Käyttötapauspohjainen testaaminen

Niina Jormanainen

21.8.2006

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Käyttötapaukset ovat hyvin yleisesti käytettyjä järjestelmän vaatimusten kuvaamiseksi käyttäjien näkökulmasta. Koska käyttötapaukset sisältävät järjestelmän toiminnallisuuden ja vaatimusten kuvaukset, on hyvin luonnollista kehittää järjestelmän toiminnalliset testitapaukset käyttötapausten pohjalta. Tässä tutkielmassa esitellään, kuinka RSI-mallin mukaisia käyttötapauksia voitaisiin hyödyntää sekä toiminnallisessa testaamisessa että käytettävyystestauksessa. Käytettävyystestauksen suorittamiseksi tarvitaan myös käyttäjäkeskeistä kehittämistä, joka pyrkii huomioimaan määriteltyjen käyttäjien määritetyssä ympäristössä suorittamaan toimintaan liittyvät ei-toiminnalliset vaatimukset. Tämän seurauksena tässä tutkielmassa esitellään prosessimalli, joka yhdistää käyttötapauspohjaisen kehittämisen ja käyttäjäkeskeisen kehittämisen RSI-käyttötapausten avulla, sekä kuvaa kuinka käyttötapauksia hyödynnetään kehitettävän järjestelmän testaamisessa.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.1, D.2.2, H.5.2, K.6.3
Avainsanat: Käyttötapauspohjainen kehittäminen, käyttäjäkeskeinen kehittäminen, käyttötapauspohjainen testaaminen.

Abstract

Use-cases are a widely used methodology for documenting system requirements from a user's point of view. Use-cases provide descriptions for a system's behaviour and requirements, and as a result it is natural to develop functional test-cases, based on the use cases. In this thesis I will present how RSI-use cases could be exploited in both functional testing and usability testing. For conducting usability tests, we need user-centred design which will try to take notice of the non-functional requirements for specific users in a specific context while doing their job. Thus, I present a process model which will combine the use-case-based design and user-centred design using RSI-use-cases. The model also shows how use-cases are exploited in testing.

ACM-Classes (ACM Computing Classification System, 1998 version): D.2.1, D.2.2, H.5.2, K.6.3
Keywords: Use-case-based design, user-centred-design, use-case-based testing.

Sisältö

1. Johdanto

Kehitettäessä tietojärjestelmää tarkoituksena on yleensä kehittää järjestelmä, joka auttaa käyttäjiään suorittamaan jonkin tehtävän. Järjestelmän tulee vastata johonkin käyttäjän tarpeeseen, joko mahdollistamalla uudenlaisen tehtävän suorittamisen tai jollakin tavalla helpottaa, nopeuttaa tai automatisoida jotakin käyttäjän vanhaa tehtävää. Järjestelmän käyttäjät asettavat siis vaatimuksia sille, mitä toimintoja järjestelmän tulisi sisältää, ja miten sen tulisi toimia. Järjestelmän kehittäjien tulee ottaa nämä vaatimukset huomioon järjestelmän kehitystyössä, jotta järjestelmä tuottaisi käyttäjilleen jotakin arvoa.

Oikeiden toimintojen lisäksi kehittäjien tulee kiinnittää erityistä huomiota järjestelmän käyttöliittymän kehittämiseen, sillä loppukäyttäjille käyttöliittymä on ohjelman ainoa konkreettinen ja näkyvä osa, vaikka se tuotteen kehittäjille on vain yksi osa koko järjestelmää ja sen kehittämistä. Jos käyttäjä kokee ohjelman käyttöliittymän huonoksi, epäkäytännölliseksi tai virheelliseksi, hän ajattelee koko ohjelman olevan sitä. Vaikka järjestelmä sisältäisikin markkinoiden optimoiduimman asiakasrekisterin hakualgoritmin, mutta sen visuaalinen esitys ei ole käyttäjien mielestä käytettävä, eivät he välttämättä koe hakutoimintoakaan hyödylliseksi.

Kun järjestelmä kehitetään yhdessä käyttäjien kanssa niin, että järjestelmä toteuttaa käyttäjien sille asettamat vaatimukset, on todennäköisempää, että käyttäjät ovat yleisesti tyytyväisempiä järjestelmään ja projektin riski epäonnistua pienenee. Lisääntyneen asiakastyytyväisyyden myötä järjestelmä voi saada markkinaedun muihin kilpailijoihin nähden. Myös käyttötuesta ja koulutuksesta aiheutuvat kustannukset asiakkaille ja kehittäjälle pienenevät, kun järjestelmä toimii käyttäjien odottamalla tavalla.

Jotta kehitettävä järjestelmä toimisi käyttäjien haluamalla tavalla, ja että se sisältäisi juuri ne toiminnot juuri sellaisina kuin käyttäjät ne haluavat, tulee järjestelmän kehittäjien kerätä tietoa käyttäjistä ja heidän vaatimuksistaan. Ketkä käyttävät kehitettävää järjestelmää? Millaisia he ovat? Mitä he haluavat järjestelmän avulla saavuttaa? Millaisessa ympäristössä he käyttävät järjestelmää? Miten he käyttävät sitä? Mikä on heidän sovellusalueen tietämys? Millainen heidän yleinen tietokoneiden käyttötaito on? Muun muassa näihin kysymyksiin järjestelmän kehittäjien tulee

löytää vastaukset ja soveltaa niitä järjestelmän kehittämiseksi, jotta järjestelmästä tulisi käyttäjien mielestä tarpeellinen, hyödyllinen ja käytettävä.

Käyttäjien vaatimusten keräämiseen ja tarpeiden selvittämiseen on olemassa hyvin monenlaisia menetelmiä, joista yksi menetelmä on järjestelmän kuvaaminen käyttötapausten avulla. Käyttötapaukset ovat yleensä järjestelmän kehittäjien ja käyttäjien yhdessä muodostamia kuvauksia kehitettävästä järjestelmästä ja sen toiminnasta. Käyttötapaukset ovat kehittäjien ja käyttäjien yhteinen kieli vaatimusten dokumentoimiseen, jonka jälkeen kehittäjät rakentavat järjestelmän käyttötapausten pohjalta. Käyttötapausten luomisessa voidaan käyttää perinteisiä käyttäjäkeskeisen kehittämisen menetelmiä käyttäjien vaatimusten ja tarpeiden selvittämiseksi, kuten muun muassa käyttäjien työskentelyn havainnointi heidän aidossa työympäristössään sekä loppukäyttäjille tehtävät kyselyt ja haastattelut.

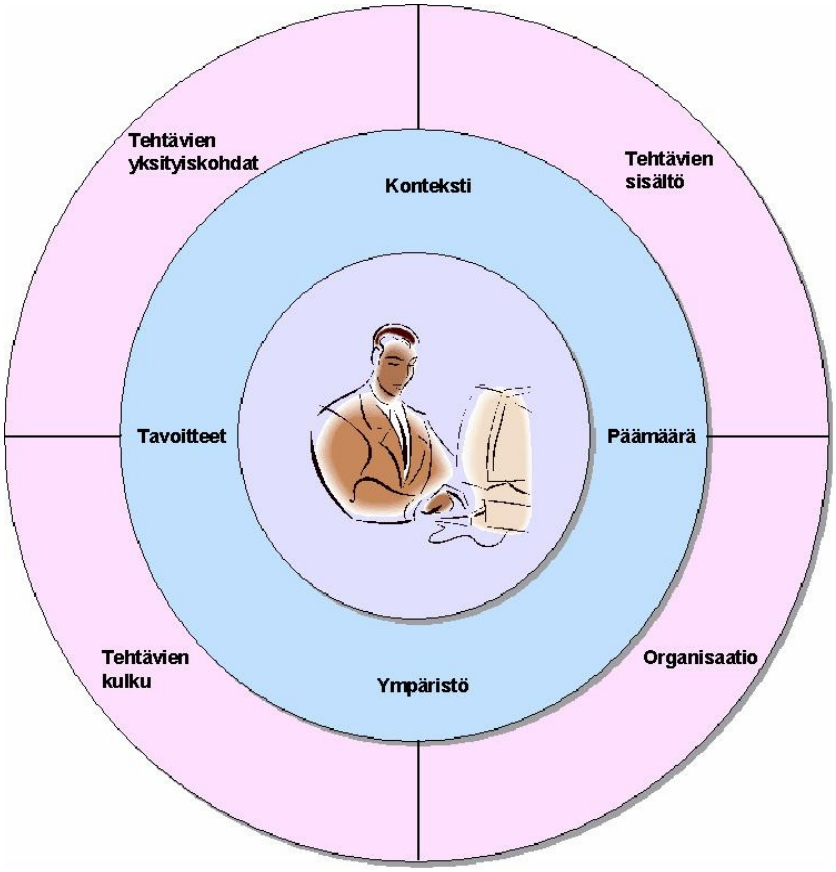
Järjestelmän testaamisen avulla varmistetaan, että kehitetty järjestelmä todella vastaa käyttäjien asettamia vaatimuksia. Lisäksi testaamisen avulla pyritään löytämään mahdollisimman suuri osa järjestelmän sisältämistä virheistä. Testaaminen voi perinteisen ohjelmakoodin testaamisen lisäksi kohdistua myös esimerkiksi ohjelmistotuotteen dokumentteihin ja käytettävyyteen. Riippuen tuotteen kehityksessä käytetystä elinkaarimallista, testaaminen voi tapahtua aivan kehitysprosessin lopussa, jolloin testataan valmista järjestelmää tai sitä voi tapahtua koko ajan kehityksen aikana, jolloin kehittäminen, testaaminen ja uudelleen kehittäminen toistuvat sykleissä kunnes järjestelmä on kokonaisuudessaan valmis.

Tässä pro gradu -tutkielmassa esittelen käyttäjäkeskeisen järjestelmäkehitysmallin, joka hyödyntää käyttötapauksia käyttäjien vaatimusten dokumentoimiseen, käytettävyyden kehittämiseen ja testitapausten muodostamiseen. Tutkielmassa käsittelen käyttäjäkeskeisen ja käyttötapauspohjaisen ohjelmistokehityksen perusperiaatteita. Lisäksi esittelen, miten kirjallisuudessa on esitetty käyttötapauksien mahdollisuuksia ohjelmistojen testaamisessa. Viimeinen osa tutkielmaani esittelee RSI-käyttötapausanalyysi menetelmän, ja sen soveltamisen ohjelmistojen testaamiseen ja käyttäjäkeskeiseen kehittämiseen. Tutkielma sisältää myös käytännön esimerkin käyttötapauspohjaisesta kehittämisestä, testaamisesta sekä käyttöliittymäkehityksestä pankkiautomaattijärjestelmäesimerkin avulla.

2. Käyttäjäkeskeinen kehittäminen

ISO 9241-11 standardin mukaan käytettävän järjestelmän avulla ennalta määritetyt käyttäjät voivat käyttää järjestelmää ennalta määritetyssä käyttötilanteessa saavuttaakseen määritellyt tavoitteet tehokkaasti ja niin, että käyttäjä on tyytyväinen järjestelmän käyttämiseen (Jokela et al., 2003). Käytettävä ohjelma on määritelty myös sellaiseksi, joka tukee tehtävien tehokasta suoritusta määritellyssä käyttöympäristössä (Karat & Dayton, 1995). Järjestelmän käytettävyyden määritelmiä on olemassa monia hieman toisistaan eroavia. Nielsen määrittelee järjestelmän käytettävyyden viiden eri ominaisuuden avulla: opittavuuden, käytön tehokkuuden, muistettavuuden, virheettömyyden ja käyttäjän kokeman tyytyväisyyden avulla (Nielsen, 1993). Deborah Mayhewn mukaan käytettävyys on järjestelmän käyttöliittymän mitattavissa olevia ominaisuuksia, joita ilmenee käyttöliittymässä enemmän tai vähemmän (Mayhew, 1994).

Käyttäjäkeskeisen kehittämisen tarkoituksena on kehittää järjestelmiä tietyille ihmisille, tiettyjen tehtävien tehokasta, helppoa ja miellyttävää suorittamista varten tietyssä työympäristössä. Kuvassa 1 on esitetty käyttäjäkeskeisen kehittämisen perusperiaate, joka on järjestelmän kehitykselle asetettavien tavoitteiden, päämäärän, käyttökontekstin ja tehtävien tarkastelu nimenomaan järjestelmän todellisten loppukäyttäjien näkökulmasta (Rubin, 1994). Rubinin mukaan paras käyttäjäkeskeisen kehittämisen määritelmä on Wesley Woodsonin määritys, joka määrittelee käyttäjäkeskeisen kehittämisen tavaksi suunnitella tuotteita niin, että käyttäjät voivat suorittaa vaaditun käytön, toiminnot, palvelut ja tukitoiminnot mahdollisimman vähällä kuormituksella mahdollisimman tehokkaasti (Rubin, 1994). Mayhew puolestaan määrittelee käyttäjäkeskeisen kehittämisen opinalana, joka tarjoaa jäsennetyt menetelmät, joiden avulla voidaan saavuttaa käyttöliittymien käytettävyys järjestelmän kehityksen aikana (Mayhew, 1994).



Kuva 1: Käyttäjäkeskeinen kehittäminen (Rubin, 1994).

Gould ja Lewis määrittelivät käyttäjäkeskeiselle kehittämiselle kolme periaatetta, jotka ovat aikainen keskittyminen käyttäjiin ja heidän tehtäviinsä, tuotteen käytön empiirinen mittaaminen ja iteratiivinen tuotteen kehitys (Gould & Lewis, 1985). Aikainen keskittyminen käyttäjiin ja heidän tehtäviinsä tarkoittaa, että järjestelmän kehittäjien tulee tuntea tuotteensa loppukäyttäjät tutustumalla muun muassa heidän kognitiivisiin ominaisuuksiin, käyttäytymistapoihin ja asenteisiin. Käyttäjien ominaisuuksien lisäksi kehittäjien tulee tietää, kuinka käyttäjät nykyisin hoitavat ne tehtävät, joiden suorittamista varten tuleva järjestelmä kehitetään. Empiirisellä mittaamisella pyritään jo kehityksen aikaisessa vaiheessa testaamaan prototyyppien ja simulaation avulla, kuinka loppukäyttäjät suoriutuvat tehtävistään käyttäen uutta järjestelmää. Mittaamisen avulla heidän toimintaa ja reaktioita tulee havainnoida, tallentaa ja analysoida, jotta järjestelmään voidaan tehdä muutoksia. Tästä seuraakin käyttäjäkeskeisen suunnittelun kolmas periaate, jonka mukaan järjestelmää kehitetään, testataan ja mitataan sekä tämän jälkeen uudelleen kehitetään mittaamisesta saadun palautteen pohjalta kerta toisensa jälkeen niin kauan kunnes järjestelmä on valmis.

Maguiren mukaaan käyttäjäkeskeisen kehittämisen eduista seuraa myös lisäkustannuksia verrattuna perinteisiin kehitysmenetelmiin. Hänen mukaansa johdon tehtävästä tulee vaikeampi, koska projektin suunnittelun tulee huomioida kehitystehtävien iterointi ja palautteen kerääminen (Maguire, 1998). Aikaa kuluu myös kehitysorganisaation eri edustajien väliseen kommunikointiin ja kompromissien tekemiseen. Kuitenkin käyttäjäkeskeinen kehittäminen tuo tullessaan monia etuja, kuten rahalliset säästöt myöhemmissä kehitysvaiheissa, jolloin muutosten tekeminen olisi paljon kalliimpaa. Lisäksi käyttäjät tuntevat järjestelmän enemmän omakseen ja myös kehitysorganisaatio hyötyy kehittäjien luovuudesta ja taitojen kehittymisestä (Maguire, 1998).

2.1 Käyttäjäkeskeisen kehittämisen prosessi

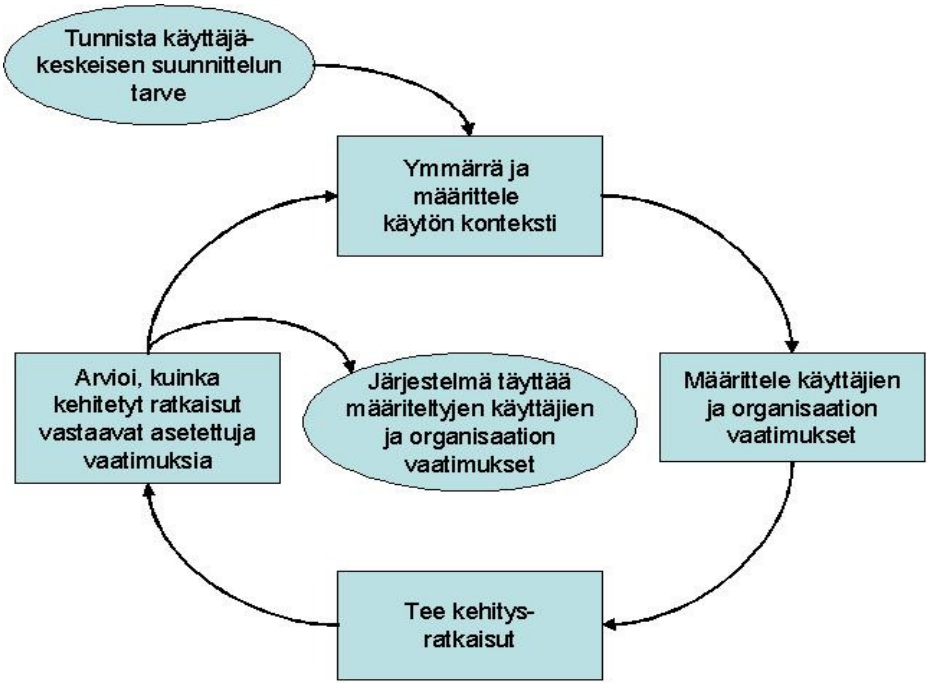
Käyttäjien kannalta hyvien järjestelmien kehittämiseksi on esitetty monia eri kuvauksia prosesseista ja tehtävistä, joita järjestelmän elinkaaren tulisi sisältää. Käyttäjäkeskeisestä järjestelmäkehityksestä on olemassa ISO standardi 13407, joka määrittelee interaktiivisten järjestelmien käyttäjäkeskeisen suunnitteluprosessin. Standardi ei kuvaa erilaisia menetelmiä käytettävyyden huomioimiseksi, vaan se määrittelee käyttäjäkeskeisen suunnittelun yleiset periaatteet ja toiminnot (Jokela et al., 2003).

Yleisiä periaatteita käyttäjäkeskeiseen kehittämiseen standardin mukaan on neljä:

1. Käyttäjien aktiivinen osallistuminen kehitystyöhön ja kehittäjien selkeä käsitys käyttäjä- ja tehtävävaatimuksista.
2. Sopiva ja tarkoituksenmukainen toimintojen jakaminen käyttäjien ja teknologian välillä.
3. Iteratiivinen kehittäminen
4. Kaikkien sidosryhmien osallistuminen kehittämiseen

Standardi korostaa, että kehitysprosessissa tulee varata aikaa ja muita resursseja sekä toimintojen iteroimiseen eli toimintojen toistoon että palautteen keräämistä ja analysointia varten (Jokela et al., 2003).

Käyttäjäkeskeisen kehittämisen prosessiin kuuluu neljä toimintoa, jotka on esitetty kuvassa 2 suorakaiteen muotoisissa laatikoissa. Nämä toiminnot ovat standardin ydin. Kuten kuvassa 2 on esitetty, kehitysprosessi aloitetaan määrittelemällä kehitettävän järjestelmän käyttöympäristö eli käytön konteksti. Toisena toimintona on määritellä loppukäyttäjien ja kohdeorganisaation vaatimukset kehitettävälle järjestelmälle. Kun tunnetaan käyttäjät, kohdealue sekä käyttökonteksti, tehdään järjestelmän kehitysratkaisut määrittelyjen perusteella. Neljäs toiminto on tehtyjen ratkaisujen arvioiminen suhteessa asetettuihin vaatimuksiin. Mikäli ratkaisut eivät täytä niille asetettuja vaatimuksia, tulee kehityssykli aloittaa alusta tarkastamalla käytön konteks-tin määritys. Kun ratkaisujen ja vaatimusten välillä ei ole enää ristiriitoja, järjestelmä täyttää käyttäjien ja kohdeorganisaation vaatimukset.



Kuva 2: Käyttäjäkeskeisen kehittämisen prosessi (Jokela et al., 2003).

Prosessina ISO 13407 standardi toteuttaa myös Gouldin ja Lewisin kolme käyttäjäkeskeisen kehittämisen periaatetta: järjestelmän kehitys perustuu käyttäjien ja näiden käyttöympäristön määrittelyyn, jolloin käyttäjät huomioidaan jo heti järjestelmäkehityksen alusta lähtien, kehityksen tulosta verrataan asetettuihin vaatimuksiin. Jos tulos ei täytä asetettuja vaatimuksia, suoritetaan kehitystoimintojen toisto, kunnes vaatimukset toteutuvat.

2.2 Käyttäjäkeskeisen kehittämisen elinkaarimalli

Käyttäjäkeskeisen kehittämisen elinkaarimallissa (*Usability Engineerin Lifecycle*) Deborah Mayhew jakaa järjestelmän elinkaaren kolmeen osaan: vaatimusanalyysiin, Suunnittelu/Testaus/Kehitys –vaiheeseen sekä asennusvaiheeseen. Näistä vaatimusten analysointivaihe sisältää ISO standardin tehtävät ”Ymmärrä ja määrittele käytön konteksti” ja ”Määrittele käyttäjien ja organisaation vaatimukset”. Suunnittelu/Testaus/Kehitys –vaihe sisältää ISO standardin kaksi muuta tehtävää ”Tee kehitysratkaisut” ja ”Arvioi, kuinka ratkaisut vastaavat asetettuja vaatimuksia”.

Mayhewn elinkaarimallissa vaatimusanalyysivaiheessa kehittäjät määrittelevät kehitettävän järjestelmän loppukäyttäjät käyttäjäprofiilin (*User Profile*) avulla (Mayhew, 1994). Käyttäjäprofiilissa määritellään käyttäjien ominaisuudet, jotka voivat vaikuttaa käyttöliittymän suunnitteluun, kuten psykologiset ominaisuudet (esim. asenteet, motivaatio), tietämys ja kokemus (esim. konekirjoitustaito, kohdealueen tuntemus), työ- ja tehtäväominaisuudet (esim. käytön määrä, tehtävä rakenne) ja fysiologiset ominaisuudet (esim. värisokeus). Käyttäjäprofiili voidaan muodostaa esimerkiksi haastattelemalla loppukäyttäjiä tai suorittamalla käyttäjäkysely.

Vaatimusanalyysiin kuuluu myös kontekstuaalisen tehtäväanalyysin (*Contextual Task Analysis*) tekeminen. Tehtäväanalyysissä määritellään käyttäjäprofiilin avulla löydettyjen eri käyttäjäryhmien nykyiset työtehtävät ja tavat, joilla he tällä hetkellä suorittavat tehtävänsä. Kun ymmärretään mitä tehtäviä käyttäjät suorittavat, ymmärretään mitä tavoitteita käyttäjien toiminnalla on (Mayhew, 1994). Tehtäväanalyysi voidaan tehdä vain havainnoimalla käyttäjien toimintaa heidän nykyisessä työympäristössään ja keskustelemalla samalla käyttäjien kanssa siitä, mitkä heidän tehtävänsä on ja kuinka he suorittavat ne. Tarkastelemalla käyttäjien toimintaa heidän aidossa työympäristössään saadaan mahdollisesti selville myös ne työvaiheet, joita he eivät tiedosta suorittavansa.

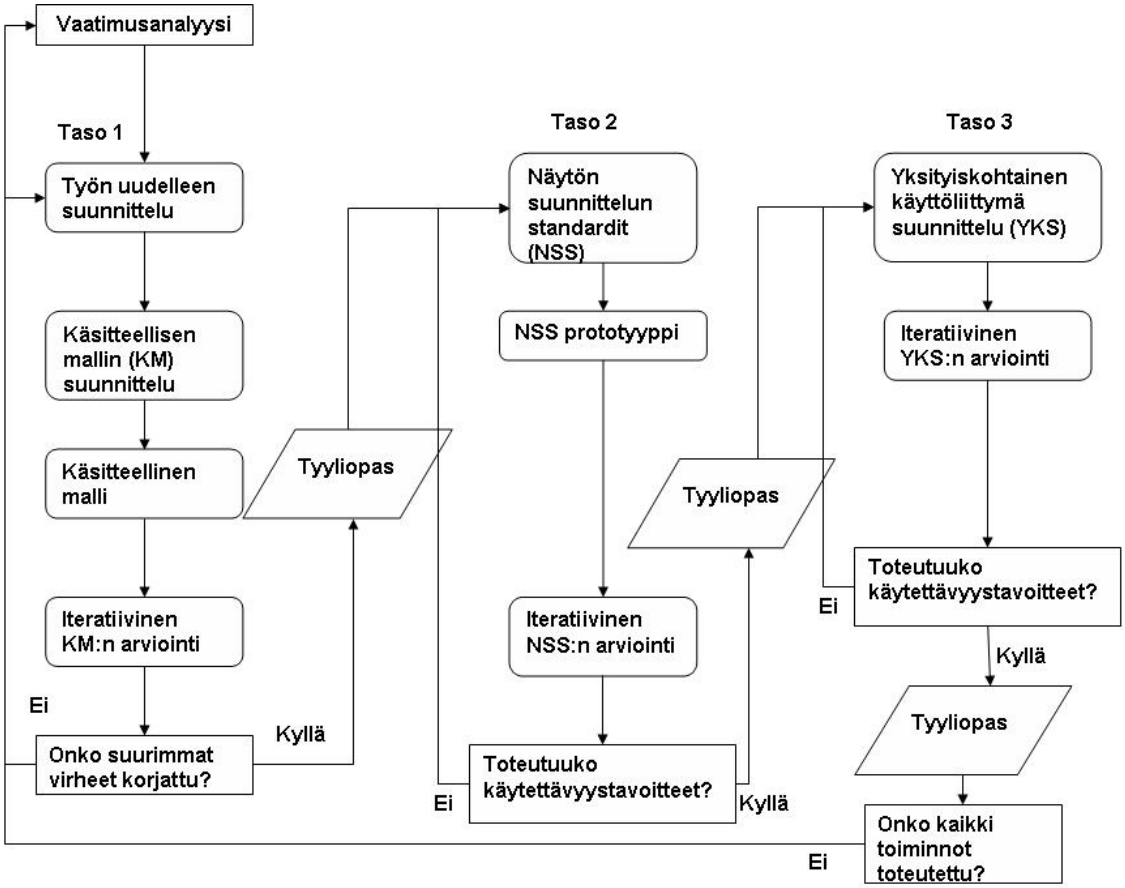
Käyttäjäprofiilin, kontekstuaalisen tehtäväanalyysin, järjestelmän teknisten rajoitteiden ja mahdollisuuksien sekä yleisten käyttöliittymäsuunnittelun periaatteiden pohjalta muodostetaan kehitettävän järjestelmän käytettävyystavoitteet (*Usability goals*). Käytettävyystavoitteet voivat olla

kvantitatiivisia tai kvalitatiivisia. *Kvantitatiiviset käytettävyystavoitteet* ovat mitattavissa olevia tavoitteita, joita voidaan käyttää myöhemmin hyväksymisehtoina käytettävyysarvioinneissa. *Kvalitatiiviset tavoitteet* muodostetaan käyttäjäprofiilin ja kontekstuaalisen tehtäväanalyysin avulla ja ne ovat yleisiä tavoitteita, joita ei voida mitata, mutta jotka ohjaavat kehitystyötä (Mayhew, 1994).

Kvantitatiiviset tavoitteet voivat liittyä käytön helppouteen (*Ease-of-Use*) tai oppimisen helppouteen (*Ease-of-Learn*) (Mayhew, 1994). Yleisesti voidaan ajatella, että jos käyttäjät suorittavat jotakin tehtävää usein, voidaan tehtävän toteutuksen tavoitteeksi asettaa, että tietyn koulutusajan jälkeen asiantuntijakäyttäjät pystyvät suoriutumaan tehtävästä tehokkaasti ja nopeasti, jolloin siis panostetaan käytön helppouteen. Jos tehtävää suoritetaan harvoin tai tarkoituksena on, että käyttäjät voivat suoriutua tehtävästä ilman koulutusta, keskitytään oppimisen helppouteen käyttöliittymän suunnittelussa.

Kaikki kvantitatiiviset tavoitteet voidaan muotoilla joko suhteellisiksi tai absoluuttisiksi tavoitteiksi. Suhteellisia tavoitteita voidaan mitata ja arvioida esimerkiksi vertailemalla käyttäjän suoritusta aikaisempaan ohjelman versioon tai kilpailijan vastaavan tuotteen kyseiseen toimintoon. Absoluuttiset tavoitteet mitataan absoluuttisilla mitoilla, kuten virheiden määränä suorituksen aikana tai suorituksen kestona sekunteina.

Kun vaatimusanalyysi on suoritettu, on suoritettu kaksi ensimmäistä ISO standardin määrittelemistä tehtävistä: tunnetaan käyttäjät, käytön konteksti ja käyttäjien sekä organisaation vaatimukset. Tämän jälkeen siirrytään Mayhewn elinkaarimallissa Suunnittelu/Testaus/Kehitys –vaiheeseen, jonka eri tasot ja tasojen eri tehtävät on esitetty kuvassa 3. Kuten kuvasta 3 huomataan, Suunnittelu/Testaus/Kehitys –vaihe on jakautunut kolmeen eri tasoon, joista ensimmäinen sisältää tehtäviä, joiden avulla määritellään ylemmän tason suunnitteluperiaatteet. Tasolla kaksi muodostetaan järjestelmän standardit ja muodostetaan ensimmäinen prototyyppi järjestelmästä. Kolmannessa vaiheessa toteutetaan itse järjestelmä. Kunkin tason tuotokset dokumentoidaan myös tyylioppaaseen. Jokainen Suunnittelu/Testaus/Kehitys –vaiheen taso sisältää arviointi vaiheen, jonka aikana testataan, vastaako kehityksen tulos käytettävyystavoitteita, joten myös kaksi viimeistä ISO standardin tehtävää tulee toteutetuksi tässä Mayhewn elinkaarimallissa.



Kuva 3: Käyttäjäkeskeisen elinkaarimallin toinen vaihe (Mayhew, 1994).

Kun kaikki toiminnot on toteutettu, siirrytään Mayhewn mallissa kolmanteen vaiheeseen eli asennusvaiheeseen. Asennuksen lisäksi vaihe sisältää käyttäjäpalautteen keräämisen. Käyttäjäpalaute voi aiheuttaa elinkaaren käynnistymisen alusta jonkin aikaisemmin määrittelemättömän käyttäjäryhmän tai toiminnon osalta.

Käyttäjäkeskeisen kehittämisen tarkoituksena on siis järjestelmän kehittäminen tietyille käyttäjille, tiettyjen tehtävien suorittamiseksi tietyssä ympäristössä. Kehitysprosessissa keskitytään erityisesti käyttöliittymän ja muun käyttäjälle näkyvän kehittämiseen. Kehitystyö perustuu käyttäjien kanssa suoritettavaan yhteistyöhön ja säännölliseen palautteeseen, joka ohjaa kehitystä.

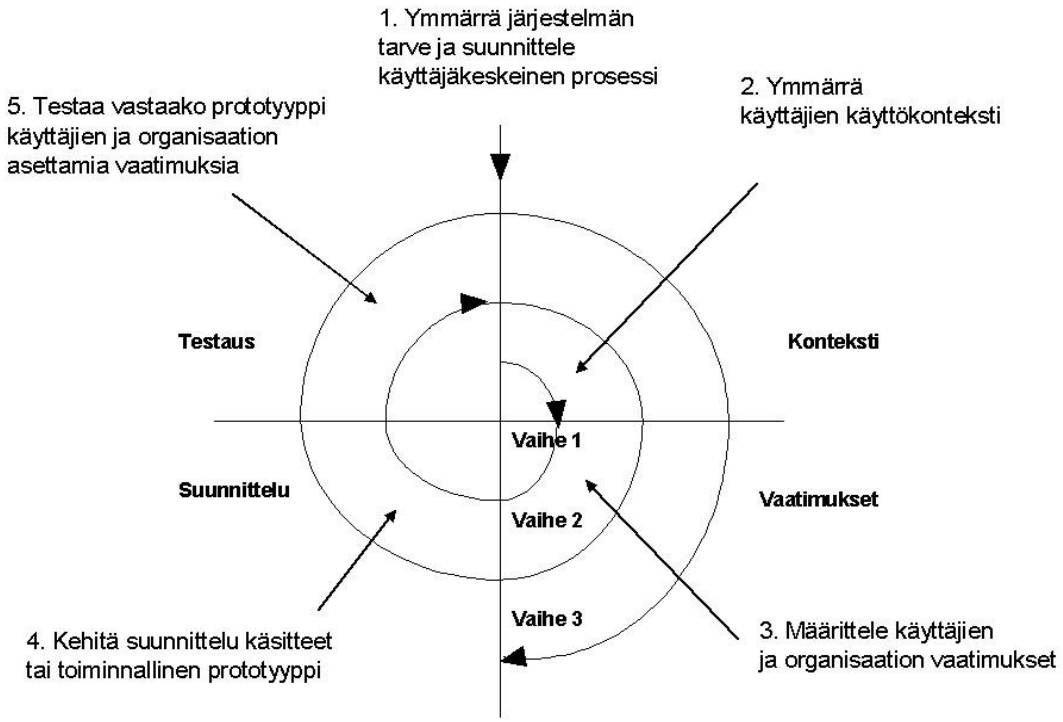
2.3 RESPECT

RESPECT on käyttäjäkeskeinen menetelmä järjestelmän vaatimusmäärittelyn tekemiseksi. Menetelmä perustuu ISO 13407 standardiin (Maguire, 1998). RESPECT-menetelmä noudattaa yleisesti hyväksyttyjä perinteisen vaatimusmäärittelyn periaatteita, mutta lisäksi se huomioi käyttäjä- ja käytettävyysnäkökulmat (Maguire, 1998). Järjestelmän kehittämiselle voidaan Maguiren mukaan asettaa kolmenlaisia vaatimuksia, jotka ovat kohdealuevaatimukset, käyttäjävaatimukset ja toiminnalliset vaatimukset sekä tekniset vaatimukset (Maguire, 1998). Kaikkien kolmen tyyppisten vaatimusten keräämisessä ja analyysissä on onnistuttava, että kehitettävästä järjestelmästä voisi tulla menestyvä. Maguiren mukaan perinteisissä vaatimusmäärittely menetelmissä käyttäjien vaatimukset kuvataan yleensä idealistisesti ja abstraktilla tasolla enemmin kuin kuvattaisiin ne käyttäjien tarpeina osana käyttäjien suorittamia tehtäviä (Maguire, 1998).

RESPECT-menetelmä yhdistää käyttäjäkeskeisen kehittämisen prosesseja perinteisiin kehitysprosesseihin kattavan vaatimusmäärittelyn kehittämiseksi. RESPECT–menetelmä täydentää perinteisiä vaatimusmäärittelyprosesseja määrittämällä korkean tason laatu- ja käyttäjävaatimusten sekä keräämällä palautetta näistä ennen tarkempien teknisten vaatimusten keräämistä.

RESPECT-menetelmä painottaa myös perinteisiä vaatimusmäärittelymenetelmiä enemmän yksityiskohtaisten käytön kuvausten ymmärtämistä tärkeiden, ei-toiminnallisten vaatimusten keräämiseksi. RESPECT-menetelmä korostaa kokonaisvaltaisen tietämyksen hankkimista käyttäjien tarpeista ja niistä muodostuvien vaatimusten todentamista käytön kuvausten avulla (Maguire, 1998). Menetelmä käyttää yksityiskohtaisia käyttöskenaarioita käyttäjille tärkeiden ei-toiminnallisten vaatimusten selvittämiseen. Tämä tarkoittaa sitä, että RESPECT-menetelmä sisältää perinteisiä käyttötapauksia ja skenaarioita tarkempia käytönkuvauksia (Maguire, 1998). Menetelmän avulla voidaan myös paremmin asettaa vaatimuksia käyttäjien kannalta tärkeysjärjestykseen. RESPECT-menetelmä on kuitenkin tarkoitettu käytettäväksi yhdessä perinteisten vaatimusmäärittelymenetelmin kanssa kohdealuevaatimusten ja tarkempien teknisten vaatimusten keräämiseksi (Maguire, 1998).

RESPECT-menetelmän mukainen vaatimusmäärittely koostuu kolmesta iteratiivisesta vaiheesta, jotka ovat käyttökontekstin määrittely ja esisuunnittelu, prototyypin kehittäminen ja käyttäjätestaus sekä käyttäjien vaatimusten dokumentoiminen (Maguire, 1998). Jokainen vaihe jaetaan neljään osa-alueeseen, jotka ovat konteksti, vaatimukset, suunnittelu ja testaus. RESPECT-prosessin eri vaiheet ja osa-alueet on esitetty kuvassa 4.

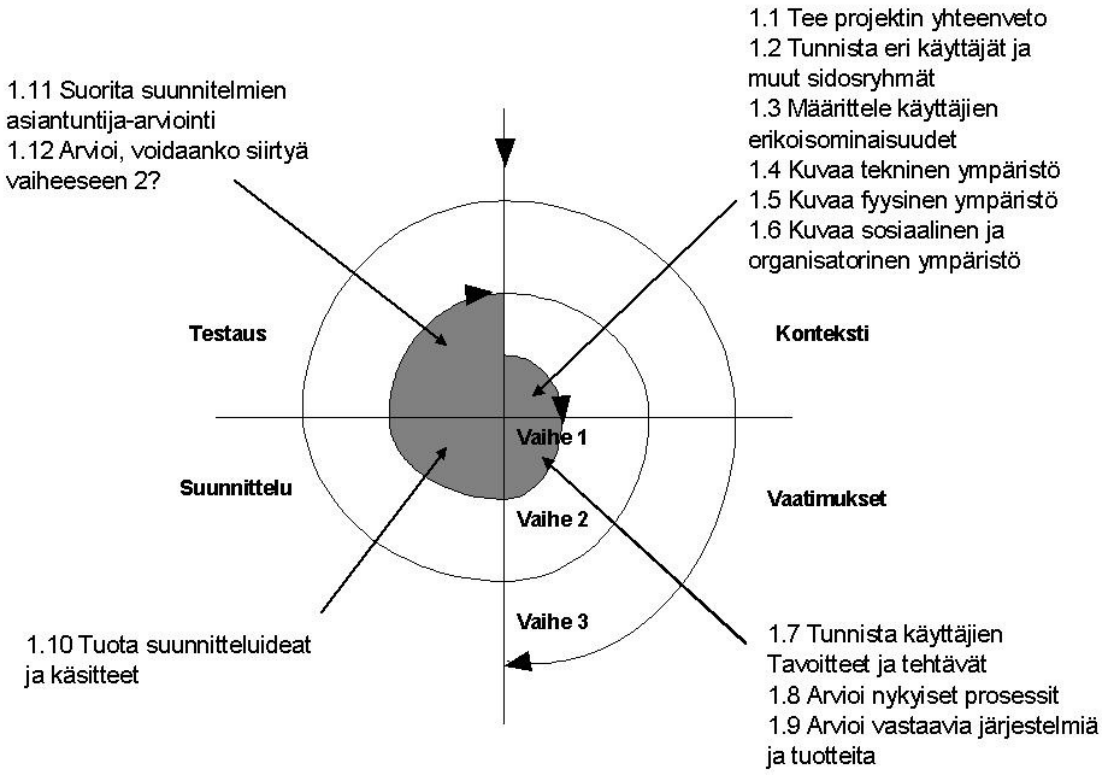


Kuva 4: RESPECT vaatimusmäärittely prosessin vaiheet (Maguire 1998).

Jokainen osa-alue sisältää tehtäviä, joiden suorittamista varten RESPECT-dokumentaatio sisältää valmiit dokumenttipohjat. RESPECT-menetelmän tarkoituksena on kerätä tietoa käyttäjistä käytön kuvausten muodostamiseksi eri tehtävien avulla. Vaiheet ovat iteratiivisia siten, että jokaisen tehtävän tuotos arvioidaan ja mahdollisesti muokataan ennen seuraavaan tehtävään siirtymistä. Myös ennen jokaisen vaiheen päättymistä ennen arvioidaan saavutetut tulokset ja seuraavaan vaiheeseen voidaan siirtyä vasta kun ennalta määritetyt tavoitteet on saavutettu.

Kuvassa 5 on esitetty tarkemmin vaiheen yksi tehtävät. RESPECT-prosessin ensimmäinen vaihe perustuu projektin ja sen tavoitteiden esimäärittelyyn. Ensimmäisen vaiheen kontekstin osalta kerätään ja analysoidaan tietoa eri käyttäjistä ja muista kehityksen sidosryhmistä sekä määritel-

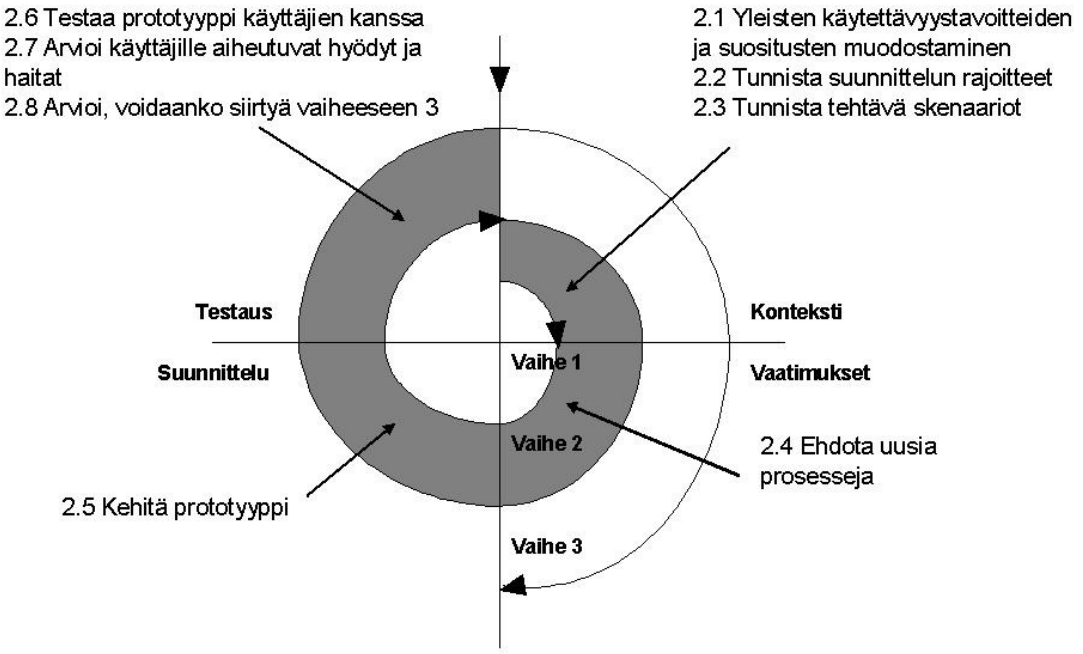
lään heidän ominaispiirteensä. Kontekstiin liittyy myös käyttöympäristön määrittely. Vaatimusten osalta tunnistetaan käyttäjien tavoitteet ja tehtäväy sekä arvioidaan nykyisiä prosesseja ja muita kehitettävää järjestelmää vastaavia järjestelmiä. Nykyisten prosessien osalta arvioidaan, mitkä käyttäjien tavoitteet ne toteuttavat ja mitä ongelmia niissä on. Muista markkinoilla olevista tuotteista voidaan hakea lisää kehitysideoita ja arvioida, mitä toimintoja niistä kehitettävän järjestelmän tulisi tai ei tulisi sisältää. Suunnitteluosassa tuotetaan suunnitteluideat ja käsitteet tähän asti kerätyn tiedon pohjalta. Kerätystä tiedosta voidaan muodostaa useampi erilainen malli arvioitavaksi. Testiosassa suoritetaan asiantuntija-arvioinnit suunnitelmille ja arvioidaan, toteuttavatko suunnitellut mallit käyttäjien tavoitteet. Jos käyttäjien tavoitteet voidaan saavuttaa kehitysratkaisujen avulla, voidaan siirtyä tasolle kaksi.



Kuva 5: RESPECT-vaatimusmäärittelyn ensimmäisen vaiheen tehtävät (Maguire, 1998).

