Hyvän käyttöliittymän varmistaminen GUIDe-prosessimallilla

Sari A. Laakso, Helsingin yliopisto, tietojenkäsittelytieteen laitos
Karri-Pekka Laakso, Interacta Design Oy

Tiivistelmä

Monissa projekteissa ohjelmiston käyttöliittymän ongelmat tulevat esille vasta käyttöönoton yhteydessä, jolloin korjaaminen voi aiheuttaa toteutusratkaisuihin kalliita muutoksia. Ongelman syynä voi olla riittämätön tai jopa kokonaan puuttuva käyttöliittymäsuunnitteluprosessi, minkä seurauksena käyttöliittymä saattaa syntyä esimerkiksi koodauksen sivutuotteena, tai jatkuvasti projektin aikana muuttuvat asiakasvaatimukset, jotka selviävät sitä mukaa kuin ohjelmiston toteutus etenee ja tuotteen toiminnallisuus konkretisoituu käyttöliittymäksi.

GUIDe-prosessimallissamme (Goals – User Interface Design – Implementation) käyttöliittymä suunnitellaan heti projektin alussa konkreettiseksi vaatimukseksi toteutukselle. GUIDen sisältämä suunnitteluprosessi tuottaa systemaattisella tavalla myös järjestelmässä tarvittavan tietosisällön ja toiminnallisuuden.

GUIDessa käyttöliittymäsuunnittelu sijoitetaan projektin alkuun, ennen toteutuksen tai esimerkiksi tietokantaratkaisujen suunnittelua, jotta käyttöliittymäratkaisuja ja järjestelmän soveltuvuutta käyttötarkoituksiinsa voitaisiin testata niin varhaisessa vaiheessa, että testitulosten vaatimat suuretkin muutokset olisivat vielä helposti toteutettavissa. Tarvittavien muutosten tekeminen käyttöliittymän kuvasarjoihin on huomattavasti vaivattomampaa ja edullisempaa kuin toimivan ohjelmakoodin käyttäminen asiakasvaatimusten tai käyttöliittymäkehityksen iterointivälineenä. Järjestelmän pienoismallina toimiva havainnollinen käyttöliittymäkuvaus on myös tehokas kommunikointiväline asiakkaan suuntaan.

Avainsanat: GUIDe, käyttöliittymäsuunnittelu, toiminnallisuus, käytettävyys, tavoitepohjaiset käyttötapaukset, asiakasvaatimukset, vaatimusmäärittely, prosessimalli.

ACM CCS: H.5.2, D.2.1

Sisällys

1 Johdanto

Monet käyttöliittymäongelmat tulevat usein esille vasta ohjelmiston valmistumisen jälkeen, kun järjestelmällä yritetään ensimmäistä kertaa saada tehtyä käyttäjien todellisia työtehtäviä. Pahimmassa tapauksessa käyttötarkoituksen kannalta keskeistäkin toiminnallisuutta voi puuttua järjestelmästä kokonaan. Osa eteen tulevista ongelmista puolestaan on käytettävyysongelmia: tehottomista ja vaikeaselkoisista käyttöliittymäratkaisuista syntyvää ajanhukkaa, virheitä ja turhaa koulutustarvetta. Osa käyttöliittymän puutteista on niin pahoja, että ne on joka tapauksessa korjattava ennen käyttöönottoa.

Monissa tapauksissa ongelmat ovat seurausta siitä, että käyttöliittymää ei missään vaiheessa suunnitella eikä sen soveltumista käyttötarkoitukseensa testata ennen kuin vasta sitten, kun järjestelmä otetaan käyttöön. Käyttöliittymän tarjoama toiminnallisuus voi syntyä asiakkaan hyväksymien toimintoluettelojen pohjalta, joiden tarkoituksenmukaisuutta ei ole missään vaiheessa suhteutettu käyttäjien työtehtäviin ja kohdeyrityksen liiketoimintaprosesseihin – joita ei välttämättä edes ole selvitetty. Vaatimusdokumentissa lueteltujen toimintojen käyttöliittymäratkaisut saattavat syntyä koodauksen sivutuotteena. Joissain projekteissa käyttöliittymäsuunnittelulle on varattu vähäinen määrä resursseja, mutta suunnittelun lähtökohdaksi tarvittavaa käyttäjien työtehtävien ja tavoitteiden selvitystä ei ole tehty lainkaan.

Sen sijaan, että järjestelmää ja sen käyttöliittymää testattaisiin todellisissa käyttötilanteissa vasta käyttöönoton jälkeen, koko järjestelmä voidaan alusta lähtien suunnitella käyttötilanteiden ja käyttäjien tavoitteiden (goals) pohjalta. GUIDe-prosessimallissamme (Goals – User Interface Design – Implementation) käyttäjien työnkulut ja tavoitteet selvitetään heti projektin alussa ja tarvittaessa työnkulkuja suoraviivaistetaan jo tässä vaiheessa. Tavoitteiden selvittämisen jälkeen käyttöliittymäsuunnittelija johtaa niitä tukevat käyttöliittymäratkaisut ja piirtää käyttöliittymän toimintalogiikasta yksityiskohtaiset kuvasarjat. Käyttöliittymäkuvat ovat ikään kuin täsmällinen pienoismalli seuraavissa vaiheissa toteutettavasta järjestelmästä, ja niistä selviää myös järjestelmän tarjoama toiminnallisuus.

Pienoismallin rakentamisen ansiosta järjestelmän käyttöliittymä eli sen tarjoamat toiminnot, toimintojen käyttötavat sekä tietosisältö ja tiedon esitystavat voidaan testata jo ennen toteutuksen aloittamista, jolloin korjausten tekeminen on vielä vaivatonta. Lisäksi myös asiakas pystyy kunnolla ottamaan kantaa konkreettisiin suunnitelmiin jo projektin alussa: Työnkulkujen ja tavoitteiden kuvauksista hän näkee, mitä järjestelmällä pitäisi pystyä tekemään, ja käyttöliittymäkuvista selviää täsmällisesti, miten työtehtävät järjestelmällä suoritetaan. Jatkossa testattu käyttöliittymäkuvaus (user interface specification) toimii syötteenä mm. käsiteanalyysille, tietokannan suunnittelulle ja järjestelmän arkkitehtuurisuunnittelulle.

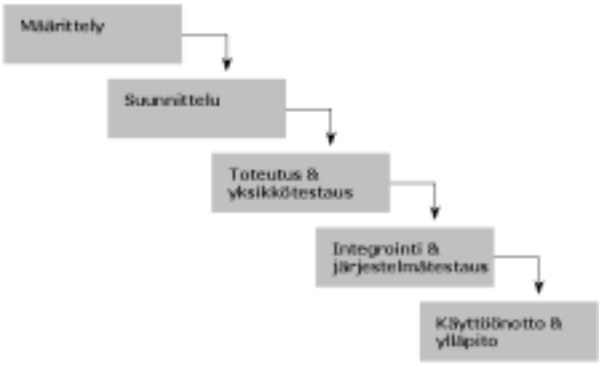
Luvussa 2 eritellään muutaman nykyisin käytössä olevan prosessimallin ongelmia erityisesti hyvän käyttöliittymän kehittämisen kannalta. Luvussa 3 kuvataan GUIDe-prosessimallin vaiheistus ja menetelmät. Luvussa 4 GUIDen menetelmiä sovitetaan muiden prosessimallien osaksi sekä arvioidaan niiden tuomia etuja ja haasteita.

2 Puutteita prosessimallien käyttöliittymäkehityksessä

Monissa tällä hetkellä käytössä olevissa prosessimalleissa ohjelmiston käyttöliittymä syntyy muiden, usein toteutusnäkökohtiin keskittyvien aktiviteettien sivutuotteena. Tässä luvussa luodaan katsaus kahdenlaisiin prosessimalleihin ja niiden ongelmiin käyttöliittymäkehityksen näkökulmasta: luvussa 2.1 eritellään vesiputousmallia noudattavia ja luvussa 2.2 inkrementaaliseen kehitykseen perustuvia menettelyjä.

