliin. Referenssiprosessimalli antaa pohjan arvioinnille, mutta referenssiprosessimallin käyttö myös varmistaa, että arviointitulokset voidaan esittää yhteisessä kontekstissa. [2]

Organisaation prosessimalli kehitetään pääasiassa vastaamaan organisaation aktiviteetteja. Prosessimallin sisällön luomiseen voidaan käyttää pääasiassa kolmea lähdettä: henkilökohtainen kokemus, referenssiprosessimallit, ja prosessityökalut. Mallit ovat enemmän tai vähemmän formaaleja. Informaalit mallit ovat pääasiassa johdettu henkilökohtaisesta kokemuksesta sellaisista aktiviteeteista, joita tarvitaan ohjelmiston kehittämisessä. Formaalit mallit taas sitä vastoin ovat strukturoitu sitä varten perustetun ryhmän toimesta, täysin organisaation tukemia, ja voivat perustua tai olla johdettuja standardeista kuten ISO/EIC 12207 (Standard for Information Technology – Software Life Cycle Processes). Organisaatio voi myös johtaa ohjelmistoprosessinsa räätälöimällä kokonaisvaltaisen ohjelmistonprosessityökalun, kuten Rational Unified Process (RUP). UPEDU on esimerkki organisaation prosessimallista, joka on johdettu RUP referenssiohjelmisto-prosessityökalusta. Prosessin parantamista tehdään, jotta saavutettaisiin etuja kuten ohjelmiston parempi laatu, alhaisemmat kehitys- ja ylläpitokustannukset, kasvanut ohjelmistotuotteiden ja prosessien ennustettavuus ja hallittavuus sekä lyhyempi markkinoillevientiaika. Ohjelmistoprosessin parannukset toteutetaan yleensä jatkuvina parannussykleinä, jossa parannus saavutetaan askeleittain tai tiettyinä parannustoimina, kuten muuttamalla ohjelmistokehityksen käytäntöjä.

UPEDU on iteratiivinen prosessi, joka nojaa saman määritellyn prosessin uudelleensuorittamiseen usean iteraation ajan. Prosessin toistuva luonteen, projektin tilasta kertovien metriikoiden ja jokaisesta iteraatiosta ja vaiheesta saatujen oppien avulla voidaan prosessia hienosäätää jokaista seuraavaa iteraatiota varten. [2]

12. Käytetyt työkaluohjelmat

Käytetyistä työkaluista toimitetaan tarvittavilta osin projektiryhmän käyttöön käyttöohje-dokumentteja. Osa ohjeistuksista voidaan välittää myös sähköpostilla. Ohjeita työkaluohjelmien käyttöön voidaan antaa myös demoamalla niitä alkuluennoissa ja wrap-up kokoontumisien yhteydessä.

Mallinnus

- StarUML
  - Tämä on suositeltu vaihtoehto, mutta myös muita voi käyttää
- esim. ArgoUML
  - voit käyttää myös ArgoUML:llään pohjautuvaa plug-in -ohjelmaa, joka integroituu Eclipseen eli toiminnallisuuden voi liittää kyseiseen kehitysalustaan
  - Eclipseen löytyy myös muita UML-kuvaukseen soveltuvia plug-in ohjelmia

Toteutus

- Eclipse ja Java
- tai asiakkaan vaatima toteutuskieli

Testaus

- Junit (yksikkötestaus ja osa regressiotestauksesta)
- Mantis (bugien seuranta: projektiryhmän löytämät bugit, asiakkaan ilmoittamat bugit ja muutospyynnöt)
  - asiakkaan ilmoittamat bugit ja muutospyynnöt voidaan tarvittaessa ottaa vastaan jollain muullakin tavalla, mutta tästä on sovittava erikseen kurssin vastuuhenkilöiden kanssa.