Vaiheen kaksi tarkempi tehtävien kuvaus on esitetty kuvassa 6. Toisessa vaiheessa RESPECT-vaatimusmäärittelyä kontekstin osalta määritellään järjestelmän käytettävyystavoitteet ja yleiset suositukset sekä tunnistetaan suunnittelun rajoitteet. Järjestelmän suunnitelman testaamista varten

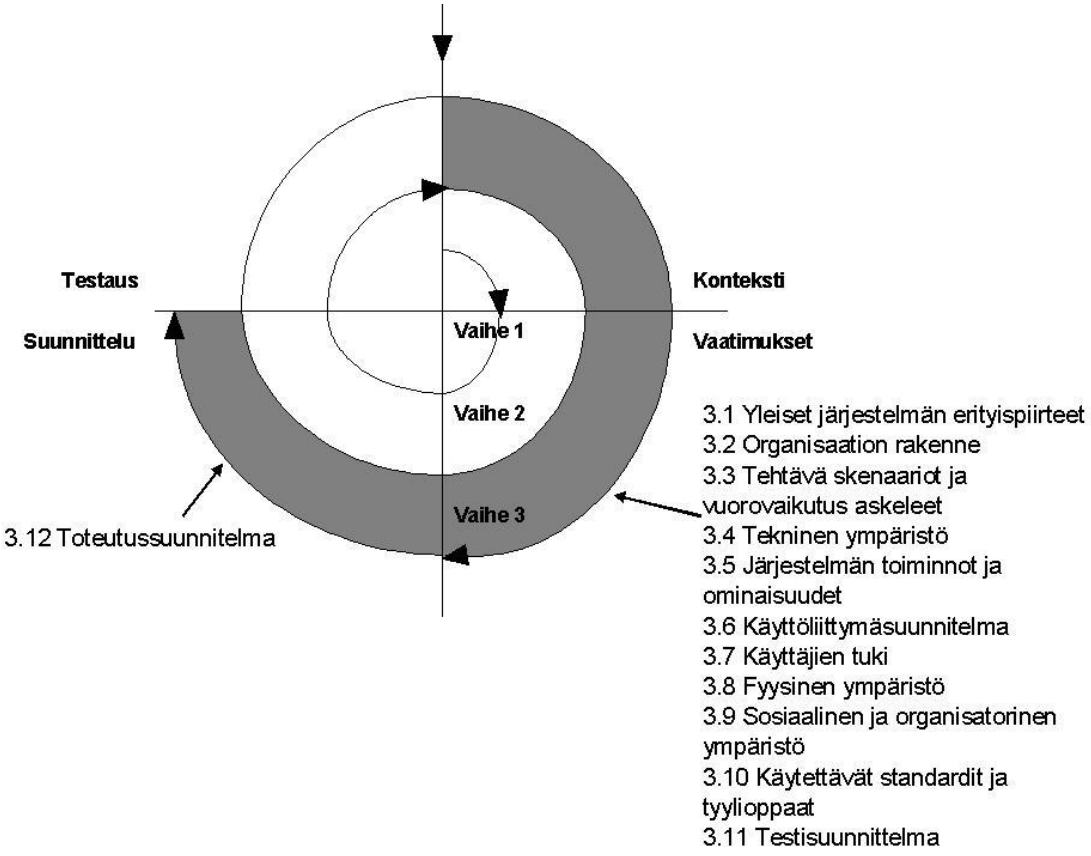
tunnistetaan tarvittavat tehtäväskenaariot. Jokaista käytettävyystavoitetta tulee vastata ainakin yksi skenaario ja jokainen skenaario kuvaa joukon käytettävyystavoitteita tietyssä kontekstissa. Vaatimusten osalta jokaisesta skenaariosta kuvataan niiden interaktiivinen prosessi käyttäjän ja järjestelmän väliltä. Samaan aikaan kehitetään myös lista mahdollisista toiminnoista ja ominaisuuksista tukemaan prosessin dokumentointia. Suunnitteluosassa luodaan järjestelmästä interaktiivinen prototyyppi edellä mainitun toiminto- ja ominaisuuslistan pohjalta. Testausvaiheessa testataan kehitetty prototyyppi käyttäjien kanssa käyttäen kehitettyjä skenaarioiden kuvauksia. Testauksessa havaitut ongelmat dokumentoidaan. Lisäksi arvioidaan jokaisen käyttäjäryhmän osalta saavuttavatko eri käyttäjäryhmät tavoitteensa tehtävien suorittamisen aikana. Jos prototyyppi ja arvio ovat hyväksyttäviä, voidaan siirtyä vaiheeseen kolme eli käyttäjien vaatimusten dokumentoimiseen, joka on esitetty kuvassa 7.



Kuva 6: RESPECT-vaatimusmäärittelyn ensimmäisen vaiheen tehtävät (Maguire, 1998).

Kaksi ensimmäistä vaihetta tuottavat joukon mahdollisia käyttäjien vaatimuksia, jotka muodostavat pohjan vaatimusmäärittelylle. Kolmannessa vaiheessa kehittäjien ja käyttäjien edustajien tulisi käydä läpi jokainen vaiheissa yksi ja kaksi esitetty vaatimus ja valita niistä kaikki potentiaaliset käyttäjien vaatimukset. Tämän jälkeen kehittäjät ja käyttäjät arvioivat vaatimukset niin, että jo-

kainen vaatimus esiintyy lopullisissa vaatimuksissa vain kerran, kaikki tarpeettomat ja ristiriitaiset vaatimukset poistetaan ja arvioinnin yhteydessä huomatut uudet vaatimukset lisätään vaatimuksiin. Tämän jälkeen vaatimukset dokumentoidaan vaatimusmäärittelyluonnokseen, jonka käyttäjät ja kehittäjät vielä arvioivat ennen vaatimusten hyväksymistä. Kuvassa 7 on esitetty vaiheen kolme aikana toteutettava käyttäjien vaatimusten dokumentointi. Vaatimusvaiheessa käyttäjien vaatimukset koostetaan järjestelmän yleisistä vaatimuksista käyttäjien tavoitteisiin ja tehtäviin sekä järjestelmän tekniseen ympäristöön. Suunnitteluvaiheessa toteutetaan toteutussuunnitelma, johon kirjataan muun muassa, miten käyttäjiä tiedotetaan kehitettävästä järjestelmästä ja millaista koulutusta he tarvitsevat.



Kuva 7: RESPECT-vaatimusmäärittelyn kolmannen vaiheen tehtävät (Maguire, 1998).

Ensimmäinen ja toinen vaihe vastaavat perinteisen vaatimusmäärittelyprosessin tehtäväanalyysivaihetta (Maguire, 1998). Ensimmäisessä vaiheessa analysoidaan, miten käyttäjät hoitavat tehtävänsä tällä hetkellä ja toisessa vaiheessa kehitetään uusia prosesseja ja toimintoja korvaamaan

ensimmäisessä vaiheessa havaitut toimintatavat. Perinteisten vaatimusmäärittelymenetelmien avulla voidaan tässä vaiheessa varmistaa, että kaikki kehitettävän järjestelmän tärkeät toiminnot tulee havaituksi, kun taas RESPECT-prosessin avulla muodostetaan yhtenäinen kuva toimintojen yhteistoiminnasta.

RESPECT-prosessin kolmas vaihe vastaa perinteisten menetelmien yksityiskohtaisen määrittelyn vaihetta, jossa kerättyjä vaatimuksia käytetään järjestelmän kuvauksen tekemiseksi järjestelmän suunnittelijoille ja toteuttajille. Kolmannessa vaiheessa tuotetaan käyttäjäkeskeinen lista käyttäjien vaatimuksista, jonka avulla tulevat loppukäyttäjät voivat nähdä, millainen kehitettävästä järjestelmästä tulee, ja arvioida, vastaako se heidän asettamia vaatimuksiaan.

3. Käyttötapauspohjainen kehittäminen

Ivar Jacobsonin vuonna 1987 esittelemät käyttötapaukset (*Use Case*) kuvaavat järjestelmän toimintaa eri näkökulmista (Jacobson, 1987). *Käyttötapaus* on tietty sarja käyttäjän (*user*) ja järjestelmän (*system*) välisiä tapahtumia. Käyttäjä on tietyn roolin (*role*) omaava toimija, joka käyttää järjestelmää. Käyttäjä ei ole vain järjestelmän ihmiskäyttäjä, vaan se voi olla myös jokin muu järjestelmä, tehtävä tai laite (Rumbaugh, 1994). *Rooli* määritellään tietyn tehtävän tai toisiinsa liittyvien tehtävien ryhmän avulla, joita järjestelmää käyttävät henkilöt suorittavat käyttäessään järjestelmää (Nygaard, 1986). Käyttötapausten tarkoituksena on auttaa kehittäjiä suunnittelemaan ja toteuttamaan järjestelmää käyttäjien näkökulmasta, että käyttäjät voisivat saada sellaisen järjestelmän, joka toimii heidän vaatimustensa mukaisesti.

Aluksi Jacobson kuvasi käyttötapauksia käsitteellisen diagrammin (*Conceptual diagram*) avulla, jossa käyttötapaukset muodostavat verkon solmut (Jacobson, 1987). Vuonna 1997 Jacobson esitteli käyttötapaukset käyttötapauskaavioina, joiden lisäksi hän käytti myös muita dokumentteja käyttötapausten kuvaamiseen (Jacobson et al, 1992). Myöhemmin käyttötapauksia on ruvettu kuvaamaan myös sanallisesti joko enemmän tai vähemmän muodollisella tavalla.

Alistair Cockburn on esittänyt, että käyttötapauksille on olemassa ainakin 18 erilaista määrittelyä, jotka eroavat toisistaan neljän eri ulottuvuuden perusteella (Cockburn, 1997). Nämä neljä eri ulottuvuutta ovat tarkoitus (*purpose*), sisältö (*content*), määrä (*multiplicity*) ja rakenne (*structure*). *Tarkoitus* voi olla kertomus (*storie*), jos käyttötapausten tarkoituksena on kuvata käyttäjien kertomuksia heidän toiminnastaan, tai vaatimus (*requirement*), jos tarkoituksena on kerätä käyttäjien järjestelmälle asettamat vaatimukset. *Sisältö* määräytyy sen mukaan, kuvataanko käyttötapaukset muodollisesti (*formal content*), yhdenmukaisessa proosamuodossa (*consistent prose*) vai ristiriitaisesti (*contradicting*). Kolmas ulottuvuus käyttötapauksissa Cockburnin mukaan on *määrä*, joka määräytyy sen mukaan, kuvaako käyttötapaus vain yhden *skenaarion* eli vain yhden tavan suorittaa tehtävä vai useita skenaarioita, jolloin käyttötapaus sisältää kuvauksen useammasta eri tavasta suorittaa sama tehtävä. *Rakenne* määräytyy sen mukaan muodostavatko käyttötapaukset muodollisen rakenteen (*formal structure*), epävirallisen rakenteen (*semiformal*) vai epämuodollisen (*unstuctured*) kokoelman käyttötapauksia.

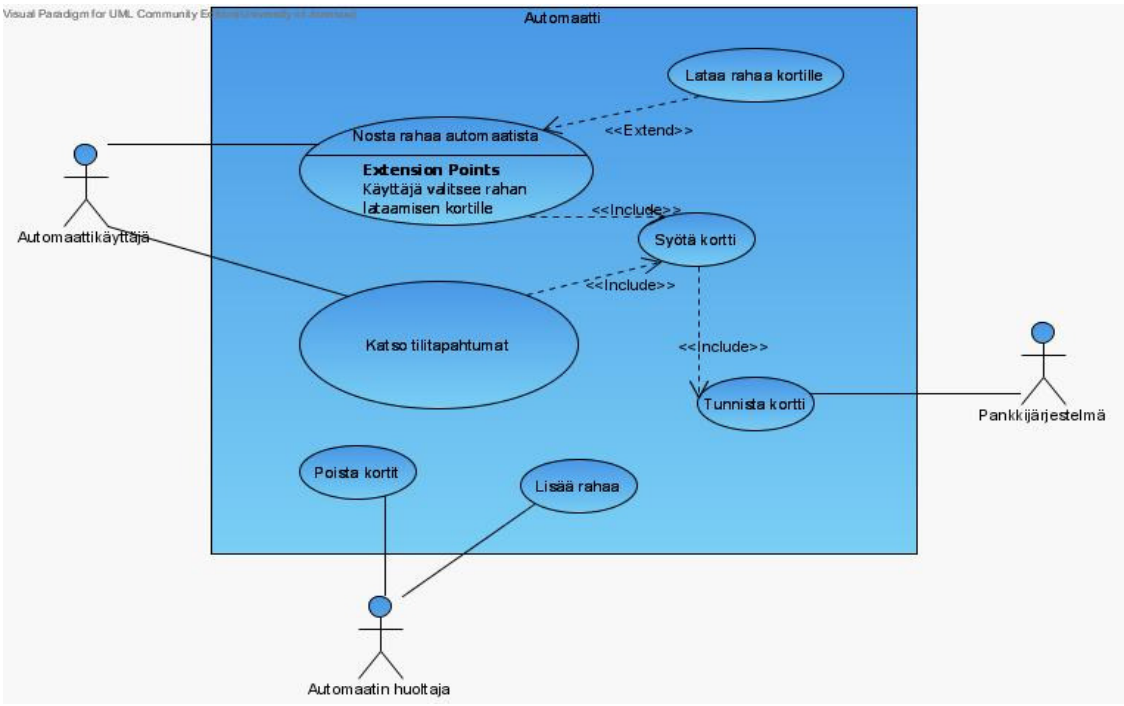
Käyttötapauksista on kehitetty niiden esittelemisen jälkeen monia eri variaatioita, ja vaikka ne on tarkoitettu käytettäväksi käyttäjien vaatimusten selvittämiseen ja tallentamiseen, niitä on käytännössä usein käytetty vain järjestelmän toimintojen ja teknisten ominaisuuksien sekä rajoitusten kuvaamiseen (Seffah et al., 2001). Monista eri käyttötapaus-käsitteen määrityksistä on seurannut, että mitkään kaksi kehitysprojektia, joissa käytetään käyttötapauksia, eivät tuota täsmälleen samanlaista kuvausta kehitettävästä järjestelmästä (Collins-Cope, 2000).

Käyttötapaukset kuvataan usein käyttötapauskaavioiden ja käyttötapausten sanallisten kuvausten avulla. Käyttötapauskaavioiden tekemiseen käytetään yleensä UML-notaatiota (OMG, 2005), jossa käyttötapauksen käyttäjät kuvataan tikku-ukkoina ja suoritettavat toiminnot eli käyttötapaukset ellipseinä. Ellipsit nimetään käyttötapauksen tunnisteella, joka muodostetaan yleensä käyttäjän suorittaman toiminnon verbistä ja toiminnan kohteesta. Esimerkiksi pankkiautomaatilla asioivan käyttäjän toiminta voisi olla rahan nostaminen tilitä, jolloin käyttötapauksen tunniste voisi olla *Nosta rahaa*. Lisäksi käyttäjä voi suorittaa automaatilla tilin saldon ja tilitapahtumien tarkastamisen. Järjestelmään liittyy myös automaatin huoltaja, joka käy lisäämässä rahaa automaattiin ja kerää automaatin talteen ottamat kortit pois pankkeihin takaisin lähetettäväksi. Lisäksi automaatti kommunikoi pankkijärjestelmän kanssa pankkikortin tunnistamiseksi, eli automaattijärjestelmään kuuluu myös yksi käyttäjä, joka on toinen järjestelmä. Pankkiautomaattijärjestelmää käytetään esimerkkinä koko pro gradu –tutkielmassa, jotta eri menetelmien vaikutus järjestelmän kehittämiseen tulisi huomatuksi.

Aluksi Jacobson määritteli käyttötapauksille kaksi suhdetta, joista toinen tarkoitti tapausta, jossa käyttötapaus pohjautuu toiseen, yleisempään käyttötapaukseen (*Build-on suhde*) (Jacobson, 1987). Toinen mahdollinen suhde kuvasi käyttötapauksen ja *entiteetin* eli kohdealueen asian tai tapahtuman välistä suhdetta, jossa entiteetillä oli pääsy käyttötapaukseen (*Access-suhde*). Myöhemmin Jacobson esitteli käyttötapauksille käyttää (<<uses>>) ja laajentaa (<<extends>>) -suhteet (Jacobson et al, 1997). Jacobson suosittelee laajennusten käyttämistä kuvaamaan itsenäisten käyttötapausten yhteydessä laajennettavan käyttötapauksen osien vaihtoehtoista suoritusta, monimutkaisia ja harvoin tapahtuvia käyttötapauksen poikkeuksia, tai kuvaamaan mitä käyttötapauksia jokin käyttötapaus mahdollistaa (Jacobson et al, 1992). UML-notaation versio 1.3 sisäl-

tää kolme käyttötapausten välistä suhdetta (OMG, 2005). Käyttötapaus voi sisältyä (<<include>>) toiseen käyttötapaukseen, laajentaa (<<extend>>) toista käyttötapausta tai käyttötapaus voi olla toisen käyttötapauksen erikoistuma (specialisation).

Edellä kuvattu pankkiautomaatti käyttötapausesimerkki voisi sisältää oman käyttötapauksen pankkikortin ja tunnusluvun syöttämiselle, sillä kortin ja tunnusluvun tunnistamiseen liittyy hyvin paljon poikkeuksia. Tällöin käyttötapaus *Syötä kortti* voisi kuulua sekä *Nosta rahaa* ja *Katso tilitapahtumat* käyttötapauksiin. Kehitettävä järjestelmä ei sisällä laajentavia käyttötapauksia, mutta esimerkin vuoksi voidaan ajatella, että pankkiautomaattijärjestelmään voitaisiin mahdollisesti tulevaisuudessa liittää toiminto, jonka avulla käyttäjä voisi rahan nostamisen sijaan ladata rahaa kortin sirulle. Käyttötapaus *Lataa rahaa kortille* voidaan siis kuvata rahan nostamisen käyttötapausta laajentavana käyttötapauksena. Tämän järjestelmän käyttötapauskaavio sisältää myös laajentavan käyttötapauksen *Lataa rahaa kortille*, vaikka sitä ei toteuta järjestelmään. Käyttötapauskaavio on esitelty kuvassa 8.



Kuva 8: Pankkijärjestelmän käyttötapauskaavio.

Käyttötapauskaavioiden lisäksi käyttötapauksia kuvataan myös sanallisesti. Itse toiminnan suorittaminen kuvataan yleensä käyttäjän tai käyttäjien ja järjestelmän välisenä dialogina. Käyttötapaus koostuu askeleista, joista joka toinen on käyttäjän suorittama ja joka toinen järjestelmän suorittama. Jos käyttötapauksessa kuvataan useampi skenaario, voi käyttötapauksen askeliin liittyä poikkeuksia, joissa kuvataan vaihtoehtoiset skenaariot. Poikkeukset voidaan ilmaista myös käyttötapaukseen liittyvinä tai laajentavina käyttötapauksina, esimerkiksi jos poikkeus voi aiheuttaa useita poikkeuksia lisää, jolloin sanallisesta kuvauksesta voi tulla raskas tehdä, lukea ja ylläpitää.

Esimerkiksi käyttötapauksen *Nosta rahaa* suoritus voisi edetä seuraavalla tavalla:

1. Käyttäjä on syöttänyt pankkikortin ja tunnusluvun onnistuneesti. Järjestelmä kysyy käyttäjältä valittavaa toimintoa.
2. Käyttäjä valitsee rahan noston.
3. Järjestelmä kysyy nostettavaa summaa.
4. Käyttäjä syöttää nostettavan summan.
5. Järjestelmä tarkastaa, että tilillä on olemassa nostettavan summan verran rahaa, ja että kortin nostoraja ei täyty nostettavasta summasta. Järjestelmä vähentää tilin saldoa käyttäjän nostaman summan verran.
6. Järjestelmä kysyy käyttäjältä, haluaako tämä tulostaa tapahtumasta kuitin.
7. Käyttäjä valitsee kuitin tulostuksen.
8. Järjestelmä pyytää käyttäjää ottamaan kortin pois.
9. Käyttäjä ottaa pankkikortin pois automaatista.
10. Järjestelmä antaa rahat ja kuitin. Käyttötapaus päättyy.

Tähän käyttötapaukseen voi liittyä esimerkiksi seuraavanlaisia poikkeuksia:

4.1 Käyttäjän tilillä ei ole nostettavaa summaa rahaa. Järjestelmä ilmoittaa käyttäjälle, että tilillä ei ole tarpeeksi saldoa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.

4.2 Kortin nostoraja ylittyy käyttäjän syöttämästä summasta. Järjestelmä ilmoittaa käyttäjälle, että tililtä ei voida nostaa niin suurta summaa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.

6.1 Kuittipaperi on loppunut automaatista. Järjestelmä ilmoittaa käyttäjälle, että kuittipaperi on loppunut eikä sitä voida tulostaa. Järjestelmä kysyy näytetäänkö saldo näytöllä. Käyttäjä valitsee saldon tulostamisen näytölle. Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.

6.1.1 Käyttäjä ei valitse saldon tulostamista näytölle. Käyttötapaus jatkuu kortin poistamisella automaatista.

7.1 Käyttäjä ei valitse kuitin tulostamista paperille, vaan saldon näyttämisen näytöllä. Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.

Poikkeukset voidaan numeroida yllä olevan esimerkin tavoin niin, että poikkeuksen ensimmäinen luku ilmaisee mihin käyttötapauksen askeleeseen poikkeus liittyy ja toinen luku ilmaisee, kuinka mones kyseiseen askeleeseen liittyvä poikkeus kyseinen poikkeus on. Jos poikkeus ei liity mihinkään tiettyyn perussuorituksen askeleeseen tai se liittyy useampaan askeleeseen, on poikkeuksen alkunumerona nolla. Käyttötapauksen laajennus kuvattaisiin sanallisessa kuvauksessa perussuorituksen jälkeisinä askeleina. Ensimmäinen askel laajennuksessa kuvaa, mistä perussuorituksen askeleesta laajentava käyttötapaus lähtee. Viimeinen askel kuvaa, missä vaiheessa käyttötapausten suoritus palaa samalle polulle, mikäli suorituspolut kohtaavat ennen käyttötapauksen päättymistä.

Toiminnon sanallisen kuvauksen lisäksi käyttötapauksen kuvaukseen voidaan liittää myös muita sanallisia kuvauksia. Esimerkiksi Bertrand Meyer ehdotti, että käyttötapauksien sanallisiin kuvauksiin lisättäisiin toiminnan alku- ja loppuehdot (Meyer, 1994). Alkuehdot kertovat, missä tilassa järjestelmä tulee olla, ennen kuin tietty käyttötapaus voidaan suorittaa. Loppuehdot kertovat missä tilassa järjestelmän tulee olla käyttötapauksen onnistuneen suorituksen jälkeen. Esimerkiksi käyttötapauksen *Nosta rahaa* alkuehtona voisi olla, että käyttäjä on syöttänyt oikean automaattikortin ja tunnusluvun automaattiin onnistuneesti käyttötapauksen *Syötä kortti* avulla. Loppuehtona voi olla, että järjestelmä antaa käyttäjälle rahat sekä kuitin tapahtumasta.

Alistair Cockburn otti käyttöön tavoitekeskeisen (*goal-based*) lähestymistavan käyttötapausanalyysiin, jolloin käyttötapauksen kuvaukseen määritellään käyttäjän tavoite, joka saavutetaan askel askeleelta käyttäjän ja järjestelmän välisen vuorovaikutuksen avulla (Cockburn, 1997). Askeleet muodostavat itsessään minikäyttötapauksia, joilla voi olla omat tavoitteensa (*sub-goal*). Tavoitekeskeinen lähestymistavan avulla halutaan korostaa, että järjestelmä toteuttaa nimenomaan käyttäjän haluamia vaatimuksia. Cockburnin mukaan käyttötapaus on kokoelma eri skenaarioita, joita yhdistää yhteinen tavoite ja hän jakaa käyttötapaukset käyttäjän kannalta koottuihin tavoitteisiin, käyttäjän tavoitteisiin, alitavoitteisiin ja vuorovaikutusdialogeihin (Cockburn, 1997). Esimerkiksi yllä esitellyssä pankkiautomaatin *Nosta rahaa* käyttötapauksessa tavoitteena voisi olla rahan nostaminen käyttäjän tililtä. Tavoite on käyttäjän ensisijainen tavoite, eli tasoltaan *Käyttäjän tavoite*, mutta ennen sitä käyttäjän tulee tunnistautua automaatille automaattikortin ja tunnusluvun avulla, joten käyttötapaus *Syötä kortti* on tällöin käyttäjä *alitavoite*.

Käyttötapaukset yhdistetään usein oliopohjaiseen ohjelmistokehitykseen, mutta niiden käyttö ei sido järjestelmän kehitystä oliopohjaiseen toteuttamiseen. Niitä käytetään myös muun tyyppisten järjestelmien kehittämisessä käyttäjien vaatimusten tallentamiseen järjestelmän kehityksen alkuvaiheessa. Käyttötapaukset itsessään eivät kuitenkaan ole käyttäjäkeskeistä järjestelmäkehitystä siinä merkityksessä, jossa käyttäjäkeskeinen kehittäminen kuvattiin luvussa 2. Käyttötapauksia voidaan käyttää apuna käyttäjäkeskeisessä kehittämisessä käyttäjien ja kehittäjien välisenä yhteisenä kielenä, mutta ne eivät ole käyttäjäkeskeisen kehittämisen edellytys eikä niiden käyttö järjestelmän kehittämiseksi takaa automaattisesti käyttäjäkeskeistä tai käytettävää järjestelmää.

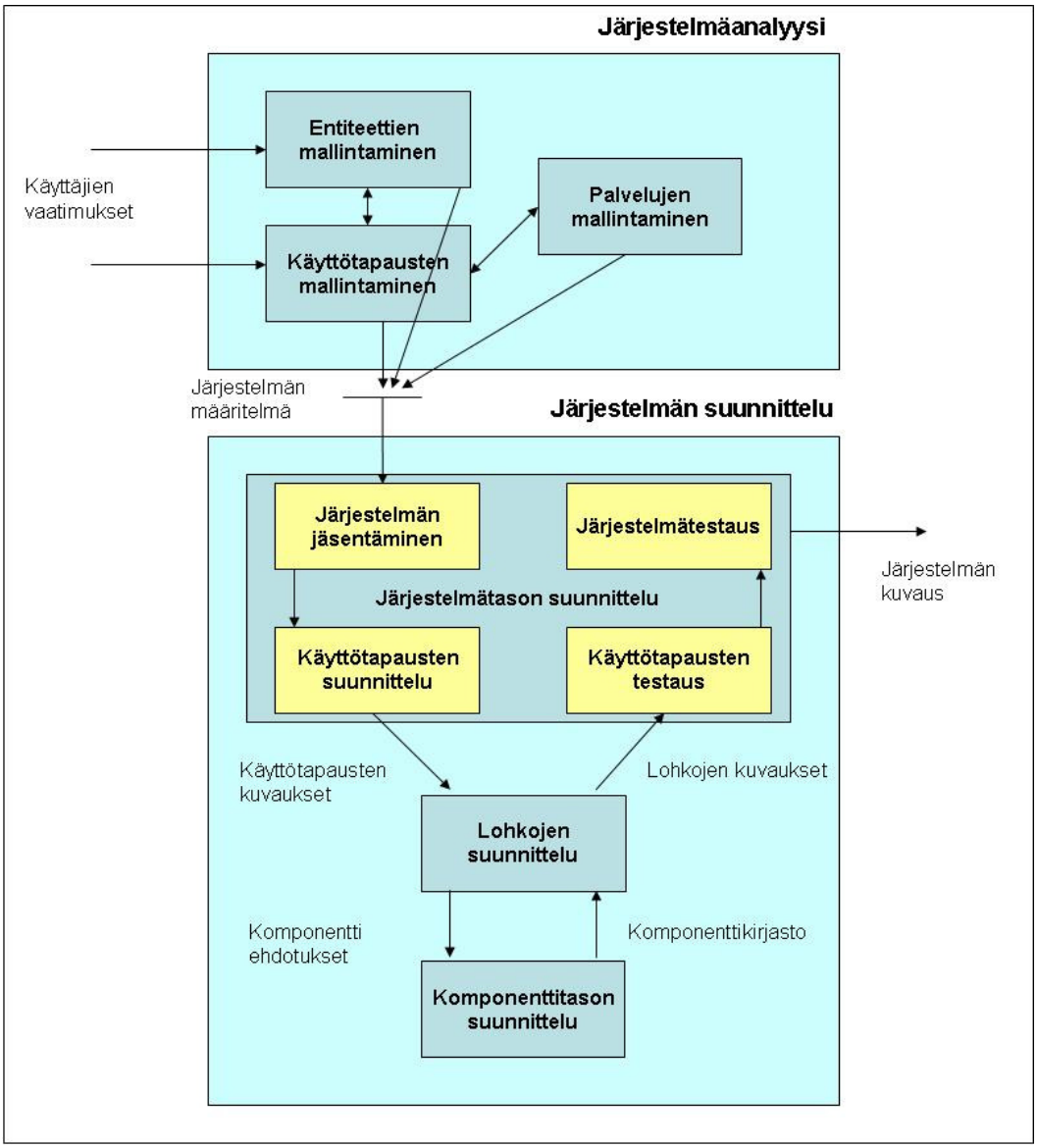
3.1 ObjectOry

ObejctOry on Ivar Jacobsonin kehittämä oliopohjainen menetelmä isojen järjestelmien kehittämiseksi oliopohjaisesti (Jacobson, 1987). ObjectOry:ssa järjestelmän ajatellaan koostuvan lohkoista (*blocks*), jotka taas koostuvat yhdestä tai useammasta komponentista (*component*). Järjestelmän kehityksen aikana asiakkaan vaatimuksista muodostetaan järjestelmän kuvaus, joka esitetään ohjelmakoodin avulla formaalisti. Järjestelmän kuvaus ja asiakkaan vaatimukset kuvaavat järjestelmän toimintaa eri tavoin, mutta ne ovat kuitenkin toisiinsa liittyvät mallit, sillä järjestelmän kuvauksen pitää toteuttaa asiakkaiden vaatimukset (Jacobson, 1987).

Järjestelmän kehitys sisältää useita muodonmuutoksia, joissa järjestelmä muuntuu asiakkaan vaatimuksista ohjelmakoodiksi. Koska ohjelman kirjoittaminen suoraan vaatimusten pohjalta on varsinkin isojen järjestelmien kohdalla käytännössä mahdotonta, tulee järjestelmää kehittää vähitellen kohti ohjelmakoodia (Jacobson, 1987). ObjectOry:n tarkoituksena on jakaa monimutkainen isojen järjestelmien kehitysprosessi pienemmiksi askeleiksi, joiden avulla luodaan kehitettävästä järjestelmästä aina uusi malli, joka auttaa kehittäjiä tarkentamaan vaatimuksia vähitellen kohti hyvin rakennettua järjestelmää (Jacobson, 1987).

Järjestelmän kehitys jaetaan ObjectOry:ssä kahteen osaan: järjestelmän analyysiin (*System Analysis*) ja järjestelmän suunnitteluun (*System Design*) (Jacobson, 1987). Analyysin tarkoituksena on määritellä järjestelmän toiminnallisuus ihanteellisissa olosuhteissa. Järjestelmäanalyysi saa

syötteenä käyttäjien vaatimukset, ja se tuottaa yhteistyössä asiakkaan kanssa tarkan määritelmän kehitettävästä järjestelmästä. Järjestelmän suunnittelun aikana kehittäjät muodostavat järjestelmän tarkan määritelmän pohjalta verifioidun kuvauksen järjestelmästä. Järjestelmän suunnittelun aikana kehitettävälle järjestelmälle määritellyt ihanteelliset olosuhteet korvataan valitulla toteutusympäristöllä. ObjectOry:n sisältämät vaiheet ja tuotokset on esitetty kuvassa 9.



Kuva 9: ObjectOry –menetelmä oliopohjaisten järjestelmien kehittämiseksi

Kuten kuvassa 9 on esitetty, analyysin aikana vaatimuksista luodaan järjestelmän tarkka määritelmä kolmen eri mallin avulla. Käyttäjien vaatimusten perusteella kuvataan järjestelmän entiteetit, jotka ovat kehitettävän järjestelmän kiinnostuksen kohteina (Jacobson, 1987). Käyttötapausmalli kuvaa järjestelmän eri näkökulmista mustana laatikkona käyttötapausten avulla (Jacobson, 1987). Palvelumalli on käsitteellinen malli, joka koostuu entiteeteistä ja käyttötapauksista sekä kuvaa järjestelmän toiminnallisuuden puolivirallisessa muodossa.

Kuvasta 9 nähdään, että analyysin tuottama määritelmä järjestelmästä toimii syötteenä suunnittelulle, joka myös sisältää kolmen eri mallin kehittämisen. Suunnittelun aikana toteutetaan järjestelmätason-, lohkojen ja komponenttitason suunnittelumallit (Jacobson, 1987). Järjestelmätason suunnittelumallissa entiteettejä ja palveluja käytetään eri ohjelman lohkojen tunnistamiseksi ja käyttötapauksia prosessien suunnitteluun.

Käyttötapausten kuvausten perusteella kehittäjät muodostavat lohkojen suunnittelun aikana ehdotukset tulevista komponenteista. Komponenttisuunnitelman pohjalta kehitetään komponenttitason malli järjestelmän sisältämistä komponenteista. Tämän kolmannen vaiheen tuotoksena on komponenttikirjasto, jota käytetään eri lohkojen kokoamiseen. Lohkojen suunnittelumallin tuloksena on kuvaus järjestelmän eri lohkoista. Kuvauksia käytetään järjestelmätason mallissa käyttötapausten testaamiseen. Käyttötapaustestauksen tuloksena saadaan testauksen kuvaukset järjestelmätestauksen syötteeksi. Kun järjestelmätestaus on suoritettu, tuottaa suunnitteluvaihe tulokseksi järjestelmän verifioidun kuvauksen.

ObjectOry on varhainen malli oliopohjaisten järjestelmien kehittämiseksi käyttötapausten avulla. Kuten Jacobsonkin toteaa, ObjectOry on varsin pelkistetty esitys järjestelmän kehittämisestä: todellisuudessa eri mallien kehitystyön tulee olla rinnakkaista, vaikka se tässä mallissa on kuvattu peräkkäisenä ja todennäköisesti tuotteen elinkaaren aikana vastaavia kehityssyklejä on useita (Jacobson, 1987). Tässä mallissa on kuitenkin jo huomioitu se, että käyttötapaukset eivät ole vain menetelmä asiakkaan vaatimusten määrittämiseksi, vaan niitä voidaan käyttää myös myöhemmässä kehitysvaiheessa, esimerkiksi testaamisen lähtökohtana.

3.2 OMT

OMT (*Object Modeling Technique*) on menetelmä oliopohjaisten järjestelmien kehittämiseen ja se sisältää graafisen ilmaisutavan oliopohjaisille käsitteille (Rumbaugh et al., 1991). Menetelmän avulla järjestelmää kehitetään mallintamalla järjestelmän kohdealue ja sen jälkeen lisäämällä malliin järjestelmän suunnittelun aikana toteutukseen liittyvät yksityiskohdat. Menetelmä koostuu neljästä vaiheesta, jotka ovat analyysi, järjestelmän suunnittelu, olioiden suunnittelu ja järjestelmän toteutus.

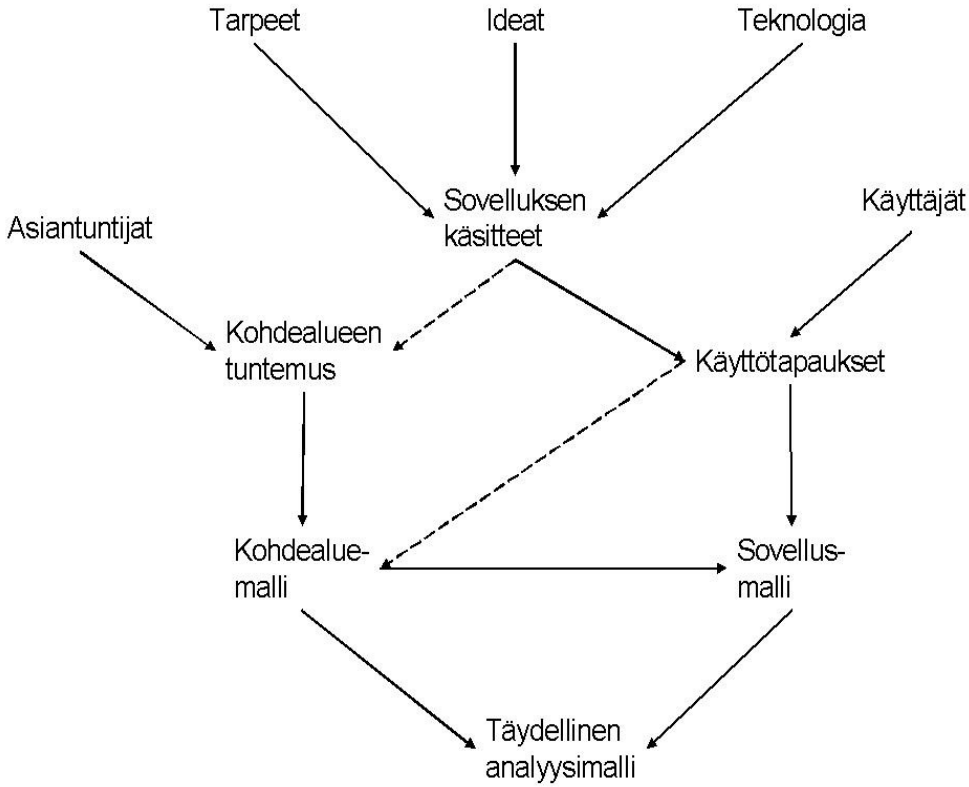
Analyysivaiheessa määritellään, millainen kehitettävän järjestelmän kohdealue on ja mitkä sen tärkeimmät ominaisuudet ovat. Analyysin tuloksena syntyy abstrakti kuvaus kehitettävästä järjestelmästä. Analyysin aikana järjestelmä kuvataan kolmen eri mallin avulla, jotka ovat oliomalli (*object model*), dynaaminen malli (*dynamic* model) sekä toiminnallinen malli (*functional model*) (Rumbaugh et al., 1991). Oliomalli kuvaa järjestelmän staattiset oliot ja niiden väliset staattiset suhteet. Dynaamisessa mallissa kuvataan järjestelmän muuttuvat osat ja sen avulla kuvataan myös järjestelmän sisäistä kontrollia. Toiminnallinen malli kuvaa tiedon muodonmuutokset järjestelmän sisällä.

Oliomalli muodostetaan tunnistamalla kohdealueen oliot ja kuvaamalla niiden *attribuutit* eli ominaisuudet sekä olioiden väliset suhteet. Oliomallia yksinkertaistetaan muodostamalla olioille hierarkkinen järjestys käyttäen perintää. Oliomallia testataan skenaariokuvausten avulla eri suorituspolkujen testaamiseksi. Lopulta olioista muodostetaan luokkakaavio ja jaetaan luokat moduuleihin. Dynaamista mallia varten tunnistetaan olioiden väliset tapahtumat ja muodostetaan sekvenssikaaviot jokaiselle havaitulle skenaariolle. Lisäksi dynaaminen malli sisältää tilakaaviot kaikille niille olioille, jotka sisältävät tärkeää, dynaamista toimintaa. Toiminnallinen malli kuvaa järjestelmän syötteitä ja tulosteita tietovuokaavioiden avulla. Toiminnallinen malli koostuu tietovuokaavioista ja tiedon rajoitteiden kuvauksista.

Järjestelmän suunnittelun aikana suunnittelijat tekevät korkean tason päätökset järjestelmän arkkitehtuurista. Kehitettävä järjestelmä jaetaan suunnittelun aikana alajärjestelmiin analyysin ja valitun arkkitehtuurin perusteella. Suunnittelu vaiheessa päätetään, mitkä ominaisuudet ja suoritus-

kykyvaatimukset ovat tärkeimpiä. Oliosuunnittelun tarkoituksena on määrittää, mitä tietorakenteita ja algoritmeja jokaisen toteutettavan luokan tulee sisältää. Toteutusvaiheessa suunnitellut luokat ja niiden suhteet toteutetaan jollakin ohjelmointikielellä. Ohjelmoinnin tulisi olla varsin pieni ja mekaaninen osa toteutusta, koska kaikki suurimmat päätökset on jo tehty suunnittelu vaiheessa (Rumbaugh et al., 1991).

Kuvassa 10 on esitetty OMT analyysimalli ja sen sisältämät vaikutussuhteet niin kuin Rumbaugh kuvasi ne vuonna 1994 (Rumbaug, 1994). Järjestelmän kehitys pohjautuu tarpeisiin, ideoihin ja teknologiaan, joiden perusteella muodostetaan sovellukseen liittyvät käsitteet.



Kuva 10: OMT analyysimalli (Rumbaugh, 1994).

Käsitteiden pohjalta luodaan käyttötapaukset yhdessä käyttäjien kanssa. Käsitteistä ja kohdealueen asiantuntijoiden tietämyksestä muodostuu kohdealueen tuntemus, joka on merkityksellinen ilman kehitettävää järjestelmääkin. Kohdealueen tuntemuksen ja käyttötapausten pohjalta voi-

daan muodostaa kohdealuemalli, joka kuvaa kohdealueen ja ne ongelmat, jotka kehitettävän järjestelmän avulla tahdotaan ratkaista. Kohdealue malli kuvaa käyttötapausten avulla järjestelmän aikariippuvaista, dynaamista toimintaa.

Käyttötapausten ja kohdealueen mallin pohjalta voidaan muodostaa sovellusmalli, jonka avulla voidaan luoda tietokoneellinen ratkaisu kohdealueen ongelmiin, kuten kuvasta 10 nähdään. Sovellusmalli sisältää toteutukseen liittyvää tietoa, mutta sen tulee pohjautua kohdealue malliin toteuttamalla se. Sovellusmallin sisältämät oliot liittyvät järjestelmän toteutukseen eikä niitä siis ole olemassa sovelluksen kohdealueella. Sovellusmallista ja kohdealue mallista muodostuu täydellinen järjestelmämalli, joka toteutetaan ohjelmakoodiksi.

Rumbaughn mukaan käyttötapaukset ovat ensisijaisesti menetelmä, jonka avulla voidaan saada selville järjestelmän vaatimukset käyttäjäkeskeisestä näkökulmasta analyysivaiheessa (Rumbaugh, 1994). Analyysivaiheessa käyttötapauksia voidaan käyttää apuna myös olioiden ja niiden välisten suhteiden selvittämiseen. Hänen mielestään ne eivät kuitenkaan ole tärkeitä analyysivaiheen jälkeen. OMT-menetelmä keskittyy kuvaamaan kehitysprosessia kehittäjien näkökulmasta, joten käyttötapauksia käytetään tässä menetelmässä nimenomaan vaatimusten määrittämiseen ja tallentamiseen eikä käyttäjäkeskeiseen kehittämiseen.

3.3 Unified Process

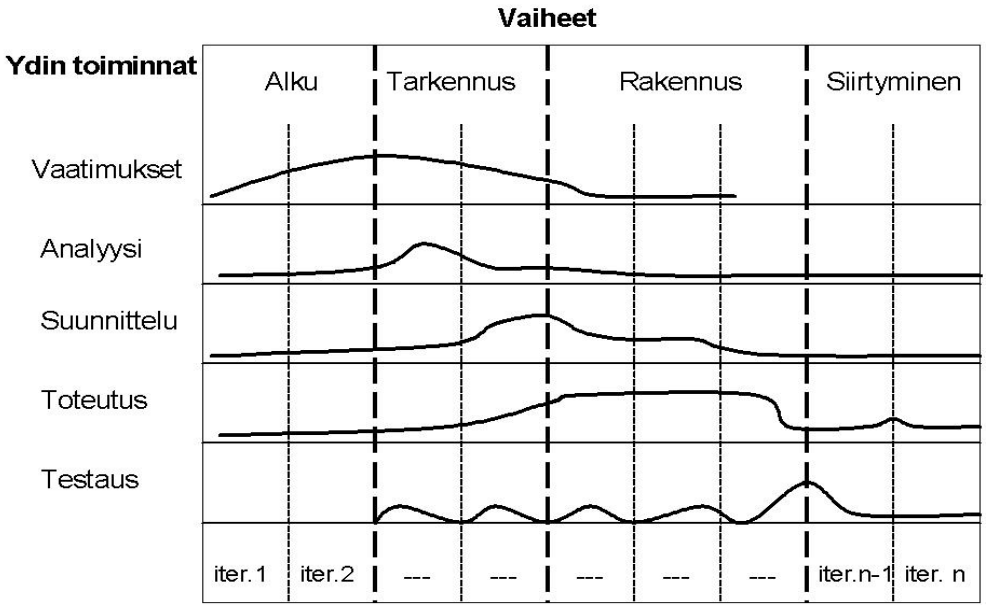
Unified Process (*UP*) on käyttötapauspohjainen, arkkitehtuurikeskeinen, iteratiivinen ja inkrementaalinen menetelmä isojen ja monimutkaisten järjestelmien kehittämiseksi (Jacobson et al., 1998). Unified Process perustuu Jacobsonin ObectOry –menetelmään sekä sitä seuranneeseen Rational Objectory Process –menetelmään (Jacobson et al., 1998). Unified Process kuvaa koko järjestelmän kehityksen toimintoina, joiden avulla käyttäjien vaatimuksista muodostetaan valmis järjestelmä. Unified Process:ssa järjestelmä kootaan komponenteista ja niiden välisistä, hyvin määritellyistä rajapinnoista. Järjestelmän kuvaamiseen eri kehityksen vaiheissa käytetään graafista mallinnuskieltä UML:ää (*Unified Modeling Language, UML*).

Unified Process on käyttötapauspohjainen menetelmä, koska kehitysprosessi koostuu toiminnoista, joissa käyttötapauksia käytetään koko ajan hyödyksi järjestelmän kehittämiseksi (Jacobson et al., 1998). Käyttötapausten avulla määritellään järjestelmän toiminnalliset vaatimukset, jonka jälkeen käyttötapaukset täsmennetään, suunnitellaan ja toteutetaan. Lopulta testaajat käyttävät käyttötapauksia testitapausten määrittämiseen.

Järjestelmäarkkitehtuuri kuvaa, miten järjestelmä on jaettu alajärjestelmiin, osiin tai lohkoihin sekä mitä eri tehtäviä tai mikä tarkoitus eri osajärjestelmillä on. Järjestelmäarkkitehtuurin tarkoituksena on antaa järjestelmän kehittäjille kokonaiskuva kehitettävästä järjestelmästä korostamalla tärkeimpiä ominaisuuksia vähentämällä yksityiskohtien määrää. Arkkitehtuurin tulee ottaa huomioon kaikki nyt ja tulevaisuudessa toteutettavat käyttötapaukset, mutta toisaalta käyttötapausten tulee sopia valittuun arkkitehtuurimalliin. Tästä seuraa se, että sekä arkkitehtuuri että käyttötapaukset kehittyvät järjestelmän kehityksen edetessä rinnakkaisesti (Jacobson et al., 1998).

Unified Process:ssa järjestelmää kehitetään inkrementaalisesti ja iteratiivisesti. Järjestelmän elinkaari syntymästä kuolemaan koostuu sykleistä. Jokainen sykli koostuu neljästä vaiheesta, jotka ovat alku, tarkennus, rakennus ja siirtyminen. Jokainen vaihe jakautuu yksittäisiin iteraatioihin. Iteraation aikana tunnistetaan ja määritellään kyseisen kehitysvaiheen merkitykselliset käyttötapaukset ja luodaan valitun arkkitehtuurin perusteella suunnitelma käyttötapausten toteuttamisesta. Käyttötapausten toiminnallisuus toteutetaan komponentteina, jotka verifioidaan käyttötapausten pohjalta. Jokaiselle vaiheelle on olemassa päätös, joka määräytyy sen mukaan, että tietyt tuotokset, esimerkiksi tietyt dokumentit tai mallit, ovat valmiina. Jos iteraation lopputulos toteuttaa sille asetetut tavoitteet, voidaan jatkaa seuraavaan iteraatioon.