2.1 Vesiputousmalli

Tiukasti vaiheistettu vesiputousmalli (kuva 2.1) ei tyypillisesti sisällä erillistä käyttöliittymän suunnitteluvaihetta, vaan ohjelman toimintovalikoimaan ja käytettävyyteen liittyviä vaatimuksia on esitetty muiden vaatimusten yhteydessä. Vaatimusmäärittelyn jälkeen dokumentointi painottuu järjestelmän arkkitehtuurisuunnitteluun ja muihin toteutusteknisiin näkökohtiin, ja käyttöliittymä ikään kuin rakennetaan etukäteen määritellyn toiminnallisuuden ja tietosisällön päälle. Se, miltä lopullinen järjestelmä tulee näyttämään ja miten se käyttäytyy loppukäyttäjien käsissä, tulee näkyviin vasta projektin loppupuolella, kun toteutus alkaa valmistua.



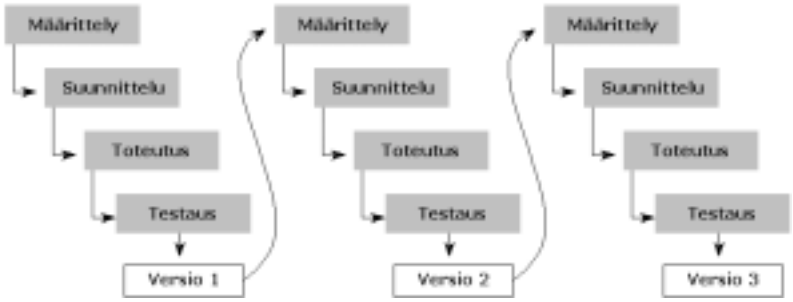
Kuva 2.1. Vesiputousmallin vaiheet.

Koska käyttöliittymä ja sen myötä myös ohjelman toimintalogiikka ja tietosisällön käyttötavat konkretisoituvat vasta projektin lopussa, järjestelmän soveltuvuutta käyttäjien työtehtäviin ei voida varmistaa, ennen kuin järjestelmä on koodattu toimivaksi tai lähes toimivaksi. Monissa tapauksissa vielä pahempana ongelmana on se, että käyttöliittymä on kehittynyt toteutusnäkökohtien päälle, mikä näkyy käyttöliittymässä toteutusratkaisujen (implementaatiomalli) ja niihin liittyvien käsitteiden ja interaktiotapojen suosimisena. Käyttäjältä vaaditaan hänen työtehtäviensä kannalta turhaa toteutusratkaisujen toimintalogiikkaan perehtymistä sekä monimutkaisia ja usein virhealttiita toimenpidesarjoja, jotka eivät tue hänen tavoitteeseensa pääsemistä. Lisäksi osa työtehtävistä voi olla mahdollista saada tehtyä loppuun ainoastaan ulkoisia apuvälineitä käyttäen, kuten kirjoittamalla muistilappuja tai kopioimalla ja käsittelemällä tietoja sopivassa taulukkolaskentaohjelmassa järjestelmän ulkopuolella.

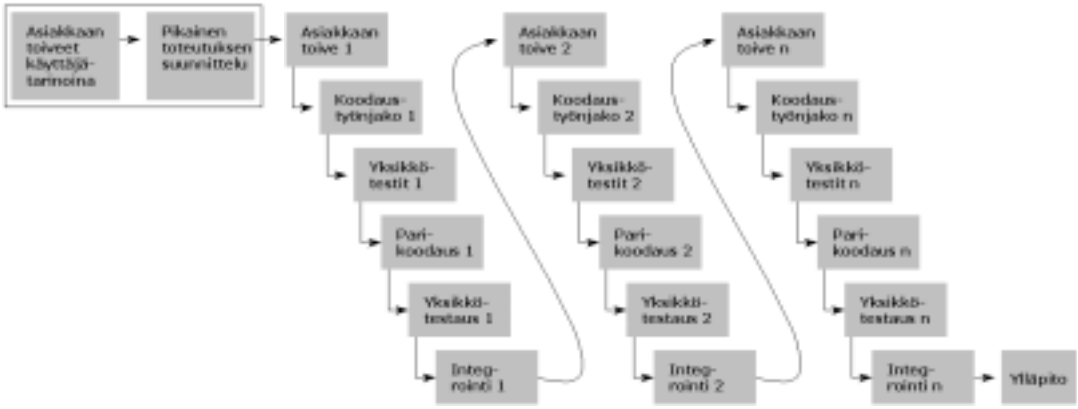
Valmiissa järjestelmässä ilmenee usein puutteita ja ongelmia, joiden korjaaminen on näin myöhäisessä vaiheessa hyvin epäoptimaalista. Käyttöönotto saattaa viivästyä, käyttäjien kokemiin ongelmiin ja virheisiin kuluu turhaa työaikaa ja korjaustoimenpiteet tulevat kalliiksi. Erään arvion mukaan muutoskustannukset kasvavat eksponentiaalisesti kymmenen potensseissa jokaisesta vesiputousmallin vaiheesta seuraavaan siirryttäessä [Boehm81]. Siten myös käyttöliittymäongelmat tulisi osata ennakoida ja niiden syntyminen tulisi pyrkiä estämään heti projektin alussa.

2.2 Inkrementaalinen ohjelmistokehitys

Inkrementaaliseen kehitykseen perustuvat prosessimallit, kuten evoluutiomalli (kuva 2.2) ja Extreme Programming (XP) –malli [Beck00] (kuva 2.3), yrittävät ratkaista liian myöhään tulevien korjausliikkeiden ongelmaa toteuttamalla ohjelmistoa pala palalta vaiheittain. Aluksi projektiryhmä suunnittelee ja koodaa vain pienen osan toiminnallisuutta, joka testataan. Tämän jälkeen suunnitellaan lisää toimintoja, jotka koodataan ja testataan, ja näin jatketaan, kunnes koko järjestelmä on saatu valmiiksi. Asiakas ei osaa etukäteen luetella vaatimuksia kattavasti, mutta niitä tulee esille vähitellen, kun järjestelmää toteutetaan ja asiakkaalle alkaa selvitä, millainen järjestelmästä tulee ja miten se toimii. ”I can’t tell you what I want, but I’ll know when I see it” [Boehm88, s. 63].



Kuva 2.2. Evoluutiomallin vaiheita.



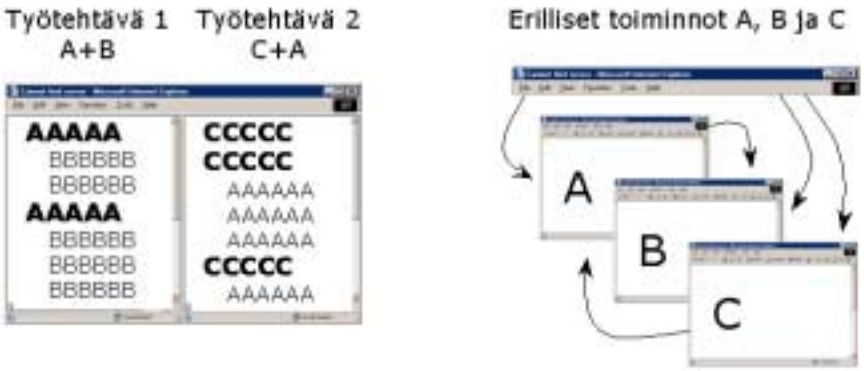
Kuva 2.3. XP-mallin vaiheita.

Hyvän käyttöliittymän tuottamisen kannalta inkrementaaliseen kehitykseen perustuvissa malleissa on jossain mielessä jopa enemmän ongelmia kuin vesiputousmallissa. Inkrementaalisissa malleissa käyttöliittymä syntyy pahimmillaan ilman varsinaista käyttöliittymäsuunnittelua koodauksen sivutuotteena (esimerkiksi XP) ja parhaimmillaankin käyttöliittymää

suunnitellaan vain pala kerrallaan jokaisen evoluutiosyklin alussa (evoluutiomalli).

Hyvää käyttöliittymäratkaisua ei synny pala kerrallaan suunnittelemalla, koska samalla käyttöliittymällä on lopulta pystyttävä suorittamaan kaikki keskeisimmät työtehtävät alusta loppuun asti. Jos työtehtävän 1 suorittamisessa tarvittavat toiminnot A ja B suunnitellaan eri vaiheissa, toimintojen käyttäminen peräkkäin työtehtävän suorittamisessa ei välttämättä enää olekaan suoraviivaista. Toiminnosta toiseen siirtyminen saattaa vaatia käyttäjältä valtavan määrän navigointia paikasta toiseen, tai toiminnon A tuottama data ei ole samanaikaisesti näkyvillä toiminnon B tuottaman datan kanssa, vaikka käyttäjän pitäisi vertailla näitä tietoja.