Konfiguraation ja muutostenhallinta

- CVS (projektivarasto, mallipohjat artefakteille)
  - lue CVS-ohje
  - yliopiston ulkopuolinen CVS:n käyttö onnistuu ssh-tunnelin avulla. Voit käyttää tunnelin luomiseen esimerkiksi Putty ohjelmaa, jonka ohje löytyy osoitteesta http://www.cs.uu.nl/technical/services/ssh/putty/puttyfw.html

Projektinhallinta

- DotProject (tehtävät, ohjeistukset)

Viitteet

[1] Unified Process for Education [UPEDU] (2004).WWW-sivusto, http://www.upedu.org/upedu/ (15.8.2006).

[2] Robillard, P.N., Kruchten, P., d'Astous, P. (2003) Software Engineering Process with the UPEDU. Addison Wesley, Boston.

[3] Robillard, P.N., Kruchten, P., d'Astous, P. (2001) YOOPEEDOO (UPEDU): A Process for Teaching Software Process. Software Engineering Education and Training. Proceedings of the 14th Conference on 19-21 Feb 2001, 18 – 26.

[4] IEEE Std 1063-2001: IEEE Standard for Software User Documentation. (lähde: http://ieeexplore.ieee.org/iel5/7687/21001/00974401.pdf)

[5] IEEE Std 830-1998: IEEE recommended Practice for Software Requirements Specifications. (lähde: http://ieeexplore.ieee.org/iel4/5841/15571/00720574.pdf?arnumber=720574)

[6] ISO 9126 (1992) Information technology - Software product evalutation - Quality characteristics and guidelines for their use. International Organisation for Standardization, Geneva.

[7] W. Suryn, A. Abran, and A. April (2003) ISO/IEC SQuaRE. The Second Generation of Standards for Software Product Quality. Proceeding (397) Software Engineering and Applications on November 3-5, Marina del Rey, CA, USA.

Liite1: Dokumenttiin kohdistuneet muutokset

Versio 1.00 (2.9.2006)
- julkaisu

Versio 1.01 (6.9.2006)
- lisätty sisällysluettelon otsikko

Versio 1.02 (7.9.2006)
- tarkennettu raporttien tarkoitusta ja merkitystä (kohta 6)
- tarkennettu evoluutiosyklin selitystä (kohta 3)
- lisätty käsite ohjelmiston elinkaari (kohta 3)
- muutettu kuvassa 1 olevan konseptuaalisen mallin suhteen merkitystä niin, että aktiviteetti ei kuluta artefaktia vaan käyttää sitä (kohta 2)

Versio 1.03 (7.9.2006)
- lisätty otsikot prosessin vaiheille (kohdat 7, 7.1-7.4)
- lisätty alkuvaiheen selitys (kohta 7.1 ja alakohdat)
- lisätty CVS-ohjeistus

Versio 1.04 (12.9.2006)
- lisätty käsite prosessityökalu (kohta 3)
- tarkennettu vaiheiden nimiä (kohta 3.1)
- lisätty elaboraatiovaiheen selitys (kohta 7.2 ja alakohdat)
- muutettu listojen merkintätapaa OpenOffice:n pdf-export yhteensopivaksi

Versio 1.05 (22.9.2006)
- tarkennettu mittaussuunnitelman selitystä (kohta 8.6.3)

Versio 1.06 (20.10.2006)
- lisätty rakennusvaiheen (construction) kuvaus (kohta 7.3 ja alakohdat)

Versio 1.07 (30.11.2006)
- muutettu kannesta laitoksen nimi tietojenkäsittelytieteen ja tilastotieteen laitokseksi
- lisätty siirtymävaiheen (transition) kuvaus (kohta 8.3 ja alakohdat)

Versio 1.08 (20.12.2006)
- lisätty kappaleet 9, 10, 11
- täydennetty kohtia sieltä täältä

Version 1.09 (30.8.2007)
- täydennetty kohtia sieltä täältä

Version 1.10 (12.9.2008)
- korjailtu kieliasua