Kuvassa 11 on esitetty yksi tuotteen elinkaaren sykli. Kuvan ylälaidassa on esitetty syklin neljä vaihetta ja vasemmassa reunassa prosessiin kuuluvat viisi ydin toimintaa järjestelmän toteuttamiseksi. Jokainen vaihe jakautuu useampaan pienempään aliprosessiin, jotka on kuvattu kuvan alareunassa iteraatioina. Kuvan vaakasuorat kaariviivat kuvaavat eri ydintoimintojen sijoittumista ja laajuutta eri syklin vaiheissa.



Kuva 11: Unified Process:n vaiheet (Jacobson et al., 1998).

Unified Process on siis iteratiivinen ja inkrementaalinen menetelmä järjestelmien kehittämiseksi. Nykyisten monimutkaisten ja laajojen järjestelmien kehittämiseksi alati muuttuvien vaatimusten keskellä vähittäisen ohjelmistokehityksen avulla voidaan yrittää minimoida kehitysprojektin epäonnistumisen riskiä sekä nopeuttaa näkyvän tuloksen aikaansaamista. Lisäksi Unified Process käyttää käyttötapauksia koko kehityksen ajan jalostaen vaatimukset ohjelmaksi niiden avulla.

RUP (*Rational Unified Process*) on Ratio Groupin ja nykyisin IBM:n (2006) kehittämä menetelmä oliopohjaisten järjestelmien kehittämiseksi käyttäen UML-mallinuskieltä. RUP-menetelmä on Unified Prosess:in kaupallinen versio, jonka soveltamista varten IBM on kehittänyt erilaisia ohjelmistoja, kuten Rational Rose ja ClearCase (IBM, 2006).

4. Käyttäjäkeskeisen ja käyttötapauspohjaisen kehittämisen yhdistäminen

Käyttäjäkeskeinen ohjelmistotuotanto (*Usability engineering*) ja perinteinen ohjelmistotuotanto (*Software engineering*) ovat kehittyneet erillään toisistaan, josta on seurannut se, että perinteiset ohjelmistotuotannon elinkaarimallit keskittyvät enemmän ohjelman rakenteelliseen oikeellisuuteen ja tehokkuuteen, kun taas käyttäjäkeskeiset elinkaaret itse käyttöliittymän oikeellisuuteen ja käytettävyyteen tiettyjen käyttäjäryhmien osalta. Vaikka nykyisiä ohjelmistotuotannon elinkaarimalleja on kehitetty niin, että ne sisältävät tiivistäkin yhteistyötä käyttäjien kanssa lähinnä oikeiden vaatimusten keräämiseksi ja varmistamiseksi, ne eivät kuitenkaan yleensä huomioi käyttäjiä käytettävyyden näkökulmasta. Toisaalta käyttäjäkeskeiset elinkaarimallit eivät kiinnitä juurikaan huomiota ohjelmistotuotannon perinteisiin prosesseihin, kuten ohjelmistoarkkitehtuurin suunnitteluun tai testausmenettelyihin muuten kuin käytettävyyden osalta. Käytännössä ohjelmiston kehittämiseksi tarvittaisiin molempien elinkaarimallien yhdistämistä.

4.1 ACUDUC

ACUDUC (*Approach Centered on Usability and Driven by Use Cases*) on prosessikeskeinen viitekehys, joka pyrkii yhdistämään perinteiset ja käyttäjäkeskeiset elinkaarimallit. Näistä perinteistä elinkaarimallia edustaa kohdassa 3.3 esitelty Unified Process ja käyttäjäkeskeistä järjestelmän kehittämistä kohdassa 2.3 esitelty RESPECT-menetelmä. ACUDUC-menetelmä yhdistää UP-menetelmän perinteisen käyttötapausanalyysin ja RESPECT-menetelmän käyttäjäkeskeinen vaatimusmäärittelyprosessin (Seffah et al., 2001). ACUDUC lähestymistapa pyrkii siis yhdistämään käyttäjäkeskeisen kehittämisen ja perinteisen ohjelmistotuotannon prosessit käyttötapauspohjaiseksi ja käyttäjäkeskeiseksi vaatimusmäärittelymenetelmäksi.

Käyttötapauskaaviot ja käyttötapausten sanalliset kuvaukset kuvaavat kehitettävän järjestelmän toiminnalliset vaatimukset ja kehittäjien toteuttamista varten tarvitsemat yksityiskohdat. RESPECT-mentelmä puolestaan kuvaa perinteisiä ohjelmistotuotannon prosesseja laajemmin järjestelmän ei-toiminnallisia vaatimuksia lisäten tietoa, joka parantaa käyttäjien ymmärrystä kehitet-

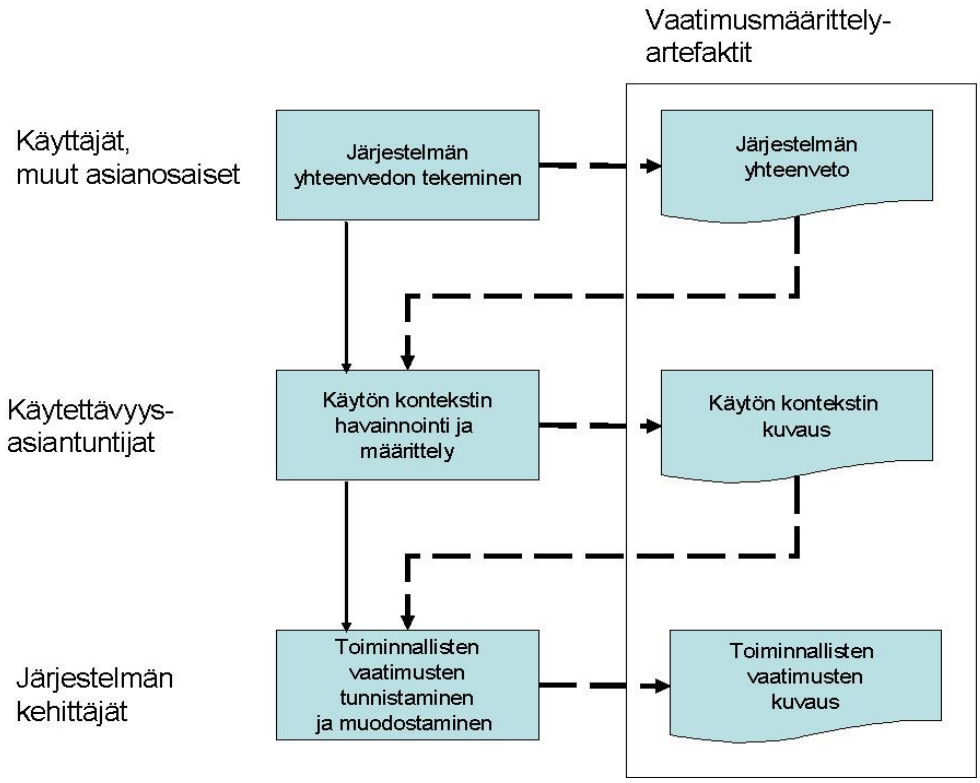
tävästä järjestelmästä (Seffah et al., 2001). Taulukossa 1 on esitetty, kuinka ACUDUC menetelmä yhdistää käyttäjäkeskeiset ja käyttötapauspohjaiset vaatimusmäärittelyprosessit.

Taulukko 1: ACUDUC- menetelmän muodostuminen käyttäjäkeskeisestä- ja käyttötapauspohjaisesta vaatimusmäärittelystä (Seffah et al., 2001).

<table>
<tr><th>Käyttäjäkeskeinen vaatimusmäärittelyprosessi</th><th>Käyttötapauspohjainen vaatimusmäärittelyprosessi</th></tr>
<tr><td>RESPECT antaa täydellisen kuvauksen käytön kontekstista ja sisältää käytettävyystavoitteet ja -mittarit sekä käyttäjien erityisominaisuuksien, tehtävien sekä fyysisen, teknisen ja kohdeorganisaation ympäristön analysoinnin.</td><td>Käytännössä käyttötapauksia on käytetty järjestelmän toimintojen ja erityispiirteiden kuten teknisten mahdollisuuksien ja rajoitteiden kuvaamiseen.</td></tr>
<tr><td colspan="2">1. ACUDUC määrittelee käytön kontekstin ja toiminnalliset vaatimukset kahtena erillisenä näkymänä samasta vaatimuksesta.
2. Toiminnallinen näkymä on kokoelma artefakteja, jotka kuvaavat toiminaalliset vaatimukset. Käytettävyysnäkymä on kokoelma artefakteja, jotka kuvaavat käytön kontekstin sekä käytettävyystavoitteet ja -mittarit.</td></tr>
<tr><td>Käytön konteksti kuvataan epäformaalissa muodossa, joka käyttäjien on helppo ymmärtää. Epäformaalista muodosta johtuen kuvaukset eivät kuitenkaan ole johdonmukaisia ja aiheuttavat epäselvyyksiä kehittäjille.</td><td>Käyttötapauskaavioiden ja sanallisten kuvausten osittain muodollinen esitystapa on kehittäjille tuttu ja helppo ymmärtää. Se voi myös tukea vaatimusten automaattista validointia.</td></tr>
<tr><td colspan="2">3. ACUDUC korostaa, että saman asian kuvaaminen eri tavoin voi edistää vaatimusmäärittelytyöhön osallistuvien henkilöiden keskinäistä kommunikaatiota.
4. Käytön kontekstiin liittyvät artefaktit tulisi kuvata tekstimuodossa. Toiminnallisia vaatimuksia tulisi käsitellä käyttötapausten avulla.
5. ACUDUC:n tulisi toimia kommunikaation välittäjänä kehittäjien, käyttäjien ja muiden sidosryhmien sekä käytettävyysasiantuntijoiden välillä.</td></tr>
<tr><td>Käytettävyysasiantuntijat käyttävät käytön kontekstia tärkeänä käytettävyystestauksen syötteenä.</td><td>Järjestelmän suunnittelijat käyttävät toiminnallisia vaatimuksia suunnittelutyön perustana.</td></tr>
<tr><td colspan="2">6. ACUDUC sisältää toimintoja kaikkien vaatimusten eheyden ja yhdenmukaisuuden arvioimiseksi ja validoimiseksi sekä käytettävyyden että kehittäjien näkökulmasta. Toimintojen pohjalta luodaan käytettävyys ja toteutussuunnitelmat.</td></tr>
</table>

ACUDUC prosessi sisältää neljä askelta, joiden avulla voidaan yhdistää käyttäjäkeskeinen vaatimusmäärittelyprosessi ja käytettävyystekniikat käyttötapauspohjaiseen vaatimusmäärittelyyn (Seffah et al., 2001). Prosessiin kuuluu kehitettävän järjestelmän yhteenvedon tekeminen käyttäjien näkökulmasta yhdessä käyttäjien kanssa. Tämän jälkeen havainnoidaan ja määritellään käytön konteksti sekä muodostetaan toiminnalliset vaatimukset. Neljäntenä askeleena on sekä toiminnallisten vaatimusten että käytön kontekstiin liittyvien vaatimusten arviointi ja todentaminen.

ACUDUC-prosessin askeleet ja niihin liittyvät tekijät ja avustavat henkilöt sekä askeleiden avulla saavutettavat vaatimusmäärittelyartefaktit on esitetty kuvassa 12.



Kuva 12: ACUDUC askeleiden, vaatimusmäärittelyartefaktien ja eri toimijoiden väliset suhteet ACUDUC prosessissa (Seffah et al., 2001).

Järjestelmän yhteenveto perustuu RESPECT-menetelmän mukaiseen projektin yhteenvetoon. ACUDUC-menetelmä esittelee projektin yhteenvetolomakkeen, johon eri käyttäjäryhmien ja muiden sidosryhmien edustajat vastaavat omasta näkökulmastaan. Käytettävyysasiantuntijat muodostavat vastausten pohjalta yhteenvedon, jonka käyttäjät, sidosryhmät ja kehittäjät yhdessä hyväksyvät. Yhteenveto edustaa eri näkökulmien yksimielisyyttä kehitettävästä järjestelmästä (Seffah et al., 2001).

Käytön kontekstin kuvaus koostuu käyttäjien ja näiden suorittamien tehtävien ominaisuuksien sekä työympäristön kuvauksista (Seffah et al., 2001). Järjestelmän suorita ja epäsuorista käyttäjäryhmistä kuvataan käyttäjien tieto- ja taitotaso, kokemus, koulutus, fyysiset ominaisuudet sekä tottumukset. Käyttäjien järjestelmän avulla suorittamista tehtävistä kuvataan tavoitteet sekä teh-

tävien suoritustiheys ja kesto. Käyttäjien työympäristö jaetaan kuvauksessa organisatorisiin näkökulmiin sekä teknisiin- ja fyysisiin tekijöihin. Lisäksi käytön kontekstin kuvaukseen määritellään järjestelmän käytettävyystavoitteet sekä käytettävyyden mittarit. Käytön kontekstin kuvaamiseen voidaan käyttää RESPECT-menetelmän mukaisia lomakkeita, tai muita käyttäjäkeskeisen kehittämisen menetelmiä.

Toiminnalliset vaatimukset kuvataan ACUDUC-menetelmässä käyttötapausten ja erityisesti käyttötapauskaavioiden avulla. Toiminnallisten vaatimusten kuvaus sisältää käyttötapausten lisäksi järjestelmän toiminnallisuuden, ominaisuuksien sekä rajoitteiden kuvaukset ja käyttöliittymäprototyyppi. Toiminnallisten vaatimusten kuvauksen muodostavat pääasiallisesti järjestelmän kehittäjät. Käyttötapausten sanallisten kuvausten ja käyttötapauskaavioiden lisäksi Seffah et al. ehdottaa, että käyttötapauksista voitaisiin lisäksi dokumentoida lisäinformaatiota käyttäjistä ja tehtävien suoritusympäristöstä esimerkiksi RESPECT -metodin sisältämän valmiiden, tarkoitusta varten kehitettyjen lomakkeiden avulla (Seffah et al., 2001).

Seffah et al. ovat havainneet, että on olemassa kolme periaatetta, jotka voivat huomattavasti parantaa vaatimusmäärittelyprosessia. Ensimmäinen periaate on, että interaktiivisten järjestelmien vaatimukset tulee määritellä kahdella tasolla. Ensimmäinen taso liittyy käytön kontekstin määrittelyyn ja toinen toiminnallisten vaatimusten kuvaamiseen. Seffah et al. ovat valinneet käytön kontekstin määrittelemiseen käytettäväksi RESPECT –menetelmän mukaiset kyselylomakkeet ja toiminnallisten vaatimusten kuvaamiseksi UML:n mukaisen käyttötapausten graafisen esityksen eli käyttötapauskaaviot.

Toinen ACUDUC:n noudattama periaate on, että käytön kontekstiin liittyvien artefaktien listaaminen varmistaa hyvän käytettävyysmäärityksen, ja se voi jopa auttaa toiminnallisten vaatimusten muodostamisessa. Tämä on heidän mukaansa äärimmäisen tärkeää, sillä se voi minimoida vaatimusten epäjohdonmukaisuuden ja parantaa järjestelmän kehittäjien ja käytettävyysasiantuntijoiden välistä kommunikaatiota.

Kolmas periaate on se, että eri toimijoiden luokittelu käyttäjiin ja muihin sidosryhmiin, käytettävyysasiantuntijoihin ja järjestelmän kehittäjiin selventää eri toimijoiden rooleja ja vastuualueita.

Lisäksi Seffah et al. toteavat, että RESPECT –menetelmän mukaiset kyselylomakkeet ovat yksinkertainen ja tehokas tapa ylläpitää yhteisymmärrystä järjestelmää eri näkökulmista tarkastelevien henkilöiden välillä, koska eri rooleissa olevat henkilöt käyttävät järjestelmän kuvaamiseksi erilaisia kuvaustapoja (Seffah et al., 2001).

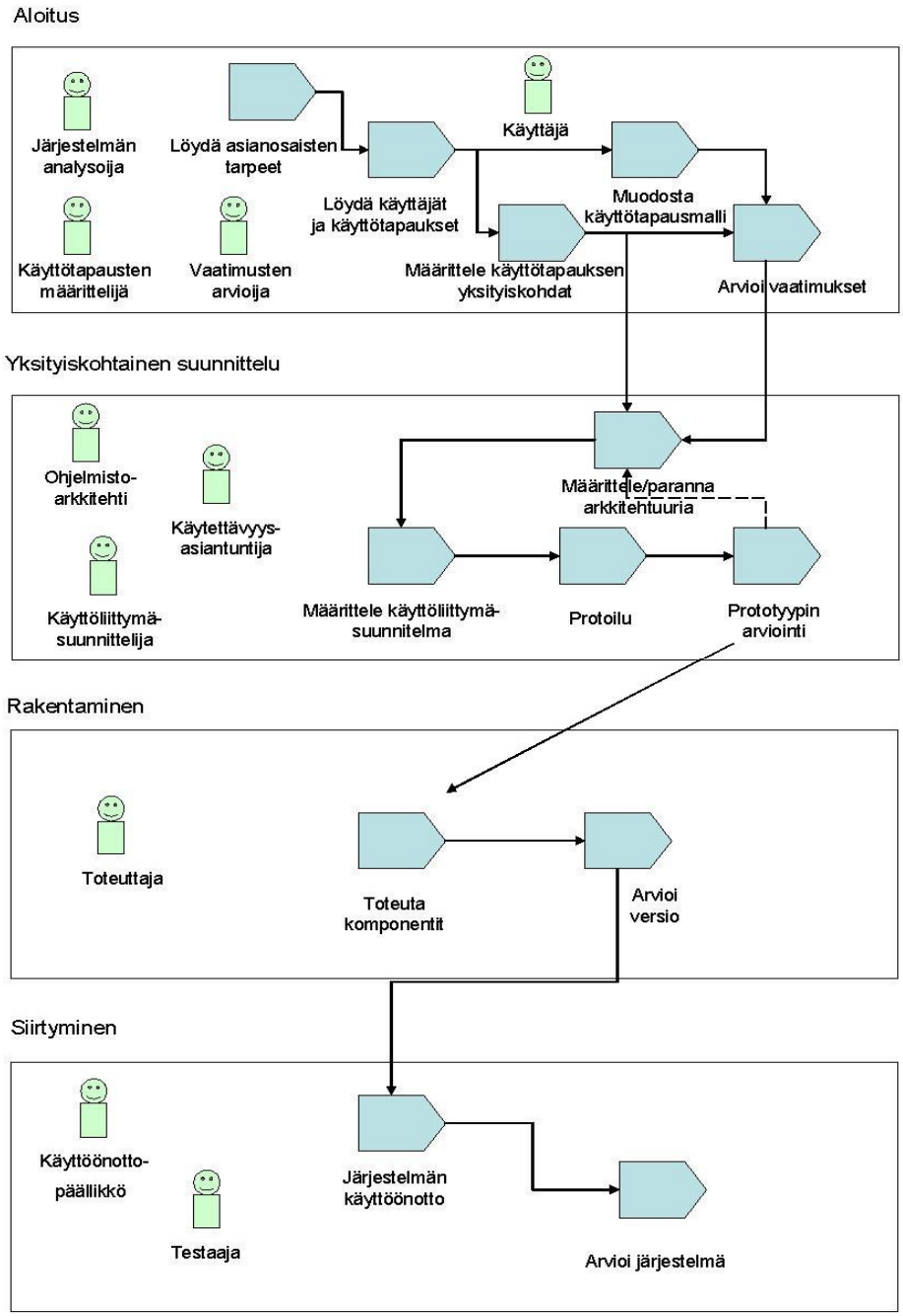
4.2 UPi

UPi on kevyt prosessimalli interaktiivisten järjestelmien kehittämiseksi. UPi yhdistää parhaita menetelmiä perinteisistä ja käyttäjäkeskeisistä elinkaarimalleista (Sousa et al., 2005). Tällä mallilla on kolme tavoitetta, jotka ovat käytettävyys, tuottavuus ja prosessien yhdistäminen. Malli pyrkii auttamaan sekä ohjelmiston kehittäjiä että käytettävyysasiantuntijoita kehittämään yhdessä interaktiivisia järjestelmiä, jotka ovat käytettäviä. Toisena tavoitteena on tehdä ihmiskeskeisestä kehittämisestä kiinteä osa perinteistä ohjelmiston elinkaarimallia lisäämällä järjestelmän kehittäjien ja käytettävyysasiantuntijoiden kommunikointia sekä tehostamalla työskentelyä. Kolmantena tavoitteena mallilla on kuvata käyttöliittymäkehityksen perusteet yhdistämällä käyttäjäkeskeisen kehittämisen periaatteet UPi-mallin esittämään kehitysprosessiin.

Kuvassa 13 on esitetty UPi-prosessin vaiheet ja toiminnot. UPi koostuu toiminnoista, joiden tarkoituksena on auttaa kehittämään käytettäviä järjestelmiä. Toiminnot pohjautuvat RUP–elinkaarimalliin, ja ne ottavat lisäksi huomioon käytettävyysnäkökulmat (Sousa et al., 2005). RUP –elinkaarimalli tyydyttää kehitysorganisaation vaatimukset antamalla järjestelmän kehitykselle rakenteen ja hyväksytyn prosessimallin (Sousa et al., 2005). UPi-prosessin vaiheet ovat prosessin aloitus, yksityiskohtainen suunnittelu, rakentaminen ja siirtyminen, kuten kuvasta 13 huomataan.

Aloitusvaiheessa toimijoina ovat järjestelmän analysoijat, käyttötapausten määrittelijät, vaatimusten arvioijat sekä käyttäjät. Kuten kuvasta 13 nähdään, aloitusvaiheen toimintoja ovat asianosaisten henkilöiden tarpeiden sekä käyttäjien ja käyttötapausten löytäminen, käyttötapausmallin muodostaminen, käyttötapausten yksityiskohtien määrittely ja vaatimusten arviointi (Sousa et al., 2005). Aloitusvaiheessa määritellään ensisijaisten käyttäjien sekä toissijaisten käyttäjien sellaiset ominaisuudet ja työympäristöt, jotka voivat vaikuttaa järjestelmän kehittämiseen sekä määritel-

lään järjestelmän ei-toiminnalliset vaatimukset. Tämän jälkeen määritellään järjestelmän toimijat eli käyttäjät ja muut järjestelmät, jotka tulevat käyttämään järjestelmää sekä toiminnot, jotka muodostuvat suoraan käyttäjien tarpeista.



Kuva 13: UPi prosessi toimintoineen (Sousa et al., 2005).

Käyttötapausmallin muodostamisen yhteydessä hiotaan edellisen toiminnon aikana muodostettuja käyttötapauksien yksityiskohtia, kuten käyttäjien ja käyttötapausten väliset suhteet. Käyttötapausten viimeistelyn yhteydessä kuvataan käyttötapauksien sisältämät tehtävät tehtävämallin avulla ja määritellään käyttötapauksiin liittyvät käytettävyysvaatimukset. Lisäksi käyttötapausten viimeistelyyn kuuluu myös järjestelmän navigoinnin määritteleminen paperiluonnosten avulla perustuen tehtävämallin hierarkkiseen rakenteeseen. Käyttötapausmallia ja viimeisteltyjä käyttötapauksia käytetään vaatimusten arvioimiseen. Vaatimusten arviointi toteutetaan yhdessä tulevien loppukäyttäjien kanssa. Vaatimusten arvioinnissa käytetään apuna edellisen toiminnan aikana muodostettuja paperiluonnoksia.

Yksityiskohtaisen suunnittelun vaiheessa järjestelmän arkkitehdit, käytettävyysasiantuntijat ja käyttöliittymäsuunnittelijat suunnittelevat käyttöliittymän toteutuksen luokkatasolla sekä visuaalisena esityksenä. Tämän jälkeen suunnitelman pohjalta muodostetaan käyttöliittymäprototyyppi, jota arvioidaan suhteessa järjestelmälle asetettuihin käytettävyysperiaatteisiin ja käyttäjien asettamiin vaatimuksiin. Arvioinnin jälkeen voidaan vielä hioa arkkitehtuuria esimerkiksi käyttöliittymäsuunnitelman aikana muodostuneilla rajaluokilla. Yksityiskohtaisen suunnittelun iteratiivisuus on myös esitetty kuvassa 13.

Rakennus vaiheessa järjestelmän toteuttaja toteuttaa edellisessä vaiheessa suunnitellut komponentit luokkineen sekä käyttöliittymän prototyypin pohjalta. Tämän jälkeen suoritetaan julkaistun version arviointi kehitysympäristössä sille asetettujen vaatimusten pohjalta, kuten kuvassa 13 on esitetty. Viimeinen vaihe UPi–prosessissa on siirtymä vaihe, jossa järjestelmä valmistetaan käyttäjille siirtoa varten. Tämän jälkeen järjestelmä arvioidaan aidossa käyttöympäristössä käyttäjien kanssa. Arvioinnin tarkoituksena on testata, vastaako kehitetty järjestelmä käyttäjien asettamia vaatimuksia ja tukeeko järjestelmä käyttäjien tehokasta tehtävien suorittamista. Järjestelmän hyväksymiseen vaikuttaa siis sekä toiminnallisten vaatimusten että käytettävyysvaatimusten toteutuminen.

Sousa et al. ovat todenneet UPi-prosessin käytännönkokemusten perusteella, että pyrittäessä perinteisten ohjelmistotuotantoprosessien ja käyttäjäkeskeisen kehittämisen prosessien yhdistämiseen, ei yhdistettyyn prosessiin tarvitse sisällyttää kaikkia molempien prosessien menetelmiä ja

käytäntöjä (Sousa et al., 2005). Riippuen mallin tavoitteesta, tulee tehdä valintoja eri prosessien välillä tavoitteiden saavuttamiseksi, sillä mitä monimutkaisempi malli on, mitä useampia erilaisia tuotoksia se sisältää ja mitä useampia erilaisia tehtäviä ja rooleja kehittäjillä on, sitä haluttomampia ammattilaiset ovat ottamaan mallin käyttöönsä. Sousa et al. mielestä tärkeintä perinteisten prosessien ja käyttäjäkeskeisten prosessien yhdistämiseksi on, että käytettävä kehitysmalli tukee erityisesti molempien alojen asiantuntijoiden välistä kommunikointia.

4.3 Oleellisten käyttötapausten mallintaminen

Oleellisten käyttötapausten mallintaminen (*Essential Use Case Modelling*) on Larry L. Constantinen ja Lucy A. D. Lockwoodin kehittämä prosessimalli (Constantine, 1995). Menetelmän tarkoituksena ei ole korvata käyttäjäkeskeisen kehittämisen menetelmiä eikä se myöskään ole menetelmä käyttöliittymien kehittämiseen (Constantine, 1995). Prosessi koostuu Jacobsonin esittelemistä, perinteisistä käyttötapauksista yhdistettynä oleelliseen mallintamiseen (Constantine, 1995). Oleellinen mallintaminen (*Essential Modeling*) pyrkii kuvaamaan järjestelmästä vain oleellisen asian, muodostamaan teknologiasta riippumattoman, ideaalisen ja abstraktin kuvan käyttäjien aikomuksista ja järjestelmän tarkoituksesta (Constantine, 1995). Oleellisten käyttötapausten mallintamisen prosessiin kuuluu kolmen itsenäisen mallin kehittäminen. Prosessiin kuuluvat mallit ovat malli käyttäjien rooleista, oleellisten käyttötapausten malli ja käytön kontekstin malli (Constantine, 1995).

Malli käyttäjien rooleista on yksinkertainen lista eri rooleista, jotka käyttäjille on tunnistettu ja se sisältää lyhyen kuvauksen jokaisen roolin ominaispiirteistä. Luvun kolme pankkiautomaatti esimerkistä voidaan käyttötapaukselle *Nosta rahaa* tunnistaa seuraavanlaisia käyttäjiä:

> **Satunnainen käyttäjä**: Nostaa rahaa harvoin, kerran tai kaksi kuussa.
> **Vakio käyttäjä**: Nostaa rahaa usein, jopa päivittäin, mahdollisesti aina saman summan.

Oleelliset käyttötapaukset (*Essential Use Case*) ovat yksinkertaistettuja ja yleistettyjä kuvauksia käyttötapauksista, ja ne kuvaavat yhden täydellisen ja toiminnan kannalta olennaisen vuorovaikutustilanteen käyttäjän ja järjestelmän välillä (Constantine, 1995). Perinteisten käyttötapausten ja

oleellisten käyttötapausten eroa voidaan kuvata seuraavan esimerkin avulla. Luvun 3 pankkiautomaattiesimerkissä käyttötapauksessa *Nosta rahaa* käyttäjän tavoitteena on rahan nostaminen. Kun mietitään kyseisen käyttötapauksen jokaista askelta, voidaan kysyä, miksi käyttäjä tekee niin: miksi käyttäjän tulee syöttää pankkikortti ennen rahan nostamista? Koska sen avulla pankki tunnistaa käyttäjän ja voi kohdistaa toiminnan oikeaan tiliin. Miksi käyttäjä syöttää tunnusluvun? Todistaakseen, että hän on kortin laillinen omistaja ja käyttäjä. Näin saamme selville, että itse asiassa käyttäjän tulee pystyä vain ilmaisemaan kuka hän on; kortti ja tunnusluku ovat yksi keino tähän, mutta muita mahdollisia keinoja olisivat esimerkiksi sormenjälkitunnistus tai äänitunnistus. Kyseisen käyttötapauksen oleellinen käyttötapaus on esitetty kuvassa 14.

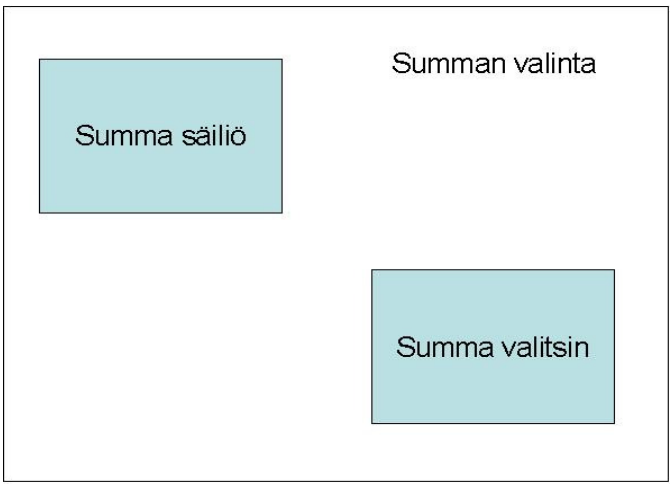
Nosta rahaa	
Käyttäjän toiminto	**Järjestelmän toiminto**
Tunnista itsesi järjestelmälle	Varmista henkilöllisyys
	Tarjoa vaihtoehtoja
Valitse	Anna rahat
Ota rahat ja lähde	

Kuva 14: Oleellinen käyttötapaus *Nosta rahaa.*

Oleelliset käyttötapaukset koostuvat siis kahdesta osasta: käyttäjän toiminnasta ja järjestelmän vastauksesta toimintaan. Nämä kaksi osaa yhdessä kuvaavat oleellisen osan käyttäjän ja järjestelmän välisestä kommunikoinnista käyttötapauksen aikana. Oleelliset käyttötapaukset kuvataan yksinkertaisessa muodossa taulukkona, jossa vasemmassa sarakkeessa on kuvattu käyttäjän toiminto ja oikeassa sarakkeessa järjestelmän vastaus käyttäen käyttäjien ja kehitettävän järjestelmän kohdealueen sanastoa.

Kolmas osa prosessia on käytön kontekstin mallintaminen. Tämän mallin avulla kehittäjät voivat esittää ja manipuloida niitä resursseja ja mahdollisuuksia, jotka käyttöliittymän tulee esittää käyttäjille. Constantinen mukaan käytön kontekstin malli tarjoaa joustavan ja abstraktin mallin kehitettävästä käyttöliittymästä sitomatta suunnittelua ulkonäöllisiin yksityiskohtiin ja tiettyihin käyttöliittymäkomponentteihin (Constantine, 1995). Käytön kontekstin malli ei ole paperiprototyyppi tai käyttöliittymä suunnitelma, vaan se on kokoelma abstrakteja käyttöliittymäelementtejä, jotka kuvaavat tarvittavia tai toivottuja mahdollisuuksia yhtä tai useampaa käyttötapausta varten. Mallit rakennetaan paperille, johon liimaillaan tiettyyn järjestykseen esimerkiksi post-it–lappuja, jotka nimetään tarkoituksen mukaan esimerkiksi säiliöksi, työskentelyalueeksi ja valitsemiksi. Laput

kuvaavat siis käyttöliittymän eri elementtejä, mutta niitä ei suoraan nimetä esimerkiksi tekstialueeksi tai painikkeeksi. Jokainen käytön kontekstin malli kehitetään tukemaan yhtä tai useampaa käyttötapausta niin, että vältetään tarpeetonta toistoa, sopimattomia käyttötapausten yhdistelmiä ja pidetään malli mahdollisimman yksinkertaisena. Kuvassa 15 on pankkiautomaatti esimerkkiin sopiva käytön kontekstin malli summan valinnalle.



Kuva 15: Kontekstuaalinen malli summan valinnasta pankkiautomaatilla.

Oleelliset käyttötapaukset eivät kahlitse järjestelmän suunnittelua mihinkään tiettyyn tapaan toteuttaa käyttäjän ja järjestelmän välinen kommunikointi. Käyttöliittymän suunnittelussa oleelliset käyttötapaukset voivat auttaa kehittäjiä keksimään uudenlaisia, luovempia tapoja järjestelmän ja käyttäjien kommunikointiin (Constantine, 1995). Käytettävyyden näkökulmasta oleellisten käyttötapausten mallintaminen voi auttaa kehittämään yksinkertaisempia ja suoraviivaisempia käyttöliittymiä, joiden avulla käyttäjät voivat tehdä sen, mikä on kaikkein oleellisinta ja näin tukea paremmin käyttäjien työskentelyä. Constantinen mukaan hyvin suunniteltu käyttöliittymän ei tulisi vaatia enempää askelia tai tietoa, kuin mitä oleellisissa käyttötapauksissa on kuvattu.

5. Käyttötapauspohjainen testaaminen

Testaaminen on prosessi, jonka aikana suoritetaan ohjelmaa tarkoituksena löytää virheitä (Myers, 1979). IEEE standardi 610.12-1990 määrittelee testaamisen toiminnaksi, jossa järjestelmä tai komponentti suoritetaan tietyillä ehdoilla, jonka jälkeen suorituksen tuloksia tarkastellaan tai tallennetaan ja arvioidaan jostakin järjestelmän tai komponentin näkökulmasta (IEEE 610, 1990). Testaaminen voi olla myös edellä mainitun toiminnan suorittamista, sarja testitapauksia, sarja testiprosesseja tai sarja testitapauksia ja testiprosesseja (IEEE 610, 1990). Binder määrittelee ohjelmiston testaamisen ohjelmakoodin suorittamiseksi käyttäen syötteiden ja valittujen tilojen yhdistelmiä virheiden löytämiseksi (Binder, 2000).

Testaamisen lisäksi voidaan puhua ohjelman *verifioinnista*, eli ohjelman toiminnan tarkastamisesta ohjelman määrityksiin nähden, *formaalista verifioinnista*, eli ohjelman oikeellisuuden todentamisesta käyttäen jotakin formaalia menetelmää tai *validoinnista*, eli ohjelman suorittamisesta todellisessa ympäristössä tarkoituksena löytää kohtia, joissa ohjelma toimii käyttäjän tarpeiden vastaisesti.

Virhe on virhetoiminto (*failure*), joka on havaittu eroavaisuus ohjelmiston toiminnan ja määritysten välillä, jos määrittely on olemassa ja se on oikein (Kaner, 1988). Tällä määrityksellä halutaan korostaa sitä, että virhe voi olla myös määrityksissä, kuten se hyvin usein onkin. Virhetoiminnon aiheuttaa vika (*fault*), joka johtuu ohjelmiston kehittäjän toimenpiteestä (*error*). Ongelma (*problem*) esiintyy silloin, kun virhetoiminnon aiheutunutta vikaa ei löydetä. Myers määrittelee ohjelmistovirheen tapahtuvan silloin, kun ohjelma ei toimi niin kuin loppukäyttäjä olettaa sen toimivan (Myers, 1976).

Ohjelmiston virheitä voidaan luokitella monella eri tavalla, esimerkiksi sen perusteella miten käyttäjä kokee virheen (esim. vähäinen, ärsyttävä, kohtalainen, estävä, vakava, kohtuuton, katastrofaalinen), missä vaiheessa järjestelmän kehitystä virhe on tapahtunut (esim. määritys-, suunnittelu-, toteutus- tai testausvirhe) tai ohjelmointiteknisesti (esim. puuttuva ehto, ylimääräinen ehto, käsittelemätön ehto). Luokittelun valinta riippuu siitä mitä tarkoitusta varten tietoa kerätään.

Virhe voi olla myös syntaksinen tai semanttinen: syntaksiset virheet liittyvät ohjelmointikielen kielioppivirheeseen, kun taas semanttinen virhe on virhe ohjelman toimintalogiikassa. Yleensä ohjelmointikielten kääntäjät etsivät ja ilmoittavat kaikista syntaksivirheistä. Semanttiset virheet jäävät ihmisten havaittavaksi, ja ne ovatkin vaikeimmin havaittavia. Semanttisen virheen havaitseminen toimintalogiikasta vaatii, että testaaja ymmärtää joko lukemalla ohjelmakoodia tai tarkastelemalla ohjelman toimintaa, miten ohjelma toimii nyt ja vertaamalla toimintaa ohjelman määrityksiin, huomaa ristiriidan määritysten ja toiminnan välillä. Virheen havaitsemisen lisäksi tulee etsiä virheen todellinen syy, joka ei välttämättä ole siinä kohdassa, jossa virhe havaitaan.

Testaamisen tarkoituksena on siis ensisijaisesti etsiä virheitä ohjelmasta. Toissijaisesti testaamisen tarkoitus voi olla osoittaa, että tietyissä olosuhteissa testatut ominaisuudet ja toiminnot toimivat. Testauksen avulla voidaan vähentää julkaisun jälkeistä ylläpidon tarvetta sekä asiakkaan luona suoritettavien jälkiasennusten määrää. Testaaminen voi myös lisätä asiakastyytyväisyyttä, kun virheitä esiintyy harvemmin.

Hyvä testitapaus on sellainen, joka todennäköisesti paljastaa toistaiseksi tuntemattomia virheitä ja onnistunut testitapaus on sellainen, joka paljastaa aikaisemmin tuntemattoman virheen (Myers, 1979). Koska ohjelmakoodin täydellinen testaaminen on käytännössä mahdotonta, tulee testitapausten valintaan kiinnittää erityistä huomiota. Testaukseen käytettävän ajan ja resurssien minimoimiseksi tulisi valita se joukko testitapauksia, jolla on suurin todennäköisyys havaita suurin osa virheistä. Testaamisen tulee olla hyvin suunniteltua ja dokumentoitua niin, että testit ovat toistettavissa ja voidaan arvioida, kuinka kattavaa testaus on ollut.

Testausmenetelmät jaetaan yleensä lasilaatikko (*white box testing* tai *glass box testing*) ja mustalaatikko (*black box testing*) menetelmiin. Oleellinen ero näiden kahden välillä on se, että lasilaatikkomenetelmissä testaajalla on käytettävissä ohjelmakoodi, kun taas mustalaatikko menetelmissä testaaja ei ole kiinnostunut ohjelman sisäisestä toiminnasta tai rakenteesta. Mustalaatikko testauksessa testaaja on kiinnostunut niistä tilanteista, joissa järjestelmä ei toimi määritysten mukaisesti. Lasilaatikkomenetelmien avulla voidaan keskittyä testaamaan ohjelmakoodia jostakin tietystä näkökulmasta.

5.1 Testitapausten muodostaminen käyttötapauksista

Jacobson ehdotti käyttötapausten hyödyntämistä myös ohjelmiston testaamiseen jo esitellessään käyttötapausten käytön järjestelmän analysointia ja suunnittelua varten (Jacobson, 1987). Käyttötapauksia käytettiin ObjectOry –mallissa testaamiseen ja käyttötapausten testaamisen tuloksena saatiin käyttötapausten testikuvaukset, jotka toimivat syötteenä järjestelmätestaukselle (Jacobson, 1987). Vaikka Rumbaughn (1994) mielestä käyttötapaukset eivät ole juurikaan käyttökelpoisia analyysivaiheen jälkeen, on niitä esitetty käytettäväksi myös muuhun kuin analysointi tarkoitukseen, muun muassa testaamiseen.

Yleensä käyttötapauksia hyödynnetään mustalaatikkotestauksessa toiminnallisten testitapausten kehittämiseksi. Käyttötapausten ympärille on kehitetty malleja, joissa käyttötapauksia kehitetään koko elinkaaren ajan ja niitä hyödynnetään esimerkiksi järjestelmätestauksessa (Regnel et al., 2000) ja hyväksymistestauksessa (Kirner et al., 1999). Käyttötapausten avulla voidaan myös testata vaatimusten oikeellisuutta ja yhdenmukaisuutta (Wiegers, 1999).

Kuitenkin Weidenhaupt et al. ovat todenneet tutkimuksessaan, että nykyiset toimintatavat harvoin mahdollistavat skenaarioiden käytön järjestelmätestauksessa (Weidenhaupt et al., 1998). Tutkimuksessa tarkasteltiin skenaarioiden käyttöä yhteensä 12:sta eri ohjelmisto-tuotantoprojektissa, neljässä eri Euroopan maassa. Lähes kaikissa tutkimukseen osallistuneissa projekteissa oltaisi haluttu perustaa järjestelmätestaus vaatimusmäärittelyvaiheessa muodostettuihin käyttötapauksiin, koska käyttötapaukset sisältävät kuvauksen järjestelmän vaatimukset käyttäjien näkökulmasta. Suurin ongelma projekteissa oli kuitenkin se, että järjestelmän kehityksen alkuvaiheessa dokumentoidut skenaariot eivät olleet ajan tasalla enää testausvaiheessa. Tästä syystä Weidenhaupt et al. totesivat, että projekteista puuttui systemaattinen tapa luoda testitapauksia skenaarioista. Lisäksi heidän mielestään kattavien testitapausten muodostaminen käyttötapauksista on ristiriidassa skenaarioiden monimutkaisuuden vähentämisen periaatteen kanssa.

Jacobsonin mukaan käyttötapauksia voidaan käyttää testitapausten muodostamisessa, sillä käyttö-tapausten avulla voidaan määritellä, mitä tulisi testata (Jacobson et al., 1992). Käyttötapausten avulla voidaan muodostaa mustalaatikkotestitapauksia, joissa testataan normaali suorituspolku ja

poikkeukselliset suorituspolut sekä testitapauksia, jotka perustuvat vaatimusmäärittelyyn. Käyttötapauksia voidaan myös käyttää käyttäjädokumentaation testaaminen (Jacobson et al., 1992). Lisäksi käyttötapauksia voidaan käyttää integraatiotestauksessa, sillä käyttötapaus toteutuu usein useamman luokan ja lohkon yhteistoiminnan tuloksena. Kun eri luokat ja lohkot on testattu yksinään yksikkötestaus vaiheessa, voidaan lohkot yhdistää ja testata niiden toimintaa yhdessä suorittamalla integraatiotestaus käyttötapausten avulla.

Suorituspolkujen testaamisen tulisi Jacobsonin mukaan käyttää toiminnallista testausta, jossa järjestelmää käytetään pidempiä aikoja suorittamalla tehtäviä niin kuin ne on tarkoitettu suoritettavaksi (Jacobson et al., 1992). Testaamisen aikana voi tapahtua sellaisia virheitä, joita voi tavalliselle käyttäjälle tapahtua tavallisessa käytössä. Tällaiset testitapaukset testaavat Jacobsonin mukaan järjestelmän luotettavuutta. Testitapaukset, jotka perustuvat vaatimusmäärittelyyn ovat sellaisia testitapauksia, jotka voidaan jäljittää suoraan vaatimusmäärittelyyn. Nämä testitapaukset voivat liittyä toiminnalliseen testaamiseen tai yksittäisten vaatimusten testaamiseen. Käyttäjädokumentaation testaaminen on Jacobsonin mukaan dokumenttien käytettävyystestausta, jossa etsitään eroavaisuuksia dokumentin sisältämän järjestelmän toiminnan kuvauksen ja järjestelmän todellisen toiminnan väliltä.

Binderin mukaan käyttötapauksia tulisi hyödyntää mustalaatikkotestaukseen, esimerkiksi ekvivalenssiosituksen tekemiseen (Binder, 2000). Koska ohjelman saamien syötteiden kaikkien mahdollisten arvojen testaaminen on mahdotonta, tulee testatessa löytää se tietty, pienempi osajoukko syötteitä, joiden avulla testata ohjelmaa (Myers, 1979). Tähän voidaan pyrkiä ekvivalenssiosituksen avulla niin, että syötteistä muodostetaan luokkia sen mukaan, että ohjelma toimii suurin piirtein samalla tavalla luokan jokaisen edustajan kohdalla.