Oletetaan, että käyttäjä tarvitsee työtehtävässä 1 samanaikaisesti toimintojen A ja B tuottamaa dataa ja työtehtävässä 2 toimintojen C ja A tuottamaa dataa (kuva 2.4). Tällöin hyvässä käyttöliittymäratkaisu tarjoaa näkymät, joissa työtehtävää 1 varten on visualisoitu samanaikaisesti näkyviin toimintojen A ja B tuottama data (kuvassa vasemmalla), ja työtehtävää 2 varten toimintojen C ja A tuottama data (kuvassa keskellä). Jos käyttöliittymää ei suunnitella yhdellä kertaa työtehtävien pohjalta vaan paloittain, käyttöliittymään todennäköisimmin muodostuu irrallisia saarekkeita (esimerkiksi ikkunoita, kuten kuvassa 2.4 oikealla), joissa yhdessä on toiminnon A, toisessa toiminnon B ja kolmannessa toiminnon C tuottama data. Ikkunoiden välillä siirtyileminen voi olla vaivalloista, eikä käyttäjä välttämättä saa toimintojen A ja B dataa samanaikaisesti näkyviin työtehtävänsä vaatimalla tavalla organisoituna ja visualisoituna.

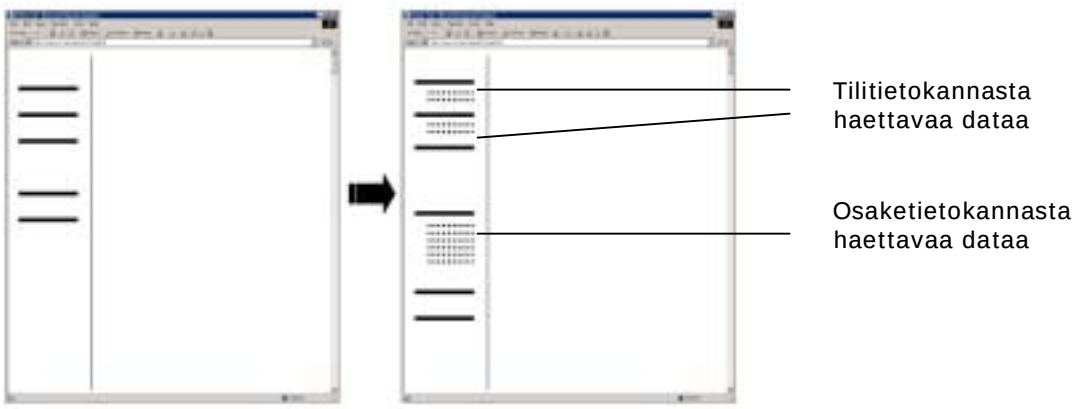


Kuva 2.4. Työtehtävien ja toimintojen riippuvuus.

Toinen ongelma pala kerrallaan syntyvästä käyttöliittymästä on se, että lisättäessä järjestelmään uusi pala toiminnallisuutta myös käyttöliittymään on lisättävä vastaava uusi pala, joka tyypillisesti muuttaa myös olemassa olevaa käyttöliittymää. Jos olemme rakentamassa käyttäjän työtehtäviin sopivaa hyvää käyttöliittymäratkaisua emmekä irrallisia toimintosaarekkeita, uuden toiminnallisuuden lisäys vaikuttaa paitsi olemassa olevaan käyttöliittymään, myös koodiin – eikä pelkästään käyttöliittymäkoodiin, vaan usein myös sen alla oleviin toteutusratkaisuihin.

Käyttöliittymäratkaisun muuttuminen toimintoja lisättäessä voi pahimmillaan muuttaa suorituskykyvaatimuksia tai niiden seurauksenna alla olevia arkkitehtuuriratkaisuja merkittävästi. Kuvassa 2.5 on esimerkki suorituskykyvaatimuksiin vahvasti vaikuttavasta käyttöliittymän muutoksesta yhden evoluutiosyklin aikana. Ensimmäisessä vaiheessa kuvitteelliseen web-pankkijärjestelmään on toteutettu vain muutama perustoiminto, joita vastaavat

linkit on sijoitettu vasemman reunan navigointipalkkiin. Käyttäjä voi näiden linkkien avulla katsoa tilitapahtumiaan yksi tili kerrallaan. Toisessa vaiheessa järjestelmään on lisätty mahdollisuus seurata myös omaa osakesalkkua reaaliajassa, minkä seurauksena navigointipalkissa kannattaa näyttää suoraan myös käyttäjän omien osakkeiden viimeisimmät muutokset. Samalla navigointipalkkiin päätetään nostaa myös tilien saldot näkyviin. Tällöin käyttäjä pystyy suorittamaan mahdollisimman monta tavoitettaan loppuun asti pelkästään avaamalla järjestelmän päänäkymän ja vilkaisemalla navigointipalkin overview-osaa.



Kuva 2.5. Käyttöliittymäratkaisun kehittyminen evoluutiosyklien välillä.

Oletetaan, että kuvan 2.5 järjestelmään toteutetaan ensimmäisen evoluutiosyklin aikana vain ne toiminnot, jotka hakevat pankkitietokannasta käyttäjän tilitietoja aina, kun hän valitsee vasemman reunan navigointipalkista jonkin tilin (kuvassa 2.5 vasemmalla). Seuraavassa evoluutiovaiheessa lisätään osakesalkkutoiminnot, minkä seurauksena navigointipalkin käyttöliittymäratkaisua päivitetään siten, että viimeisimmät osakemuutokset näytetään suoraan palkissa (kuvassa 2.5 oikealla). Vanhassa käyttöliittymässä haku kohdistuu aina yhteen pankin tietokantaan kerrallaan ja vain silloin, kun käyttäjä valitsee navigointipalkista jonkin linkin. Uusi käyttöliittymäratkaisu vaatii navigointipalkin overview-näkymässä olevan datan hakemista kaikista tili- ja osaketietokannoista heti sisäänkirjautumisen jälkeen, minkä seurauksena tarvittavien hakujen määrä moninkertaistuu. Silti käyttöliittymän pitäisi latautua nopeasti sekä sisäänkirjautumisen jälkeen että käytön aikana.

Kaikista edellisen kaltaisista tilanteista ei välttämättä synny tarvetta kantavien arkkitehtuuriratkaisujen uudelleen suunnittelulle, mutta monissa tilanteissa olisi huomattavan edullista nähdä lopullinen, kokonainen käyttöliittymäratkaisu jo ennen toteutusratkaisujen suunnittelua, koska käyttäjän kannalta hyvät käyttöliittymäratkaisut tuovat paljon uusia vaatimuksia toteutuksen suunnittelulle. Esimerkiksi järjestelmän tilan säilyttäminen käynnistyskerrasta toiseen, epätäydellisten syötteiden tukeminen, jatkuva tallentaminen (autosave) tai perumistoiminnot (undo) voivat jälkeenpäin lisättäessä aiheuttaa valtavia muutoksia arkkitehtuuriratkaisuihin tai esimerkiksi tietokannan rakenteeseen.

3 GUIDe-prosessimalli

Vesiputousmallin ongelmana on se, että toteutettavan järjestelmän toiminta konkretisoituu asiakkaalle ja käyttäjille liian myöhään, minkä seurauksena tarvittavien muutosten tekeminen on vaikeaa ja kallista. Inkrementaaliseen kehitykseen perustuvissa prosessimalleissa taas eräänä lähtökohtana on uskomus siitä, ettei vaatimuksia kuitenkaan voida saada riittävän hyvin selville heti alussa. Järjestelmää lähdetään kehittämään paloittain yrittäen näin minimoida muutoksista tulevaa ylimääräistä työtä edes jonkin verran. Tämä kuitenkin johtaa lähinnä huonon käyttöliittymän syntymiseen paloittain asiakkaan mielipiteiden varassa, ja prosessin aikana tehdään valtavasti turhaa koodaustyötä.

GUIDe-mallin kehitys on lähtenyt olettamuksesta, että vaatimukset pystytään saamaan jo projektin alussa selville riittävän täsmällisesti, kunhan käytössä on tarkoituksenmukaiset menetelmät ja vaatimukset saadaan kuvattua sopivassa muodossa. GUIDeen on kehitetty toimintatavat, joilla riittävän täsmälliset vaatimukset saadaan selville projektin alussa, ja ratkaisuja siihen, kuka vaatimukset selvittää, missä muodossa niistä kannattaa kommunikoida asiakkaan ja käyttäjien kanssa sekä miten niiden oikeellisuus testataan ennen toteutusta luotettavammilla menetelmillä kuin kysymällä asiakkaan tai käyttäjien edustajien mielipiteitä.

Luvussa 3.1 annetaan yleiskuva GUIDen vaiheistuksesta. Vaiheiden sisällöt kuvataan yksityiskohtaisemmin luvuissa 3.2-3.5.

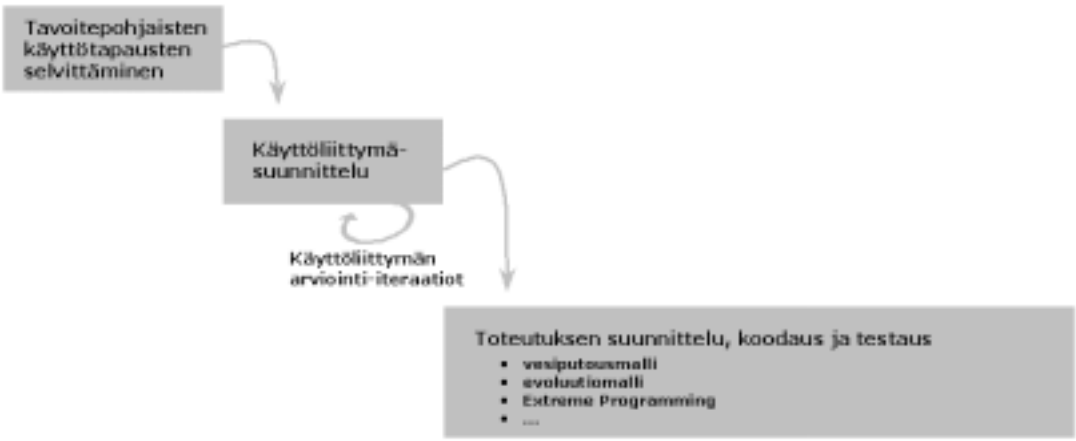
3.1 GUIDen vaiheet

GUIDe on käyttäjien työtehtäviä kuvaaviin tavoitepohjaisiin käyttötapauksiin (goal-based use cases) ja käyttöliittymäsuunnitteluun (goal-derived design) perustuva prosessimalli, jossa käyttöliittymä suunnitellaan heti projektin alussa vaatimukseksi toteutukselle. GUIDe tuo vaatimusmäärittelyyn uutena osana tavoitepohjaiset käyttötapaukset, jotka on kuvattu loppukäyttäjän näkökulmasta, ja luo sillan tekstimuotoisten käyttötapauskuvausten ja näyttökuvina esitettyjen käyttöliittymäratkaisujen välille.

Käyttöliittymän suunnittelu sijoitetaan projektin alkuun, ennen toteutuksen tai esimerkiksi tietokantaratkaisujen suunnittelua, jotta käyttöliittymäratkaisuja ja järjestelmän soveltuvuutta käyttötarkoituksiinsa voitaisiin testata niin varhaisessa vaiheessa, että testitulosten vaatimien suurtenkin muutosten tekeminen olisi vielä helppoa ja nopeaa. Käyttöliittymäkuvaus myös konkretisoi vaatimusmäärittelyä ja toimii yhtenä vaatimuksena toteutuksen suunnittelulle.

GUIDen vaiheistuksen keskeisimmät kohdat on kiteytetty kuvaan 3.1. Heti projektin vaatimusmäärittelyvaiheessa käyttöliittymäsuunnittelijat selvittävät käyttäjien työnkulut ja niistä laaditut tavoitepohjaiset käyttötapaukset ensisijaisesti kenttätutkimusten avulla. Sen jälkeen he johtavat käyttäjien tärkeimmistä työtehtävistä ja tavoitteista käyttöliittymäratakisun, jota testataan sopivilla arviointimenetelmillä, tyypillisimmin käyttäjien läpikäyntipalavereilla (walkthroughs) ja simulointipohjaisilla asiantuntija-arvioilla. Kun käyttöliittymäkuvaus on valmistunut ja testattu, siirrytään toteutuksen suunnitteluun. Toteutusvaiheessa voidaan edetä tapauksesta riippuen vesiputoustyyppisen lähestymistavan sijaan esimerkiksi

prototyyppimallin (prototyping model) tai XP:n kaltaisten evoluutiosyklien mukaan.



Kuva 3.1. GUIDe-mallin keskeisimmät vaiheet.

3.2 Tavoitepohjaisten käyttötapausten selvittäminen

Projektin alussa käyttöliittymäsuunnittelijat selvittävät käyttäjien konkreettisia työnkulkuja ja tavoitteita tyypillisesti kenttätutkimusten avulla. Tässä vaiheessa ei ryhdytä laatimaan luetteloita järjestelmän toiminnoista, vaan tarvittava toiminnallisuus selviää myöhemmin käyttöliittymästä.

Kenttätutkimusvaiheessa tehdään tyypillisesti kontekstuaalisia käyttäjähaastatteluja (contextual interviews) ja käyttäjätarkkailuja (user observations), jotka paljastavat käyttäjien olemassa olevia työnkulkuja, työtehtäviä ja niiden takana olevia tavoitteita. Selville saaduista työnkuluista irrotetaan tavoitepohjaisia käyttötapauksia, koska pelkkien skenaariomuotoisten työnkulkukuvausten avulla on vaikeaa päästä irti nykykäytännöistä ja suunnitella samojen tavoitteiden saavuttamiseksi tehokkaampia työtapoja ja käyttöliittymäratkaisuja.

Skenaarioiden takaa on osattava katsoa riittävän (muttei liian) korkean tason tavoitteita, jotka ovat juuri ja juuri suunniteltavan järjestelmän toimintojen yläpuolella. Kuvassa 3.2 on esimerkki tavoitepohjaisesta käyttötapauksesta, joka on laadittu kirjastojärjestelmän käyttöliittymäsuunnittelua varten. Käyttötapauksessa ei jäädä kirjojen hakemisen ja varausten tekemisen tasolle, vaan etsitään järjestelmän yläpuolella oleva konkreettinen tavoite.

Käyttötapaus: Cardin visualisointikirja yliopiston muissa kirjastoissa

Tutkijan tavoite: Hannu Toivonen tietää, että Cardin kirjassa olisi hyviä glyyfiesimerkkejä hänen Tutkimustiedonhallinnan peruskurssin luentoaan varten, mutta hänellä ei ole Cardin kirjaa.

Tilatietoja:

Ajankohdat

- Tänään on ma 1.9. klo 9.30.
- Luento on ti 9.9. klo 10-12.

Tarvittava kirja

- Hannu on aiemmin lukenut kirjaa, mutta hänellä ei ole omaa kappaletta.
- Hannu muistaa, että kirjan nimi on jokin ’Information Visualization’ ja yksi tekijöistä on Card.

- Kirjan todelliset lähdetiedot: Card Stuart, MacKinlay Jock, Shneiderman Ben, Readings in Information Visualization: Using Vision to Think.

Kirjan saatavuus

- Omassa TKTL:n lähikirjastossa on 3 kpl, kaikki lainassa.
- Lääketieteellisen kirjastossa on 3 kpl: lainassa 2, saatavilla 1. Hannu ei ole aiemmin käynyt siellä, ei tiedä kirjaston sijaintia.
- Kasvatustieteellisen ja psykologian kirjastoissa on 2 kpl, kaikki lainassa.

Kuva 3.2. Esimerkki tavoitepohjaisesta käyttötapauskuvauksesta (lyhennelmä).

Tavoitepohjaisia käyttötapauksia laadittaessa saattaa aluksi vaikuttaa siltä, että näin konkreettisten käyttötapausten määrittely on turhaa, koska erilaisia käyttötapauksia näyttää löytyvän kymmeniä tai satoja. Todellisuudessa kuitenkin käyttötapaukset kategorisoituvat hämmästyttävän pieneen edustavaan otokseen, jolla on yllättävän suuri kattavuus. Esimerkiksi elokuvalippujen varausjärjestelmän tai pääkaupunkiseudun julkisen liikenteen reittihakujärjestelmän suunnittelussa noin 3-6 eri kategoriaan sijoittuvaa käyttötapausta vaikuttaa olevan kattava määrä järjestelmän toiminnallisuuden ja käyttöliittymän suunnittelua varten. Projektin alussa kuitenkin on selvitettävä huomattavasti suurempi määrä käyttötapauksia, koska monet niistä osuvat samaan kategoriaan eli paljastuvat lähemmän tarkastelun jälkeen saman käyttötapauksen variaatioiksi.