Esimerkiksi luvun 3 pankkiautomaatin käyttötapauksesta *Nosta rahaa* voidaan muodostaa nostettavan rahan osalta yksi kelvollinen ja kolme epäkelpoa luokkaa. Kelvollisessa luokassa nostettava summa on suurempi tai yhtä suuri kuin 20 euroa, koska automaatit eivät anna sitä pienempiä seteleitä. Ollakseen kelvollinen nostettava summa ei kuitenkaan saa olla suurempi kuin nostoraja, jos sellainen on määritelty tai tilin saldo, jos nostoraja on suurempi kuin saldo tai nostorajaa ei

ole määritelty. Epäkelpo summa on tällöin alle 20 euron summat ja yli nostorajan tai katteen menevät summat. Kelvollinen luokka voidaan myös ilmaista seuraavasti:

```
20 € ≤ X ≤ nostoraja ≤ saldo
```

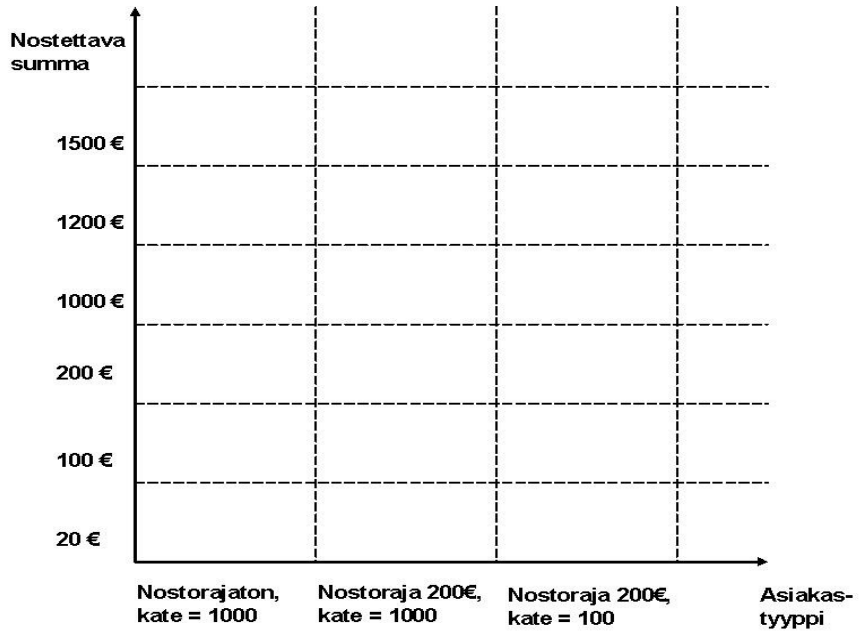
Epäkelvot luokat voidaan taas ilmaista seuraavalla tavalla:

```
X < 20 €
X > nostoraja
X > saldo
```

Ekvivalenssiositus on Binderin mukaan hieman vanhentunut menetelmä, joten sen soveltaminen käyttötapauspohjaiseen testaamiseen vaatii huomiota testitapausten valitsemisen ja testausjärjestyksen osalta (Binder, 2000). Koska UML:n mukaiset käyttötapaukset eivät sisällä syötteiden ja tulosteiden määrittelyä eikä syötteiden ja tulosteiden suhteiden määrittelyä tai ehtoja, jotka määrittelisivät perus- ja vaihtoehtoisia tehtäväpolkuja, on testitapausten valitseminen Binderin mielestä vaikeaa. Binderin mukaan Jacobson et al. (1992) ei kerro, missä järjestyksessä käyttötapauksia tulisi testata. Koska testitapauksiin liittyy suoritusjärjestyksestä johtuvia rajoitteita ja riippuvuuksia eri testitapausten välillä, tulisi testitapausten suoritusjärjestys miettiä tarkkaan (Binder, 2000).

Jacobson ja Ng ovat todenneet, että käyttötapaukset antavat kehyksen testitapauksille, sillä käyttötapaukset sisältävät perussuorituksen lisäksi kuvauksen muista mahdollisista tavoista suorittaa tehtävän, ja sisältää siis tällöin useita eri testitapauksia (Jacobson & Ng, 2005). Koska käyttötapauspohjaisessa kehittämisessä pyritään mahdollisimman varhaiseen testaamiseen sekä testaamiseen ennen toteuttamista, voidaan käyttötapauksia käyttää testitapauksien muodostamiseksi ennen käyttötapauksen toteuttamista. Testitapausten kirjoittamista ennen testattavan koodin kirjoittamista on Jacobsonin ja Ngn mukaan kritisoitu, koska ennen toteutusta on mahdollista muodostaa vääriä testitapauksia. Heidän mielestään käyttötapausten pohjalta voidaan kuitenkin luoda alustava testitapausten joukko (Jacobson & Ng, 2005).

Käyttötapauksia ei voi kuitenkaan suoraan pitää testitapauksina, sillä ne on tarkoitettu erilaisille kohteille ja niillä on erilaiset tarkoitukset: käyttötapaukset ovat kehittäjiä ja asiakasta varten, testitapaukset testaajia varten ja käyttötapauksia käytetään kuvaamaan ja selventämään asiakkaan tavoitteita ja vaatimuksia, kun taas testitapausten tarkoituksena on arvioida käyttötapauksen toteutusta sen määrityksiin nähden (Jacobson & Ng, 2005). Testitapaukset voidaan erottaa käyttötapauksista tapahtumavirtojen avulla. Kun käyttötapaus koostuu useammasta skenaariosta, eli se sisältää perustapahtumakuvauksen lisäksi vaihtoehtoisia kuvauksia käyttötapauksen suorittamisesta, tulee jokaisesta skenaariosta muodostaa oma testitapauksensa testiaineistoineen ja ympäristöineen.



Kuva 16: Testitapausten tunnistaminen käyttötapauksesta *Nosta rahaa*.

Esimerkiksi kuvassa 15 on esitetty luvussa 3 esitetyn pankkiautomaatti esimerkin käyttötapaus *Nosta rahaa* niin, että x-akselilla on kuvattu katteen ja nostorajan osalta erilaiset asiakastyypit ja y-akselilla nostettavat summat. Käyttötapauksesta voidaan muodostaa kolme erilaista testitapausta, joissa testataan kunkin kolmen eri asiakastyypin rahan nostoa eri summilla. Testitapaukset on esitetty taulukossa 2. Kolme erilaista asiakastyyppiä muodostavat käyttäjät, joiden nostoraja on vähemmän kuin tilin saldo, nostoraja on enemmän kuin tilin saldo sekä nostorajattomat käyttäjät.

Jos rahan noston tulisi onnistua testattavalla rahasummalla, on taulukkoon 2 merkattu ”X”. Jos rahan nosto ei saa onnistua, on taulukkoon 2 merkattu ”-” kyseisen rahasumman kohdalle.

Taulukko 2: Käyttötapauksesta muodostettavat testitapaukset.

Testitapaus	Nostoraja (200€)	Kate	80 €	100 €	120 €	200 €	220 €	1 000 €	1 200 €
Testi 1	Ei	1000	X	X	X	X	X	X	-
Testi 2	Kyllä	1000	X	X	X	X	-	-	-
Testi 3	Kyllä	100	X	X	-	-	-	-	-

Jacobson ja Ng painottavat, että testauksen suunnittelun tulee olla systemaattista ja että tietyn toteutuksen lopetusehdoiksi tulee kehittää testitapaukset jo ennen toteuttamisen aloittamista (Jacobson & Ng, 2005). Heidän mukaansa käyttötapausten avulla voidaan tunnistaa ja organisoida testitapauksia järjestelmällisesti, kun jokainen käyttötapaus käydään erikseen läpi kyseiseen käyttötapaukseen liittyvien testitapausten tunnistamiseksi. Tämä on erityisen tärkeää siksi, että virheiden ja ongelmatilanteiden tunnistaminen ja rajaaminen tiettyyn ohjelman osaan olisi helpompaa.

5.2 Käyttötapauspohjainen järjestelmätestaus

Binderin mukaan käyttötapaukset sopivat hyvin järjestelmätestauksen testitapausten perustaksi, sillä ne sisältävät paljon tietoa toiminnallisista vaatimuksista (Binder, 2000). Täydellinen järjestelmätestaus vaatii kuitenkin lisäksi myös muunlaisia, toteutuksesta riippuvaisia testausmenetelmiä. Tässä kohdassa esitellään Binderin käyttötapauspohjainen malli järjestelmätestauksen suunnittelemiseksi ja toteuttamiseksi.

Binder olettaa, että oliopohjainen järjestelmäkehitys on aina inkrementaalista, jolloin jokaisen inkrementin jälkeen on mahdollista suorittaa järjestelmätestaus inkrementin tuottamalle yksikkö- ja integrointitestatulle ohjelmaversiolle. Järjestelmätestauksen tarkoituksena on tällöin Binderin mukaan paljastaa virheet, jotka johtuvat version sisältämien komponenttien tai alajärjestelmien yhteistoiminnasta kokonaisena järjestelmänä, testata toteuttaako inkrementin tuottama järjestelmä sille asetetut vaatimukset sekä selvittää kehittäjille, joko tuote on toimitettavissa.

Binder ehdottaa yleiseksi järjestelmätestauksen strategiaksi seitsemän vaiheista menetelmää, jossa käytetään käyttötapauksia testitapausten muodostamisessa. Menetelmän vaiheet ovat seuraavat (Binder, 2000):

1. Kehitä laajentavat käyttötapaukset koko järjestelmästä ja testattavat, toteutuksesta riippuvat vaatimukset tarvittavilta osin.
2. Suorita yksikkö- ja integraatiotestaus hyväksyttävästi.
3. Kehitä käyttötapaussarjat jokaiselle laajennetulle käyttötapaus testille.
4. Aja testisarjat, etsi ja poista virheet sekä suorita regressio- eli rasitustestaus.
5. Kehitä ja aja toteutuksesta riippuvat testitapaukset.
6. Suorita testaus/virheen etsintä ja korjaus/uusinta testaus –sykliä, kunnes luotettavuustavoitteet saavutetaan tai aika ja resurssit loppuvat.
7. Julkaise järjestelmä.

Koska käyttötapaukset kuvataan luonnollisella kielellä, ei niitä Binderin mukaan voida suoraan käyttää testitapauksina. Binderin mukaan käyttötapauksia ei kannata muokata testaukseen soveltuviksi, vaan tehdä käyttötapauksista erillisiä, laajennettuja käyttötapauksia (*Extended Use Cases*), joista testitapaukset muodostetaan tietyn prosessin tuloksena. Laajennetut käyttötapaukset eivät tässä yhteydessä tarkoita siis samaa kuin käyttötapauksien laajentaa (*extend*) –suhde, vaan se tarkoittaa jokaisesta käyttötapauksesta kehitettävää, erillistä kuvausta, jota Binder kutsuu laajentavaksi käyttötapaukseksi. Laajennettu käyttötapaus sisältää täydellisen listan käyttötapaukseen liittyvistä toiminnallisista muuttujista (*operational variable*), täydellisen määritelmän jokaisen toiminnallisen muuttujan rajoitteista sekä käyttötapauksen toiminnallisen suhteiden kuvauksen. Taulukossa 3 on esitelty muutama pankkiautomaattiesimerkkiin liittyvä skenaariokuvaus, joita käytetään tässä kohdassa esitettyjen testitapausesimerkkien lähtökohtana.

Pankkiautomaatti esimerkin tapauksessa toiminnallisia muuttujia ovat muun muassa nostettava summa, automaattiasiakkaan tilin tila (suljettu, auki), kortin tila (käytettävissä, varastettu, vanhentunut), automaatin sisältämän rahan määrä sekä automaatin tila (suljettu, auki, yhteys pankkiin katkennut). Näille toiminnallisille muuttujille on olemassa rajoitteita, kuten esimerkiksi nos-

tettavan summan tulee olla vähintään 20 €, mutta pienempi kuin tilin saldo, nostoraja tai automaatin sisältämä rahamäärä.

Taulukko 3: Muutama pankkiautomaattiin liittyvä käyttötapaus sekä mahdollisia skenaarioita (Binder, 2000).

Käyttötapaus	Toimija	Mahdollinen syöte/tuloste yhdistelmä
Syötä kortti	Automaatti asiakas	1. Väärän tunnusluvun syöttäminen; oikean tunnusluvu syöttäminen;alkuvalikko 2. Tunnusluku oikein; asiakkaan pankkiin ei saada yhteyttä; ilmoitus: Yritä myöhemmin uudelleen. 3. Tunnusluku oikein; asiakkaan tili on suljettu; ilmoitus: Soita pankkiisi 4. Syötetty kortti on varastettu; oikea tunnusluku syötetty; kortin poisotto.
Nosta rahaa	Automaatti asiakas	1. Pyydä nostettavaksi 50 €; tili auki; saldo 100 €; automaatti antaa rahat. 2. Pyydä nostettavaksi 100 €; tili auki; saldo 50 €; ilmoitus: Ei katetta 3. Pyydä nostettavaksi 200 €; tili auki; nostoraja 100 €; ilmoitus: Nostorajan ylitys

Toiminnalliset suhteet saadaan muodostamalla toiminnallisista muuttujista päätöstaulu. Taulukossa 4 on esitelty käyttötapauksen *Syötä kortti* toiminnallisten muuttujien suhteet. Taulukkoa luetaan niin, että taulukon jokainen rivi muodostaa muunnoksen (*variant*). Jokaisen muunnoksen tulee olla yksilöllinen, eli jokaista tiettyä toiminnallisten muuttujien arvojen joukkoa vastaa vain yksi muunnos.

Taulukko 4: Käyttötapauksen *Syötä kortti* toiminnalliset suhteet (Binder, 2000).

	Toiminnalliset muuttujat				Odotettu tulos	
Muuttuja	**Kortti**	**Tunnusluku**	**Pankin vastaus**	**Asiakkaan tilin tila**	**Viesti**	**Kortin toiminto**
1	Epäkelpo	-	-	-	Syötä automaattikortti	Poisto automaatista
2	Kelpo	Vastaa korttia	Pankki tunnistaa	Suljettu	Ota yhteys pankkiin	Poisto automaatista
3	Kelpo	Vastaa korttia	Pankki tunnistaa	Avoin	Valitse toiminto	-
4	Kelpo	Vastaa korttia	Pankki ei tunnista kort-	-	Yritä uudelleen myöhemmin	Poisto automaatista
5	Kelpo	Ei vastaa korttia	-	-	Syötä tunnusluku	-
6	Peruutettu	-	Pankki tunnistaa	-	Kortti on suljettu	Ota automaattiin
7	Peruutettu	-	Pankki ei tunnista kort-	-	Kortti ei kelpaa	Poisto automaatista

tia

Esimerkiksi taulukossa 4 ensimmäinen rivi tarkoittaa sitä, että jos käyttäjä syöttää automaattiin muun kuin automaattikortin, kuten esimerkiksi jonkin kauppaketjun bonuskortin, automaatti pyytää käyttäjää syöttämään automaattikortin ja poistaa väärän kortin automaatista. Toisella rivillä käyttäjä syöttää automaattikortin ja sitä vastaavan oikean tunnusluvun, pankki tunnistaa kortin, mutta asiakkaan tili on suljettu, jolloin automaatti kehottaa käyttäjää ottamaan yhteyttä omaan pankkiin ja poistaa kortin automaatista.

Taulukko 5: Minimalistinen testitapaussarja käyttötapaukselle *Syötä kortti* (Binder 2000).

	Toiminnalliset muuttujat				**Odotettu tulos**	
Muuttuja	**Kortti**	**Tunnusluku**	**Pankin vastaus**	**Asiakkaan tilin tila**	**Viesti**	**Kortin toiminto**
1	Epäkelpo	-	-	-	Syötä automaattikortti	Poisto automaatista
1T	Kelpo	%*@#	-	-	Syötä automaattikortti	Poisto automaatista
1F	Mikä tahansa T testi muunnoksille 2-7.					
2	Kelpo	Vastaa korttia	Pankki tunnistaa kortin	Suljettu	Ota yhteys pankkiin	Poisto automaatista
2T	1234	1234	Tunnistaa	CLSD	Ota yhteys pankkiin	-
2F	Mikä tahansa T testi muunnoksille 1. 3-7.					
3	Kelpo	Vastaa korttia	Pankki tunnistaa kortin	Avoin	Valitse toiminto	-
3T	1234	1234	Tunnistaa	OPEN	Valitse toiminto	-
3F	Mikä tahansa T testi muunnoksille 1, 2, 4-7.					
4	Kelpo	Vastaa korttia	Pankki ei tunnista korttia	-	Yritä uudelleen myöhemmin	Poisto automaatista
4T	1234	1234	Ei tunnista	-	Yritä uudelleen myöhemmin	Poisto automaatista
4F	Mikä tahansa T testi muunnoksille 1-3, 5-7.					
5	Kelpo	Ei vastaa korttia	-	-	Syötä tunnusluku	-
5T	1234	1134	-	-	Syötä tunnusluku	-
5F	Mikä tahansa T testi muunnoksille 1-4, 6, 7.					
6	Peruutettu	-	Pankki tunnistaa kortin	-	Kortti on suljettu	Ota automaattiin
6T	5555	-	Tunnistaa	-	Kortti on suljettu	Ota automaattiin
6F	Mikä tahansa T testi muunnoksille 1-6, 7.					
7	Peruutettu	-	Pankki ei tunnista korttia	-	Kortti ei kelpaa	Poisto automaatista
7T	5555	-	Ei tunnista	-	Kortti ei kelpaa	Poisto automaatista
7F	Mikä tahansa T testi muunnoksille 1-6.					

Kun laajennetut käyttötapaukset on määritelty yllä esitellyiltä osin, voidaan muodostaa testitapaukset. Testitapaukset muodostetaan niin, että jokainen yllä esitetty muunnos on sekä tosi että epätosi ainakin kerran. Tosi testitapauksessa kaikki toiminnallisten muuttujien arvot täyttävät muunnoksen ehdot. Epätodessa testitapauksessa muutetaan ainakin yhtä toiminnallisen muuttujan arvoa niin, että arvo ei täytä muunnoksen ehtoa. Taulukossa 5 on esitelty minimalistinen testitapaussarja käyttötapaukselle *Syötä kortti*. Epätosi testitapaus voi usein olla tosi testitapaus jollekin muulle muunnokselle, kuten taulukon 5 esimerkissä, koska tämän käyttötapauksen toiminnallinen suhde on loogisesti täydellinen ja minimalistinen.

Että järjestelmätestaus olisi kattavaa, kaikki vaatimukset pitää tulla testatuiksi. Tästä johtuen jokainen laajennetun käyttötapauksen muunnos tulisi testata ainakin kerran. Binder ehdottaa kattavuuden mittariksi (XUVC) käyttötapauspohjaiselle testaamiselle seuraavanlaista kaavaa (Binder, 2000):

$$XUVC = \frac{\text{Toteutettujen käyttötapausten määrä}}{\text{Tarvittavien käyttötapausten määrä}} \times \frac{\text{Testattujen muunnosten määrä}}{\text{Muunnosten yhteislukumäärä}} \times 100$$

Lisäksi taulukossa 6 on esitelty, miten käyttötapauksiin liittyvät testitapaukset voidaan jäljittää. Tällaisen taulukon avulla voidaan helposti tarkastaa, onko kaikki käyttötapauksia varten ainakin yksi testitapaus. Jotta testaus voitaisiin lopettaa, tulisi jokaisen testitapauksen mennä läpi hyväksyttävästi.

Taulukko 6: Käyttötapauksiin liittyvien käyttötapausten jäljittäminen (Binder 2000).

	Testi 1	Testi 2	Testi 3	Testi 4	Testi 5	Testi 6	Testi 7	…	Testi 9999
Käyttötapaus 1				X					
Käyttötapaus 2		X			X				X
Käyttötapaus 3						X			
Käyttötapaus 4	X				X		X		
Käyttötapaus 5									
…									
Käyttötapaus 999		X							X

Laajennettuja käyttötapauksia voidaan käyttää testauksessa silloin, kun suurin osa testattavan järjestelmän tärkeimmistä vaatimuksista voidaan ilmaista käyttötapausten avulla. Esimerkiksi sil-

loin, jos testattava käyttötapaus sisältää paljon järjestelmän sisäistä toimintaa, joka ei näy käyttäjällä, kuten vaativaa laskentaa, ei käyttötapausten pohjalta voida välttämättä muodostaa riittäviä testitapauksia.

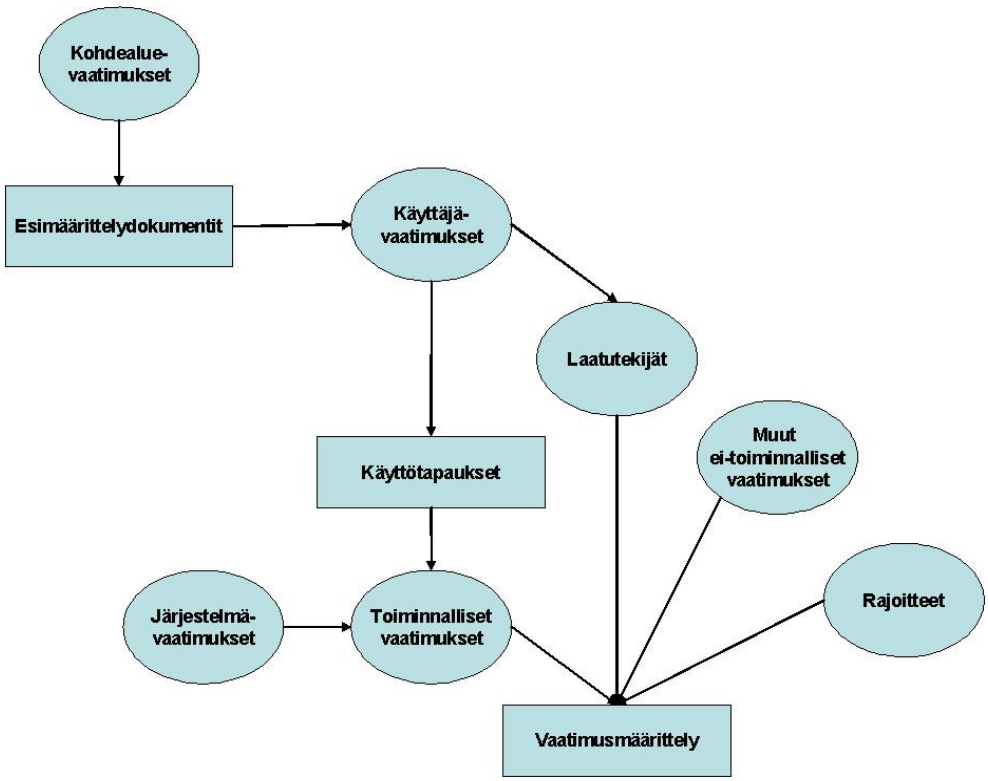
Tyypillisesti yksittäisestä käyttötapauksesta voidaan muodostaa hyvin suuri joukko testitapauksia. Tästä syystä testitapausten valintaan tulee kiinnittää huomiota. Tehokas testaaminen edellyttää Binderin mukaan käyttötapausten rajoitteiden ja suhteiden mallintamista sekä riittävän suuren testitapausjoukon kehittämistä, että voitaisiin varmistua testauksen kattavuudesta (Binder, 2000).
Binderin mukaan käyttötapauspohjaiseen testaamiseen liittyy kuitenkin sekä hyviä että huonoja puolia. Koska käyttötapauksille ei ole olemassa yhtä sovittua abstraktion tasoa tai yksityiskohtien määrää eri käyttötapausmenetelmien välillä, tulee testitapausten muodostamisessa käytettävän menetelmän ratkaista epäselvyydet. Käyttötapaukset eivät myöskään yleensä ilmaise esimerkiksi suorituskykyvaatimuksia tai vikasietoisuusvaatimuksia, joten näitä vaatimuksia varten on muodostettava ja suoritettava omat testitapauksensa (Binder, 2000).

Kuitenkin Binderin mukaan käyttötapauspohjaisella testaamisella on huomattavasti enemmän hyviä kuin huonoja puolia. Binder laskee menetelmän eduiksi muun muassa sen, että käyttötapaukset ovat laajasti käytössä, ne ovat usein ainoa käytettävissä oleva vaatimusmäärittelydokumentti, käyttötapaukset kuvaavat testattavan järjestelmän asiakkaan ja käyttäjien näkökulmasta ja laajennetut käyttötapaukset mahdollistavat systemaattisen tavan suunnitella sekä kehittää testitapauksia. Lisäksi Binderin mukaan laajennetut käyttötapaukset voivat paljastaa esimerkiksi myös suunnitteluvirheitä, jos käyttötapaukset ovat monimerkityksellisiä, epäjohdonmukaisia tai epätäydellisiä.

5.3 Käyttötapaukset vaatimusten testaamisessa

Järjestelmän kehitys perustuu sille asetettuihin vaatimuksiin ja ohjelmiston ajatellaan olevan toimiva silloin, kun kehitetty järjestelmä vastaa sille asetettuja vaatimuksia. Vaatimukset ovat dokumentoituja esityksiä asiakkaan, käyttäjien ja kehittäjien näkemyksistä siitä, mitä järjestelmän tulee tehdä ja mitä ehtoja tai rajoitteita toiminnalle on olemassa. Oleellista kehitysprojektin onnis-

tumisen kannalta on, että ohjelma vastaa oikeaksi todettuja vaatimuksia, eli että vaatimusmäärittelyyn, jonka pohjalta järjestelmä on kehitetty, on valittu oikeat vaatimukset ja ne on kuvattu oikein.



Kuva 17: Vaatimusmäärittelyn eri komponentit ja niiden väliset suhteet (Wiegers, 1999).

Vaatimukset voidaan jakaa eri tasoihin, esimerkiksi kohdealuevaatimuksiin (*Business requirements*), käyttäjävaatimuksiin (*User requirements*), toiminnallisiin vaatimuksiin (*Functional requirements*) sekä moniin ei-toiminnallisiin vaatimuksiin (Wiegers, 1999). Kuvassa 17 on esitetty nämä eri tasot ja niiden väliset suhteet. Kuten kuvasta 17 nähdään, kohdealuevaatimukset kuvaavat järjestelmän kohdealueen korkean tason tavoitteet, jotka dokumentoidaan esimäärittelydokumentteihin. Käyttäjävaatimukset kuvaavat ne tehtävät, jotka käyttäjien tulee olla mahdollista suorittaa järjestelmän avulla. Käyttäjävaatimusten dokumentoimiseen voidaan käyttää esimerkiksi käyttötapauksia, kuten kuvassa 17 on esitetty. Käyttötapauksien kuvaamat toiminnalliset vaatimukset määrittävät sen, mitä kehittäjien tulee järjestelmään rakentaa, että käyttäjät voivat suorittaa järjestelmällä niitä tehtäviä, jotka käyttötapauksissa on määritelty. Lopullinen vaatimusmää-

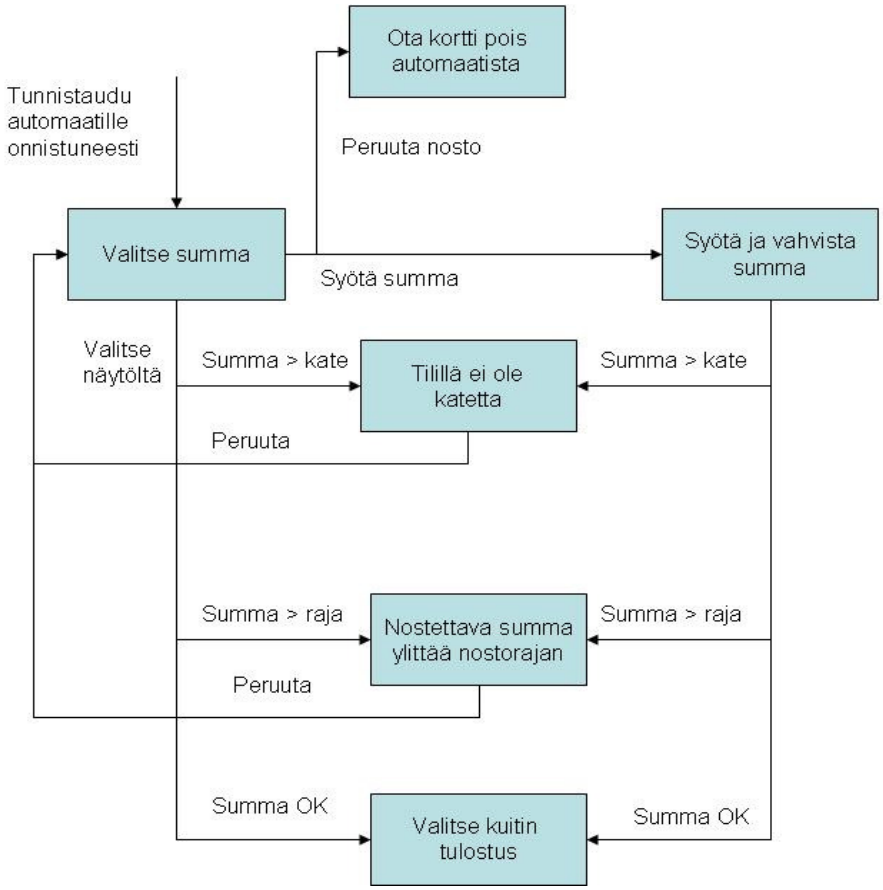
rittely koostuu toiminnallisten vaatimusten lisäksi ei-toiminnallisista vaatimuksista, kuten laatutekijöistä, rajoitteista ja muista ei-toiminnallisista vaatimuksista, kuten kuvasta 17 nähdään.

Mitä myöhäisemmässä vaiheessa järjestelmän kehitystä vaatimusmäärittelyssä havaitaan virhe, sitä kalliimpaa ja vaikeampaa virheen korjaaminen on, koska sitä useampaa asiaa joudutaan todennäköisesti muuttamaan. Wiegersin mukaan tutkimuksissa on osoitettu, että asiakkaan havaitseman määrittelyvirheen korjaaminen maksaa 68-110 kertaa enemmän kuin virheen korjaaminen jo määrittelyvaiheessa (Wiegers, 1999). Hänen mukaansa myös virheen korjaamiseen menee huomattavasti enemmän aikaa, sillä järjestelmätestauksessa havaitun virheen korjaamiseen menee noin 5-17 tuntia, kun vaatimusmäärittelyvaiheessa korjaus kestää vain 30 minuuttia. Vaatimusmäärittelyn virheistä johtuukin Wiegersin mukaan jopa 40-60 % kaikista ohjelmistoprojektin aikana löydetyistä virheistä. Lisäksi Wiegersin mukaan tutkimuksessa, jossa tutkittiin 8380 eri ohjelmistoprojektia, kaksi suurinta syytä projektin epäonnistumiselle olivat käyttäjiltä kerättävän tiedon puuttuminen ja vaillinainen vaatimusmäärittely (Wiegers, 1999). Virheiden suuresta määrästä, niiden korjaamisen hitaudesta ja kalleudesta johtuen olisi tärkeää myös testata, että ohjelma ei ainoastaan täytä sille asetettuja vaatimuksia, vaan se myös kehitetään oikeiden vaatimusten pohjalta.

Vaatimusmäärittelylle voidaan suorittaa samanlaisia muodollisia katselmointeja ja tarkastustilaisuuksia, joita voidaan tehdä muillekin ohjelmistoprojektin aikana tehtäville dokumenteille. Wiegersin mukaan tarkastusmenettelyt ovat tehokas tekniikka vaatimusmäärittelyn laadun parantamiseksi. Lisäksi Wiegersin mukaan käyttötapauksista voidaan muodostaa toiminnallisia testitapauksia kehityksen hyvin aikaisessa vaiheessa ja jo testitapausten suunnittelu ilman niiden varsinaista suorittamista ohjelman avulla paljastaa monia virheitä ja ongelmia vaatimuksissa. Wiegers ehdottaa, että testitapauksia käytettäisiin dokumentoitujen vaatimusten ja analyysimallien, kuten dialogikarttojen (*dialog maps*) verifioimiseen ja prototyyppien arvioimiseen.
Esimerkiksi luvussa kolme esitetyn pankkiautomaattiesimerkin käyttötapauksen *Nosta rahaa* osalta voitaisiin olettaa kohdealuevaatimus, että käyttäjä voi nostaa rahaa automaatin avulla omalta tililtään vuorokauden ajasta tai viikonpäivästä riippumatta. Käyttötapauksessa tämä vaatimus huomioidaan siten, että käyttäjän tulee tunnistautua automaatille rahan nostamiseksi. Toi-

minnallinen vaatimus tälle käyttäjän toiminnallisuudelle voisi olla seuraava: jos käyttäjän tunnistautuminen automaatille onnistuu, järjestelmä kysyy käyttäjältä nostettavaa summaa, jonka jälkeen käyttäjä voi valita joko summan syöttämisen itse tai jonkin järjestelmän ehdottamista summista.

Kuvassa 18 on esitetty tähän käyttötapaukseen liittyvä dialogikartta, joka toteuttaa yllä esitetyn toiminnallisuuden. Kuvan laatikot kuvaavat käsitteellisesti järjestelmän ja käyttäjän välisen vuorovaikutuksen osia tai vaiheita ja nuolet kuvaavat mahdollista siirtymistä osien tai vaiheiden välillä.

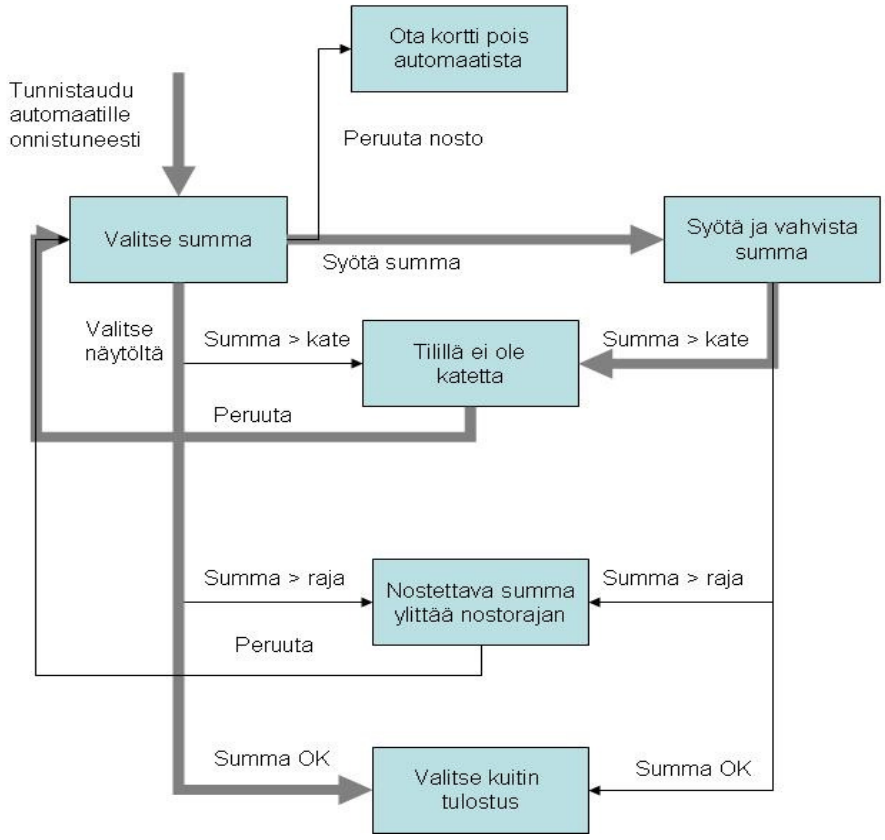


Kuva 18: Osa käyttötapauksen *Nosta rahaa* dialogikartasta.

Kuten kuvasta 18 huomataan, tämä käyttötapaus sisältää monta eri suorituspolkua sekä erilaisia poikkeustilanteita, jolloin käyttötapauksesta voidaan muodostaa hyvin monia eri testitapauksia. Käyttötapauksen ja toiminnallisen vaatimuksen kuvausten perusteella testitapausten kirjoittaja

kirjoittaa testitapaukset sen mukaan, miten hän olettaa käyttäjän kommunikoivan järjestelmän kanssa. Testitapauksen kuvaus voisi olla seuraavanlainen:

> *Tunnistaudu automaatille onnistuneesti, jolloin järjestelmä siirtyy dialogiin, jossa pyydetään valitsemaan nostettava summa. Valitse, että syötät summan itse etkä valitse järjestelmän ehdottamista summista. Järjestelmä siirtyy dialogiin, jossa pyydetään syöttämään summa ja vahvistamaan se. Syötä tilin saldoa suurempi summa ja vahvista se. Järjestelmä siirtyy dialogiin, jossa ilmoitetaan liian suuren summan nostamisesta. Järjestelmä siirtyy takaisin dialogiin, jossa käyttäjää pyydetään valitsemaan nostettava summa. Valitse järjestelmän ehdottama summa, joka ei ylitä saldoa tai nostorajaa. Järjestelmä siirtyy dialogiin, jossa kysytään saldon tulostamista kuitille tai näytölle.*



Kuva 19: Testitapauksen jäljittäminen dialogikartasta.

Tämän jälkeen testitapaukset voidaan jäljittää käyttötapauksesta kuvassa 19 esitetyllä tavalla. Kuvassa 19 testitapauksessa suoritettava suorituspolku on esitetty paksummilla nuolilla. Testaus-

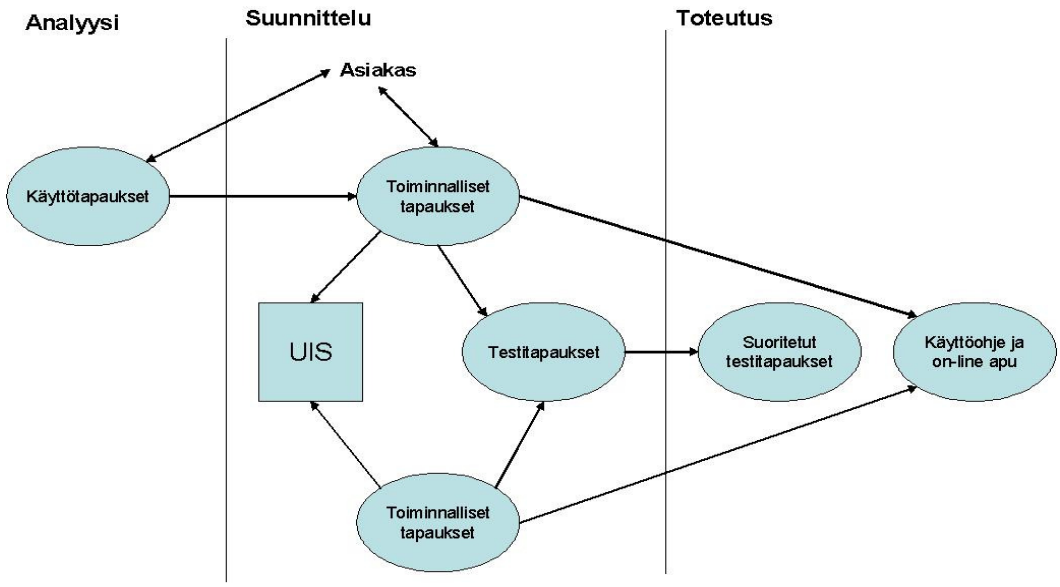
vaiheessa toiminnalliset vaatimukset käydään läpi niin, että jokaiset testitapauksen kohdalla mietitään, voitaisiinko testitapaus ”ajaa” nykyisten vaatimusten perusteella. Lisäksi jokaista toiminnallista vaatimusta pitää vastata ainakin yksi testitapaus (Wiegers, 1999). Käymällä läpi kaikki dialogikartan eri suorituspolut jokaisen testitapauksen osalta, voidaan Wigersin mukaan löytää virheellisiä ja puuttuvia vaatimuksia.

Jos jokin dialogikartan laatikoiden välisistä nuolista jäisi kokonaan käyttämättä missään testitapauksessa, voi olla, että kyseistä siirtymistä ei ole määritelty järjestelmän toiminnaksi. Jos jokin vaatimus sisältää siirtymän, voi kyseessä olla virheellinen vaatimus, joka tulee korjata. Toinen mahdollisuus on, että siirtymä on mahdollinen, mutta sen testaava testitapaus puuttuu vielä. Jos jokin testitapaus sisältää sellaisen siirtymän dialogilaatikoiden välillä, jota ei dialogikarttaan ole tehty eikä kyseistä testitapausta voi siis silloin suorittaa, on mahdollista, että virhe on testitapauksessa tai siirtymä on mahdollinen, mutta se puuttuu dialogikartasta ja on näin ollen puuttuva vaatimus (Wiegers, 1999).

Wiegersin mukaan vaatimusten käsitteellisen testaamisen avulla voidaan löytää puuttuvat, virheelliset ja tarpeettomat vaatimukset kauan ennen kuin yhtään riviä koodia on kirjoitettu. Tämä auttaa kehitysprojektin kustannusten ja aikataulun hallitsemisessa, sillä vaatimuksissa olevat virheet voidaan löytää mahdollisimman aikaisessa vaiheessa. Vaatimusten aikainen testaaminen auttaa Wiegersin mukaan kehittämään laadukkaampia järjestelmiä lyhyemmässä ajassa ja vähemmillä kustannuksilla.

5.4 Evolutiiviset käyttötapaukset testaamisessa

Kirner et al. ovat kehittäneet menetelmän, jonka avulla perinteisiä käyttötapauksia voidaan kehittää tuotteen elinkaaren aikana palvelemaan myös muita kehitysvaiheita kuin pelkästään tuotteen vaatimusten määrittelyä (Kirner et al., 1999). Heidän kehittämässään mallissa käyttötapauksia jalostetaan järjestelmän kehityksen aikana toiminnallisiksi tapauksiksi ja testitapauksiksi. Kolme erilaista käyttötapauspohjaista mallia on heidän mielestään kokonaisuutena yksinkertainen käsitteellinen malli sekä järjestelmän kehittäjille että asiakkaille (Kirner et al., 1999). Kuvassa 20 on esitetty, miten käyttötapaukset kehittyvät järjestelmän kehityksen aikana.



Kuva 20: Käyttötapausten kehittyminen tuotteen elinkaaren aikana (Kirner et al., 1999).

Kuten kuvasta 20 nähdään, perinteisiä käyttötapauksia käytetään analyysivaiheessa vaatimusten keräämiseksi asiakaskeskeisestä näkökulmasta. Kirner et al. käyttävät tässä mallissa käyttötapauksia hyvin perinteisessä muodossa niin, että käyttötapaukset koostuvat käyttötapauksen nimestä ja yhteenvedosta, kyseisen käyttötapauksen käyttäjien määrittelystä, alku- ja loppuehdoista sekä käyttötapauksen toiminnan ja sen poikkeusten kuvauksesta. Käyttötapausten sisältämä kuvaus järjestelmän toiminnasta on yksityiskohtainen kuvaus, mutta se ei sisällä käyttöliittymän toteutukseen liittyviä yksityiskohtia.

Suunnitteluvaiheessa käyttötapauksia kehitetään toiminnallisiksi tapauksiksi (*behaviour cases*) niin, että yksi tai useampi toiminnallinen tapaus sisältää yhden käyttötapauksen toteutukseen liittyvän kuvauksen (Kirner et al., 1999). Toiminnallinen tapaus muodostetaan lisäämällä sen perustana olevan käyttötapauksen alku- ja loppuehtoihin sekä toiminnan ja poikkeusten kuvaukseen toteuttamiseen liittyviä yksityiskohtia. Lisäksi toiminnallinen tapaus sisältää tiedon siitä, mitä välineitä ja mahdollisuuksia käyttäjällä on vaikuttaa toiminnan suorittamiseen. Toiminnallinen tapaus kuvaa muun muassa sen, mitä tietoa järjestelmä antaa käyttäjälle, miten tieto näytetään käyttäjälle ja millä ehdoilla toiminto on mahdollinen suorittaa. Toiminnallisten käyttötapausten arvo kehitysprosessissa Kirnerin et al. mielestä on, että toiminnalliset tapaukset voivat määritellä

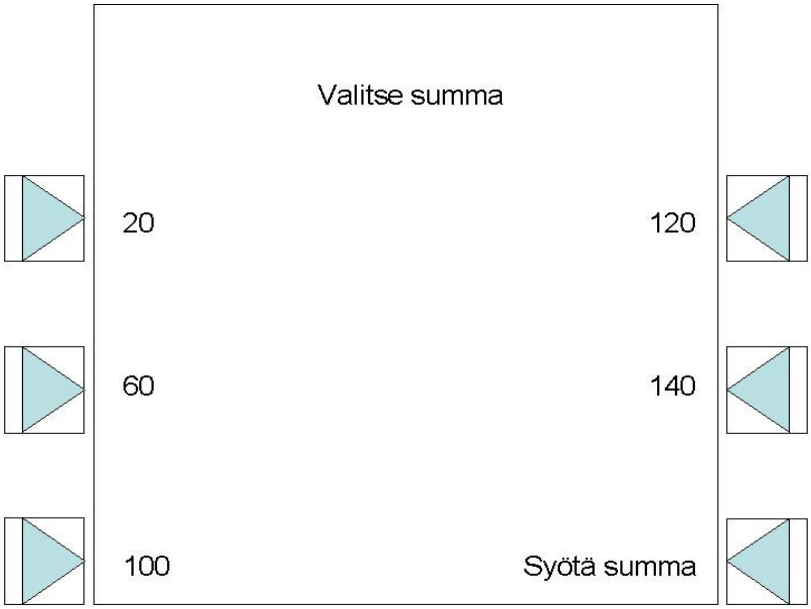
käyttöliittymän ja tuotteen toiminnallisuuden laajentamalla käyttötapauksen sisältämien vaatimusten kuvausta ilman käyttötapauksissa olevia yksityiskohtia (Kirner et al., 1999). Taulukossa 7 on esitetty toiminnallisena tapauksena osa luvussa 3 esitetyn pankkiautomaatti esimerkin sisältämää käyttötapausta *Nosta rahaa*. Taulukossa esitetty tapaus *Valitse summa* on osa käyttötapausta, ja se kuvaa järjestelmän ja käyttäjän välistä kommunikointia nostettavan rahasumman valinnan osalta.

Taulukko 7: Toiminnallinen tapaus *Valitse summa.*

Nimi	*Valitse summa*
Yhteenveto	Käyttäjä valitsee nostettavan summan
Käyttäjät	Peruskäyttäjä
Toiminta mekanismi	Käyttäjä valitsee nostettavan summan näytön vieressä olevien nuolinäppäimien avulla.
Alkuehdot	Käyttäjä on syöttänyt automaattiin pankkikortin ja siihen liittyvän tunnusluvun oikein.
Syöte mekanismi	Näppäimistö
Kuvaus	1. Käyttäjä valitsee nostettavan summan ja painaa OK-painiketta näppäimistöllä. 2. Järjestelmä tarkastaa tilin katteen ja nostorajan. 3. Järjestelmä vertaa valittua summaa tilin katteeseen ja nostorajaan. 4. Järjestelmä vähentää tilin saldoa valitulla summalla. 5. Järjestelmä kysyy käyttäjältä saldon tulostamista, jonka jälkeen käyttäjä saa kortin, kuitin ja rahat.
Poikkeukset	Saldo ei riitä: Järjestelmä antaa käyttäjälle ilmoituksen riittämättömästä katteesta. Summa on suurempi kuin nostoraja: Järjestelmä antaa käyttäjälle ilmoituksen nostorajan ylittymisestä. Käyttäjä valitsee summan syöttämisen: Järjestelmä kysyy nostettavaa summaa.
Loppuehdot	Tililtä vähennetään käyttäjän syöttämä summa ja automaatti antaa rahat käyttäjälle.
Toiminna käyttöönotto/ käytöstä poisto	Toiminto on käytettävissä vain kun käyttäjä on syöttänyt automaattiin pankkikortin ja korttia vastaavan, oikean tunnusluvun.