3.3 Käyttöliittymäsuunnittelu

Käyttötapauskuvausten laatimisen jälkeen työnkulkuja ja yrityksen liiketoimintaprosesseja suoraviivaistetaan tarvittaessa. Esimerkiksi kuvassa 3.2 esitetyn käyttötapautapauksen tukemiseksi pystymme oikaisemaan työnkulkua viimeistään käyttöliittymän suunnitteluvaiheessa. Nykyisen työnkulun mukaan Hannu voi saavuttaa tavoitteensa varaamalla tavoittelemansa kirjan Lääketieteellisen tiedekunnan kirjastosta ja noutamalla sen myöhemmin itse sieltä. Voimme kuitenkin suoraviivaistaa tavoitteen saavuttamista muuttamalla työnkulkua siten, että Hannu voi tällaisessa tilanteessa kirjan varaamisen ja noutamisen sijaan tilata sen yliopiston sisäpostissa. Siten järjestelmään ei tarvittukaan varaustoimintoa, vaan mahdollisuus sisäpostitilauksiin ja tieto arvioidusta toimitusajasta. Uusi ratkaisu ja sen toteuttaminen saattaa olla kustannuksiltaan jopa edullisempi kuin aiempi. Joissain tapauksissa käy niin, että prosessien suoraviivaistamisen tuloksena joitain järjestelmän osia ei tarvita lainkaan tai jopa kokonainen järjestelmä saatetaan todeta tarpeettomaksi ja jättää toteuttamatta.

Käyttöliittymän suunnitteluprosessi alkaa siten, että käyttöliittymäsuunnittelija valitsee yhden työtehtävän (korkean tason käyttötapauksen) ja laatii sen suorittamiseksi tarvittavat toiminnot ja niiden käyttöliittymäratkaisut, mutta välttää tarkasti lisäämästä järjestelmään yhtään sellaista toimintoa tai ylimääräistä tietoa, jota ei tarvita juuri tämän käyttötapauksen suorittamisessa. Seuraavaksi hän integroi käyttöliittymään vähitellen tukea yhä uusille työtehtäville. Suunnittelun edetessä hän jatkuvasti simuloi käyttöliittymän toimintaa suorittamalla yhden työtehtävän kerrallaan. Näin testataan, etteivät uudet mukaan otetut työtehtävät ole hankaloittaneet aiemmin suunniteltujen töiden suorittamista.

Suunniteltavaan järjestelmään ei voi syntyä toimintoja (features), joita ei tarvita käyttötapausten suorittamisessa, koska jokaisen käyttötapauksen kohdalla laaditaan vain ko. tavoitetta palvelevat toiminnot ja niiden käyttöliittymäratkaisut. Jos suunnitteluryhmälle tai asiakkaalle tulee projektin

aikana mieleen muita potentiaalisia hyödyllisiä toimintoja, ne asetetaan käyttöliittymäsuunnittelijoiden tutkittavaksi: onko löydettävissä jokin huomiotta jätetty tavoitepohjainen käyttötapaus, jossa tätä uutta toimintoa tarvittaisiin? Jos tarvetta toiminnon käytölle ei ole, se dokumentoidaan tarpeettomaksi, kunnes joku pystyy osoittamaan käyttötilanteen, jossa ko. toiminto tuo lisäarvoa käyttäjälle. Jos taas huomiotta jätetty käyttötapaus löytyy, se lisätään vaatimusmäärittelyyn ja otetaan suunnitteluun mukaan, mistä tyypillisesti seuraa käyttöliittymään muitakin muutoksia kuin uuden ehdotetun toiminnon lisääminen.

GUIDen käyttöliittymäsuunnitteluprosessi muistuttaa pitkälle XP:n toteutusiteraatioita [Beck00], joiden lähtökohtana olevat asiakkaiden laatimat käyttäjätarinat (user stories) on korvattu käyttöliittymäasiantuntijoiden laatimilla tavoitepohjaisilla käyttötapauksilla. Käyttöliittymäsuunnittelu tehdään parityönä (vrt. XP:n pariohjelmointi), jossa käyttötapauksen vaatima toiminnallisuus lisätään aina edellisen ratkaisun päälle tehden samalla vaadittavat muutokset olemassa olevaan ratkaisuun. Inkrementaalisesti syntyvää käyttöliittymäratkaisua korjataan tarvittaessa paljonkin prosessin aikana (refaktorointi). Käyttötapaukset on kirjoitettu etukäteen (XP:n yksikkötestit), ja kaikki testit ajetaan jokaista uutta käyttötapausta vastaavan toiminnallisuuden lisäämisen jälkeen. Erona on se, että käyttöliittymäsuunnittelussa kehityssyklit ovat vielä lyhyempiä kuin XP:ssä, ja vastuu käyttötapauksista on asiantuntijoilla, ei asiakkaalla, vaikka asiakas viime kädessä hyväksyy käyttötapauskuvaukset ja niiden priorisoinnin.

Kun koko käyttöliittymän ensimmäinen versio on saatu suunniteltua kokonaan, suunnittelija laatii käyttöliittymäkuvauksen (user interface specification), jossa näytetään yksityiskohtaisesti kuvasarjojen avulla, kuinka käyttäjä vaihe vaiheelta kulkee kohti tavoitettaan eli saa työtehtävänsä tehdyksi. Sekä suunnitteluprosessin aikaisia kehitysversioita että lopullista käyttöliittymäkuvausta voidaan testata käyttäjillä.

3.4 Käyttöliittymäratkaisun arviointi

Kun käyttöliittymäratkaisu on ensin suunniteltu luvussa 3.3 kuvatun inkrementaalisen kehitysprosessin avulla kokonaiseksi versioksi, käyttöliittymäratkaisuja arvioidaan ikään kuin järjestelmätestauksen tasolla ennen toteutuksen suunnitteluun siirtymistä. Käyttöliittymää kannattaa ryhtyä arvioimaan varmistamalla ensin, että käyttöliittymässä ylipäätään on kaikki työtehtävien tekemisessä tarvittava data ja toiminnot (utility) ja testaamalla käyttöliittymäratkaisujen tehokkuus (efficiency). Vasta kun järjestelmän toiminnallisuus- ja tehokkuusnäkökohdat ovat kunnossa, kannattaa ryhtyä korjaamaan mahdollisia opittavuuteen ja intuitiivisuuteen liittyviä ongelmia, koska opittavuutta on helpompi lisätä oikean toiminnallisuuden ja tehokkaan käyttöliittymäratkaisun päälle kuin päinvastoin.

Puuttuvaa toiminnallisuutta sekä tehokkuusongelmia saadaan kustannustehokkaasti esille käyttäjien kanssa tehtävien *käyttöliittymän läpikäyntipalaverien* (walkthroughs) [Bias91] avulla, joita kannattaa tehdä jo käyttöliittymän suunnitteluvaiheessa ensimmäiselle kokonaiselle käyttöliittymäluonnokselle. Läpikäyntipalavereissa on tyypillisesti samanaikaisesti paikalla pari käyttäjää sekä 1-2 käyttöliittymäasiantuntijaa, joista toinen ohjailee läpikäyntiä palautteen keräämisen kannalta tarkoituksenmukaisiin suuntiin. Läpikäyntipalaverin aikana tulee hyvin esille puutteita ja väärinkäsityksiä työtehtävien ymmärtämisessä sekä mm. sellaisia poikkeustapauksia, joihin ei olla aiemmissa kenttätutkimuksissa osuttu.

Opittavuusongelmiin ei näissä läpikäyntipalavereissa kannata kovin paljon painottua.

Toinen hyvä tapa selvittää toiminnallisuus- ja tehokkuusongelmia on työtehtävien suorittamisen *simulointiin perustuva asiantuntija-arvio* (expert review, usage simulations). Koska simulointitestaukseen ei kuitenkaan osallistu loppukäyttäjiä lainkaan, sen avulla ei saada kiinni kovin paljon työtehtäviin ja niiden kautta järjestelmän toiminnallisuuteen ja tietosisältöön liittyviä puutteita. Simuloinnin aikana kyllä nousee esille lukuisia työtehtävien suorittamiseen liittyviä avoimia kysymyksiä, joiden kautta päästään välillisesti käsiksi myös toiminto- ja tietosisältöpuutteisiin. Avoimia kysymyksiä voidaan selvittää käyttäjiltä asiantuntija-arvioin tekemisen jälkeen esimerkiksi puhelimitse.

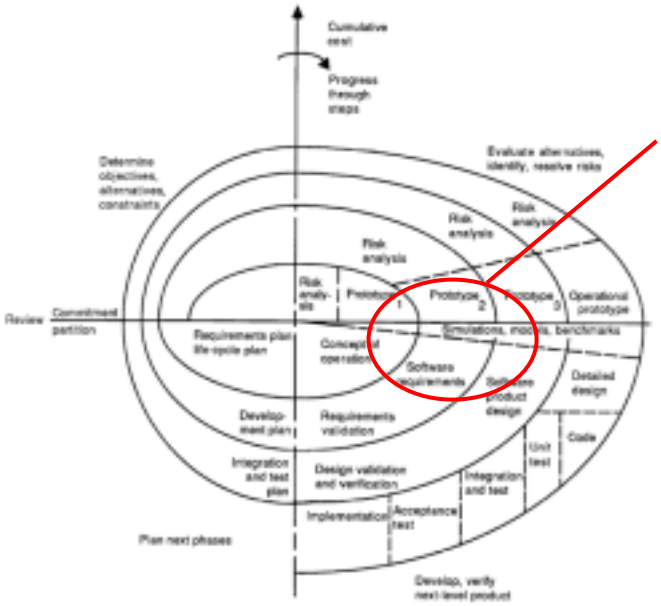
Käyttöliittymäratkaisujen opittavuuden ja intuitiivisuuden ennustaminen ilman käyttäjiä on monesti vaikeaa; esimerkiksi hyvin pienet erot kohteiden sijoittelussa, visuaalisissa yksityiskohdissa ja sanavalinnoissa voivat muuttaa käyttäjien tekemiä tulkintoja yllättävällä tavalla. Opittavuusongelmia voidaan lopuksi selvittää uusilla käyttöliittymän läpikäyntipalavereilla tai *käytettävyystesteillä* (usability tests) [Nielsen93, luku 6], joissa yksi käyttäjä kerrallaan tekee testin ohjaajan antamia testitehtäviä.

Jokaisen arviointikierroksen jälkeen löytyneet ongelmat korjataan ja käyttöliittymäkuvausta päivitetään. Käyttöliittymäratkaisut ja järjestelmässä tarvittava toiminnallisuus on edelleen olemassa vain kuvina, joiden muuttaminen testitulosten perusteella on nopeaa. Tarkoituksena ei kuitenkaan ole iteroida vain iteroinnin vuoksi, vaan minimoida iterointitarve ja luoda heti ensimmäisestä käyttöliittymän versiosta niin hyvä, ettei sitä tarvitsisi kovin paljon korjata enää testauksen yhteydessä.

3.5 Toteutuksen suunnittelu, koodaus ja testaus

Projektin alussa tehtävien käyttötapausselvitysten, käyttöliittymäsuunnittelun ja käyttöliittymän testauksen jälkeen alkaa toteutusvaihe, joka sisältää ainakin toteutuksen suunnittelua, koodausta ja testausta sekä monissa tapauksissa myös mm. käsiteanalyysia ja tietokantasuunnittelua. Toteutusvaihe voidaan tehdä esimerkiksi vesiputousmallin mukaan, jos toteuttajilla on paljon kokemusta vastaavanlaisten järjestelmien tekemisestä ennestään, tai esimerkiksi evoluutiomallin tai XP-mallin mukaisissa vaiheittain toiminnallisuutta lisäävissä sykleissä.

Kaikille vaihtoehdoille yhteistä on se, että toteutusvaihe saa yhtenä vaatimuksena syötteekseen käyttöliittymäkuvauksen. Käyttöliittymäkuvauksen laatimisprosessin tarkoituksena on pudottaa mahdollisimman alas vääränlaisen käyttöliittymän tuottamiseen liittyvät riskit, jotka ovat interaktiivisten järjestelmien suunnittelussa yksi keskeisimpien riskien ryhmä [Boehm88]. Kuvaassa 3.3 on Boehmin spiraalimalli, johon on sijoitettu keskeisimmät GUIDe-menetelmät. GUIDen käyttötapausselvitysten ja käyttöliittymäsuunnittelun lisääminen projektin alkupuolelle vähentää käyttöliittymäriskejä ja vie projektin alkuosaa vesiputousmallia kohti, ja projektin loppuosa voidaan tehdä jäljellä olevista riskeistä riippuen esimerkiksi toteutustekniikkojen prototypisointiin tai vesiputousmallin suuntaan painottuen.



GUIDen käyttötapausselvitykset ja käyttöliittymäsuunnittelu

Kuva 3.3. Boehmin spiraalimalli [Boehm88].

Vaikka käyttöliittymästä olisi laadittu täsmällinen kuvaus projektin alussa, toteutuksen suunnittelun ja koodauksen kuluessa käyttöliittymäratkaisuihin joudutaan usein tekemään joitain toteutuskompromisseja. Esimerkiksi tietyt käyttäjän kannalta hyvät ratkaisut saattavat osoittautua liian vaivalloisiksi tehdä projektissa käytetyillä toteutusvälineillä, jolloin niiden käytettävyyttä kannattaa hieman pudottaa esimerkiksi toteutuksesta syntyvien säästöjen vuoksi. Kun GUIDe-mallia käyttävässä projektissa törmätään kompromissitarpeeseen, toteutuksen suunnittelija tai ohjelmoija kuvaa ongelman käyttöliittymäsuunnittelijalle ja delegoi uuden kompromissiratkaisun suunnittelemisen hänelle. Tämä työnjako varmistaa, etteivät ohjelmoijat joudu oman työnsä ohella tekemään käyttöliittymäsuunnittelua, josta heillä ei yleensä ole ennestään osaamista. Kun käyttöliittymäasiantuntija suunnittelee myös toteutuskompromissit, niistä ei pitäisi syntyä kohtuuttoman suuria uusia käytettävyysongelmia.

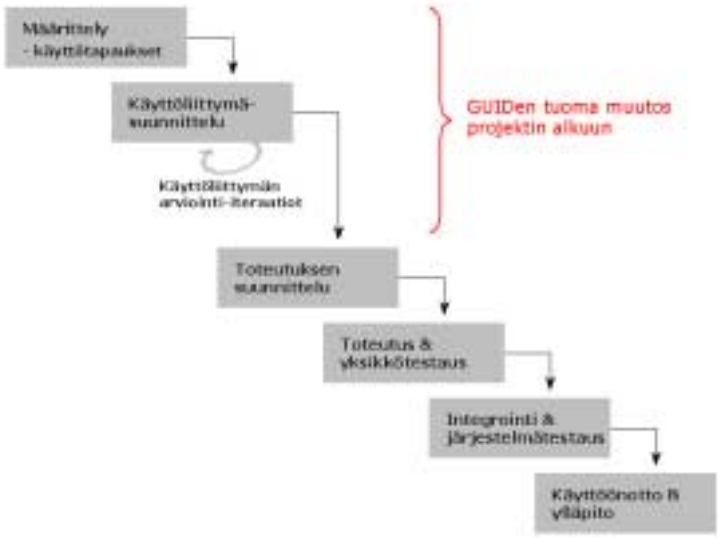
Erityisesti niissä tapauksissa, joissa toteutettava järjestelmä tulee suuren yleisön laajaan käyttöön, suosittelemme vielä lopuksi käytettävyystestaamaan toimivaa järjestelmää muutamalla testikäyttäjällä. Näin lopussa voidaan korjata joitain pieniä opittavuuteen vaikuttavia seikkoja (esimerkiksi yhteen kuuluvien komponenttien ryhmittely, otsikkotekstit), joiden korjaaminen on nopeaa mutta jotka kuitenkin voivat joissain kohdissa vaikuttaa paljon käyttäjän tulkintaan.

4 GUIDe-mallin sovittaminen projektiin

GUIDen keskeisimpänä parannuksena on projektin alkuun sijoitettava käyttöliittymäsuunnitteluvaihe, joka päättyy täsmällisen käyttöliittymäkuvauksen tuottamiseen. Luvussa 4.1 tarkastellaan tämän menettelyn suhteuttamista vesiputous- ja XP-malleihin. Luvussa 4.2 arvioidaan GUIDen tuomia etuja ja haasteita koko projektin kannalta.