Toiminnallisia tapauksia puolestaan käytetään sekä testitapausten että käyttöliittymämäärittelyn (*User Interface Specification, UIS*) kehittämiseksi. Kirnerin et al. mukaan toiminnallisia tapauksia voitaisiin muuntaa käyttöohjeiksi ja ohjelmien on-line aputoiminnoiksi toteutusvaiheessa, kuten kuvasta 20 nähdään. Kuvassa 21 on käyttötapaukseen *Nosta rahaa* ja siitä kehitettyyn toiminnalliseen tapaukseen *Valitse summa* liittyvä käyttöliittymämäärittely. Kuten kuvasta 21 nähdään,

käyttäjä kommunikoi järjestelmän kanssa näppäinten avulla painamalla sitä näppäintä, joka osoittaa käyttäjän haluamaan rahasummaan.



Kuva 21: Toiminnallisen tapauksen *Valitse summa* kuvaama käyttöliittymämäärittely.

Suunnitteluvaiheessa muodostetaan myös testitapaukset toiminnallisten tapausten pohjalta. Testitapaukset sisältävät kuvauksen toiminnoista, joiden avulla testaaja varmistuu, että testattava osa vastaa sille asetettuja vaatimuksia. Erona käyttötapausten ja toiminnallisten tapausten dokumentoimiseen on, että testitapauksissa kuvataan testin tarkoitus, riippuvuudet muista testitapauksista ja testin odotettu tulos sekä tarkka kuvaus siitä, mitä testaaja tekee toiminnan testaamiseksi. Testitapaus ei sisällä toiminnan poikkeusten kuvauksia, sillä jos toiminta poikkeaa testitapauksessa kuvatusta toiminnasta, on kyseessä virhe järjestelmän toiminnassa. Toteutusvaiheessa järjestelmälle suoritetaan testaus mustalaatikko-menetelmänä suunniteltujen testitapausten pohjalta, kuten kuvassa 20 näkyy.

Yhdestä toiminnallisesta tapauksesta muodostuu todennäköisesti useita testitapauksia, koska testitapauksessa voidaan kattaa vain yksi suorituspolku kerrallaan, kun taas toiminnallisessa tapauksessa voidaan kuvata myös toiminnan poikkeukset. Taulukossa 8 on esitetty aiemmin esitettyyn toiminnalliseen tapaukseen liittyvä testitapaus, jossa testataan nostorajan tarkastamista.

Taulukko 8: Testitapaus toiminnalliselle tapauksella *Valitse summa.*

Nimi	*T1 Valitse summa*
Yhteenveto	Käyttäjä valitsee nostettavan summan, joka on suurempi kuin tilille määritetty nostoraja.
Käyttäjät	Peruskäyttäjä
Alkuehdot	Käyttäjä on syöttänyt automaattiin pankkikortin ja siihen liittyvän tunnusluvun oikein.
Testin tarkoitus	Testitapauksen avulla varmistutaan nostorjan tarkastus automaattinoston yhteydessä.
Riippuvuus muista testitapauksista	Ennen testitapauksen suorittamista tulee suorittaa testitapaus T0, jossa käyttäjä syöttää kortin ja tunnusluvun automaattiin onnistuneesti.
Odotettu tulos	Tililtä ei voi nostaa rahaa ja järjestelmä antaa virheilmoituksen.
Kuvaus	1. Valitsee nostettavaksi summaksi 120 ja painaa OK-painiketta näppäimistöllä. 2. Järjestelmän tulee ilmoittaa, että nostettava summa on suurempi kuin tilin nostoraja. Järjestelmä kysyy nostettavaa summaa uudelleen.

5.5 Käyttötapausten rakenteellinen testaaminen

Koska käyttötapauksilla on hyvin tärkeä ja suuri rooli järjestelmän vaatimusten määrittämisessä, ovat Carniello et al. kehittäneet uudenlaisen joukon käyttötapauksiin pohjautuvia testauskriteerejä. Testauskriteerien avulla on tarkoitus arvioida käyttötapauksista muodostettujen testitapausten laatua sekä testata itse käyttötapauskuvaus (Carniello et al., 2005). Carniello et al. mukaan on olemassa näyttöä siitä, että rakenteellinen testaaminen, toiminnallinen testaaminen ja tarkastusmenettelyt täydentävät toisiaan, sillä löydetyt virheet ovat eri tyyppisiä eri testausmenetelmää käytettäessä.

Ennen varsinaista ohjelmakoodia toteutetut esitykset järjestelmästä, kuten käyttötapaukset, ovat yleensä nimenomaan toiminnallisen testaamisen testitapausten lähtökohtana. Carniello et al. ovat kuitenkin sitä mieltä, että myös näillä kuvauksilla on olemassa määritelty rakenne, jota voidaan käyttää rakenteellisen testaamisen perustana. Heidän mukaansa tällaisten kuvausten rakenteellisella testaamisella voidaan löytää esityksistä sellaisia virheitä, joita ei löydettäisi pelkästään toiminnallisten testien avulla. Carniello et al olettavat, että toteutettujen esityksiä voidaan testata jäljittelemällä järjestelmätestauksessa havaittavaa toimintaa ennen varsinaisen ohjelmakoodin

toteuttamista. Carniello et al. ovat käyttäneet tätä oletusta käyttötapausten testaamiseen (Carniello et al., 2005).

Käyttötapaukset voidaan määritellä sarjana järjestelmän suorittamia toimintoja, joiden tarkoituksena on tuottaa havaittavaa arvoa käyttäjilleen. Käyttötapaukset ja niiden väliset suhteet kuvataan käyttötapauskaavioiden avulla. Suhteet määrittävät näin ollen diagrammin rakenteen. Carniello et al. esittelemät testauskriteerit pyrkivät testaamaan käyttötapauskaavioiden sisäisen rakenteen, eli käyttötapausten väliset suhteet.

Kuten luvussa 3 kuvattiin, käyttötapauskaaviot muodostuvat käyttötapauksista, jotka kuvataan käyttötapauksen tunnuksella nimetyillä ellipseillä, käyttäjistä, jotka kuvataan tikku-ukkoina sekä käyttötapausten välisistä suhteista, jotka kuvataan käyttötapausten välisinä nuolina. Lisäksi kuten luvussa 3 kerrottiin käyttötapauksilla voi olla erityyppisiä suhteita, joista Carniello et al. määrittelevät sisältää, laajentaa sekä käyttää –suhteet. Carniello et al. esittelemät testauskriteerit vaativat, että jokainen käyttötapausten välinen suhde suoritetaan ainakin kerran jossakin testitapauksessa (Carniello et al., 2005).

Sisältää ja laajentaa –suhteiden käyttö kuvattiin tarkemmin jo luvussa 3. Käyttötapausten A ja B väliset sisältää ja laajentaa -suhteet suoritetaan sellaisen testitapauksen avulla, jossa käyttötapaus A joko sisältää tai laajentaa käyttötapauksen B toiminnan. Käyttää-suhde on käyttötapauksen ja käyttäjän välinen suhde, joka suoritetaan sellaisen testitapauksen avulla, jossa käyttäjän tarvitsee käyttää käyttötapauksen sisältämää toiminnallisuutta.

Testikriteerit perustuvat käyttötapausten välisiin suhteisiin ja laajentavien käyttötapausten kombinaatioihin. Ensimmäinen testikriteeri (TK1) on kaikkien käyttää-, laajentaa- ja sisältää- suhteiden suorittaminen. Tämä tarkoittaa sitä, että testijoukon T tulee suorittaa jokainen käyttötapauskaavion D suhteet ainakin kerran. Luvun 3 pankkiautomaatti esimerkissä tämä tarkoittaa seuraavien suhteiden suorittamista:

- Käyttää-suhde: Nosta rahaa, Katso tilitapahtumat, Lisää rahaa automaattiin, Poista kortit
- Sisältää-suhde: Nosta rahaa – Syötä kortti, Katso tilitapahtumat – Syötä kortti

Laajentaviin käyttötapauksiin liittyy ehto, jonka täytyy toteutua ennen kuin laajentava käyttötapaus suoritetaan. Tähän perustuen Carniello et al. ehdottavat testikriteerin, joka vaatii jokaisen laajentavan käyttötapauksen suorittamisen ja suorittamattajättämisen. Tästä seuraa se, että testisarjojen tulee sisältää suorittamisen ja suorittamattajättämisen eri kombinaatiot. Luvun kolme pankkiautomaatti esimerkki ei sisällä laajentavat käyttötapauksia, mutta esimerkin vuoksi voidaan käyttää mahdollista käyttötapausta *Lataa rahaa kortille*, joka voisi laajentaa käyttötapausta *Nosta rahaa*. Tällöin tätä käyttötapauksen suhdetta tulee vastata ainakin kaksi testitapausta, jossa toisessa laajentava käyttötapaus suoritetaan ja toisessa se jätetään suorittamatta. Eli toisessa testitapauksessa käyttäjä suorittaa rahan nostamisen normaalisti ja toisessa käyttäjä lataa rahaa kortille.

Testitapausten suorittamista ja testaamisen kattavuuden arviointia varten käyttäen yllä määriteltyjä testauskriteerejä on kehitetty UCT (*Use Case Tester*) testausväline (Carniello et al., 2005). Tätä testausvälinettä varten on kehitetty oma notaatio käyttötapausten esittämiseksi. Kun käyttötapaukset kuvataan määritellyn notaation avulla, voidaan käyttötapausten suorittaminen ja kattavuusanalyysi automatisoida. Carniello et al. mukaan tämän menetelmän avulla voidaan löytää käyttötapauksista muun muassa sellaisia virheitä kuten skenaarioita, joita ei testata toiminnallisen testaamisen avulla ja semanttisia ristiriitoja käyttötapauksissa. Notaation mukainen käyttötapauksen kuvaus käyttötapauksesta *Nosta rahaa* voidaan kuvata seuraavasti:

```
Use_case_name(Nosta rahaa);
Actors(start Käyttäjä);
Include_relationships(Nosta rahaa -> Syötä kortti);
Extend_relationships(Nosta rahaa <- Lataa rahaa kortille);
Initial_states({kortilleLataus == true, kortilleLataus ==
false});
Actions_stream(
      Inclusion(Syötä kortti);
      If (korttiOK == true) {
            Extension_condition(Lataa rahaa):
            If(kortilleLataus == true)
                  Extension(Lataa rahaa kortille);
            If(kortilleLataus == false)
                  Action(Anna rahat käyttäjälle);
      }
)
```

6. RSI-käyttötapausanalyysi

RSI-lähestymistapa käyttötapausanalyysiin on Ratio Group:in ja Mark Collins-Copen kehittämä menetelmä, joka yhdistää useita eri käyttötapausanalyysin menetelmiä, joita esiteltiin jo luvussa 3. Ratio Group on erikoistunut olio- ja komponenttipohjaiseen ohjelmistosuunnitteluun ja heillä on paljon kokemusta käyttötapausanalyysin opettamisesta ja konsultoimisesta (Ratio, 2005). Käyttötapausanalyysi voi kuulostaa yksinkertaiselta ja helpolta menetelmältä, mutta kokemus on osoittanut, että tehtäessä käyttötapausanalyysiä tulee esille kysymyksiä muun muassa analyysin laajuudesta, yksityiskohtaisuudesta, analyysin kohdeyleisöstä sekä siitä, miten käyttöliittymä tulisi kuvata ja miten käyttötapaukset liittyvät oliomalliin. RSI-käyttötapausanalyysi tarjoaa tarkan prosessikuvauksen järjestelmän vaatimusten sekä käyttöliittymän suunnittelemiseksi ja kuvaamiseksi (Collins-Cope, 2000).

RSI-käyttötapausanalyysi koostuu kolmenlaisista käyttötapauksista: vaatimuskäyttötapauksista (*Requirements*), palvelukäyttötapauksista (*Service*) ja käyttöliittymäkäyttötapauksista (*Interface*). *Vaatimuskäyttötapaukset* määrittelevät käyttäjien vaatimukset ja ne vastaavat kysymyksiin, mitä asiakas haluaa ohjelmalta sekä mitä ohjelman tulee tehdä, että käyttäjän asettamat tavoitteet saavutetaan. Vaatimuskäyttötapauksissa kuvataan järjestelmän ja käyttäjän välinen vuorovaikutus, mutta varsinaiset käyttöliittymälomakkeet kuvataan käyttöliittymäkäyttötapauksissa. *Käyttöliittymäkäyttötapaukset* kuvaavat käyttäjän antamat syötteet järjestelmän ymmärtämässä muodossa. Käyttöliittymäkäyttötapaukset ilmaisevat mitä informaatiota järjestelmä tarvitsee palvelukäyttötapausten toteuttamiseksi. Käyttöliittymäkäyttötapauksia ovat myös järjestelmän tuottamat tulosteet. *Palvelukäyttötapaukset* kuvaavat järjestelmän sisäistä, itsenäistä toimintaa ilman käyttäjän vuorovaikutusta. Palvelukäyttötapauksia on kahdenlaisia: sellaisia jotka päivittävät järjestelmän tilaa ja sellaisia, jotka palauttavat tietoa järjestelmästä muuttamatta sen tilaa.

RSI-lähestymistapa pohjautuu moniin eri käyttötapausanalyysin menetelmiin, mutta yksi tärkeimmistä lähtökohdista RSI-mallin kehittämiselle on ollut Alistair Cockburnin työ (Cockburn, 1997). RSI-menetelmä eroaa Cockburnin menetelmästä käyttötapausten luokittelussa sekä se erottelee käyttötapauksista käyttöliittymän ja palvelut omiksi alatyypeikseen (Collins-Cope,

2000). Vaatimuskäyttötapaukset voidaan lajitella kolmeen eri tasoon niiden sisältämän toiminnan mukaan käyttäjän tavoite, koottu tavoite ja alitavoite –tason käyttötapauksiksi.

Cockburnin työstä on myös lähtöisin RSI-mallin tavoitelähtöisyys. Vaatimuskäyttötapausten sanallisissa kuvauksissa askel askeleelta etenevä kuvaus järjestelmän ja käyttäjän toiminnasta kertoo kuinka käyttäjän tavoite saavutetaan. Tätä kuvausta kutsutaan käyttötapauksen perussuoritukseksi eli onnistuneen perussuorituksen skenaarioksi. Jos järjestelmän tai käyttäjän toiminta voi poiketa onnistuneen perussuorituksen skenaariosta, kirjataan poikkeukset ja niistä toipuminen käyttötapauksen poikkeuksiksi.

Käyttötapausten välisistä suhteista RSI-menetelmä keskittyy lähinnä sisältyä-suhteen käyttämiseen, mutta se ei kuitenkaan sulje pois muidenkin suhteiden käyttöä. Esimerkiksi laajennuksella voidaan kuvata ominaisuutta, joka laajentaa nykyisen käyttötapauksen toiminnallisuutta. Laajennus voi olla nykyiseen toimintoon liittyvä lisäominaisuus, joka voidaan toteuttaa myöhemmin järjestelmän elinkaaren aikana. Laajennus voi olla myös perussuorituksen jonkun askeleen vaihtoehtoinen, yhtä todennäköinen suoritustapa Laajennusten käyttö käyttötapausanalyysissä voi helpottaa käyttötapausten ylläpitoa, kun voidaan keskittyä vain uusiin ominaisuuksiin. Laajennukset voivat myös helpottaa paremman järjestelmäarkkitehtuurin kehittämisessä, kun tiedetään millaisia ominaisuuksia voi tulevaisuudessa tulla. Toisaalta laajennukset voivat vaikeuttaa käyttötapausten ymmärrettävyyttä. Laajennukset voidaan aina kuvata myös niin, että yksi suoritustapa valitaan käyttötapauksen perussuoritukseksi ja vaihtoehtoiset suoritustavat kuvataan tällöin kyseisen suoritustavan poikkeuksina, jotka johtavat samaan lopputulokseen kuin onnistuneen perussuorituksen skenaario.

Lisäksi RSI:ssä käytetään Steve Cookin ja John Danielsin käyttötapausten abstraktion tasoja. Cook ja Daniels havaitsivat Collins-Copen mukaan, että käyttötapauksia voidaan mallintaa kolmesta eri näkökulmasta, jotka ovat käsitteellinen, yksityiskohtainen ja toteutuksellinen näkökulma (Collins-Cope, 2000). Käsitteellinen näkökulma kuvaa järjestelmän ja kohdealueen olioita sekä niiden välisiä suhteita käyttäjien näkökulmasta. Yksityiskohtaisessa näkökulmassa keskitytään järjestelmän käyttämien olioiden ja niiden välisten suhteiden kuvaamiseen. Toteutusnäkö-

kulma kuvaa oliot kokonaisuudessaan. RSI keskittyy näistä kolmesta näkökulmasta käsitteelliseen ja yksityiskohtaiseen näkökulmaan.

RSI-käyttötapausanalyysin käyttötapausten jako kolmeen eri kategoriaan on lähtöisin Mark van Harmelenin tavasta yhdistää käyttöliittymäsuunnittelu ja oliosuunnittelu (van Harmelen, 1996). Van Harmelenin mallissa käyttötapaukset koostuvat tapahtumista, jotka käsittelevät käyttäjien ymmärtämiä objekteja ja niiden välisiä suhteita, ydintapahtumista sekä vuorovaikutustapahtumista. Ydintapahtumia ovat tapahtumat, jotka kuvaavat käyttäjien ymmärtämiä olioita, jotka kuuluvat kehitettävään järjestelmään. Vuorovaikutustapaukset kuvaavat käyttäjien ja järjestelmän välisen vuorovaikutuksen rakenteita ja tapahtumia.

Käyttötapausanalyysin koulutus- ja konsultointitapahtumista keräämiensä kokemusten ja käyttötapausanalyysin tekijöiden kohtaamien ongelmien innoittamana Ratio Group järjesti epävirallisen tutkimuksen käyttötapausten käytöstä yrityksissä (Collins-Cope, 2000). Tutkimus toteutettiin vuonna 1998 ja siihen kerättiin vapaaehtoisia osallistujia ympäri maailman Usenet:n comp.object -uutisryhmän kautta. Tutkimukseen otettiin mukaan yhteensä kymmenen eri esimerkkiä käyttötapauksista. Tutkimuksen perusteella Ratio Group huomasi, että hyvin usein käyttötapausanalyysissä unohdettiin käyttöliittymien suunnittelu ja eri kohdeyleisöt vaativat käyttötapauksilta erilaista rakennetta ja erilaisia kuvauksia (Collins-Cope, 2000). Lisäksi tutkimuksen perusteella pääteltiin, että vaikka käyttötapauksissa oli käytetty erilaisia määrityksiä siitä, miten käyttötapaukset muodostuvat, kaikissa tapauksissa oli kuitenkin pidetty käyttötapausanalyysin tekemistä hyödyllisenä. Tutkimuksessa huomattiin myös, että yrityksissä on jo käytössä vapaamuotoiset käyttötapausten yksityiskohtien luokittelut ja näitä pystytään kehittämään eteenpäin. Lisäksi käyttötapauksissa käytettiin tehokkaasti erilaisia abstraktion ja yksityiskohtien tasoja. Nämä Ratio Groupin päätelmät tutkimuksessa mukana olleista käyttötapauksista ovat vaikuttaneet merkittävästi RSI-menetelmän kehittämiseen muiden käyttötapausanalyysimenetelmien lisäksi.

6.1 Vaatimuskäyttötapaukset

Vaatimuskäyttötapausten tarkoituksena on dokumentoida järjestelmän liiketoiminnalliset vaatimukset järjestelmällisellä tavalla. Vaatimuskäyttötapaukset kuvaavat järjestelmän ja käyttäjän

välisen vuorovaikutuksen. Vaatimuskäyttötapaukset koostuvat käyttötapauskaavioista ja käyttötapausten sanallisista kuvauksista. Käyttötapauskaavio muodostetaan tunnistamalla järjestelmän kaikki mahdolliset eri käyttäjät, jonka jälkeen määritellään kunkin käyttäjän tavoitteet, jotka he haluavat toteuttaa järjestelmän avulla. Tämän jälkeen voidaan piirtää käyttötapauskaavio, jossa näkyy eri käyttäjien tavat käyttää järjestelmää.

Käyttötapausten sanallisissa kuvauksissa esitetään käyttötapauksen tarkoitus, eli käyttäjän toiminnan tavoite, kuten Cockburn esitti (Cockburn, 1997). Käyttötapauksissa tulisi myös kuvata Myerin ehdottamalla tavalla käyttötapauksen alku- ja loppuehdot (Meyer, 1994). Itse käyttötapauksen suoritus kuvataan onnistuneen perussuorituksen skenaarion avulla ja vaihtoehtoiset skenaariot perussuorituksen poikkeuksina. Sekä käyttötapauskaaviot että käyttötapausten sanalliset kuvaukset ovat RSI-menetelmässä tarkoitettu ensisijaisesti asiakkaan ja loppukäyttäjien käyttöön, mutta niitä hyödynnetään myös käyttöliittymä- ja palvelukäyttötapausten tekemisessä.

Vaatimuskäyttötapaukset kuvataan erillisen kaavaimen avulla, jolloin käyttötapaukset ovat osittain muodollisia ja niillä on yhtenäinen sekä selkeä esitystapa. Taulukossa 9 on esitetty RSI:n mukainen vaatimuskäyttötapaus luvun 3 käyttötapauksesta *Nosta rahaa*.

Taulukko 9: Vaatimuskäyttötapaus *Nosta rahaa.*

Tunniste	Nosta rahaa
Toimija(t)	Automaatin peruskäyttäjä
Tavoitteet	Käyttäjä nostaa rahaa omalta tililtään pankkiautomaatin avulla.
Taso	Käyttäjän tavoite
Sisältyvät käyttötapaukset	Syötä kortti
Laajennetut käyttötapaukset	
Alkuehdot	Käyttäjä on syöttänyt automaattikortin ja tunnusluvun automaattiin suorittamalla onnistuneesti käyttötapauksen Syötä kortti.
Loppuehdot	Käyttäjä saa takaisin automaattikortin, rahat ja kuitin. Järjestelmä veloittaa käyttäjän tiliä nostettavalla summalla.
Onnistuneen perussuorituksen skenaario	1. Käyttäjä on syöttänyt pankkikortin ja tunnusluvun onnistuneesti. Järjestelmä kysyy käyttäjältä valittavaa toimintoa. 2. Käyttäjä valitsee rahan noston. 3. Järjestelmä kysyy nostettavaa summaa. 4. Käyttäjä syöttää nostettavan summan. 5. Järjestelmä tarkastaa, että tilillä on olemassa nostettavan summan verran ra-

<table>
<tr><td></td><td>haa, ja että kortin nostoraja ei täyty nostettavasta summasta. Järjestelmä vähentää tilin saldoa käyttäjän nostaman summan verran.
6. Järjestelmä kysyy käyttäjältä, haluaako tämä tulostaa tapahtumasta kuitin.
7. Käyttäjä valitsee kuitin tulostuksen.
8. Järjestelmä pyytää käyttäjää ottamaan kortin pois.
9. Käyttäjä ottaa pankkikortin pois automaatista.
10. Järjestelmä antaa rahat ja kuitin. Käyttötapaus päättyy.</td></tr>
<tr><td>Poikkeukset</td><td>
<table>
<tr><th>Kuvaus poikkeuksesta</th><th>Kuvaus käyttäjän ja järjestelmän välisestä vuorovaikutuksesta</th></tr>
<tr><td>4.1 Käyttäjän tilillä ei ole nostettavaa summaa rahaa.</td><td>Järjestelmä ilmoittaa käyttäjälle, että tilillä ei ole tarpeeksi saldoa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.</td></tr>
<tr><td>4.2 Kortin nostoraja ylittyy käyttäjän syöttämästä summasta.</td><td>Järjestelmä ilmoittaa käyttäjälle, että tililtä ei voida nostaa niin suurta summaa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.</td></tr>
<tr><td>6.1 Kuittipaperi on loppunut automaatista.</td><td>Järjestelmä ilmoittaa käyttäjälle, että kuittipaperi on loppunut eikä sitä voida tulostaa. Järjestelmä kysyy näytetäänkö saldo näytöllä. Käyttäjä valitsee saldon tulostamisen näytölle. Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
<tr><td>6.1.1 Käyttäjä ei valitse saldon tulostamista näytölle.</td><td>Käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
<tr><td>7.1 Käyttäjä ei valitse kuitin tulostamista paperille, vaan saldon näyttämisen näytöllä.</td><td>Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
<tr><td>7.2 Käyttäjä ei valitse kuitin tulostamista paperille, eikä saldon näyttämisen näytöllä vaan tapahtuman lopettamisen ilman saldon tulostamista.</td><td>Käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
</table>
</td></tr>
<tr><td>Tekniset kaaviot</td><td>Sekvenssikaavio Nosta rahaa</td></tr>
<tr><td>Käyttöliittymämääritykset</td><td>Käyttöliittymäkäyttötapaukset Nostosumman valinta, Kuitti</td></tr>
<tr><td>Muutoshistoria</td><td>NJ, 5.8.2006 Käyttötapauksen ensimmäinen versio
NJ, 10.8.2006 Lisätty poikkeukset 7.1 ja 7.2</td></tr>
</table>

6.2 Palvelukäyttötapaukset

Kun vaatimuskäyttötapaukset kuvaavat sen, miten järjestelmän toiminta ilmenee käyttäjälle, palvelukäyttötapaukset kuvaavat sen, mitä järjestelmän sisällä tapahtuu käyttäjälle näkymättömissä. Palvelukäyttötapaukset ovat itsenäisiä eivätkä ne sisällä käyttäjän vuorovaikutusta. Tällöin siis

vaatimuskäyttötapausten automatisoidut eli järjestelmän suorittamat askeleet ovat hyvin todennäköisesti palvelukäyttötapauksia.

Palvelukäyttötapauksia on sellaisia, jotka päivittävät järjestelmän tilaa, sekä sellaisia, jotka palauttavat tietoa järjestelmän tilasta muuttamatta sitä. Toinen tapa jaotella palvelukäyttötapauksia, on erotella niistä välttämättömät palvelukäyttötapaukset. Välttämättömät käyttötapaukset toteuttavat vaatimuskäyttötapausten edellyttämiä asioita riippumatta siitä, kuinka käyttöliittymä on toteutettu. Tämän erottelun avulla voidaan analysoida vaatimuskäyttötapauksissa tapahtuvien muutosten vaikutukset järjestelmän toimintaan.

Palvelukäyttötapaukset koostuvat sanallisista kuvauksista, jotka sisältävät palvelukäyttötapauksen alku- ja loppuehdot, käyttötapauksen vastaanottamat parametrit kutsuvasta ympäristöstä sekä käyttötapauksen palauttamat palautusarvot. Päivittävät palvelukäyttötapaukset saavat parametrinaan yleensä kyselykäyttötapausten palautusarvoja. Käyttöliittymä sitoo yhteen sekä päivittävät että kyselevät palvelukäyttötapaukset. Collins Cope ei suosittele sisältää-suhteen käyttämistä palvelukäyttötapausten tekemisessä, sillä se puuttuisi liikaa ohjelmiston tekniseen suunnitteluun (Collins-Cope, 2000).

Luvun 3 pankkiautomaattiesimerkkiin voisi liittyä esimerkiksi palvelukäyttötapaus *Veloita tiliä*, jossa käyttäjän tilin saldoa vähennetään käyttäjän valitseman nostosumman verran. Tämä palvelu liittyy rahan nostamiseen, ja se suoritetaan käyttötapauksen *Nosta rahaa* aikana sen jälkeen, kun käyttäjä on valinnut nostettavan summan ja järjestelmä on todennut summan olevan kelvollinen. Tämä palvelukäyttötapaus voisi olla esimerkiksi seuraavanlainen:

Veloita tiliä

Syötteet: nostettavaSumma, Tili
Tulosteet: status (ok/failed)
Alkuehdot: Tili.tila != suljettu, Sulkulista.isClosed(Tili.korttiNro) == false,
nostettavaSumma ≥ 20, nostettavaSumma ≤ Tili.saldo,
nostettavaSumma ≤ Tili.nostoraja
Loppuehdot: Tili.saldo = Tili.saldo – nostettavaSumma,
Tili.nostoraja = Tili.nostoraja – nostettavaSumma

Tässä palvelukäyttötapauksessa syötteeksi saadaan nostettava summa sekä käsiteltävä tili oliona. Käyttötapaus palauttaa totuusarvon sen mukaan, onnistuuko tilin velottaminen vai ei. Palvelun alkuehtona on, että nostettava tili on avoinna, automaattiin syötetty kortti ei ole sulkulistalla, eli se ei ole varastettu tai kadonnut, ja että nostettava summa on kelvollinen. Palvelukäyttötapauksen jälkeen käyttäjän tilin saldoa sekä päivittäistä nostorajaa on pienennetty käyttäjän valitseman nostosumman verran. Tämä palvelukäyttötapaus on siis tyypiltään pääpalvelukäyttötapaus sekä päivittävä palvelukäyttötapaus. Kyselevä palvelukäyttötapaus liittyen käyttötapaukseen *Nosta rahaa* voisi olla esimerkiksi seuraavanlainen:

```
Tarkasta nostettava summa

Syötteet: nostettavaSumma, Tili
Tulosteet: status (ok/failed)
Alkuehdot: Tili.tila != suljettu, Sulkulis-
ta.isClosed(Tili.korttiNro) == false
```

Palvelukäyttötapauksessa *Tarkasta nostettava summa* tarkastetaan onko nostettava summa pienempi tai yhtä suuri kuin tilin nostoraja sekä onko nostettava summa pienempi tai yhtä suuri kuin tilin saldo. Koska tämä palvelukäyttötapaus on kyselevät palvelukäyttötapaus, siinä ei ole loppuehtoja, sillä järjestelmän tila ei muutu käyttötapauksen aikana. Palvelukäyttötapaus palauttaa vain tiedon siitä, onko nostettava summa kelvollinen vai ei.

Palvelukäyttötapaukset menevät huomattavan paljon syvemmälle järjestelmän toimintaan kuin vaatimuskäyttötapaukset, joten näiden käyttötapausten kohdeyleisönä ovat lähinnä vain järjestelmän suunnittelijat ja kehittäjät, ei niinkään loppukäyttäjät tai asiakas

6.3 Käyttöliittymäkäyttötapaukset

Käyttöliittymäkäyttötapausten tavoitteena on antaa yksityiskohtainen kuvaus käyttöliittymästä ja sen toiminnasta. Käyttöliittymäkäyttötapaukset kuvataan usein joko prototyypin avulla tai piirtämällä hahmotelma käyttöliittymästä paperille. Käyttöliittymäkäyttötapauksissa kuvataan kaikki käyttäjän järjestelmälle antamat syötteet, syötteiden tyypit ja rajoitteet, sekä painikkeiden ja valikoiden käynnistämät toiminnot. Käyttöliittymäkäyttötapaukset kuvaavat siis informaatiota, jonka

järjestelmä saa palvelukäyttötapausten suorittamiseksi tai tietoa, jonka järjestelmä tuottaa palvelukäyttötapausten avulla käyttäjälle näytettäväksi. Käyttöliittymäkäyttötapausten toteuttamisesta vastaa yleensä käyttöliittymäsuunnittelija ja näiden käyttötapausten kohteena ovat sekä loppukäyttäjät että järjestelmäsuunnittelijat.

Alla oleva pankkiautomaatti esimerkkiin liittyvä käyttöliittymäkäyttötapaus *Kuitti* esittää automaatin tulostamaa kuittia rahan noston yhteydessä. Tästä käyttöliittymäkäyttötapauksesta voidaan nähdä, mitä tietoja tilistä ja suoritetusta tapahtumasta tulostetaan noston jälkeen kuitille. Käyttöliittymäkäyttötapaukset koostuvat kahdesta osasta: käyttöliittymän muotoilun kuvauksesta sekä toiminnallisuuden kuvauksesta. Muotoiluosassa vaihtuvat tekstit, kuten esimerkiksi automaattikohtainen osoite ja numero, on kuvattu väkässuluissa <> erotuksena käyttöliittymään sellaisenaan tulevasta tekstistä. Numeeriset tiedot on merkitty #-merkeillä. Nostosumma merkitään lisäksi miinuksella ja tilin tiedot plussalla. Toiminnallisuudesta kuvataan, mitä palvelukäyttötapauksia käyttöliittymään liittyy. Alla olevaan käyttöliittymäkäyttötapaukseen liittyy kyseleviä palvelukäyttötapauksia, joiden avulla haetaan eri tietoja kuitille.

Muotoilu:

Kuitti

<Automaatin osoite>
<Automaatin numero>

Päiväys	**Kello**	**Tapahtuman nro**	**Kortin nro**
pp.kk.vv	**hh.mm**	**####**	**#####**

<kortin pitkä numero ja tyyppi>
Otto euroa **####.##-**

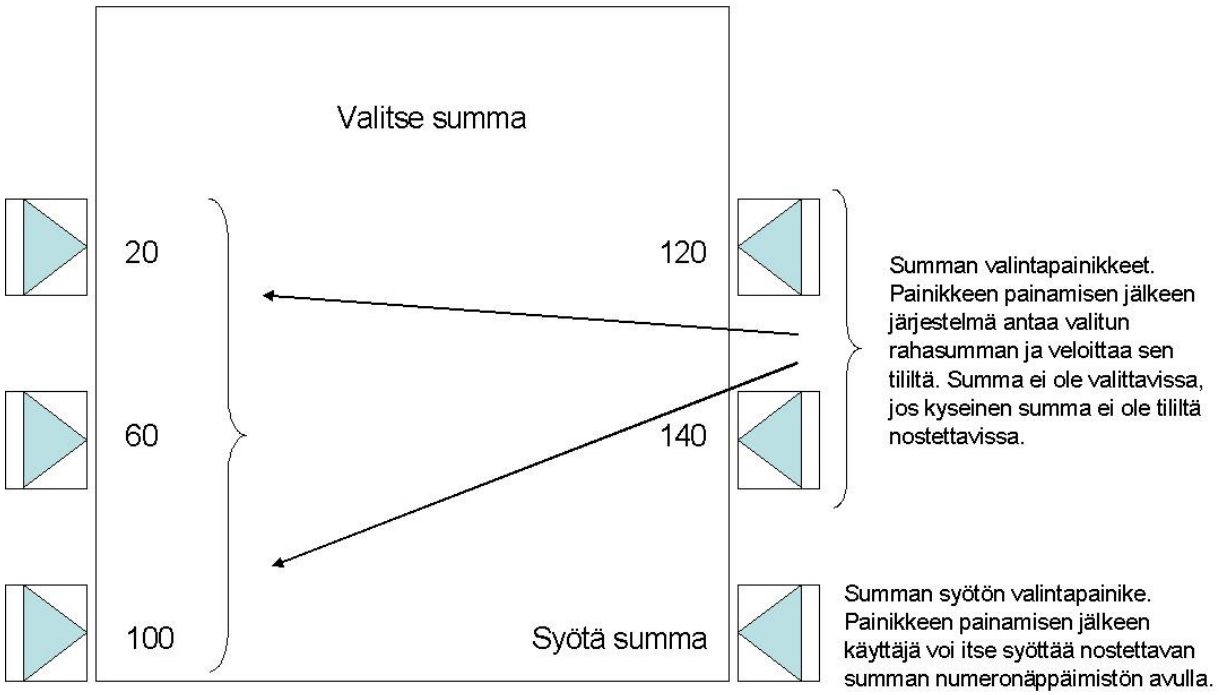
Tilitiedot	**Yksikkö €**
Tilin saldo	**########.##+**
Tililtä nostettavissa	**########.##+**
Kortin käteisnostovara	**####.##+**

<Viesti>

Toiminnallisuus:
1. **Kuitin muodostaminen: <<kysely>> *Muodosta kuitti***

Palvelukäyttötapauksia voidaan käyttää apuna käyttöliittymäkäyttötapausten muodostamisessa, koska palvelukäyttötapausten syötteet vaativat yleensä käyttäjältä kerättävää tietoa, kuten esimerkiksi automaattikortin ja tunnusluvun, ja tulokset voivat olla tietoa, joka tulee esittää käyttäjälle, kuten esimerkiksi tapahtuman tietojen tulostaminen kuitille.

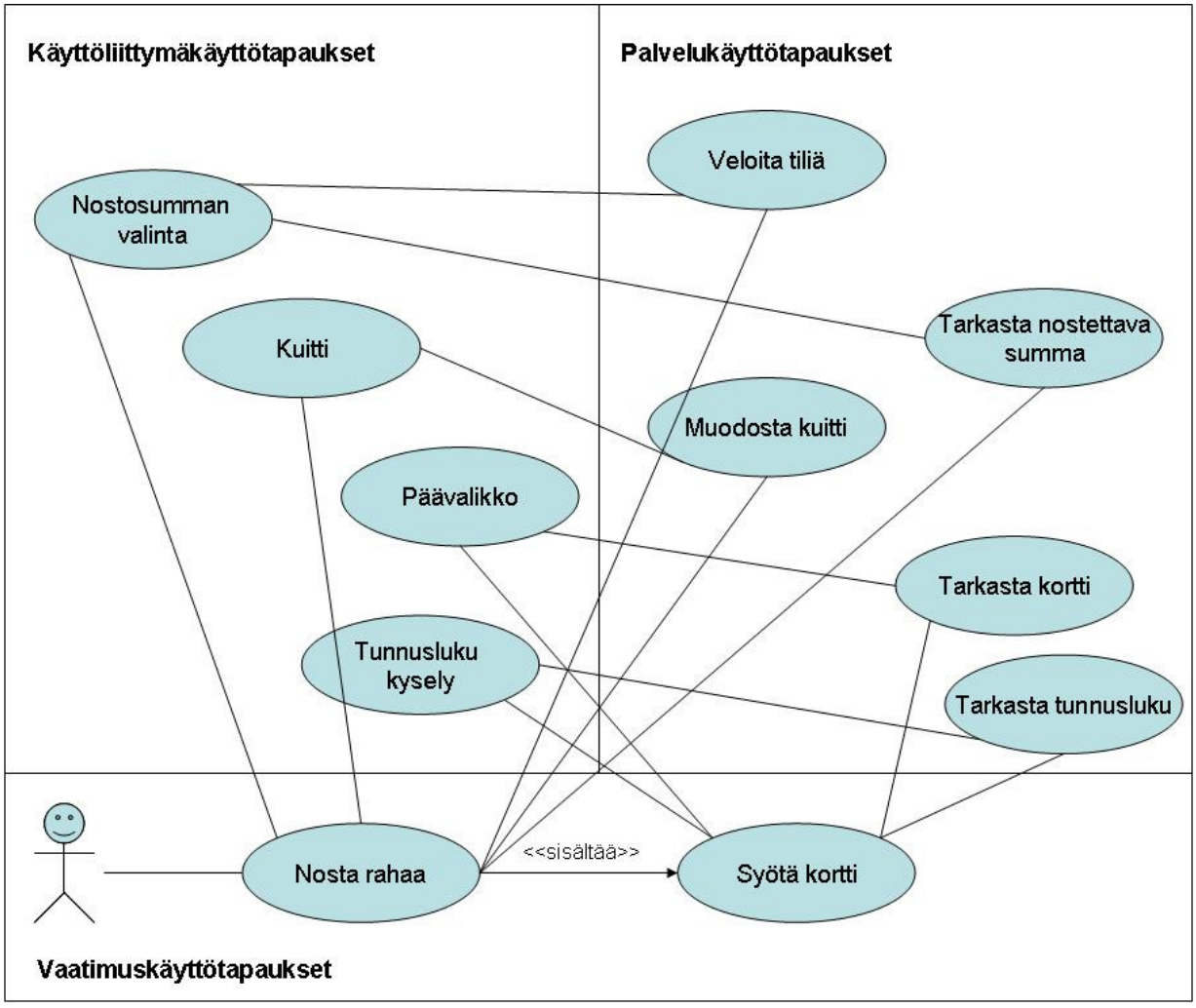
Käyttöliittymiin voi liittyä myös päivittäviä palvelukäyttötapauksia, esimerkiksi jos käyttöliittymään kuuluu hyväksymispainike, jonka painamisen yhteydessä järjestelmä tallentaa käyttäjän muuttamat tiedot. Palvelukäyttötapausten alkuehdot voivat asettaa käyttöliittymälle vaatimuksia, joiden on toteuduttava ennen palvelun suorittamista. Tällöin käyttöliittymä voidaan suunnitella niin, että käyttäjän tekemien virheiden mahdollisuus olisi mahdollisimman pieni. Tämä voi onnistua, jos käyttöliittymä suunnitellaan niin, että alkuehto toteutuu aina. Esimerkiksi pankkiautomaatin tapauksessa käyttöliittymä voitaisiin suunnitella niin, että käyttäjä voisi valita nostettavaksi summaksi vain sellaisia summia, jotka ovat enemmän kuin 20 € ja vähemmän tai yhtä paljon kuin tilin nostoraja tai saldo. Tällöin käyttäjä ei voi valita virheellistä summaa ja rahan nosto onnistuu aina, jos käyttäjä pääsee summan valintaan asti. Kyseinen käyttöliittymäkäyttötapaus *Nostosumman valinta* esimerkki on esitetty kuvassa 22.



Kuva 22: Käyttöliittymäkäyttötapaus *Nostosumman valinta*.

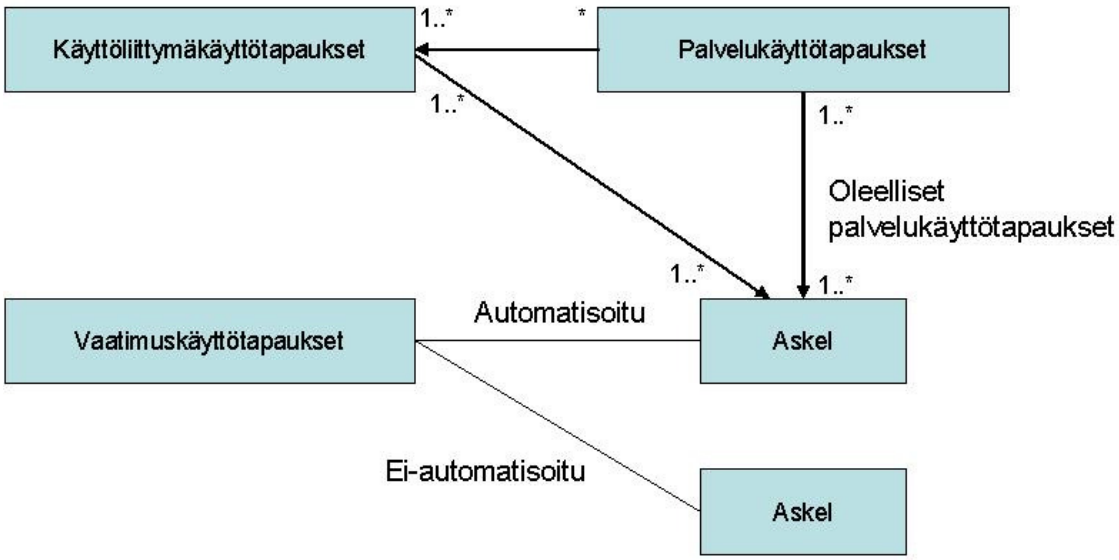
6.4 Jäljitettävyysmalli

RSI-käyttötapausanalyysin neljäs osa on jäljitettävyysmalli, joka yhdistää eri käyttötapaukset yhteen. Jäljitettävyysmalli on oleellinen osa RSI-analyysin käytettävyyttä, sillä mallin avulla voidaan jäljittää riippuvuudet eri käyttötapausten välillä. Jäljitettävyysmalli muodostetaan jäljittämällä oleelliset palvelukäyttötapaukset vaatimuskäyttötapauksista, kaikki käyttöliittymiin liittyvät palvelut sekä vaatimuskäyttötapausten askelista käyttöliittymäkäyttötapauksiin. Kuvassa 23 on esitetty pankkiautomaattijärjestelmän jäljitettävyysmalli rahan nostamisen osalta.



Kuva 23: Pankkiautomaatti järjestelmän jäljitettävyysmalli käyttötapauksen *Nosta rahaa* osalta.