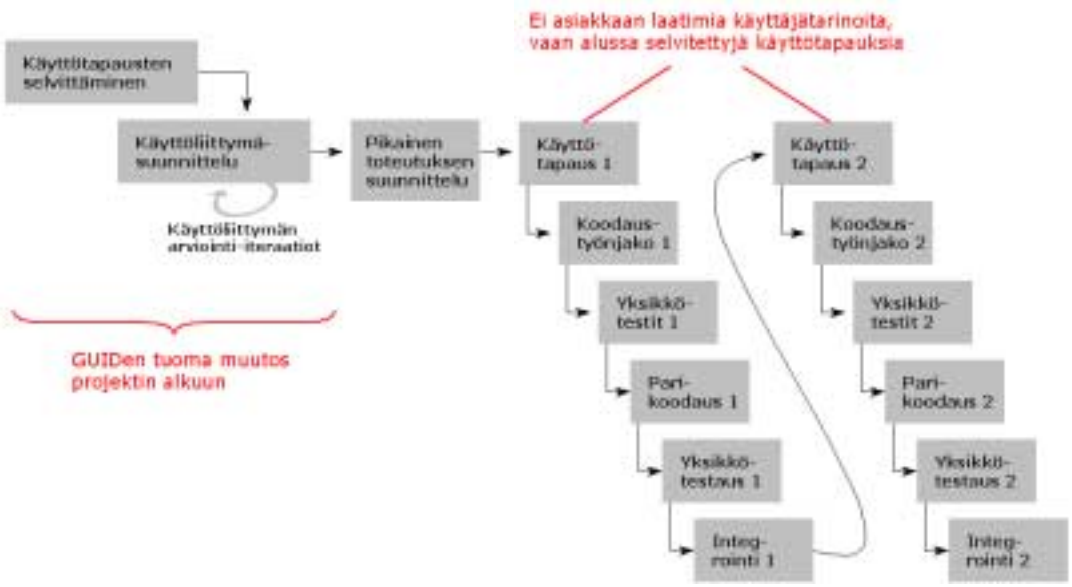
4.1 GUIDen vaiheet vesiputous- ja XP-mallin osana

Jos sijoitamme käyttöliittymäsuunnittelun ja käyttäjien työtehtävien selvittämisen omina vaiheinaan projektin alkuun, saamme kerralla ratkaistua vesiputousmallin ongelman liian myöhään esille tulevista muutostarpeista (kuva 4.1). Käyttäjien työnkuvaukset ja käyttöliittymäkuvat ovat se osa järjestelmästä, jota asiakkaat ja käyttäjät ymmärtävät, ja käyttöliittymäkuvien avulla iterointi on hyvin edullista koodin avulla iterointiin nähden. Suurimmat ja kalleimmat riskit alkavat olla vaatimusmäärittelyn ja käyttöliittymäkehityksen sijaan toteutusratkaisujen puolella.



Kuva 4.1. GUIDe vesiputousmallin osana.

GUIDe voidaan sovittaa myös inkrementaaliseen ohjelmistokehitykseen perustuviin malleihin. Kuvassa 4.2 GUIDen vaiheet on yhdistetty XP-malliin.



Kuva 4.2. GUIDe XP-mallin osana.

Kuten kuvasta 4.2 näkyy, GUIDen mukaiset käyttötapausselvitykset ja käyttöliittymäsuunnittelu on sijoitettu projektin alkuun. Lisäksi erona alkuperäiseen XP-malliin on mm. se, että vastuuta vaatimusten kartoittamisesta ei jätetä asiakkaalle, joka laatisi vaatimuskuvauksia kirjoittamalla käyttäjätarinoita (user stories), vaan käyttöliittymäasiantuntijat selvittävät loppukäyttäjien työtehtävät, työnkulut, liiketoimintaprosessit ja asiakkaan tarpeet tarkoituksenmukaisilla menetelmillä. Toki asiakkaankin kannattaa osallistua työnkulkujen selvittämiseen ja käyttötapausten priorisointiin, etenkin jos asiakkaan edustajat kuuluvat myös toteutettavan järjestelmän käyttäjäjoukkoon, mutta vastuuta työnkulkujen selvittämisestä ei jätetä asiakkaan varaan. Järjestelmän tilaajalla ei tyypillisesti ole osaamista selvitysmenetelmien käytöstä eikä välttämättä resurssejakaan selvitysten tekemiseen.

Kun projektissa siirrytään XP:n mukaisiin testaus- ja toteutussykleihin, järjestelmää lähdetään toteuttamaan yksi käyttötapaus tai käyttötapauksen osa kerrallaan. Pääsääntöisesti korkeimmalle priorisoiduissa käyttötapauksissa tarvittava toiminnallisuus koodataan ensin, jotta järjestelmällä voitaisiin mahdollisimman varhaisessa vaiheessa saada aikaan hyödyllisiä työtehtäviä tai niiden osia.

GUIDen erona XP:hen on se, että GUIDe yrittää pudottaa puuttuvien vaatimusten esiintymistodennäköisyyden mahdollisimman alas käyttöliittymäsuunnittelun avulla, kun taas XP:ssä lähdetään siitä, ettei ongelmaa voida välttää eikä vaatimuksia kuitenkaan voida saada etukäteen selville, joten rakennetaan koko projekti varautumaan muuttuviin vaatimuksiin. Sijoittamalla GUIDe-menetelmät XP-mallin alussa olevan lyhyen suunnitteluvaiheen (Planning Game) alkuun saamme minimoitua projektin aikana tulevien muutosten riskiä ja parannettua käyttöliittymän laatua. Silti toteutus voi monissa tapauksissa olla kustannustehokkainta XP:n periaatteita käyttäen.

4.2 GUIDen tuomia etuja ja vaatimuksia

GUIDen suurimpiin etuihin sisältyy se, että GUIDe antaa systemaattisen menettelyn järjestelmässä tarvittavien toiminto- ja tietosisältömäärittelyjen tuottamiseksi. Lisäksi järjestelmän käyttöliittymän suunnittelu tehdään niin varhaisessa vaiheessa, että käyttöliittymäratkaisuja voidaan testata silloin, kun korjausten tekeminen on vielä edullista. Käyttöliittymän arvioinnissa käytettävät käyttäjäläpikäynnit ja käyttötapaussimuloinnit ovat huomattavasti luotettavampia menetelmiä kuin esimerkiksi asiakkailta kysytyt mielipiteet.

Käyttäjät ja asiakkaatkin osaavat erittäin hyvin ottaa kantaa käyttöliittymäsuunnittelun lähtökohdaksi laadittaviin tavoitepohjaisiin käyttötapauksiin, koska ne kuvaavat käyttäjien jokapäiväistä työtä, jonka ymmärtämisessä käyttäjät ovat parhaita asiantuntijoita. Työnkulkuja voidaan suoraviivaistaa ennen käyttöliittymäsuunnittelun aloittamista, millä voi olla lopputulokseen paljon suurempia vaikutuksia kuin käyttöliittymä- tai toteutusratkaisuilla. Joissain tapauksissa tarpeettomia vaiheita ja jopa kokonaisia tietokonejärjestelmiä pystytään poistamaan työnkulusta kokonaan.

Täsmällisen käyttöliittymäkuvauksen ansiosta myös kommunikointi asiakkaan ja loppukäyttäjien kanssa tehostuu, kun he näkevät kuvasarjoista, miltä valmis järjestelmä näyttää ja miten se toimii. Asiakkaan edustajat pystyvät paremmin ottamaan kantaa konkreettiseen käyttöliittymäkuvaukseen kuin laajoihin kirjallisesti esitettyihin vaatimusluetteloihin, joiden kattavuutta on vaikea arvioida ja varmistaa.

GUIDen käytön edellytyksenä on se, että projektissa pitäisi olla tavoitepohjaisten käyttötapausten selvittämisen ja käyttöliittymäsuunnittelun osaamista. Lisäksi käyttöliittymäratkaisuja pitäisi osata arvioida ainakin jollain arviointimenetelmällä. Jos käyttöliittymäsuunnittelun osaamista ei ole riittävästi, käyttöliittymäsuunnittelun sijoittaminen projektin alkuun ei välttämättä vähennä käyttöliittymäriskejä riittävästi. Tästä huolimatta interaktiivisten järjestelmien kehitysprojekteissa kannattanee käyttää GUIDen lähestymistapaa, jossa käyttöliittymäkuvaus laaditaan hyvin varhaisessa vaiheessa, koska koodaajien tuottamat käyttöliittymäratkaisut eivät hyvin todennäköisesti ole ainakaan parempia kuin alussa laadittu käyttöliittymäkuvaus, jota testataan ennen koodausta edes jollain arviointimenetelmällä.