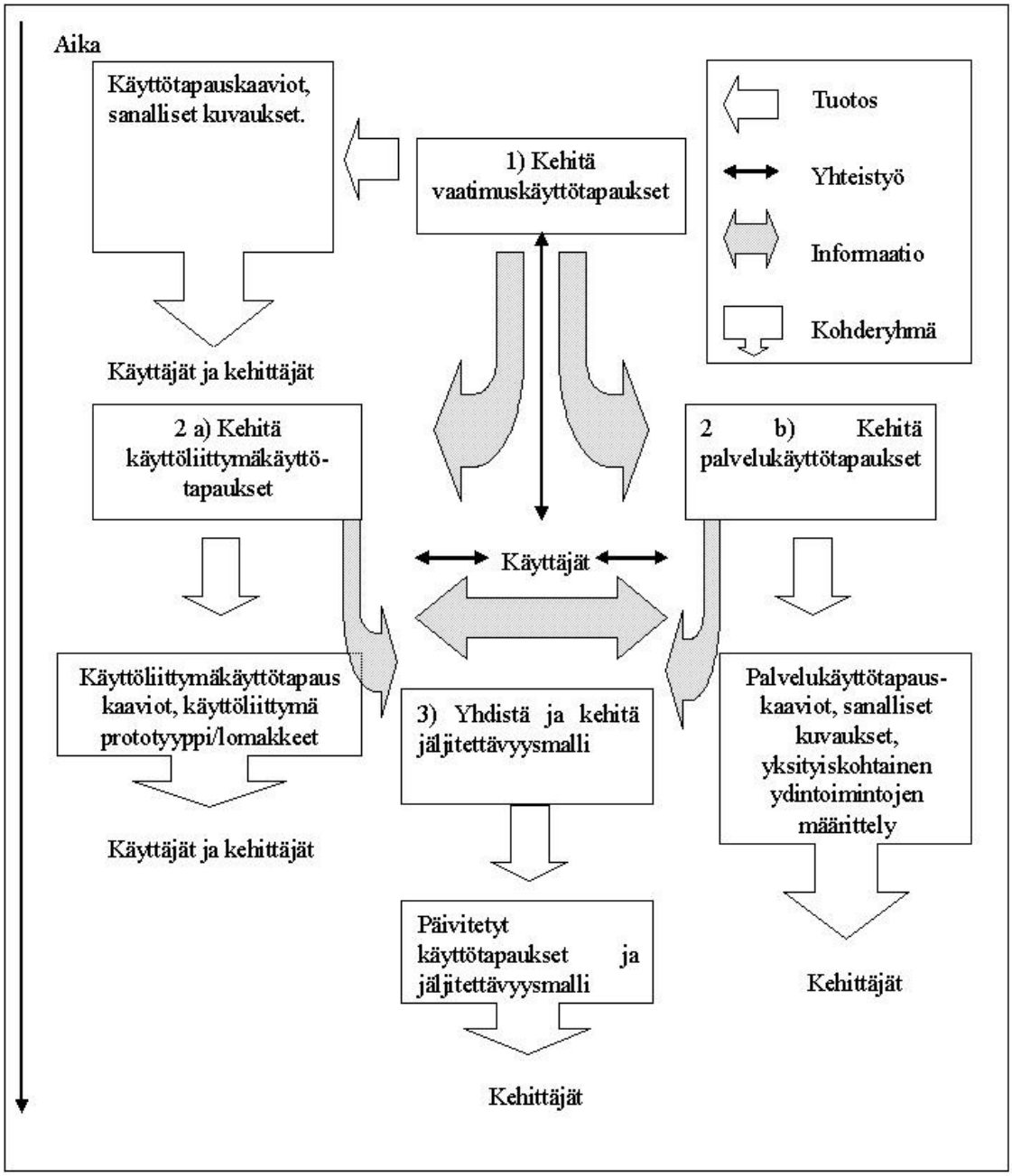
Jäljitettävyysmallista tulee tarkastaa että jokaista vaatimuskäyttötapauksen automatisoitua eli järjestelmän suorittamaa askelta vastaa palvelukäyttötapaus ja käyttöliittymäkäyttötapaus. Jokainen palvelukäyttötapaus liittyy ainakin yhteen käyttöliittymäkäyttötapaukseen ja yhteen vaatimuskäyttötapaukseen. Lisäksi jokaista käyttöliittymäkäyttötapauksen tulee vastata ainakin yksi automatisoitava vaatimuskäyttötapauksen askel ja jokaiseen käyttöliittymäkäyttötapaukseen liittyy ainakin yksi palvelukäyttötapaus. Kuvassa 24 on esitetty nämä jäljitettävyysmallin käyttötapausten väliset suhteet.



Kuva 24: RSI-jäljitettävyysmalli eri käyttötapaustyyppien välillä.

6.5 Käyttötapausanalyysi prosessi

RSI-käyttötapausanalyysi on kolmivaiheinen prosessi. Prosessin ensimmäisenä tavoitteena on vaatimuskäyttötapausten muodostaminen, jonka jälkeen vaatimuskäyttötapausten avulla tehdään palvelukäyttötapaukset ja käyttöliittymätapaukset niiltä osin, mitkä toiminnot automatisoidaan. Palvelu- ja käyttöliittymäkäyttötapaukset kehitetään rinnakkain. Kun kaikki käyttötapaukset ovat valmiina, muodostetaan jäljitettävyysmalli, jonka avulla linkitetään eri käyttötapaukset yhteen. Prosessin työskentelymenetelmät ovat sekä inkrementaalisia (käyttöliittymä- ja palvelukäyttötapausten osalta) että iteratiivisia. Kuva 25 havainnollistaa RSI-prosessin työvaiheita, tuotoksia, tuotosten kohderyhmiä sekä eri vaiheiden välistä tiedon siirtoa.



Kuva 25: RSI prosesssi (Collins-Cope 2000).

Pienissä projekteissa järjestelmän suunnitteluprosessi voidaan aloittaa suoraan vaatimuskäyttötapausten suunnittelusta, mutta suuremmissa projekteissa voi olla tarpeen tehdä esimäärittelydokumentti asiakkaan kanssa. Vaatimuskäyttötapausten tekemisen aliprosesseja ovat mahdollisen esimäärittelyn tarkasteleminen käyttäjien kanssa ja liiketoimintaprosessien sekä järjestelmän pää-

käyttötapojen sekä niiden poikkeuksien ja poikkeusten poikkeuksien tunnistaminen kunnes ei ole enää lisää poikkeuksia. Näiden perusteella laaditaan käyttötapauskaaviot ja sanalliset kuvaukset, joita viimeistellään ja arvioidaan. Jos arvioinnissa löydetään parannettavaa, toistetaan vaatimuskäyttötapausten aliprosesseja, kunnes parannettavaa ei enää ole.

Kun vaatimuskäyttötapaukset ovat valmiina, voidaan aloittaa käyttöliittymäkäyttötapausten ja palvelukäyttötapausten tekeminen. Käyttöliittymäkäyttötapausten syötteenä toimivat vaatimuskäyttötapausten lisäksi myös yhtä aikaa kehitettävät palvelukäyttötapaukset. Käyttöliittymäkäyttötapausten avulla kuvataan siis vaatimuskäyttötapausten suorittamiseksi tarvittavat käyttöliittymät, joiden avulla käyttäjä pyytää järjestelmää suorittamaan palveluita, jotka yhdessä toteuttavat käyttäjän asettamat tavoitteet. Käyttöliittymä- ja palvelukäyttötapausten kehittäminen on inkrementaalista, eli osa sekä palvelu- että käyttöliittymäkäyttötapauksista kehitetään kokonaisuudessaan iteratiivisesti, jonka jälkeen aloitetaan seuraavien palvelu- ja käyttöliittymäkäyttötapausten kehittäminen alusta.

Kun tarvittavat käyttöliittymät ja palvelut on dokumentoitu, tulee tarkastaa, että jokaista vaatimuskäyttötapauksen automatisoitua askelta vastaa ainakin yksi käyttöliittymä ja yksi palvelu. Käyttöliittymä- ja palvelukäyttötapausten kehittäminen on myös iteratiivista, jolloin käyttötapauksia muokataan, lisätään ja poistetaan tarvittaessa, mikäli esimerkiksi palvelukäyttötapauksissa tapahtuu muutoksia, jotka vaikuttavat käyttöliittymiin.

7. Testaaminen RSI-käyttötapausanalyysissä

Sen lisäksi, että käyttötapauksia käytetään järjestelmän vaatimusten määrittämiseen ja dokumentoimiseen, on hyvin houkutteleva ajatus käyttää niitä myös myöhemmissä järjestelmän kehitysvaiheissa, kuten testaamisessa, koska ne sisältävät jo valmiiksi järjestelmän toiminnan ja toiminnan rajoitteiden kuvaukset. RSI-käyttötapausanalyysi vie käyttötapausten käytön normaalia vaatimusmäärittelyvaihetta pidemmälle esittelemällä erilliset käyttötapaustyypit järjestelmän sisäisen toiminnan ja käyttöliittymän kuvaamiselle. Myös näitä käyttötapaustyyppejä voidaan käyttää apuna käyttötapauspohjaisessa testaamisessa. Vaatimuskäyttötapauksia voidaan käyttää lähinnä itse käyttötapausmallin testaamiseen sekä toiminnallisten testitapausten kehittämiseksi esimerkiksi järjestelmä- ja hyväksymistestausta varten. Palvelukäyttötapauksia voidaan mahdollisesti hyödyntää järjestelmän sisäisen rakenteen testaamisessa. Käyttöliittymäkäyttötapauksia voidaan puolestaan käyttää kehitettävän järjestelmän hyvin varhaisena prototyyppinä, joten niille voidaan suorittaa sekä perinteistä käytettävyystestausta että käytettävyyden arviointia.

RSI-käyttötapausanalyysin vaatimuskäyttötapaukset vastaavat perinteisiä käyttötapauksia, jotka sisältävät siis käyttötapauskaaviot sekä käyttötapausten sanalliset kuvaukset. Käyttötapauskaavioita voidaan käyttää käyttötapausten rakenteelliseen testaamiseen, joka esiteltiin kohdassa 5.5. Käyttötapauskaavioiden testaamisen avulla voidaan varmistua itse käyttötapausmallin oikeellisuudesta. Käyttötapausten sanallisten kuvausten testaamiseksi on esitetty muun muassa erityisiä tarkastustilaisuuksia, joiden tarkoituksena on ennemminkin etsiä virheitä käyttötapausmallista eikä niinkään saavuttaa yksimielisyyttä vaatimuksista (Anda & Sjøberg, 2002). Koska koko järjestelmän kehitys nojaa käyttötapauksiin ja niitä käytetään järjestelmän vaatimusten dokumentoimiseen, on tärkeää varmistua, että itse käyttötapaukset on tehty oikein. Käyttötapausten sanallisia kuvauksia voidaan käyttää myös vaatimusten testaamiseen, niin kuin kohdassa 5.3 kuvattiin Wiegersin mallin mukaisesti.

Myöhemmässä kehitysvaiheessa käyttötapausten sanallisista kuvauksista voidaan Binderin esittämän mallin mukaisesti kehittää laajentavia käyttötapauksia, joista voidaan muodostaa toiminnallisia testitapauksia järjestelmätestausta varten. Laajentavat käyttötapaukset ja niistä muodostettavat testitapaukset esiteltiin kohdassa 5.2. Testitapauksia voidaan käyttää myös hyväksymis-

testauksessa, sillä hyväksymistestauksessa testataan, että järjestelmä toteuttaa käyttäjien asettamat vaatimukset, ja koska käyttötapaukset sisältävät nimenomaan käyttäjien vaatimukset heidän näkökulmastaan, soveltuvat ne myös hyväksymistestauksen testitapauksiksi. Lisäksi Nielsenin mukaan järjestelmän toiminnan kuvaavia skenaarioita, kuten käyttötapausten sanallisia kuvauksia, voidaan käyttää järjestelmän suunnittelun aikaisena prototyyppinä, jolle voidaan suorittaa käytettävyyden arviointia (Nielsen, 1993).

Palvelukäyttötapaukset ovat formaalissa muodossa esitettyjä kuvauksia järjestelmän sisäisestä toiminnasta ilman käyttäjän vuorovaikutusta. Koska nämä käyttötapaukset sisältävät formaalin notaation, voisi olla mahdollista kehittää testaustyökalu, joka voisi automaattisesti testata palveluiden rakenteellista oikeellisuutta, kuten Carniello et al. esittelemä UCT-testaustyökalu, jonka avulla testataan käyttötapauskaavioiden rakenteellista oikeellisuutta (Carniello et al., 2005). Testaustyökalu voisi sisältää myös toiminnon koko järjestelmän rakenteen testaamiseksi jäljitettävyysmallin avulla. Jäljitettävyysmallihan sisältää koko järjestelmän rakenteen esittelemällä eri käyttötapaustyyppien väliset suhteet.

Käytettävyystestauksen tarkoituksena on varmistaa, että kehitettävästä järjestelmästä tulee helppo oppia ja käyttää, sen käyttäminen on tyydyttävää sekä hyödyllistä ja että se on käytännöllinen loppukäyttäjien mielestä (Rubin, 1994). Muita testaamisen avulla saavutettavia tavoitteita ja hyötyjä ovat vertailutietojen tuottaminen seuraavia testauksia varten, käyttötuesta aiheutuvien kustannusten ja riskien minimointi, myynnin lisääminen sekä kilpailuedun saavuttaminen (Rubin, 1994).

Käyttöliittymäkäyttötapaukset voivat toimia järjestelmän prototyyppinä, jolle voidaan näin ollen suorittaa myös käytettävyystestausta ja käytettävyyden arviointia. Käytettävyyden arviointimenetelmät ja käytettävyystestaus eroavat toisistaan siten, että käytettävyystestaus suoritetaan aina aitojen loppukäyttäjien kanssa, kun taas käytettävyyden arviointimenetelmissä yksi tai useampi henkilö, jotka tuntevat yleiset käytettävyyden periaatteet, suorittavat käyttöliittymän arvioinnin käyttäen jotakin tiettyä menetelmää.

Koska käyttöliittymäkäyttötapaukset ovat paperille luonnosteltuja kuvauksia mahdollisesta käyttöliittymästä, eivätkä ne siis sisällä ohjelmakoodia, voidaan järjestelmän kehityksen aikaisessa vaiheessa kehittää samasta käyttöliittymästä useita, hyvin erilaisia käyttöliittymäkäyttötapauksia, joiden avulla voidaan suorittaa vertailevaa käytettävyystestausta. Vertailevan testaamisen tarkoituksena on yleisesti todentaa, mikä vaihtoehto on helpompi käyttää tai oppia, tai parantaa kehittäjien ymmärrystä siitä, mitä hyviä ja huonoja puolia eri suunnittelumalleilla on (Rubin, 1994). Rubinin mukaan parhaita tuloksia ja luovimpia ratkaisuja saadaan silloin, kun vertailtavana on hyvin erilaisia ja merkittävästi toisistaan eroavia vaihtoehtoja.

Käytettävyyden arviointimenetelmistä voidaan käyttöliittymäkäyttötapauksien arviointiin käyttää esimerkiksi heuristista arviointia. Heuristinen arviointi on Jacob Nielsenin kehittämä arviointimenetelmä edulliseen ja nopeaan käytettävyyden arviointiin (Nielsen & Mack, 1994). Arvioinnin suorittaa pieni joukko asiantuntijoita, jotka arvioivat käyttöliittymän käytettävyyttä *heuristiikkojen* eli käytettävyysperiaatteiden avulla (Nielsen & Mack, 1994). Heuristisen arvioinnin avulla löydetään sekä vakavia että vähemmän vakavia käytettävyysongelmia. Yleensä löydetyt ongelmat ovat vähemmän vakavia käytettävyysongelmia, jotka kuitenkin voivat oleellisesti haitata järjestelmän käyttöä. Heuristisen arvioinnin ongelmana on, että arvioinnissa saattaa jäädä huomaamatta kohdealueen tietämykseen liittyviä käytettävyysongelmia, koska arvioijat ovat yleensä käyttöliittymäsuunnittelun ja käytettävyyden asiantuntijoita.

Osallistava ryhmäläpikäynti on käytettävyyden arviointimenetelmä, joka eroaa muista arviointimenetelmistä siinä, että sen suorittamiseen tarvitaan aina vähintään yksi loppukäyttäjä. Lisäksi arviointiin osallistuu ainakin yksi suunnittelija ja yksi käytettävyysasiantuntija (Kotkaluoto, 2005). Ryhmäläpikäynnissä jokainen osallistuja käy läpi näyttökuvat yksi kerrallaan ja kirjaa itsenäisesti etenemisehdotuksensa kuvaan, jonka jälkeen seuraa ryhmäkeskustelu, jossa käydään läpi eri vaihtoehdot sekä mahdolliset ongelmatilanteet (Kotkaluoto, 2005). Osallistavassa ryhmäläpikäynnissä voidaan käyttää käyttöliittymäkäyttötapauksia käytettävyyden arviointiin, kun käyttöliittymän kuvauksesta poistetaan arvioinnin ajaksi kaikki käyttöliittymää selittävät kommentit.

Kotkaluodon mukaan osallistavan ryhmäläpikäynti -menetelmän käyttöön liittyy vielä joitakin epäselviä kysymyksiä, koska käytöstä ei ole vielä tarpeeksi käyttökokemustietoa (Kotkaluoto, 2005). Selvittämistä tarvitaan vielä esimerkiksi siihen, miten paljon käyttäjiä pitäisi osallistua, että määrä olisi ihanteellinen löydettyjen ongelmien ja käytettyjen resurssien suhteen tai miten paljon yksittäisen käyttäjän mielipiteisiin voi luottaa. Lisäksi, jos resursseja on runsaasti käytettävissä tuotteen kehityksen aikana, kuinka monta osallistavaa ryhmäläpikäyntiä kannattaa järjestää. Menetelmä mahdollistaa kuitenkin käyttäjien osallistumisen käytettävyyden arviointiin jo kehityksen aikaisessa vaiheessa, eikä arviointi vaadi niin paljon resursseja, kuin esimerkiksi käyttäjien kanssa suoritettu käytettävyystestaus, joten se voi toimia korvaavana menetelmänä aikaisessa käytettävyystestauksen vaiheessa.

Kun käytettävyyden testaaminen otetaan mukaan järjestelmän kehitykseen, joudutaan myös muuttamaan kehitysprosessia, koska perinteiset ohjelmistotuotannon prosessit eivät huomioi käytettävyysnäkökulmia järjestelmän kehityksessä. Käytettävyyden arvioinnista ei myöskään ole suurta hyötyä, jos käyttäjien erikoisominaisuuksia ei ole otettu huomioon jo kehityksen aikaisessa vaiheessa, sillä muutosten tekeminen käyttöliittymään käytettävyyden parantamiseksi voi olla hyvin vaikeaa ja kallista. Kohdassa 6.5 esitellyssä RSI-käyttötapausanalyysin prosessikuvauksessa ei ole otettu huomioon käytettävyysnäkökulmia, joten kyseistä prosessin kuvausta tulee muuttaa käyttäjäkeskeisemmäksi, jos käyttötapauksia halutaan käyttää laajamittaisesti testaamisessa.

7.1 Käyttäjäkeskeinen RSI-käyttötapausanalyysiprosessi

Tässä luvussa kuvattu elinkaarimalli yhdistää OMT-menetelmän, joka edustaa perinteisiä, oliopohjaisia ohjelmistotuotannon prosesseja, käyttäjäkeskeisen RESPECT-menetelmä, joka on käyttäjäkeskeisen kehittämisen ISO 13407 -standardin käytännön sovellus sekä RSI-käyttötapausanalyysin, joka on käyttötapauspohjainen vaatimusmäärittelymenetelmä toiminnallisten vaatimusten keräämiseen. Näiden eri mallien yhdistämiseen käytetään ACUDUC-mallissa esitettyjä periaatteita perinteisten ja käyttäjäkeskeisten mallien yhdistämiseksi käyttötapausten avulla. Nämä neljä eri menetelmää yhdistämällä saadaan käyttäjäkeskeinen, käyttötapauspohjainen, iteratiivinen ja inkrementaalinen elinkaarimalli oliopohjaisten, interaktiivisten järjestelmien

kehittämiseksi ottaen huomioon käytettävyysnäkökulmat, tekninen suunnittelu sekä kehitysorganisaation työn organisointi.

Tähän malliin päädyin, koska kuten tämän tutkielman aikana on monesti todettu, ovat käyttötapaukset laajalti käytössä oleva menetelmä järjestelmän vaatimusten määrittämiseen. Käyttötapausten käyttö on helppoa ja nopeaa oppia, luonnollisella kielellä kuvattuna ne tarjoavat kehittäjille ja käyttäjille molempien osapuolien ymmärtämän kommunikointitavan ja ne muodostavat järjestelmän kuvauksen nimenomaan käyttäjien näkökulmasta.

Käyttäjäkeskeinen kehittäminen huomioi puolestaan kehitettävän järjestelmän eri käyttäjäryhmät, ja heidän ominaisuuksiinsa liittyvät erityisvaatimukset. Tämä parantaa järjestelmän käytettävyyttä, joka taas lisää käyttäjien tyytyväisyyttä ja mahdollisesti parantaa kehitysorganisaation kilpailuasemaa. Lisäksi paremman käytettävyyden avulla voidaan saada aikaan kustannussäästöjä esimerkiksi vähentyvän käyttötuen ja koulutuksen kautta. Kehitysprojektin epäonnistumisen riskiä sekä kustannuksia voidaan myös vähentää jatkuvalla testaamisella. Kun tehtyjä kehitysratkaisuja ja toteutusta arvioidaan säännöllisesti, voidaan virheet havaita nopeammin ja aikaisemmassa vaiheessa, jolloin virheiden korjaaminen on myös nopeampaa ja edullisempaa. Jatkuvasti muuttuvassa toimintaympäristössä myös järjestelmälle asetettavat vaatimukset voivat muuttua nopeasti, jolloin inkrementaalisen eli vähittäisen kehittämisen avulla voidaan sopeutua muutoksiin myös kehitysprosessin aikana.

Käytettävyystestauksen mukaan ottaminen RSI-analyysiprosessiin muuttaa myös itse prosessia. Prosessin tulee sisältää toimintoja, joiden avulla selvitetään esimerkiksi ketä järjestelmän käyttäjät ovat, millaisia erityisominaisuuksia heillä on, miten he tällä hetkellä suorittavat työtehtävänsä ja mitä käytettävyystavoitteita järjestelmälle asetetaan. ISO 13407 -standardin mukaan käyttäjäkeskeisen järjestelmäkehityksen tulisi olla iteratiivista ja sisältää yhteistyötä käyttäjien kanssa, kuten kohdassa 2.1 todettiin. Lisäksi prosessin tulisi mahdollistaa kehittäjien, käytettävyysasiantuntijoiden ja käyttäjien välistä kommunikointia. Kuten Sousa et al. totesivat, kommunikoinnin mahdollistaminen ja sen lisääminen ovat tärkeässä osassa perinteisten prosessien ja käyttäjäkeskeisten prosessien yhdistämiseksi (Sousa et al., 2005). Että kehitettäviä käyttötapauksia voitaisiin käyttää tehokkaasti apuna testaamisessa, tulisi menetelmän myös tukea systemaattista testitapaus-

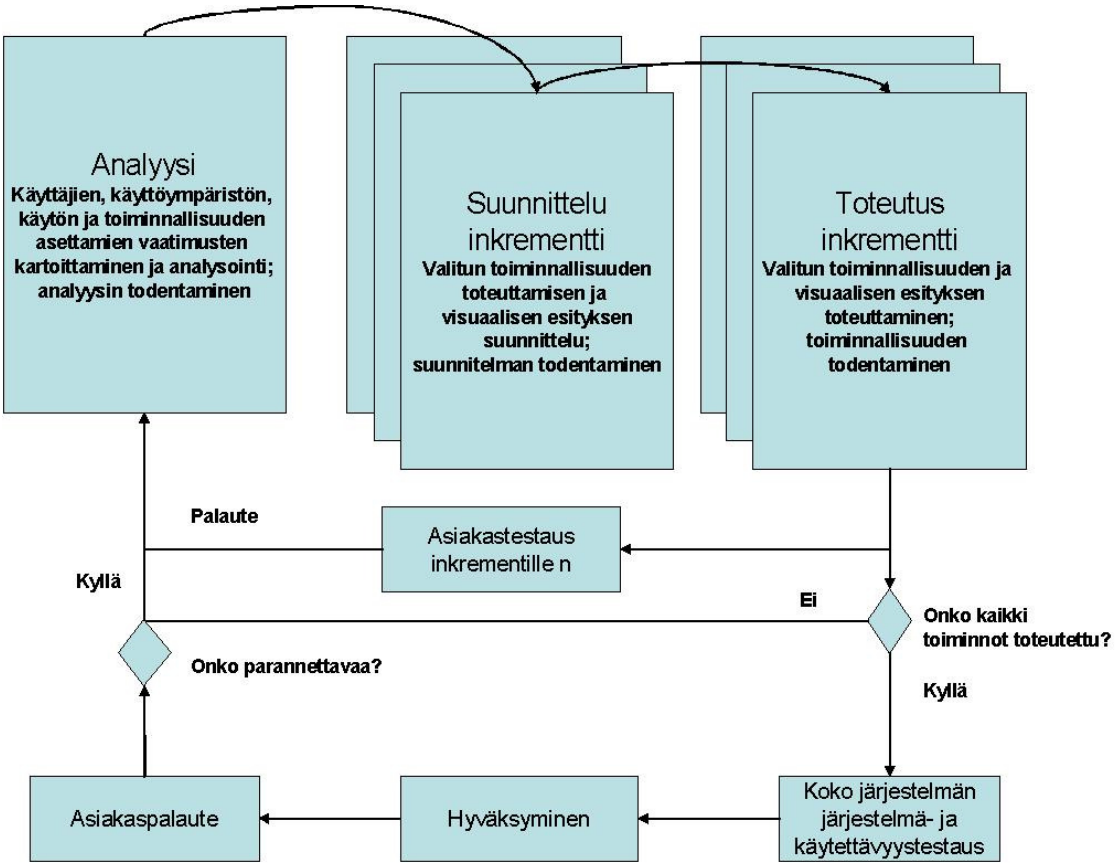
ten luomista käyttötapauksista, kuten Weidenhaupt et al. totesivat (Weidenhaupt et al., 1998). Myös testitapausten jäljitettävyys niiden perustana olevaan käyttötapaukseen on tärkeää.

RESPECT-menetelmässä vaatimusmäärittely lähtee käyttäjien ja käytön kontekstin määrittelystä, jonka jälkeen määritellään käyttäjien ja organisaation vaatimukset. ACUDUC-menetelmän periaatteiden mukaan käytön konteksti ja vaatimukset kirjataan erillisillä notaatioilla kuvaamaan järjestelmää eri näkökulmista. Erillisten notaatioiden yhdistäminen lisää eri kehittäjätahojen yhteisymmärrystä kehitettävästä järjestelmästä. Käyttäjäkeskeisessä RSI-mallissa kehitys alkaa käyttäjien ja käytön kontekstin määrittelemisellä. Tämä on kuvattu tässä tutkielmassa pankkiautomaatti järjestelmän esimerkin avulla käyttäen RESPECT-menetelmän sisältämiä valmiita dokumenttipohjia. Tämän jälkeen määritellään toiminnalliset vaatimukset RSI-menetelmän mukaisten vaatimuskäyttötapausten avulla. Perinteisiä RSI-menetelmän mukaisia vaatimuskäyttötapauksia ja käyttöliittymäkäyttötapauksia muokkaamalla saadaan aikaiseksi yhdistetyt kuvaukset, jotka kuvaavat sekä toiminnalliset vaatimukset että kontekstiin liittyvät käyttäjien vaatimukset. Lisäksi käyttäjäkeskeinen RSI-menetelmään yhdistetään OMT-menetelmä, joka kuvaa järjestelmän teknistä analyysiä ja kehitystä.

Kuvassa 26 on esitetty korkean tason kuvaus järjestelmän elinkaarimallista, jossa käytetään RSI:n mukaisia käyttötapauksia, mutta huomioidaan myös käytettävyysnäkökulmat. Järjestelmän kehittäminen koostuu kolmesta eri vaiheesta: analyysistä, suunnittelusta ja toteutuksesta. Jokainen vaihe sisältää syklin eli iteraation, jonka aikana suoritetaan vaiheeseen kuuluvat tehtävät, jonka jälkeen toteutetut tuotokset todennetaan testaamalla. Jos tuotoksissa havaitaan virheitä, kehittäjien tulee palata vaiheen tehtäviin uudelleen, kunnes korjattavaa ei enää löydetä.

Analyysivaiheessa määritellään järjestelmän sisältämien toimintojen tärkeysjärjestys niin, että järjestelmän kehittäminen jaetaan inkrementteihin. Ensimmäiseen inkrementtiin valitaan pienin mahdollinen joukko toteutettavia toimintoja, joiden avulla järjestelmä on jo käytettävissä. Tämän jälkeen järjestelmää laajennetaan toimintojen prioriteettien mukaisesti, kunnes kaikki toiminnot on toteutettu. Järjestelmä voidaan antaa asiakkaan testattavaksi jo ennen lopullisen järjestelmän valmistumista. Asiakkaan suorittamasta testauskäytöstä saadaan palautetta, joka voi todentaa tai kumota analyysin päätelmät käyttäjistä, käyttökontekstista ja toiminnasta.

Kun koko järjestelmä on valmis, suoritetaan koko järjestelmän järjestelmä- ja käytettävyystestaus, jonka jälkeen järjestelmä hyväksytetään asiakkaalla. Hyväksymisen jälkeen järjestelmä siirtyy ylläpitovaiheeseen, jonka aikana kerätään säännöllisin väliajoin asiakaspalautetta, joka voi aloittaa kehitysprosessin alusta joidenkin toimintojen osalta.



Kuva 26: Käyttäjäkeskeinen RSI-käyttötapausanalyysiprosessi.

RSI-prosessissa projektin osallistujiksi eritellään järjestelmän suunnittelijat, käyttäjät, asiakas ja muut sidosryhmät sekä järjestelmän markkinoijat (Collins-Cope, 2000). Sekä kehitysorganisaation että asiakasorganisaation henkilöillä on itse asiassa olemassa monia eri rooleja siten, että yhdellä henkilöllä voi olla useampia rooleja. Järjestelmän tilaaja, eli projektin asiakas, joka kustantaa projektin toteuttamisen ei välttämättä ole järjestelmän käyttäjä, tai ainakaan järjestelmän ainoa käyttäjä. Järjestelmällä voi olla monenlaisia eri käyttäjiä, eli erilaisia käyttäjäryhmiä. Luvussa 3 esitetyn pankkiautomaatti esimerkin käyttäjäryhmiä voivat olla pankkiautomaatin käyttäjän

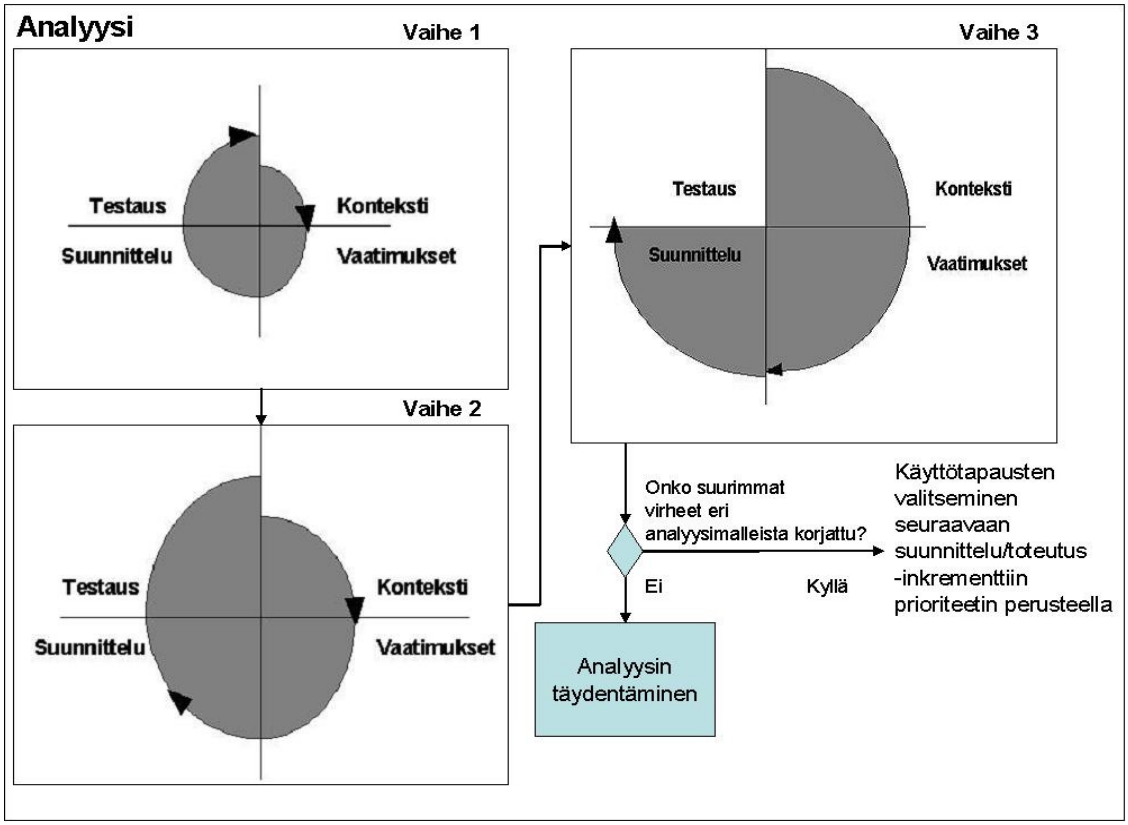
lisäksi automaatin huoltaja ja pankkivirkailija. Myös automaatin käyttäjät voivat muodostaa useamman erilaisen käyttäjäryhmän. Kehittäjilläkin on erilaisia rooleja, jotka organisaation koosta riippuen voivat jakautua tai olla jakautumatta usealle eri henkilölle. Kehittäjiin voidaan laskea kuuluvaksi esimerkiksi järjestelmän markkinoijat, projektijohto, suunnittelijat, koodaajat, käyttöliittymäsuunnittelijat, käytettävyysasiantuntijat, testaajat ja dokumentoijat. Kuten jo aiemmin todettiin, kehitysprosessin tulisi tukea näiden eri roolien välistä kommunikointia, että rajat perinteisten ohjelmistotuotannon prosessien ja käyttäjäkeskeisten prosessien välillä madaltuisivat, ja kaikki osapuolet ymmärtäisivät toistensa lähtökohdat paremmin.

Seuraavissa alakohdissa on tarkemmin esitelty käyttäjäkeskeisen RSI-käyttötapausanalyysiprosessin vaiheet, niiden sisältämät tehtävät ja tehtäviin osallistuvat roolit. Tehtävien suorittamiseen on olemassa monia eri käytännön menetelmiä, joista projektikohtaisesti valitaan juuri kyseiseen projektiin sopivat menetelmät. Esimerkiksi käyttäjäkeskeisten menetelmien käyttö riippuu siitä, millaista järjestelmää ollaan kehittämässä, koska muun muassa nettipalvelun käyttäjät voivat olla lähes ketä tahansa, jolloin todellisten loppukäyttäjien tavoittaminen voi olla vaikeaa, samoin jos järjestelmälle ei ole selkeää asiakasta.

Tämän luvun alakohdissa on esitelty joitakin esimerkkidokumentteja pankkiautomaatti järjestelmän kehittämiseksi käyttäjäkeskeisen RSI-menetelmän avulla. Esimerkeissä on käytetty RESPECT-menetelmän mukaisia lomakepohjia käyttäjäkeskeisen analyysin osalta, mutta käyttäjien, käytön kontekstin ja käyttäjien vaatimusten määrittämiseksi voitaisiin käyttää myös mitä tahansa muuta käyttäjäkeskeistä menetelmää, joita esiteltiin enemmän tämän tutkielman kohdassa 2.3.

7.2 Analyysi

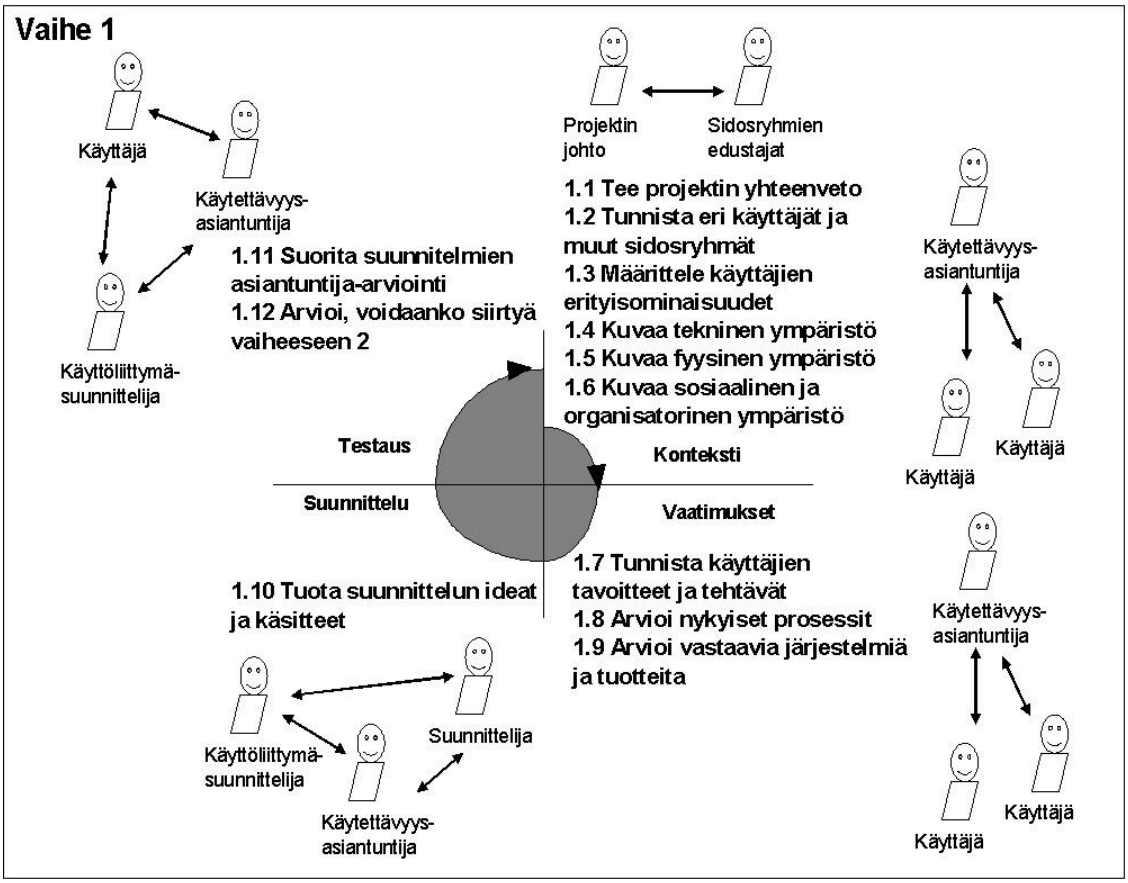
Analyysivaiheessa kartoitetaan ja analysoidaan ongelma. Analyysi jakautuu kolmeen erilliseen vaiheeseen RESPECT-vaatimusmäärittelyn tavoin. Kuten RESPECT-menetelmässä, analyysin ensimmäisen vaiheen aikana kartoitetaan nykytilanne. Toisessa vaiheessa kehitetään uusia toimintamalleja ongelman ratkaisemiseksi ja suoritetaan OMT-mallin mukainen analyysi kohdealueesta. Kolmannessa vaiheessa dokumentoidaan analyysintulokset, kuten RESPECT-menetelmässäkin. Analyysi vaiheen yleiskuvaus on esitetty kuvassa 27.



Kuva 27: Käyttäjäkeskeisen analyysin eri vaiheet.

Kuvassa 28 on esitetty analyysin ensimmäisen vaiheen tehtävät ja tekijät yksityiskohtaisemmin. Tämän vaiheen pääasiallisia tekijöitä ovat käytettävyysasiantuntija, käyttöliittymäsuunnittelija ja käyttäjät. Analyysin ensimmäisessä vaiheessa projektin johto ja mahdollinen asiakas voivat muodostaa järjestelmän yhteenvedon esimerkiksi RESPECT-lomakkeen 1.1 avulla varsinkin, jos kehitettävä järjestelmä on hyvin laaja tai kriittinen. Projektin yhteenveto pankkiautomaattijärjestelmän osalta on kuvattu liitteessä 1 kohdassa 1.1.

Tämän jälkeen käytettävyysasiantuntija selvittää, mitä eri käyttäjäryhmiä järjestelmällä on, mitä erityispiirteitä käyttäjäryhmien edustajilla on ja millaisessa ympäristössä he käyttävät järjestelmää. Käyttäjä- ja kontekstianalyysi voidaan suorittaa esimerkiksi RESPECT-lomakkeiden 1.2 – 1.10 avulla, haastattelemalla mahdollisia loppukäyttäjiä, suorittamalla käyttäjäkysely ja havainnoimalla käyttöympäristöä ja käyttötilanteita. Liitteessä 1 on kuvattu osittain pankkiautomaatin käyttäjä- ja konteksti analyysi RESPECT-lomakkeiden avulla.

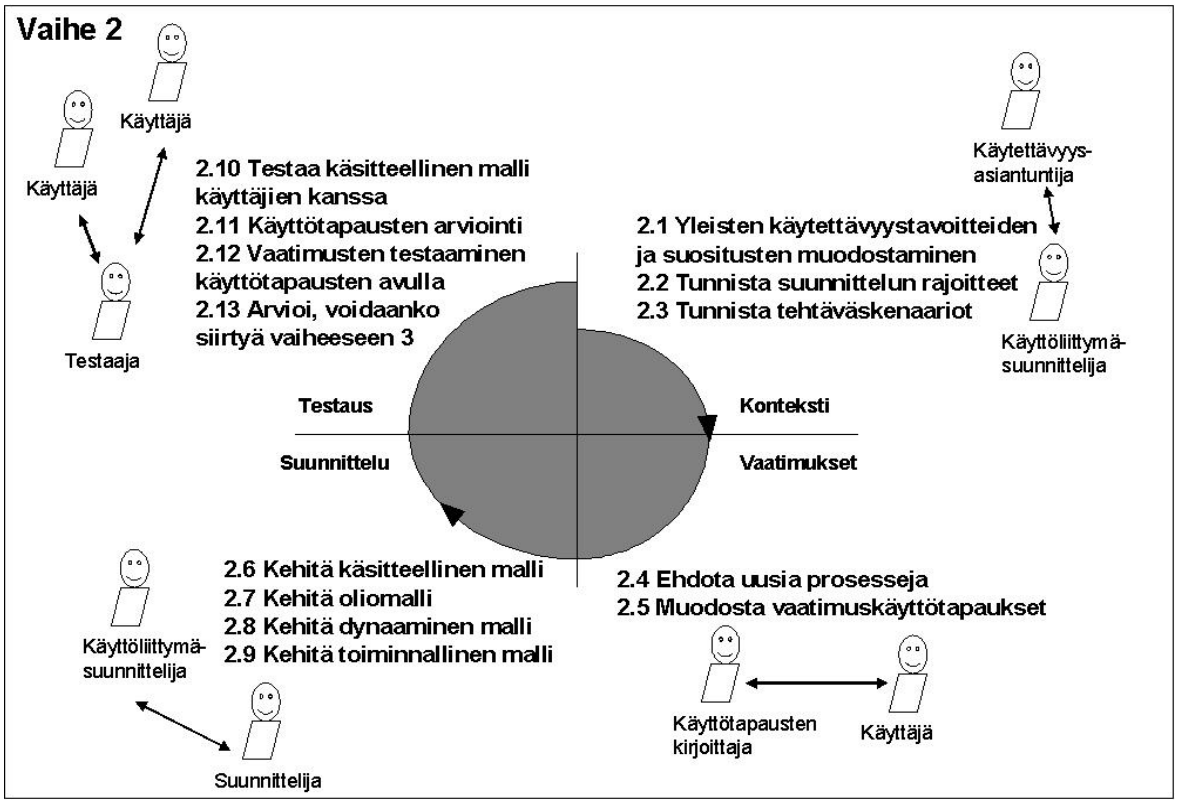


Kuva 28: Analyysin ensimmäinen vaihe.

Tämän jälkeen, kuten kuvasta 28 nähdään, käytettävyysasiantuntija selvittää, mitä vaatimuksia eri käyttäjät asettavat järjestelmälle sekä mitkä ovat heidän tavoitteensa ja tehtävänsä. Käyttäjien vaatimuksia ja tavoitteita voidaan selvittää arvioimalla käyttäjien nykyisiä prosesseja sekä mahdollisesti vastaavia, jo valmiita järjestelmiä ja tuotteita esimerkiksi kilpailija-analyysin avulla. Tämän jälkeen käyttöliittymäsuunnittelija, suunnittelija ja käytettävyysasiantuntija voivat yhdessä hahmotella järjestelmän käsitteet ja ideoida mahdollista ratkaisua ongelmaan. Ennen kuin ratkaisun toteutusta voidaan ruveta suunnittelemaan, tulee vielä varmistua siitä, että tehdyt havainnot käyttäjistä ja ympäristöstä ovat oikein esimerkiksi suorittamalla sisäinen katselmointi tai tarkastus löydöksille. Löydösten tarkastamiseen osallistuvat kaikki tekijät, sekä mahdollisesti myös muita kehitysorganisaation jäseniä ja käyttäjien edustajia. Liitteessä 1 on kuvattu pankkiautomaatti esimerkkiin liittyvät käyttäjien tavoitteet ja vaatimukset kuvattuna RESPECT-menetelmän

lomakkeilla 1.7 ja 1.8. Liitteessä 1 kohdassa 1.9 ja 1.10 on kuvattu pankkiautomaatti järjestelmän suunnittelu ideat ja käsitteet sekä niiden arviointi RESPECT-menetelmän mukaisesti.

Pankkiautomaattijärjestelmän esimerkissä havaitaan analyysin ensimmäisen vaiheen jälkeen, että suurimmat nykyisten automaattien ongelmat liittyvät käyttäjien kannalta automaatin turvallisuuteen ja vikatilanteiden raportoiminen, kuten rahan tai kuittipaperin loppuessa tai kortin juuttuessa automaattiin. Lisäksi muun muassa automaatin ergonomiaan ja valaistukseen liittyy ongelmia, jotka häiritsevät esimerkiksi näkö- ja liikuntarajoitteisia. Automaatin käyttäjien ensisijaisia tavoitteita ovat toiminnan nopeus ja turvallisuus. Pankkien ensisijaisia tavoitteena on lisätä automaattien käyttöä myös vanhusten ja esimerkiksi liikunta- sekä näkörajoitteisten joukossa huomioimalla heidän erityistarpeitaan automaatin ergonomian ja laitteiston avulla. Lisäksi automaatin halutaan toimivan luotettavasti. Mahdollisista virhetilanteista pitäisi pystyä ilmoittamaan ja toipumaan mahdollisimman nopeasti sekä niin, että käyttäjät ymmärtävät, mikä vika automaattiin tuli.



Kuva 29: Analyysin toinen vaihe.

Analyysin vaiheessa kaksi siirrytään nykytilan havainnoinnista ja arvioinnista uusien toimintatapojen suunnitteluun, aivan kuten RESPECT-menetelmässäkin. Vaiheen kaksi tehtävien ja tekijöiden tarkempi kuvaus on esitetty kuvassa 29. Tämän vaiheen ensimmäisenä tehtävänä on muodostaa järjestelmälle asetettavat käytettävyystavoitteet ja yleiset suositukset järjestelmän kehittämiselle. Pankkiautomaattiesimerkin käytettävyystavoitteet ja järjestelmän kehittämisen yleiset suositukset on kuvattu liitteessä 2 kohdassa 2.1. Käytettävyystavoitteet määrittelevät käytettävyysasiantuntija ja käyttöliittymäsuunnittelija yhteistyössä.

Käytettävyystavoitteiden ja suunnittelun rajoitteiden määrittelemisen jälkeen siirrytään uusien tehtäväskenaarioiden kehittämiseen. RESPECT-mentelmän tehtäväskenaariot eroavat perinteisistä käyttötapausten skenaarioista muun muassa kuvaamalla myös käytön kontekstin. Jokaista järjestelmän käytettävyystavoitetta varten tulee kehittää ainakin yksi tehtäväskenaario (Maguire, 1998). Lisäksi tehtäväskenaarioiden tulisi kuvata sekä tavallisia tehtäviä, vaikeita tehtäviä että kriittisiä tehtäviä, ja erityisen tärkeää on, että tehtäväskenaariot heijastavat uuden järjestelmän tuomia muutoksia. Taulukossa 10 on esitetty RESPECT-mentelmän mukainen tehtäväskenaario RESPECT-lomakkeen 2.3 avulla luvun 3 pankkiautomaattiesimerkille. Kuten taulukosta 10 nähdään, tehtäväskenaarion skenaariokuvaus on vapaamuotoisempi kuin perinteisissä käyttötapauksissa ja se sisältää myös käytön kontekstiin liittyviä poikkeuksia. Nämä poikkeukset tulee huomioida vaatimuskäyttötapauksissa, jotta käyttötapauksista muodostettavat testitapaukset olisivat mahdollisimman kattavia myös käytön kontekstin osalta.

Tehtäväskenaarioiden pohjalta suoritetaan kohdeorganisaation työn uudelleen organisointi tarvittavilta osin. Uusien prosessien kehittämiseksi voidaan käyttää esimerkiksi RESPECT-menetelmän lomaketta 2.4. Uusien prosessien kehittämisen tarkoituksena on arvioida nykyisiä prosesseja ja kehitettyjä skenaarioita sekä kehittää uusista prosesseista sellaisia, että käyttäjien tavoitteet saavutetaan (Maguire, 1998). Jokaista skenaariota varten kuvataan joukko käyttäjän ja järjestelmän välisiä vuorovaikutustilanteita, jotka kuvaavat, miten järjestelmää tulisi käyttää. Jokaista vuorovaikutustilannetta varten listataan joukko toimintoja ja ominaispiirteitä, jotka tukevat vuorovaikutuksen suorittamista. Liitteessä 2 kohdassa 2.2 on esitetty taulukossa 10 esitetyn pankkiautomaatin tehtäväskenaarion ensimmäiseen skenaarioon liittyvä uuden prosessin kuvaus RESPECT-lomakkeen 2.4 avulla.

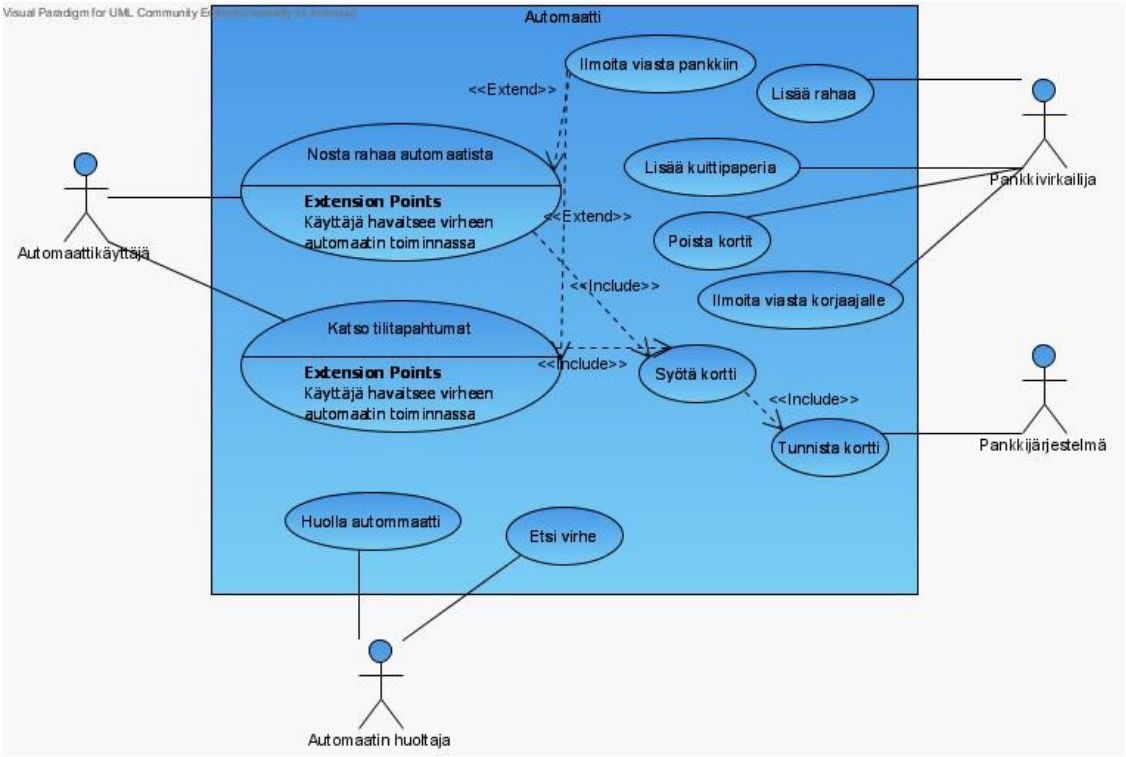
Taulukko 10: Rahan nostamisen tehtäväskenaario esimerkki (Maguire, 1998).

2.3 Tehtäväskanaario: Rahan nostaminen	
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**	
Skenaario	**Toiminnan päämäärä**
Käyttäjä syöttää kortin ja tunnusluvun. Käyttäjä valitsee rahan noston. Käyttäjä valitsee nostettavaksi summaksi 50 € ja haluaa kuitin paperille. Käyttäjä ottaa kortin, rahat ja kuitin automaatista.	90% nykyisistä pankkiautomaatin käyttäjistä tulisi pystyä suorittamaan tehtävä yhden minuutin sisällä. 70% ensikertaa automaattia käyttävistä tulis myös suoritua tehtävästä minuutissa.
Käyttäjä syöttää kortin ja tunnusluvun. Käyttäjä valitsee nostettavaksi summaksi 300 €, kun nostoraja on 250 €.	Käyttäjien tulisi vaivattomasti ymmärtää, että he ovat ylittäneet nostorajansa ja voivat onnistuneesti nostaa 250 €.
Käyttäjä syöttää kortin ja tunnusluvun. Käyttäjä nostaa 100 €, mutta päättää perua toiminnon.	Käyttäjien tulisi olla mahdollista lopettaa tapahtuma ja saada korttinsa takaisin onnistuneesti.
Käyttäjä syöttää kortin ja tunnusluvun. Käyttäjä tuntee olonsa uhatuksi takana seisovan henkilön takia. Mitä käyttäjän tulisi tehdä?	Käyttäjien tulisi olla mahdollista lopettaa tapahtuma nopeasti ja mahdollisesti tehdä hälytys.
Automaatista loppuu raha tai kuittipaperi pankin aukioloaikana. Asiasta ilmoitetaan pankkiin jollakin tavalla täydennystä varten.	Automaatin tulisi käsitellä tämä tilanne käyttäjän kannalta avuliaalla tavalla.
Käyttäjä syöttää kortin katsomatta sitä. Automaatti ei hyväksy korttia, joten käyttäjä kiinnittää siihen uudelleen huomiota ja syöttää kortin uudelleen. Käyttäjä unohtaa tunnusluvun, ja yrittää syöttää uuden tunnusluvun useita kertoja.	Automaatin tulisi käsitellä tämä tilanne käyttäjän kannalta avuliaalla tavalla.
Käyttäjä syöttää kortin ja tunnnusluvun. Käyttäjä nostaa 100 €. Käyttäjä saa kuitin, mutta väärän summan rahaa.	Automaatin tulisi käsitellä tämä tilanne käyttäjän kannalta avuliaalla tavalla.

Uudelleen organisoidut prosessit kuvataan myös käyttötapauskaavioina ja vaatimuskäyttötapausten sanallisina kuvauksina. Vaatimuskäyttötapausten kehittämiseen osallistuvat lähinnä käytettävyysasiantuntija, käyttötapausten kirjoittajat sekä käyttäjät. Käyttötapausten kirjoittajilla ei yleensä ole tietämystä järjestelmän toteutuksesta, vaan kirjoittajat voivat olla jopa järjestelmän tulevia loppukäyttäjiä.

Kuvassa 30 on esitetty pankkiautomaattijärjestelmän käyttötapauskaavio, kun järjestelmä on kehitetty myös käyttäjäkeskeisiä menetelmiä hyödyntäen. Verrattaessa kuvaa 30 luvussa 3 esitettyyn pankkiautomaatin käyttötapauskaavioon kuvassa 8 sivulla 18 havaitaan, että käyttäjäkeskeisen kehittämisen seurauksena järjestelmä sisältää joitakin lisätoimintoja, joiden tarkoituksena on lisätä käyttäjien tyytyväisyyttä. Esimerkiksi automaatissa havaittavan vian ilmoittaminen pankkiin lisää käyttäjien kontrollin tunnetta; heillä on mahdollisuus selviytyä myös virhetilanteista. Lisäksi joitakin toimintoja on jaettu eri tavalla eri käyttäjien kesken, josta on seurannut yhden

uuden käyttäjäryhmän, pankkivirkailijan, löytyminen. Rahan ja kuittipaperin lisääminen sekä automaatin käytöstä poistamien korttien poistaminen automaatista ovatkin uudessa järjestelmässä pankkivirkailijan tehtäviä pankin ollessa auki, koska huoltomiehen kutsuminen kyseisiin tehtäviin veisi paljon aikaa ja olisi kallista pankille.



Kuva 30: Käyttäjäkeskeisen pankkiautomaatti järjestelmän käyttötapauskaavio.

Taulukossa 11 on esitetty pankkiautomaatti järjestelmän vaatimuskäyttötapaus *Nosta rahaa*, joka on muokattu käyttäjäkeskeisen RSI-käyttötapausanalyysiprosessin mukaiseksi. Kuten taulukosta 11 havaitaan, vaatimuskäyttötapauksen käyttötapauskaavain muuttuu verrattuna perinteiseen RSI-käyttötapausanalyysin vaatimuskäyttötapauksen kaavaimeen, joka esiteltiin taulukossa 9 sivulla 66. Käyttäjäkeskeisyys näkyy kaavaimessa myös kontekstiin liittyvien poikkeusten huomioimisena. Näin käyttötapauksesta muodostettavat testitapaukset kattavat laajemmin myös käytettävyysnäkökulmat. Uudesta käyttäjäkeskeisestä vaatimuskäyttötapauksesta voidaan melko helposti muodostaa sekä vain toiminnallisuuden kuvaus ottamalla käyttöön skenaarion kuvaus ja siihen liittyvät toiminnalliset poikkeukset, tai käytön kuvaus tarkastelemalla vain skenaariota ja sen kontekstiin liittyviä poikkeuksia. Tämä vastaa ACUDUC-menetelmän tapaa kuvata järjestelmää

sekä toiminnallisesta että ei-toiminnallisesta näkökulmasta. Lisäksi kaavain sisältää käyttötapauksiin liittyvät kvantitatiiviset käytettävyystavoitteet. Näiden avulla käyttötapauksesta voidaan helposti muodostaa myös käytettävyystestauksen testitapauksia. Käytettävyystavoitteita käytetään tällöin testitapauksen hyväksymisperiaatteina.

Testitapausten muodostamista varteen kaavaimeen on lisätty rivi käyttötapausten toiminnallisia muuttujia varten. Kaavain sisältää myös oman rivin toiminnallisille testitapauksille, jotka perustuvat kyseiseen käyttötapaukseen. Toiminnallisilla testitapauksilla tarkoitetaan tässä Binderin mallin mukaisesti toteutettavia testitapauksia, jotka pohjautuvat käyttötapauksiin, käyttötapauksen toiminnallisiin muuttujiin ja niiden välisiin suhteisiin. Luettelo käyttötapauksiin liittyvistä testitapauksista parantaa testitapausten jäljitettävyyttä sekä helpottaa käyttötapausten testaamista kertomalla, mitkä testitapaukset tulee suorittaa käyttötapauksen testaamiseksi.

Taulukko 11: Käyttäjäkeskeisen RSI-käyttötapausanalyysin mukainen vaatimuskäyttötapaus *Nosta rahaa*.

Tunniste	Nosta rahaa
Toimija(t)	Automaatin peruskäyttäjä, pankkivirkailija
Tavoitteet	Käyttäjä nostaa rahaa omalta tililtään pankkiautomaatin avulla.
Taso	Käyttäjän tavoite
Sisältyvät käyttötapaukset	Syötä kortti
Laajennetut käyttötapaukset	
Alkuehdot	Käyttäjä on syöttänyt automaattikortin ja tunnusluvun automaattiin suorittamalla onnistuneesti käyttötapauksen Syötä kortti.
Loppuehdot	Käyttäjä saa takaisin automaattikortin, rahat ja kuitin. Järjestelmä veloittaa käyttäjän tiliä nostettavalla summalla.
Onnistuneen perussuorituksen skenaario	1. Käyttäjä on syöttänyt pankkikortin ja tunnusluvun onnistuneesti. Järjestelmä kysyy käyttäjältä valittavaa toimintoa. 2. Käyttäjä valitsee rahan noston. 3. Järjestelmä kysyy nostettavaa summaa. 4. Käyttäjä syöttää nostettavan summan. 5. Järjestelmä tarkastaa, että tilillä on olemassa nostettavan summan verran rahaa, ja että kortin nostoraja ei täyty nostettavasta summasta. Järjestelmä vähentää tilin saldoa käyttäjän nostaman summan verran. 6. Järjestelmä kysyy käyttäjältä, haluaako tämä tulostaa tapahtumasta kuitin. 7. Käyttäjä valitsee kuitin tulostuksen. 8. Järjestelmä pyytää käyttäjää ottamaan kortin pois. 9. Käyttäjä ottaa pankkikortin pois automaatista. 10. Järjestelmä antaa rahat ja kuitin. Käyttötapaus päättyy.

Taulukko 11 jatkuu

Toiminnalliset poikkeukset	**Kuvaus poikkeuksesta**	**Kuvaus käyttäjän ja järjestelmän välisestä vuorovaikutuksesta**
	0.1 Käyttäjä päättää perua rahan nostamisen missä tahansa tapahtuman vaiheessa ennen rahasumman valintaa esim. tuntiessaan olonsa uhatuksi.	Käyttäjä valitsee tapahtuman keskeyttämisen. Järjestelmä poistaa kortin automaatista veloittamatta sitä. Järjestelmä tulostaa kuitin, missä ilmoitetaan tilin saldo ja että tapahtuma keskeytettiin.
	0.2 Käyttäjä ryöstetään automaatilla.	Käyttäjä voi painaa hälytyspainiketta, joka käynnistää voimakkaan sireenin sekä lähettää automaattisen hälytyksen ryöstöstä hätäkeskukseen.
	4.1 Käyttäjän tilillä ei ole nostettavaa summaa rahaa.	Järjestelmä ilmoittaa käyttäjälle, että tilillä ei ole tarpeeksi saldoa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.
	4.2 Kortin nostoraja ylittyy käyttäjän syöttämästä summasta.	Järjestelmä ilmoittaa käyttäjälle, että tililtä ei voida nostaa niin suurta summaa. Järjestelmä pyytää käyttäjää syöttämään uuden summan.
	6.1 Kuittipaperi on loppunut automaatista.	Järjestelmä ilmoittaa käyttäjälle, että kuittipaperi on loppunut eikä sitä voida tulostaa. Järjestelmä kysyy näytetäänkö saldo näytöllä. Käyttäjä valitsee saldon tulostamisen näytölle. Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.
	6.1.1 Käyttäjä ei valitse saldon tulostamista näytölle.	Käyttötapaus jatkuu kortin poistamisella automaatista.
	7.1 Käyttäjä ei valitse kuitin tulostamista paperille, vaan saldon näyttämisen näytöllä.	Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.

Taulukko 11 jatkuu.

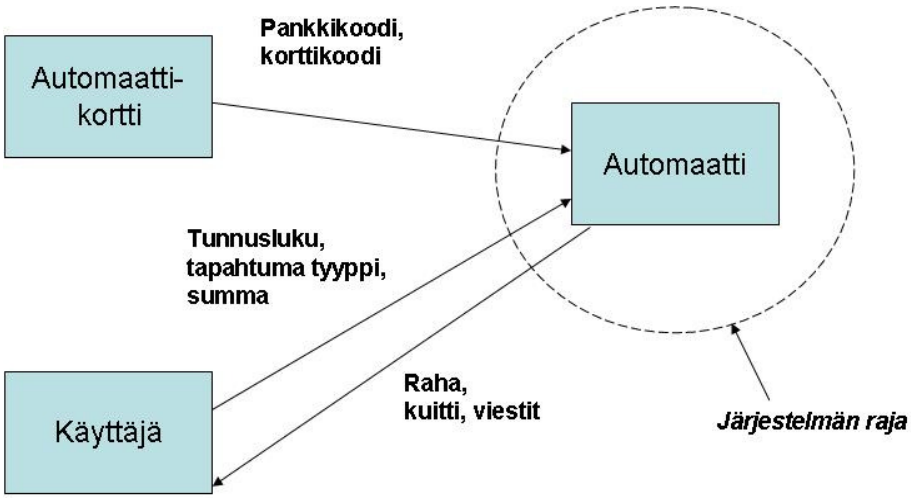
<table>
<tr><td>Kontekstiin liittyvät poikkeukset</td><td>
<table>
<tr><th>Kuvaus poikkeuksesta</th><th>Kuvaus käyttäjän ja järjestelmän välisestä vuorovaikutuksesta</th></tr>
<tr><td>0.1 Käyttäjä päättää perua rahan nostamisen missä tahansa tapahtuman vaiheessa ennen rahasumman valintaa esim. tuntiessaan olonsa uhatuksi.</td><td>Käyttäjä valitsee tapahtuman keskeyttämisen. Järjestelmä poistaa kortin automaatista veloittamatta sitä. Järjestelmä tulostaa kuitin, missä ilmoitetaan tilin saldo ja että tapahtuma keskeytettiin.</td></tr>
<tr><td>0.2 Käyttäjä ryöstetään automaatilla.</td><td>Käyttäjä voi painaa hälytyspainiketta, joka käynnistää voimakkaan sireenin sekä lähettää automaattisen hälytyksen ryöstöstä hätäkeskukseen.</td></tr>
<tr><td>6.1 Kuittipaperi on loppunut automaatista. Pankki on kiinni.</td><td>Järjestelmä ilmoittaa käyttäjälle, että kuittipaperi on loppunut eikä sitä voida tulostaa. Järjestelmä kysyy näytetäänkö saldo näytöllä. Käyttäjä valitsee saldon tulostamisen näytölle. Järjestelmä näyttää tilin saldon näytöllä ja käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
<tr><td>6.1.1 Käyttäjä ei valitse saldon tulostamista näytölle.</td><td>Käyttötapaus jatkuu kortin poistamisella automaatista.</td></tr>
<tr><td>6.1.2 Kuittipaperi on loppunut automaatista. Pankki on auki.</td><td>Käyttäjä ilmoittaa automaatin avulla pankkiin loppuneesta kuittipaperista. Pankkivirkailija saapuu 5 minuutin sisällä lisäämään paperia automaattiin.</td></tr>
</table>
</td></tr>
<tr><td>Käytettävyystavoitteet</td><td>
<table>
<tr><td>0. 90% nykyisistä pankkiautomaatin käyttäjistä tulisi pystyä suorittamaan rahan nostaminen yhden minuutin sisällä. 70% ensikertaa automaattia käyttävistä tulis myös suoritua tehtävästä minuutissa.</td></tr>
<tr><td>6.1 Automaatin tulisi ilmoittaa pankkiin paperin loppumisesta, pankin ollessa auki pankkivirkailijan tulisi lisätä paperia 5 minuutin sisällä ilmoituksesta. Pankin ollessa kiinni järjestelmän tulee käsitellä tapahtuma käyttäjien kannalta mahdollisimman avuliaalla tavalla.</td></tr>
</table>
</td></tr>
<tr><td>Tekniset kaaviot</td><td>Sekvenssikaavio Nosta rahaa</td></tr>
<tr><td>Käyttöliittymämääritykset</td><td>Käyttöliittymäkäyttötapaukset Nostosumman valinta, Kuitti</td></tr>
<tr><td>Toiminnalliset muuttujat</td><td>Nostettava summa, automaatin sisältämän rahan määrä, tilin nostoraja, tilin saldo.</td></tr>
<tr><td>Testitapaukset</td><td>T1, T2, T3, T4, T5, T6</td></tr>
<tr><td>Muutoshistoria</td><td></td></tr>
</table>

Kuten kuvasta 29 huomataan, analyysin toisessa vaiheessa suunnitellaan järjestelmän käsitteellinen malli kehitettävästä järjestelmästä, erona RESPECT-menetelmään, jossa tässä vaiheessa toteutettaisiin järjestelmän prototyyppi. Käsitteellinen malli luo pohjan käyttöliittymän suunnittelulle, ja sen avulla kuvataan yleiset käyttöliittymän suunnitteluperiaatteet, ei kuitenkaan käyttöliittymän yksityiskohtia. Käsitteellisen mallin avulla määritellään muun muassa, onko kehitettävä järjestelmä tuote- vai prosessiorientoitunut, esitysehdot prosesseille ja tuotteille, määritellään päänäytöt ja pääasialliset navigointipolut eri näyttöjen välillä. Käsitteellisen mallin muodostaa yleensä käyttöliittymäsuunnittelija, mutta kehitystyöhön voi osallistua arvioimalla ja antamalla palautetta kuka tahansa niistä henkilöistä, jotka osallistuivat käyttäjäprofiilin tai tehtäväanalyysin tekemiseen. Käsitteellisiä malleja voidaan myös muodostaa useita erilaisia vaihtoehtoja, joille suoritetaan vertailevaa käytettävyystestausta käyttäjien kanssa parhaan mahdollisen tuloksen saavuttamiseksi.

Lisäksi tässä vaiheessa kehittäjien kohdealueen tuntemus on syventynyt sille tasolle, että voidaan suorittaa OMT-menetelmän mukainen analyysi. Tämä tarkoittaa siis oliomallin, dynaamisen mallin ja toiminnallisen mallin kehittämistä. Oliomallia varten tunnistetaan järjestelmän käsitteelliset oliot, jotka eivät sisällä järjestelmän toteutukseen liittyviä olioita. Pankkiautomaatti järjestelmässä automaattiin ja sen käyttöön liittyviä olioita ovat esimerkiksi tili, pankkikortti, pankki, pankkiyhteisö, asiakas ja tapahtuma. Tämän jälkeen määritellään olioiden väliset suhteet ja kuvataan ne alustavan luokkakaavion avulla. Löydetyille olioille määritellään mahdolliset attribuutit ja pyritään yksinkertaistamaan mallia luomalla luokkahierarkia käyttäen perintää. Oliomallin tuloksena on tarkennettu luokkakaavio. Pankkiautomaatin alustava luokkakaavio sekä tarkennettu luokkakaavio on kuvattu liitteessä 2 kohdassa 2.3.

Dynaamisen mallin kehittämisessä käytetään apuna skenaarioiden kuvaamia tyypillisiä vuorovaikutussarjoja. Jokaisen skenaarion osalta tunnistetaan olioiden väliset tapahtumat ja määritellään skenaarion tapahtumien kulku olioiden välisinä tapahtumina. Tapahtumien perusteella kehitetään tapahtumavirta kaavio koko järjestelmästä. Lisäksi dynaamiseen malliin kuuluu tilakaavioiden määrittäminen kaikille niille luokille, jotka sisältävät järjestelmän kannalta tärkeää dynaamista toimintaa. Pankkiautomaattiin liittyvät dynaamisen mallin kaaviot on kuvattu liitteessä 2 kohdassa 2.4.

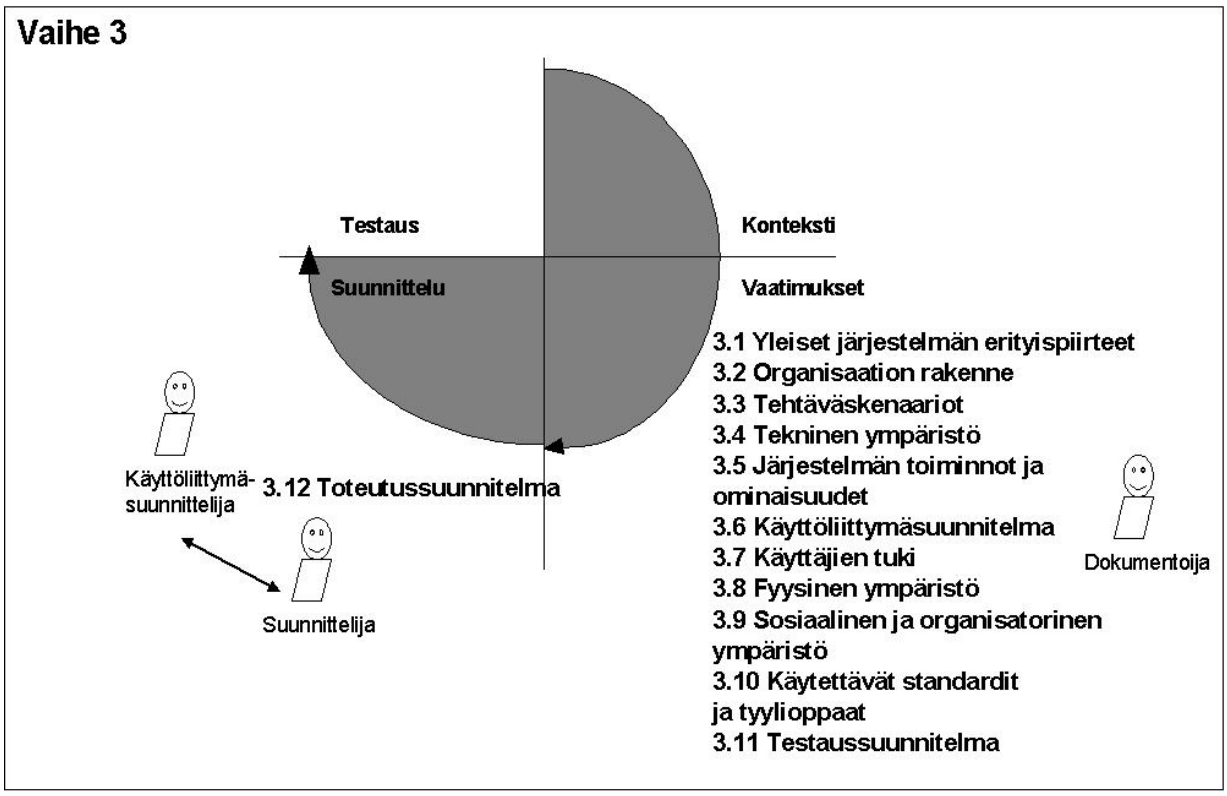
Toiminnallista mallia varten tunnistetaan järjestelmän syöte- ja tulostearvot, jotka ovat järjestelmän ja järjestelmän ulkopuolisen maailman välisten tapahtumien parametreja. Kuvassa 31 on esitetty pankkiautomaattijärjestelmän syötteiden ja tulosteiden yleiskuvaus. Syötteiden ja tulosteiden pohjalta muodostetaan tietovuokaaviot, jotka kuvaavat miten jokainen ulostuloarvo muodostetaan sisääntuloarvoista. Toiminnallinen malli koostuu tietovuokaavioiden lisäksi olioiden välisistä rajoitteista. Esimerkiksi pankkiautomaatin osalta olioiden välisiä rajoitteita ovat muun muassa ”pankkitilin saldo ei voi olla negatiivinen” ja ”nostettava summa ei voi olla suurempi kuin tilin päivittäinen nostoraja”. Toiminnallinen malli muodostaa pohjan RSI-mallin mukaisten palvelukäyttötapausten tekemiselle, sillä toiminnallinen malli kuvaa palvelukäyttötapausten parametrit ja olioiden väliset rajoitteet sekä palvelukäyttötapausten alku- ja loppuehdot. Pankkiautomaatti esimerkkiin liittyvät tietovuokaaviot on kuvattu liitteessä 2 kohdassa 2.5.



Kuva 31: Pankkiautomaatti järjestelmän syötteet ja tulosteet (Rumbaugh et al., 1991).

Vaiheen tuotokset tarkastetaan testaamalla käsitteellinen malli yhdessä käyttäjien kanssa sekä suorittamalla käyttötapausten arviointi. Käyttötapausten avulla voidaan myös testata vaatimukset, kuten jo aikaisemmin esitettiin. OMT-menetelmän mukaisille analyysimalleille voidaan suorittaa tarkastustilaisuudet tai katselmoinnit.

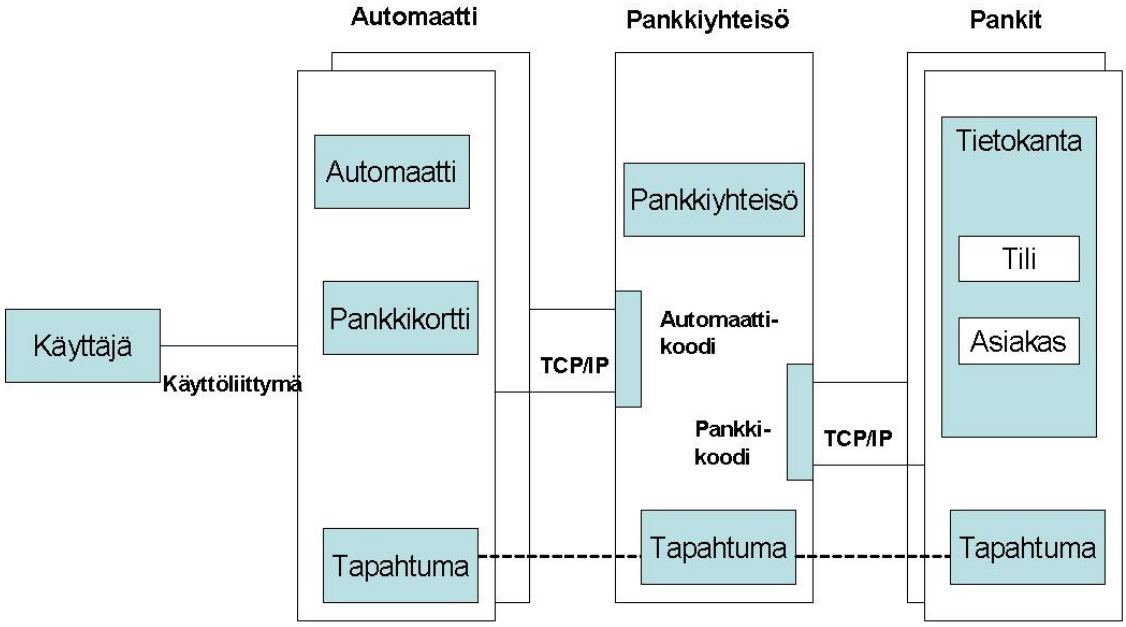
Analyysin kolmannessa vaiheessa suoritetaan RESPECT-mallin mukainen vaatimusten dokumentoiminen, joka esiteltiin jo kohdassa 2.3. Kuvassa 32 on esitetty analyysin kolmannessa vaiheessa tuotettavat dokumentit. RESPECT-mentelmän mukaisten dokumenttien lisäksi analyysin kolmannessa vaiheessa muodostetaan toteutussuunnitelma, joka sisältää OMT-menetelmän mukaisen järjestelmän korkean tason suunnitelman eli järjestelmäarkkitehtuurin kuvauksen.



Kuva 32: Analyysin kolmas vaihe.

Korkean tason suunnitelmassa järjestelmä jaetaan OMT-analyysin pohjalta alajärjestelmiksi, jotka jakautuvat komponenteiksi ja luokiksi, jotka lopulta toteuttavat järjestelmän toiminnallisuuden. Järjestelmäarkkitehtuurin suunnittelemisen avulla toteutuksesta tulee järkevä kokonaisuus, joka on ylläpidettävissä ja laajennettavissa myöhemmin. Jakamalla järjestelmä alajärjestelmiksi mahdollistetaan useiden alajärjestelmien rinnakkainen ja osittain toisistaan erillinen toteuttaminen, joka taas tukee inkrementaalista kehittämistä. Kuvassa 33 on esitetty pankkiautomaattijärjestelmän järjestelmäarkkitehtuuri. Kuten kuvasta 32 huomataan, pankkiautomaatti kommunikoi itse asiassa pankkiyhteisön keskuspalvelimen kanssa, joka puolestaan välittää viestit automaatin ja

käyttäjän pankin välillä. Käyttäjän ja tämän tilin tiedot sijaitsevat käyttäjän oman pankin tietokannassa.



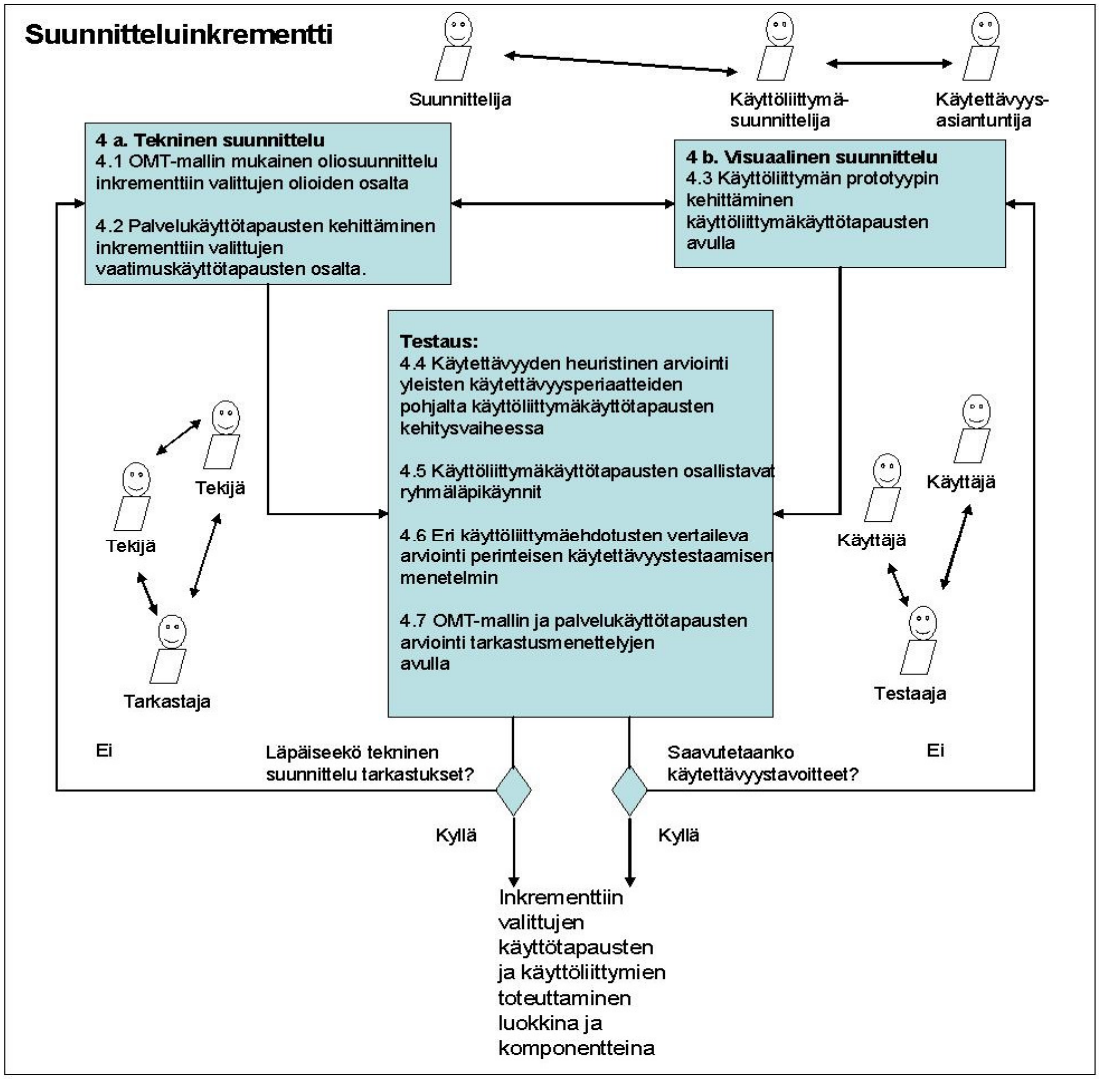
Kuva 33: Pankkiautomaatin järjestelmäarkkitehtuuri (Rumbaugh et al., 1991).

Kuten kuvassa 26 on esitetty, analyysin eri vaiheiden suorittamisen jälkeen voidaan arvioida, onko suurimmat virheet eri analyysimalleista korjattu. Mikäli korjattavaa löytyy, tulee kehittäjien täydentää analyysiä tarvittavilta osin. Analyysivaiheen jälkeen kehittäjillä tulisi olla selkeä kokonaiskuva kehitettävästä järjestelmästä ja sille asetetuista vaatimuksista. Kehittäjillä tulisi olla myös selkeä kuva siitä, mitkä toiminnot ovat eri käyttäjien kannalta oleellisempia. Toteutus jaetaan vaatimuskäyttötapausten tärkeysjärjestyksen perusteella osiin, jotka toteutetaan yksi kerrallaan. Ensimmäiseen suunnittelu/toteutus –inkrementtiin valitaan tärkeysjärjestyksen perusteella minimijoukko vaatimuskäyttötapauksia, joiden avulla järjestelmää voidaan käyttää. Lopullisten inkrementtien määrä riippuu toteutettavan järjestelmän koosta.

7.3 Suunnitteluinkrementti

Toteutuksen suunnitteluvaiheessa suoritetaan valittujen toimintojen toteutuksen suunnittelu luokka- ja komponenttitasolla, sekä järjestelmän visuaalisen esityksen eli käyttöliittymän suunnittelu.

Kehittäjät muodostavat valittujen vaatimuskäyttötapausten osalta niihin liittyvät käyttöliittymäkäyttötapaukset ja palvelukäyttötapaukset. Toteutuksen suunnittelu vastaa OMT-menetelmän oliosuunnittelun vaihetta, jossa suunnittelijat muodostavat lopulliset luokkahierarkiat ja toteutusalgoritmit.



Kuva 34: Suunnitteluinkrementin osat ja tehtävät sekä eri tekijät.

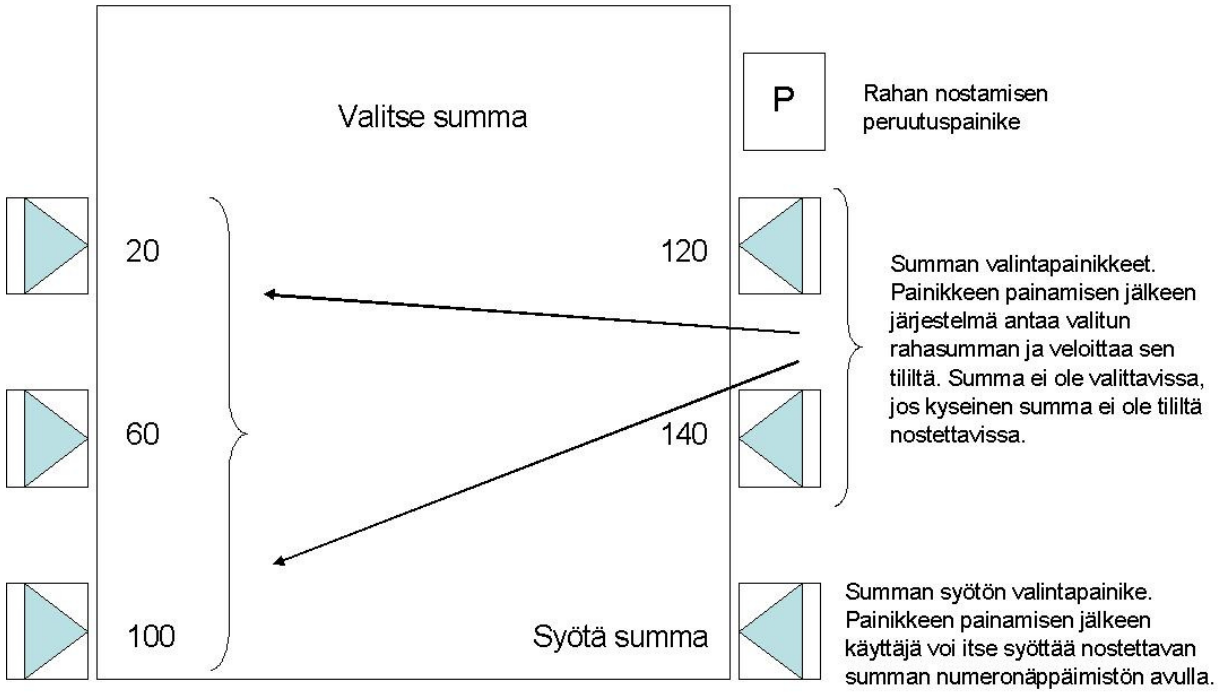
Järjestelmän luokkatason suunnittelu ja käyttöliittymäsuunnittelu ovat rinnakkaisia, sillä esimerkiksi käyttöliittymän suunnittelu voi synnyttää luokkia, joita ei analyysimallien perusteella tiedetty olevan. Ennen suunnitelmien toteuttamista testataan, että suunniteltu toteutus vastaa sille ase-

tettuja vaatimuksia. Jos suunnitelma ei vastaa analyysimalleja, tulee suunnitelmaa korjata ennen toteuttamiseen siirtymistä. Suunnitteluinkrementti on kuvattu tarkemmin kuvassa 34.

Kuten kuvasta 34 huomataan, tekninen suunnittelu ja käyttöliittymäsuunnittelu ovat rinnakkaisia prosesseja. Teknisen suunnittelun aikana muodostetaan RSI-käyttötapausanalyysin mukaiset palvelukäyttötapaukset suunnitteluinkrementtiin valittujen vaatimuskäyttötapausten osalta. Palvelukäyttötapaukset tehdään analyysivaiheessa muodostetun toiminnallisen mallin mukaan. Teknisestä suunnittelusta vastaavat tekniset suunnittelijat, jotka toimivat yhteistyössä käyttöliittymäsuunnittelijoiden kanssa. Käyttöliittymäsuunnittelu sisältää käyttöliittymäkäyttötapausten muodostamisen. Käyttöliittymäkäyttötapaukset suunnittelevat ja toteuttavat käyttöliittymäsuunnittelijat yhdessä käytettävyysasiantuntijoiden kanssa.

Kuvassa 35 on esitetty käyttäjäkeskeinen käyttöliittymäkäyttötapaus *Nostosumman valinta*. Käyttöliittymäkäyttötapaus sisältää toiminnan ja muotoilun kuvauksen lisäksi käyttötapaukseen liittyvät kvalitatiiviset käytettävyystavoitteet, joiden tarkoituksena on ohjata käyttöliittymän lopullista toteuttamista. Verrattaessa kuvan 35 käyttöliittymäkäyttötapausta alkuperäiseen käyttöliittymäkäyttötapaukseen kuvassa 22 sivulla 71 huomataan, että ensimmäisen käytettävyystavoitteen toteuttamiseksi automaattijärjestelmään lisätään peruutus painike, jonka avulla käyttäjä voi keskeyttää.

Koska käyttöliittymäkäyttötapaukset ovat kuvauksia käyttöliittymästä, niille voidaan suorittaa käytettävyyden arviointina arvioivaa ja vertailevaa käytettävyystestausta sekä osallistavia ryhmäläpikäyntejä. Käyttöliittymäkäyttötapausten kehitysvaiheessa käytettävyysasiantuntijat voivat osallistua kehitystyöhön esimerkiksi suorittamalla käyttötapausten heuristisen arvioinnin. Käyttöliittymäkäyttötapausten osalta testaamisen tavoitteena on tarkastaa, toteutuuko järjestelmälle asetetut käytettävyystavoitteet. Kun käytettävyystavoitteet on saavutettu, voidaan siirtyä inkrementtiin valittujen toimintojen osalta toimintojen toteuttamiseen.



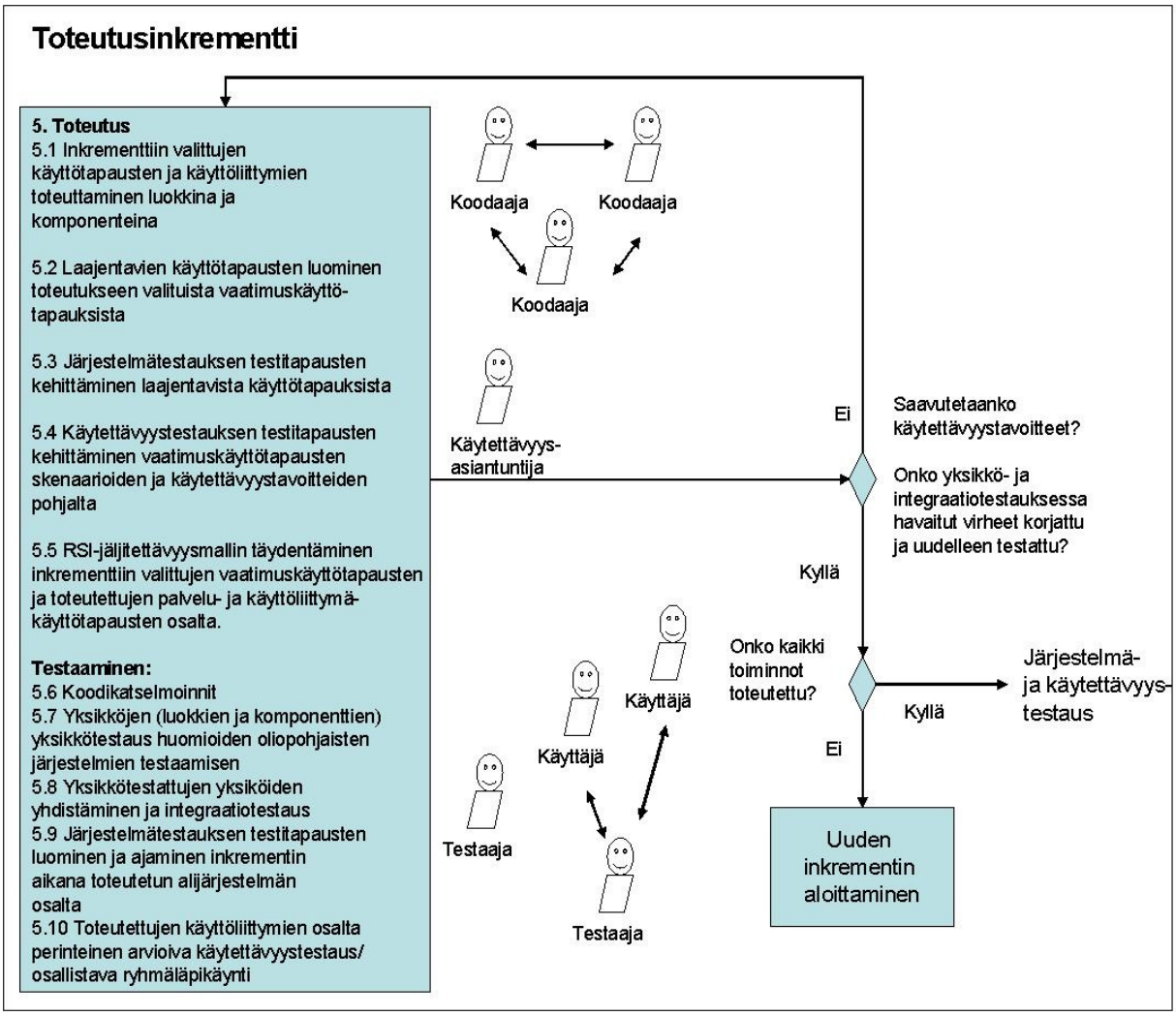
Käytettävyystavoitteet:

- Käyttäjien tulisi olla mahdollista lopettaa tapahtuma ja saada korttinsa takaisin onnistuneesti.
- Käyttäjän tulisi vaivattomasti ymmärtää, että hän on ylittänyt nostorajansa ja voi onnistuneesti nostaa nostorajan mukaisen summan.
- Käyttäjän tulisi vaivattomasti ymmärtää, että hän on ylittänyt tilin saldon ja voi onnistuneesti nostaa tilitä tilin saldon tai nostorajan mukaisen summan.