Eräänä GUIDen haittapuolena on käyttöliittymäkehityksen sijoittuminen koko projektin kriittiselle polulle. Tätä ongelmaa voidaan lieventää laatimalla käyttöliittymäsuunnittelun rinnalla toteutuskokeiluja ja protoja kriittisiksi tiedetyistä tai huonosti ymmärretyistä, riskialttiista toteutuskohdista.

GUIDessa riskinä on myös se, että joitain keskeisiä mutta harvoin esiintyviä käyttötapauksia voi jäädä kokonaan huomaamatta projektin alkupuolella, minkä seurauksena käyttöliittymä ei tue niitä ollenkaan. Tällöin ongelma tulee esille vasta ohjelman käyttöönoton jälkeen, kun käyttäjät yrittävät suorittaa valmiilla ohjelmalla todellisia käyttötapauksiaan. Toisaalta sama riski esiintyy missä tahansa vaatimusmäärittelyyn nojaavassa projektissa paljon suurempana. GUIDen käyttöliittymäsuunnitteluvaihe nimenomaan pyrkii minimoimaan tämän riskin toteutumisen todennäköisyyttä verrattuna esimerkiksi vesiputousmalliin kuuluvaan vaatimusmäärittelyyn.

Evoluutiosykleihin perustuvissa projekteissa huomaamatta jääneen käyttötapauksen ongelma saataisiin selville ennen käyttöönottoa vain, jos

puuttuva vaatimus tulisi jollekin osapuolelle mieleen kesken kehityksen tai jos se tulisi eteen jonkin kehityssyklin jälkeen käyttöönotetussa versiossa. GUIDessa puute pyritään saamaan esille demoamalla koko järjestelmän pienoismallia käyttöliittymäkuvina asiakkaalle ja testaamalla käyttöliittymäkuvausta myös loppukäyttäjien avulla jo projektin alussa. Tämän ansiosta valtaosan mahdollisista puutteista pitäisi tulla esille ennen toteutusta. Jos projektiryhmän osaaminen on käyttötapausten selvitysmenetelmien, käyttöliittymäsuunnittelun tai käyttöliittymän arviointimenetelmien osalta heikkoa, käyttöliittymäriskejä kannattaa vähentää ensin GUIDe-menettelyn mukaisesti, minkä jälkeen voidaan turvautua käyttöliittymäkuvauksen pohjalta suunniteltuun XP-mallin mukaiseen toteutusiterointiin. Näin GUIDen käyttö vähentää puuttuvien vaatimusten riskiä alussa ja parantaa käyttöliittymän laatua, mutta XP:n toteutusiteraatioiden avulla saadaan kiinni loputkin puutteet mahdollisimman varhaisessa vaiheessa.

Vastuu vaatimusten kattavuudesta ja projektissa syntyvien käyttöliittymäratkaisujen arvioinnista on yksinkertaista jättää asiakkaalle, kuten esimerkiksi XP:ssä tehdään, mutta haittapuolena on se, että tällä tavalla ei pystytä varmistamaan hyvää käyttöliittymää ja tarkoituksenmukaista toiminnallisuutta. Asiakkaalla ei tyypillisesti ole käyttöliittymän arviointi- saati suunnittelumenetelmien osaamista, joten hän ei pysty asiantuntevasti sanomaan, mikä käyttöliittymäratkaisu on hyvä ja mikä ei. Asiakas ei myöskään voi tietää, millaisia vaatimuksia käyttöliittymäsuunnittelua varten olisi tarpeen selvittää, eikä hän tyypillisesti ennestään osaa eikä ehdi itse ryhtyä tekemään käyttötapausselvityksiä. Sen sijaan asiakkaat ja etenkin loppukäyttäjät kyllä ovat parhaita asiantuntijoita järjestelmän käyttötarkoitusten ja niihin liittyvien työnkulkujen ja tavoitteiden tuntemisessa. GUIDe hyödyntää heidän asiantuntemustaan juuri näillä alueilla muttei yritä työntää vastuuta heille sellaisista tehtävistä, joista heillä ei voi olla osaamista.

5 Yhteenveto

Monissa nykyään käytössä olevissa prosessimalleissa, kuten vesiputous- tai evoluutiomallissa, käyttöliittymäkehitystä ei ole huomioitu lainkaan tai juuri lainkaan. Tämän seurauksena näiden mallien pohjalta suunnitelluissa interaktiivisissa järjestelmissä on paitsi tehottomia ja vaikeaselkoisia käyttöliittymäratkaisuja, myös puutteita työtehtävien suorittamisessa tarvittavassa toiminnallisuudessa ja tietosisällössä. Järjestelmiä ei ole suunniteltu tukemaan käyttäjiensä työnkulkuja, vaan soveltuminen työtehtävien suorittamiseen testataan pahimmassa tapauksessa vasta käyttöönoton jälkeen kentällä.

Joidenkin prosessimallien, kuten XP:n, kehittämisen eräänä lähtökohtana on oletus siitä, ettei vaatimuksia voida saada projektin alussa selville riittävän kattavasti, koska asiakas ei kuitenkaan osaa kertoa, mitä hän haluaa. Siten järjestelmää ryhdytään kehittämään paloittain ja asiakkaalta pyydetään palautetta ja muutospyyntöjä kehityksen aikana. Tämäkään menettely ei kuitenkaan juurikaan vähennä järjestelmän käyttöliittymäongelmia, koska koodaajat eivät yleensä osaa suunnitella hyviä käyttöliittymäratkaisuja eikä asiakkailla ole osaamista niiden arviointimenetelmistä, vaan käyttöliittymäkorjaukset yleensä perustuvat pelkkiin asiakkaiden esittämiin mielipiteisiin. Lisäksi näissä malleissa asiakkaalta tulevien muutostarpeiden vaatima iterointi tehdään koodatun ohjelman tai prototyypin avulla, mistä seuraa tarpeetonta koodaustyötä.

Puutteellisista vaatimuksista ja asiakkaan muutospyynnöistä syntyvien ongelmien hyväksymisen sijaan GUIDe-mallissa pyritään ratkaisemaan nämä ongelmat. GUIDessa järjestelmän käyttäjälle näkyvän toiminnan tarkoituksenmukaisuus (käyttöliittymä) on todettu tyypillisimmäksi ja suurimmaksi riskiksi interaktiivisen ohjelman kehitysprosessissa, minkä seurauksena on kehitetty systemaattinen menettely tämän riskin eliminoimiseksi. Projektin alkuun on sijoitettu käyttöliittymäsuunnitteluvaihe, jossa kenttätutkimuksen avulla selvitetyistä tavoitepohjaisista käyttötapauksista johdetaan tarvittavat toiminnot ja käyttöliittymäratkaisut sisältävä käyttöliittymäkuvaus. Käyttöliittymäkuvauksen avulla voidaan jo ennen toteutusratkaisujen suunnittelua testata, tukeeko kuvattu järjestelmä käyttäjiensä työtehtävien tekemistä:

- onko käyttöliittymässä juuri se toiminnallisuus ja tietosisältö, jota käyttötilanteissa tarvitaan (utility) ja
- ovatko käyttöliittymäratkaisut suoraviivaisia ja tehokkaita sekä yksikäsitteisen ymmärrettäviä (usability).

Lähteet

Beck00	Beck K., Extreme programming explained: embrace change. Addison-Wesley, Reading, MA, 2000.
Bias91	Bias R., Walkthroughs: Efficient Collaborative Testing. IEEE Software, Vol. 8, Nro. 5, 1991, s. 94-95. [pdf]
Boehm81	Boehm B. W., Software Engineering Economics. Prentice Hall, 1981.
Boehm88	Boehm B. W., A Spiral Model of Software Development and Enhancement. IEEE Computer, Vol. 21, No. 5, May 1988, s. 61-72. [pdf]
Nielsen93	Nielsen J., Usability Engineering. Academic Press, San Diego, CA, 1993.

Päivitetty 3.10.2004 / Sari A. Laakso, sähköposti: salaakso@cs.helsinki.fi