Kuva 35: Käyttäjäkeskeinen RSI-käyttöliittymäkäyttötapaus.

7.4 Toteutusinkrementti

Toteutusinkrementissä inkrementtiin valitut vaatimuskäyttötapaukset ja niihin liittyvät palvelu- ja käyttöliittymäkäyttötapaukset toteutetaan luokkina ja komponentteina. Toteuttamisesta vastaavat koodaajat. Samalla inkrementtiin valituista vaatimuskäyttötapauksista muodostetaan laajentavia käyttötapauksia. Laajentavista käyttötapauksista muodostetaan Binderin ehdotuksen mukaisesti järjestelmätestauksen testitapaukset. Tässä vaiheessa kehitetään myös käytettävyystestauksen testitapaukset vaatimuskäyttötapausten sisältämien skenaarioiden ja käytettävyystavoitteiden pohjalta. Kuvassa 36 on esitetty toteutusvaiheen tehtävät ja tekijät.



Kuva 36: Toteutusinkrementin tehtävät ja tekijät.

Toteutuksen aikana voidaan suorittaa koodin todentamiseksi koodikatselmointeja ja tarkastuksia. Toteutetut luokat jaetaan sopiviksi yksiköiksi, joille suoritetaan yksikkötestaus, koska kuten Binder toteaa, yksittäinen luokka ei yleensä ole itsessään merkityksellinen, vaan merkitys syntyy suhteesta muihin luokkiin (Binder, 2000). Yksikkötestauksen tarkoituksena on varmistua, että testattavat yksiköt toimivat itsenäisesti oikein. Yksikkötestatut yksiköt yhdistetään ja niille suoritetaan integraatiotestaus. Integrointitestauksessa testataan yhdistettyjen yksiköiden rajapintojen toiminta.

Jokaisen inkrementin tuloksena on toimiva, yksikkö- ja intergaatiotestattu järjestelmä, jolle voidaan suorittaa järjestelmätestaus toteutettujen käyttötapausten osalta sekä käytettävyystestaus toteutettujen käyttöliittymien osalta. Toteutettujen käyttöliittymien osalta voidaan suorittaa perinteinen arvioiva käytettävyystestaus tai osallistava ryhmäläpikäynti. Kuten kuvasta 28 nähdään, voidaan järjestelmä luovuttaa asiakkaan kokeellisen tuotantokäyttöön toteutusinkrementin jälkeen. Asiakkaan koekäytöstä kerätään palautetta, joka otetaan syötteeksi analyysiprosessiin.

Jos järjestelmä ei toteutusinkrementin jälkeen vielä sisällä kaikkia toimintoja siirrytään jälleen analyysin kautta seuraavan inkrementin suunnitteluun ja toteutukseen, kunnes voidaan todeta, että järjestelmä toteuttaa sovitut toiminnot. Vaikka jokaisen inkrementin jälkeen järjestelmä todettaisiin toimivaksi järjestelmätestauksen avulla, tulee koko järjestelmälle suorittaa kattava järjestelmä- ja käytettävyystestaus. Löydetyt virheet analysoidaan ja korjataan, kunnes järjestelmä läpäisee testaukset. Tämän jälkeen järjestelmä voidaan hyväksyttää asiakkaalla. Hyväksymisen jälkeen järjestelmä siirtyy kehittämisestä ylläpitovaiheeseen. Ylläpitovaiheessa järjestelmää voidaan edelleen kehittää esimerkiksi korjaamalla tuotantokäytössä löytyvät virheet. Uusi analyysivaihe järjestelmän jatkokehittämiseksi voi alkaa esimerkiksi asiakaspalautteen tai toiseen toimintaympäristöön mukauttamisen takia.

.

8. Yhteenveto

Perinteisesti ohjelmistontuotannossa ajatellaan, että järjestelmä on tehty oikein silloin kun se täyttää käyttäjien asettamat tavoitteet toiminnallisuuden osalta. Tämä todetaan testaamalla järjestelmän käyttöä ja vertaamalla havaittavaa toimintaa kehityksen alkuvaiheessa määriteltyihin vaatimuksiin. Mutta mitä hyötyä käyttäjille on järjestelmästä, joka toimii oikein, mutta sen käyttäminen tai käytön oppiminen on käyttäjille vaikeaa esimerkiksi kohdealueen sanaston puuttumisen, väärän tehtävien jaon tai ympäristöön sopimattoman toiminnan takia? Laajemmassa merkityksessä järjestelmän oikeellisuus perustuu oikeaan toiminnallisuuteen, joka sisältää juuri ne toiminnot ja vain ne toiminnot, jotka käyttäjät tarvitsevat, sekä muun muassa loppukäyttäjien käytön tehokkuuteen, tyytyväisyyteen ja hallinnan tunteeseen. Järjestelmän tulee siis myös tukea loppukäyttäjien tehokasta toimintaa niin, että käyttäjät tuntevat tietävänsä ja hallitsevansa järjestelmää.

Perinteiset ohjelmistotuotannonprosessit eivät yleensä huomioi käytettävyysnäkökulmia järjestelmän kehittämisessä, vaan käytettävyyteen liittyviä asioita kuvataan vain järjestelmän ei-toiminnallisina vaatimuksina. Ei-toiminnalliset vaatimukset keskittyvät yleensä lähinnä teknisen ympäristön ja teknisten rajoitteiden kuvaamiseen, eikä niinkään käyttöympäristön tai käyttötilanteen kuvaamiseen. Käytettävyyden huomioimiseksi järjestelmän kehittämisessä on olemassa useita käyttäjäkeskeisiä elinkaarimalleja, jotka taas puolestaan määrittävät eri käyttäjäryhmät, käytön kontekstin ja eri käyttäjäryhmien sekä toiminnalliset että ei-toiminnalliset vaatimukset. Käyttäjäkeskeisen kehittämisen elinkaarimallit eivät kuitenkaan kuvaa järjestelmän teknistä suunnittelua, vaan keskittyvät yleensä käytettävyyden ja käyttöliittymän kehittämiseen.

Käyttötapaukset ovat hyvin yleisesti käytetty menetelmä kehitettävän järjestelmän toiminnallisuuden kuvaamiseksi käyttäjien näkökulmasta. Käyttötapausten käyttö on hyvin suosittua, koska niiden tekemisen oppimista pidetään niin helppona, että menetelmän käyttö voidaan opettaa myös järjestelmän tilaavalle asiakkaalle. Käyttötapauksia voidaan käyttää nykyisin osana melkein minkä tahansa interaktiivisen järjestelmän kehitysprojektia, ja niitä käytetään menetelmänä sekä perinteisissä ohjelmistotuotantoprosesseissa että käyttäjäkeskeisissä prosesseissa. Käyttötapauksia olisi siis myös mahdollista käyttää laajemmassa merkityksessä kuvaamaan käyttäjien toiminnal-

listen vaatimusten lisäksi myös käyttöympäristöön ja –tilanteeseen liittyviä ei-toiminnallisia vaatimuksia.

Koska käyttötapaukset sisältävät kuvauksen järjestelmän toiminnasta, on hyvin luonnollista haluta käyttää niitä myös järjestelmän testaamiseen. Käyttötapaukset kuvaavat järjestelmän vaatimukset käyttäjien näkökulmasta, jolloin käyttötapaukset läpäisevän järjestelmän voidaan todeta täyttävän käyttäjien vaatimukset. Kuten tässä tutkielmassa totesin, käyttötapauksia voidaan käyttää toiminnallisten testitapausten kehittämiseksi mustalaatikkotestaamista varten. Lisäksi käyttötapauksia voidaan käyttää esimerkiksi järjestelmän vaatimusten testaamiseen, jolloin vaatimuksista johtuvat virheet voidaan havaita mahdollisimman aikaisessa vaiheessa kehitysprojektin resurssien säästämiseksi.

RSI-käyttötapaukset ovat mielestäni erittäin hyvä lähtökohta perinteisten ohjelmistotuotannon prosessien ja käyttäjäkeskeisten prosessien yhdistämiseksi. RSI-mallin mukaiset käyttötapaukset sisältävät omat käyttötapauksensa sekä järjestelmän tekniselle suunnittelulle palvelukäyttötapausten muodossa että käyttöliittymälle käyttöliittymäkäyttötapausten muodossa. Lisäksi se sisältää järjestelmän toiminnallisuuden ja vaatimusten kuvauksen vaatimuskäyttötapausten muodossa. RSI-käyttötapauksia voitaisiin käyttää perinteisiä käyttötapauksia laajemmin hyödyksi järjestelmien testaamisessa, sillä käyttöliittymäkäyttötapaukset mahdollistavat myös järjestelmän käytettävyyden arvioimisen kehityksen hyvin aikaisessa vaiheessa, jolloin kehitysratkaisuja voidaan edelleen muuttaa. Että käytettävyystestausta olisi järkevää toteuttaa, tulisi koko analyysiprosessia muuttaa niin, että se huomioisi käytettävyysnäkökulmat.

Tässä tutkielmassa päädyin valitsemaan perinteisistä ohjelmistotuotannon prosesseista 1990-luvulla kehitetyn OMT-menetelmän, joka on oliopohjainen menetelmä järjestelmien kehittämiseksi. Käyttäjäkeskeisistä menetelmistä valitsin käytettäväksi RESPECT-menetelmän, joka perustuu ISO 13407 -standardin määritykseen käyttäjäkeskeiseen kehittämiseen tarvittavista prosesseista. Yhdistin nämä prosessimallit yhdeksi elinkaarimalliksi käyttäen ACUDUC-menetelmän kuvaamia periaatteita perinteisten ja käyttäjäkeskeisten prosessien yhdistämiseksi. Sekä OMT- että RESPECT-menetelmä sisältää käytön kuvaukset skenaarioiden muodossa, mutta omassa mallissani korvasin osittain nämä skenaariot RSI-mallin mukaisilla käyttötapauksilla. Tuloksena

oli käyttötapauspohjainen, käyttäjäkeskeinen, iteratiivinen ja inkrementaalinen prosessimalli interaktiivisten, oliopohjaisten järjestelmien kehittämiseksi. Lisäksi kehittämäni malli kuvaa kehitysorganisaatioiden eri roolien ja käyttäjien välisen kommunikaation sekä tehtävien jakautumisen eri roolien välillä. Malli kuvaa myös, miten eri vaiheiden tuotokset voidaan todentaa sekä käytettävyyden että toiminnallisuuden osalta hyödyntäen RSI-käyttötapauksia.

Käyttäjäkeskeisen kehittämisen yhdistäminen RSI-malliin aiheutti sen, että myös RSI-käyttötapaukset muuttuivat. ACUDUC-mallin mukaisesti vaatimuskäyttötapaukset yhdistävät toiminnallisten ja ei-toiminnallisten vaatimusten kuvaukset niin, että perinteiseen vaatimuskäyttötapaukseen lisätään käytön kontekstiin liittyviä poikkeuksia sekä määritellään käytettävyyden kvantitatiiviset tavoitteet, jotka tulee ottaa huomioon käyttötapauksen toteutuksessa. Lisäksi käyttöliittymäkäyttötapauksiin lisätään kyseiseen käyttöliittymään liittyvät käytettävyyden kvalitatiiviset tavoitteet, jotka näin ohjaavat käyttöliittymän kehittämistä. Käytettävyystavoitteiden lisääminen käyttötapauksiin helpottaa myös kyseiseen käyttötapaukseen liittyvien käytettävyystestauksen testitapausten kehittämisessä.

Vaikka kuvaamassani prosessimallissa käyttäjäkeskeisenä mallina käytettiin RESPECT-mallia, se ei sido käytännön toteutusta vain RESPECT-mallin mukaisten käytettävyysmenetelmien käyttämiseen. Tärkeintä mielestäni on, että prosessi toteuttaa ISO 13407 -standardin määrittelemät toiminnot: käyttäjien ja käytön kontekstin määrittelyn, käyttäjien ja organisaation vaatimusten määrittelyn, kehitysideoiden kehittämisen ja arvioinnin asetettuja vaatimuksia vasten yhdessä käyttäjien kanssa sekä mahdollisesti toimintojen iteroinnin. Toimintojen toteuttamiseksi voidaan käyttää myös muita käyttäjäkeskeisiä menetelmiä kuin RESPECT-mallin menetelmiä.

Valitsemani OMT-menetelmä järjestelmän tekniseen kuvaamiseen on mielestäni hieman vanhanaikainen ja mielestäni se sopii hieman huonosti inkrementaaliseen kehittämiseen. Tulevaisuudessa käyttäjäkeskeisiä RSI-käyttötapauksia voitaisiin ajatella yhdistettäväksi ketteriin menetelmiin, jolloin järjestelmä jakautuisi mahdollisesti pienempiin inkrementteihin, analyysivaihe voisi keventyä ja tuotettavien dokumenttien määrää voitaisiin vähentää. Myös testaaminen voisi siirtyä vielä lähemmäs testattavan toteutuksen toteuttamista varsinkin analyysivaiheen osalta.

Lähdeluettelo

Anda B. & Sjøberg D. I. K. (2002) Towards an inspection method for use case models. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, ACM Press, New York, NY, USA, pp. 137-134.

Binder R. V. (2000) Testing Object-Oriented Systems. Models, Patterns and Tools. Addison-Wesley, Reading, Massachusetts, USA.

Carniello A., Jino M. ja Chaim M. L. (2005) Structural Testing with Use Cases. *Journal of Computer Science and Technologie*, **5**(2), pp. 100-106.

Cockburn A. (1997) Goals and Use Cases. *The Journal of Object-Oriented Programming*, **10**(5), pp. 35-40.

Collins-Cope, M. (2000) *RSI – A Structured Approach to Use Cases and HCI Design.* http://www.ratio.co.uk (3.9.2000).

Constantine L. L. (1995) Essential Modeling: Use Cases for User Interfaces. *Interactions*, **2**(2), pp. 34-46.

Gould J.D. & Lewis C. (1985) Designing for Usability: Key Principles and What Designers Think. *Communications of ACM*, **2**(3), pp. 300-311.

IBM. Rational Unified Process. www-sivu, URL: http://www-306.ibm.com/software/awdtools/rup/ (17.8.2006).

Jacobson I. (1987) Object Oriented Development in an Industrial Environment. Conference proceedings on Object-oriented programming systems, languages and applications, pp. 183-191.

Jacobson I., Booch G. ja Rumbaugh J. (1998) *The Unified Software Development Process*. Addison-Wesley, Upper Saddle River, NJ, USA.

Jacobson I., Christerson M., Johnsson P. ja Övergaard G. (1992) *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, Redwood City, CA, USA.

Jacobson I. ja Ng P.-W. (2005) *Aspect-Oriented Software Development with Use Cases*. Addison-Wesley, Upper Saddle River, NJ, USA.

Jokela T., Iivari N., Matero J. ja Karukka M. (2003) The Standard of User-Centered Design and the Standard Definition of Usability: Analyzing ISO 13407 against ISO 9241-11. *Proceedings of the Latin American conference on Human-computer interaction*, ACM Press, New York, NY, USA, pp. 53-60.

Kaner C. (1988) *Testing Computer Software*. TAB Books, USA.

Karat J. & Dayton T. (1995) Practical education for improving software usability. *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co, New York, NY, USA, pp.162-169.

Kirner D., Porter R., Punniamoorthy P., Schuh M., Shoup D., Tindall S. ja Umphress D. (1999) Extending Use Cases throughout the Software Lifecycle. *ACM Sigsoft Software Engineering Notes*, **24**(3), pp. 66-68.

Kotkaluoto S. (2005) *Osallistava ryhmäläpikäynti*. Käytettävyystutkimuksen menetelmät B-2005-1, Tietojenkäsittelytieteen laitos, Tampereen yliopisto, Tampere, Suomi.

Maguire M. C. (1998) *User-Centered Requirements Handbook*. EC Telematics Applications Program, Project TE 2010 RESPECT, Wp5 Deliverable D5.3.

Mayhew D. J. (1994) *The Usability Engineering Lifecycle*. Academic Press, USA.

Meyer B. (1994) *Object-oriented Software Construction*. Prentice Hall, Cambridge, UK.

Myers G. J. (1976) *Software Reliability: Principles and Practices*. John Wiley & Sons, USA.

Myers G. J. (1979) *The Art of Software Testing*. John Wiley & Sons, USA.

Nielsen J. (1993) *Usability Engineering*. Academic Press, USA.

Nielsen J. & Mack R. L. (1994) *Usability inspection methods*. John Wiley & Sons, USA.

Nygaard K., (1986) Program development as a Social Activity, *INFORMATION PROCESSING 86*, (toim. H.-J. Kugler), Elsevier Science Publishers B.V. (North Holland), pp. 189-198.

OMG (2005) *UML – Unified Modelling Language.* www-sivusto. http://www.omg.org/cgi-bin/doc?ad/99-06-09, (14.11.2005).

Ratio (2005) *Ratio – training, consulting, software development.* www-sivusto. http://www.ratio.co.uk (14.11.2005).

Rubin J. (1994) *Handbook of usability testing*. John Wiley & Sons, USA.

Rumbaugh J. (1994) Getting started; Using use cases to capture requirements. *Journal of Object-Oriented Programming*, **7**(5), pp. 8 – 12, 23.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. ja Lorensen W. (1991) *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey, USA.

Seffah A., Djouab R. ja Antunes H. (2001) Comparing and Reconciling Usability-Centered and Use Case-Driven Requirements Engineering Processes. *Proceedings of the 2nd Australasian conference on User interface*, IEEE Computer Society Washington, DC, USA, pp.132-139.

Sousa K., Furtado E. ja Mendonça H. (2005) UPi – A Software Development Process Aiming at Usability, Productivity and Integration. *Proceedings of the 2005 Latin American conference on Human-computer interaction*, ACM Press, New York, NY, USA, pp. 76-87.

The Institute of Electrical and Electronic Engineers (1990) *ANSI/IEEE standard 610.12-1990: glossary of software engineering terminology*. New York, NY, USA.

Weidenhaupt K., Pohl K., Jarke M. ja Haumer P. (1998) Scenarios in System Development: Current Practice. *IEEE Software*, **15**(2), pp. 34-45.

Wiegers K. E. (1999) *Software Requirements*. Microsoft Press, Redmond, Washington, USA.

LIITE 1: Pankkiautomaatin analyysin ensimmäisen vaiheen dokumentit

1.1 Projektin yhteenveto

1.1 Projektin yhteenveto	
Kysymys	**Oletus**
Mikä kehitettävä järjestelmä tai palvelu on?	Uusi pankkiautomaatti järjestelmä
Mitä toimintoja järjestelmän on aiottu sisältävän?	Perinteiset palvelut rahan nostamiseksi ja tilin saldon tarkastamiseksi. Mahdolliset uudet toiminnot voisivat olla seteleiden vaihtaminen kolikoihin, tilisiirtojen tekeminen omien tilien välillä ja lainan hakeminen.
Mitkä ovat projektin tavoitteet?	Luoda uusia palveluja pankin asiakkaille käytettäväksi automaatin kautta, tarjota luotettava palvelua sekä tarjota turvallisempi ja turvatumpi palvelu.
Kenelle järjestelmä on tarkoitettu?	Pankeille.
Ketkä tulevat käyttämään järjestelmää?	Yleisesti pankin asiakkaana olevat henkilöt, mutta erityisesti vetoaa uusiin käyttäjiin, kuten vanhuksiin ja rajoitteisiin henkilöihin.
Miksi järjestelmää tarvitaan?	Edistämään automaattien käyttöä.
Missä järjestelmää käytetään?	Pankeissa sekä yleisissä rakannuksissa, kuten asemien, kauppojen jne. yhteydessä.
Kuinka järjestelmää käytetään?	Käytetään nykyistä tapaa, eli käyttäjä seuraa näytölle annettavia ohjeita ja antaa komentoja näppäimistön avulla, mutta myös muita kommunikointi tapoja, kuten ääni tai erillinen kaukosäädin, voidaan käyttää.
Kuinka käyttäjä hankkii järjestelmän?	-
Kuinka järjestelmä asennetaan?	Järjestelmä asennetaan automaattilaitteeseen etukäteen ennen toimitusta.
Kuinka järjestelmää ylläpidetään?	Erillisen huolto-käyttöliittymän avulla, joka kertoo järjestelmän tilasta. Pankkihenkilökunta koulutetaan perus ylläpitoa varten ja erilliset huoltajat vastaavat suurimmista ongelmista.

1.2 Käyttäjien ja muiden sidosryhmien määrittely

1.2 Käyttäjät ja muut sidosryhmät		
Järjestelmä: Uusi pankkiautomaatti järjestelmä		
Käyttäjä	**Rooli järjestelmässä tai järjestelmän käyttö**	**Käyttäjien vaatimukset dokumentoidaan**
Pankin asiakkaat	Käyttävät automaattia käyttääkseen palveluja	X
Pankin henkilökunta	Vastaavat päivittäisestä huollosta, kuten kuitin ja rahan lisääminen, korjaavat pienempiä vikoja ja raportoivat suuremmista vioista	X
Automaatin huoltajat	Suorittavat perustarkastuksen puolen vuoden välein ja tulevat korjaamaan suuremmat viat.	X
Sidosryhmät	**Rooli järjestelmässä tai järjestelmän käyttö**	**Käyttäjien vaatimukset dokumentoidaan**
Pankin markkinointi henkilöstö	Päättävät, mitä palveluja automaatti tarjoaa ja mitä mainoksia automaatti esittää, kun se ei ole käytössä.	

1.3 Käyttäjien erityispiirteet

1.3 Käyttäjien erityispiirteet		
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**		
Omainaispiirteet	**Mahdollinen käyttäjän vaatimus**	**Viite**
Käyttäjäryhmän koko: Koko Suomen väestö ja ulkomaalaiset turistit.		
Ikä: 13:sta ylöspäin.	Tulee kiinnittää huomiota vanhempiin ikäryhmiin, joilla voi olla ennakkoluuloja teknologiaa kohtaan.	1.3.1
Sukupuoli: Suunnilleen yhtä paljon naisia kuin miehiäkin.	Käytettävyystestauksessa yhtä paljon naisia kuin miehiäkin testihenkilöinä.	1.3.2
Kieli ja kulttuuri: Suomi on yleiskieli, joillakin alueilla yli 50% väestöstä puhuu ruotsia. Turisteja etenkin EU-maista ja joillakin alueilla etenkin Venäjältä.	Käytetään sekä suomen että ruotsin kieltä alueesta riippuen, lisäksi 8:n muuta kielivaihtoehtoa. Käytetään yksinkertaista sanastoa ja paljon kuvia.	1.3.3 1.3.4
Koulutustaso: Mikä tahansa.	Käyttöliittymä suunniteltava niin, että se huomioi lukuvaikeuksiset henkilöt.	1.3.5
Fyysiset rajoitteet: Kaikki mahdolliset. Sisältää henkilöt, jotka ovat näkörajoitteisia tai muuten rajoittuneita.	Varmistettava, että näppäimistö ja näyttö on asennettu standardi korkeudelle. Käytetään helppoja syöttölaitteita: näppäimistöjä, joissa on suuret näppäimet ja vaihtoehtoisia tunnistautumismenetelmiä.	1.3.6 1.3.7
Erityistaidot: Ei ole		
Kokemus vastaavien järjestelmien käytöstä: 70% käyttäjistä on käyttänyt automaatteja aikaisemmin.	Järjestelmän tulisi noudattaa olemassa olevia pankkiautomaatteihin liittyviä standardeja.	1.3.8
IT-kokemus: Vaihteleva, oletusarvoisesti ei kokemusta ollenkaan.	Käytetään käyttäjien toimintaa tukevia dialogeja. Käytetään käyttäjille miellyttäviä käyttöliittymiä.	1.3.9 1.3.10
Tehtävien tuntemus: Vaihteleva, oletusarvoisesti ei tuntemusta ollenkaan.	Käytettään käyttäjien toimintaa tukevia käyttöliittymiä, joissa on selkeä rakenne. Käytetään termejä, jotka käyttäjät ymmärtävät.	1.3.11 1.3.12
Aikaisempi koulutus: Ei ole		
Käyttötiheys: Enimmäkseen harvoin käyttäviä.	Käytetään dialogeja, jotka ovat helppo oppia ja muistaa.	1.3.13
Käytön motivaatio: Voi olla haluttomia käyttää.	Tehdään järjestelmästä houkutteleva käyttää.	1.3.14

Käyttäjien valinnanvapaus: Voiko käyttäjä valita käyttääkö vai eikö käytä tuotetta? Voi olla käyttämättä järjestelmää mistä tahansa syystä.	Tehdään järjestelmästä houkutteleva ja helppo käyttää. Varmistetaan, että tavoitteet voidaan saavuttaa nopeasti.	1.3.15 1.3.16
Käyttäjien huolenaiheet: Huoli varastetuksi tulemisesta.	Tarjotaan turvallisuus ominaisuuksia, kuten hälytys-painike. Varmistetaan, että automaatti tarjoaa käytön ajaksi yksityisyyttä.	1.3.17 1.3.18
Muita oleellisa ominaispiirteitä: Tulisi houkutella satunnaisia käyttäjiä.	Tehdään järjestelmästä mahdollisimman houkutteleva ja helppo käyttää ensikertalaisille.	1.3.19

1.4 Teknisen ympäristön kuvaus

1.4 Tekninen ympäristö		
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**		
Omainaispiirteet	**Mahdollinen käyttäjän vaatimus**	**Viite**
Laitteisto, jota käyttäjät käyttävät: Perinteiset pankkiautomaatit, jotka muunnetaan vastaamaan uuden järjestelmän vaatimuksia. Mahdollisesti voidaan käyttää kosketusnäyttöä. Kaijutin ja mikki voidaan asentaa ääni syötteen ja ulostulon mahdollistamiseksi.	Laitteiston tulisi olla kestävä. Käyttöliittymä elementtien ja näppäimistön asettelun tulisi vastata olemassa olevia standardeja	1.4.1 1.4.2
Käytettävät ohjelmistot: IBM Public Terminal	Käytetään IBM:n Public Terminal:ia.	1.4.3
Ohjelmisto, jota käytetään järjestelmän kehittämiseen: Public ATM builder	Käyttäjille tulisi esitellä muita, jo olemassa olevia järjestelmiä, jotka on kehitetty valitulla ohjelmistolla.	1.4.4
Muut käytön vaatimat välineet: Puhelin, jolla voidaan ottaa yhteyttä ylläpitoon ongelmatilanteissa.	Automaatti voisi lähettää ilmoituksen automaattisesti, jos suurempi virhe tapahtuu	1.4.5
Materiaali, jota tarvitaan käytön opetteluun tai tehtävien suorittamiseen: Ohjekirja	Automaattikortin tulisi sisältää toiminnan pääkohdat, joita noudattamalla toiminta voidaan suorittaa.	1.4.6

1.5 Fyysinen ympäristö

1.5 Fyysinen ympäristö		
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**		
Omainaispiirteet	**Mahdollinen käyttäjän vaatimus**	**Viite**
Ilmasto: Suomalainen ilmasto rannikoilta sisämaahan ja pohjoisen tuntureille.	Välineiden tulisi toimia seuraavanlaisissa olosuhteissa: lämpötila -40 C - +30 C, kosteus 30-50 %, vesi- ja lumisateita.	1.5.1
Auditiivinen ympäristö: Suomalainen kaupunki, katuäänet.	Äänipalautet voi hukkua katumeteliin, ellei käytössä ole äänen säätöä tai korvakuuloketta.	1.5.2
Värähtely tai epävakaisuus: Ei vaikutusta		
Visuaalinen ympäristö: Automaattia käytetään päivällä ja yöllä. Aurinko voi aiheuttaa heijastumia näytölle.	Näytön tulisi ehkäistä häikäisyä. Mattapintaiset näytöt tulee testata käytössä. Tarvitaan erillinen valo yökäyttöä varten.	1.5.3
Tila ja varusteet: Automaatin tulisi olla käytettävissä laajalle joukolle ihmisiä.	Automaatti tulisi asentaa 1 m maan pinnan yläpuolelle, seinään upotettuna.	1.5.4
Käyttäjien asento: Automaattia käytetään yleensä seisten. Pyörätuoli käyttäjät käyttävät automaattia istualtaan.	80% pyörätuolikäyttäjistä tulisi pystyä käyttämään automaattia korkeuden ja asennon puolesta.	1.5.5
Sijainti: Kadulla, yleisillä kulkuväylillä.	Automaatin tulee olla helposti havaittavissa ja löydettävissä, mikäli käyttäjä etsii automaattia.	1.5.6
Terveys ja turvallisuus riskit: Vaara, että rahaa nostavat henkilöt ryöstetään ja pahoinpidellään.	Automaatti tulee sijoitaa avoimelle paikalle, jossa on valoisaa.	1.5.7
Suojaavat vaatteet/varusteet: Talvivaatetus sisältää hanskat tai rukkaset.	Nappuloiden tulee olla paineltavissa hansikoiduin käsin.	1.5.8

1.6 Sosiaalinen ja organisatorinen ympäristö

1.6 Sosiaalinen ja organisatorinen ympäristö		
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**		
Omainaispiirteet	**Mahdollinen käyttäjän vaatimus**	**Viite**
Henkilökunta- ja johtorakenne: ei merkitystä		
Kommunikointi rakenne: Ei merkitystä		
IT käytäntö: Jokaisella pankilla on oma automaattinsa ja he rohkaisevat automaatin käyttöä henkilöstön ajankäytön vähentämiseksi.	Henkilökunnan tulee pystyä opastamaan pankin asiakkaita automaatin käytössä. Pankissa tulisi olla aina henkilö, joka voi suorittaa asiakkaan haluamat toiminnot, jos asiakas ei halua käyttää automaattia.	1.6.1
Organisaation tavoitteet: Ei merkitystä		
Liiketoiminnalliset yhteydet: Ei merkitystä		
Toiminnan seuranta: Käyttäjät odottavat suhteellisen nopeita ja yhdenmukaisia vasteaikoja.	Automaattien vasteaikoja ja suorituksia/päivä tulee seurata säännöllisesti.	1.6.2
Palaute toiminnasta: Pankin henkilökunnan tulee pystyä manuaalisesti tarkastamaan järjestelmän tila ja rahan sekä kuittipaperin määrä.	Henkilökunnan tulisi pystyä keskeyttämään automaatin käyttö pikaista tarkastusta varten.	1.6.3
Ryhmätyöskentely: Käyttäjät ovat normaalisti yksin automaatilla, toisinaan parin kanssa.	Kahden käyttäjän tulee pystyä katselmaan/käyttämään automaattia ilman epämukavuutta.	1.6.4
Vaatii avustusta/apua on tarjolla: Apua voi mahdollisesti saada sisältä pankista tai muilta henkilöiltä jonossa. Noviisikäyttäjätä voivat vaatia apua, tai henkilöt, joiden kortti on kadonnut.	Apua ei ole normaalisti saatavissa pankin ulkopuolisilla automaateilla. Käyttäjä tarvitsee menetelmän, jonka avulla havainnoida ongelma ja saada siihen apua pian havaitsemisen jälkeen.	1.6.5
Keskeytykset/stressaavat olosuhteet: Jonoja voi muodostua kiireisinä aikoina.	Käyttäjät tarvitsevat mahdollisuuden keskeyttää toiminnon, jos he eivät voi edetä eteenpäin tai tuntevat itsensä uhatuksi.	1.6.6
Turvallisuus: Vaara tulla ryöstetyksi tai pahoinpidellyksi automaatilla.	Automaatti voi sisältää painikkeen hälytyksen tekemiseen, jos ryöstö tapahtuu.	1.6.7
Yksityisyys: Vaara, että muut henkilöt näkevät käyttäjän tilitiedot tai kortin tunnusluvun.	Automaatin tulisi estää muita jonossa olevia henkilöitä näkemästä tapahtuman suorittaminen.	1.6.8
Työn tarkoitus: Ei merkitystä		
Työn kesto: Ei merkitystä		
Työn joustavuus: Ei merkitystä		
Arvostetut taidot: Ei merkitystä		

1.7 Käyttäjien tavoitteet ja tehtävät

1.7 Käyttäjien tavoitteet ja tehtävät			
Järjestelmä: Uusi pankkiautomaatti järjestelmä			
	Asiakas	**Pankin henkilökunta**	**Ylläpito**
G1: Palvelun suorittaminen nopeasti ja turvallisesti.	X		
G2: Rahan lisääminen		X	
G3: Kuittipaperin lisääminen		X	
G4: Virheen ilmoittaminen automaatin avulla	X	X	
G5: Virheen korjaaminen		X	X

1.8 Nykyisten prosessien arvioiminen

1.8 Nykyinen prosessi			
Käyttäjäryhmä: Automaatin käyttäjät **Tavoite: Palvelun suorittaminen nopeasti ja turvallisesti**			
Tehtävän askel	**Ongelma/tehtävän variaatio**	**Mahdollinen käyttäjän vaatimus**	**Viite**
1. Asetu automaatti jonoon odottamaan vuoroa.			
2. Syötä kortti.	Kortti syötetään väärinpäin.	Korttiin lovi. Automaattiin kuva oikeasta kortin asennosta.	1.8.1 1.8.2
3. Syötä tunnusluku.	Käyttäjä unohtaa tunnusluvun. Automaatti hajoaa. Jonossa oleva henkilö kiinnittää liikaa huomiota tapahtuman suorittamiseen.	Salli sormenjälkitunnistus. Nopea reset-toiminto. Lisää automaatin käytön yksityisyyttä.	1.8.3 1.8.4
4. Valitse rahan nosto.			
5. Valitse tai syötä nostettava summa.	Valittu suomma on suurempi kuin pankkitilin saldo. Automaatista loppuu raha. Käyttäjä päättää keskeyttää tapahtuman. Käyttäjä valitsee saman nostettavan summa lähes aina.	Näytä nostettavissa oleva summa Mahdollista tapahtuman peruuttaminen nopeasti. Mahdollista oikopolut usein valittuihin vaihtoehtoihin.	1.8.5 1.8.6 1.8.7
6. Valitse tulostetaanko kuitti.	Automaatista loppuu paperi.		
7. Ota kortti.	Kortti ei palaudu.		
8. Ota rahat ja kuitti (jos valitsit kuittitulostuksen).	Rahaa ja/tai kuittia ei anneta. Automaatti antaa väärän summan rahaa. Automaattikäyttäjä ryöstetään.	Mahdollista virheen raportoiminen sen sattuessa. Lisää automaattiin kamera, joka nauhoittaa automaatin tapahtumat.	1.8.8 1.8.9

1.9 Suunnittelu ideoiden ja käsitteiden kehittäminen

1.10 Suunnittelu ideat ja käsitteet			
Järjestelmä: Uusi pankkiautomaatti järjestelmä			
	Ideat ja käsitteet	**Kommentit**	**Viite**
Yleiset ideat			
	Aputoiminto äänen avulla.		
Tavoitteisiin liittyvät ideat			
G1: Palvelun suorittaminen nopeasti ja turvallisesti.	Kysymys-vastaus -toiminto aloittelijoille		
G2: Rahan lisääminen	Yksinkertainen lisäyssysteemi, jota voidaan käyttää niin että automaattia ei tarvitse sulkea tai sammuttaa.		
G3: Kuittipaperin lisääminen	Yksinkertainen lisäyssysteemi, jota voidaan käyttää niin että automaattia ei tarvitse sulkea tai sammuttaa.		
G4: Virheen ilmoittaminen automaatin avulla	Lisää painike, jonka avulla käyttäjät voivat ilmoitta pankille, jos heidän korttinsa juuttuu automaattiin.		
G5: Virheen korjaaminen	Tee mahdolliseksi, että pankin henkilökunta voisi korjata joitakin virheitä niin, että ylläpito henkilöitä ei tarvitse kutsua paikalla joka kerta.		

1.10 Suunnittelu ideoiden ja käsitteiden arvioiminen

1.10 Suunnittelu ideat ja käsitteet			
Järjestelmä: Uusi pankkiautomaatti järjestelmä			
	Ideat ja käsitteet	**Kommentit**	**Viite**
Yleiset ideat			
	Aputoiminto äänen avulla.	Kallis toteuttaa, mutta voisi auttaa näkörajoitteisia käyttäjiä.	1.11.1
Tavoitteisiin liittyvät ideat			
G1: Palvelun suorittaminen nopeasti ja turvallisesti.	Kysymys-vastaus -toiminto aloittelijoille	Käyttökelpoinen idea, mutta voi hidastaa tapahtuman suorittamista.	1.11.2
G2: Rahan lisääminen	Yksinkertainen lisäyssysteemi, jota voidaan käyttää niin että automaattia ei tarvitse sulkea tai sammuttaa.	Ei suositeltava, koska voi aiheuttaa elektroniikkavikoja.	
G3: Kuittipaperin lisääminen	Yksinkertainen lisäyssysteemi, jota voidaan käyttää niin että automaattia ei tarvitse sulkea tai sammuttaa.	Ei suositeltava, koska voi aiheuttaa elektroniikkavikoja.	
G4: Virheen ilmoittaminen automaatin avulla	Lisää painike, jonka avulla käyttäjät voivat ilmoitta pankille, jos heidän korttinsa juuttuu automaattiin.	Hyvä idea, jos painike on käytettävissä vain silloin, kun kortti on oikeasti juuttunut automaattiin.	1.11.3
G5: Virheen korjaaminen	Tee mahdolliseksi, että pankin henkilökunta voisi korjata joitakin virheitä niin, että ylläpito henkilöitä ei tarvitse kutsua paikalla joka kerta.	Harkitsemisen arvoinen. Vaarana on, että henkilökunta ylikuormittuu.	1.11.4

LIITE 2: Pankkiautomaatin analyysin toisen vaiheen dokumentit

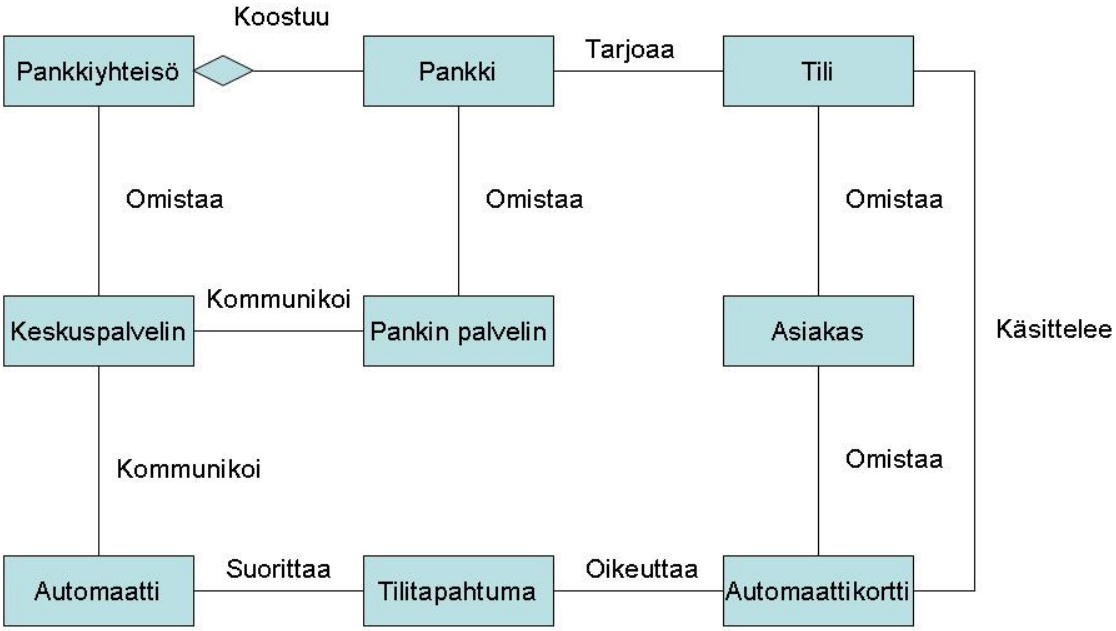
2.1 Yleiset käytettävyystavoitteet

2.1 Yleiset käytettävyystavoitteet		
Järjestelmä: Uusi pankkiautomaatti järjestelmä **Käyttäjäryhmä: Automaatin yleiset käyttäjät**		
Käytettävyystavoite	**Käyttäjän tavoite, joka toteuttaa käytettävyystavoitteen**	**Tärkeä tavoite**
Tehokkuus: Suoritettujen tehtävien laatu tai määrä.	On tärkeää, että käyttäjä voi suorittaa tehtävät virheettömästi	X
Käytön tehokkuus: Tehtävän suorittamiseen kuluva aika, aika verrattuna kokeneen käyttäjän aikaan.	Käyttäjät odottavat käyttävänsä automaattia nopeasti ja he tulevat kärsimättömiksi, jos vasteajat ovat pitkät.	X
Käytön tyydyttävyys: Järjestelmän käytöstä havaittu ilo tai tyydytys.	Tehtävän tyydyttävyys syntyy kyvystä suorittaa tehtäväs nopeasti ja onnistuneesti.	
Opittavuus: Kyky käyttää järjestelmän aputoimintoa tai käyttöohjetta tehtävän suorittamiseen.	Automaatti voi sisältää lyhyet ohjeet, mutta järjestelmän käytön ei oleteta nojaavan aputoiminnon käyttöön.	
Vaistonvaraisuus: Kyky suorittaa tehtävä rajatulla määrällä ohjeita.	Uudet käyttäjät eivät halua käyttää järjestelmää jatkossa, jos sen käyttö ei ole vaistonvaraista.	X
Käyttäjien tukeminen/auttaminen: Kyky selvitä vastaantulevista ongelmista.	Järjestelmän tulisi auttaa käyttäjää, mikäli tämä ei pääse tehtävässä eteenpäin.	
Kontrolloitavuus: Käyttäjillä on tunne, että he kontrolloivat järjestelmän toimintaa.	Käyttäjien tulisi olla luottavaisia, että he osaavat käyttää järjestelmää ja voivat selviytyä mahdollisista virhetilanteista.	
Välttää muistin kuormittamista: Henkisten ponnistelujen havaitseminen, fyysisten merkkien tarkkaileminen.	Järjestelmän ei tulisi vaatia käyttäjää muistamaan pitkiä tunnuslukuja.	
Välttää fyysistä kuormittamista: Sykkeen ja hengitystiheyden mittaaminen.	Henkilöiden, joilla on motorinen vamma tai jotka käyttävät pyörätuolia, tulisi pystyä käyttämään automaattia ilman epämukavuutta.	X
Turvallisuus: Mahdollisuus käyttää järjestelmää turvallisesti.	Automaatti tulee asentaa paikkaan, jossa käyttäjät tuntevat olonsa turvalliseksi ja joka on hyvin valaistu.	

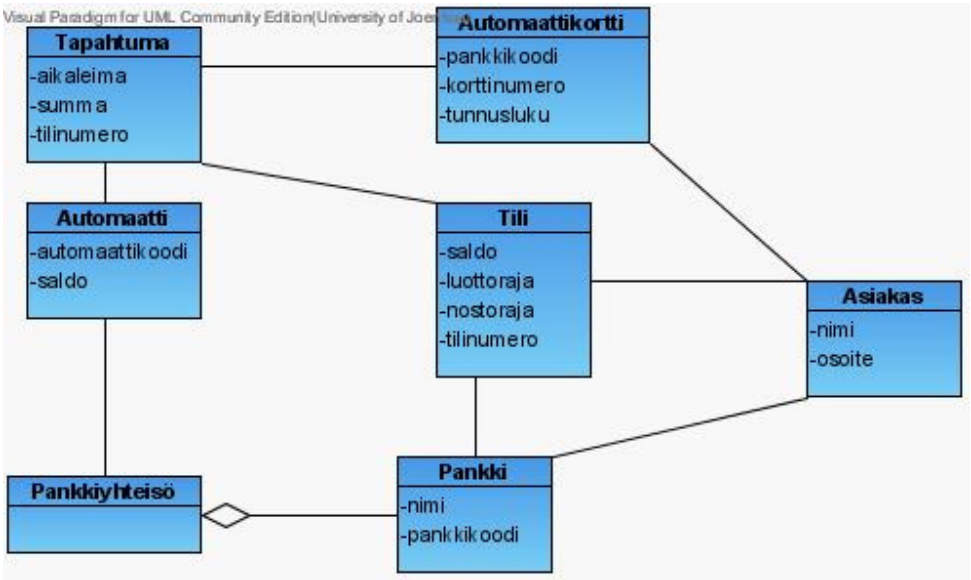
2.2 Uuden prosessin kuvaus

2.4 Uusi prosessi			
Käyttäjäryhmä: Automaatin käyttäjät **Tavoite: Palvelun suorittaminen nopeasti ja turvallisesti**			
Skenaario: **Käyttäjä syöttää kortin ja tunnusluvun. Käyttäjä valitsee rahan noston. Käyttäjä valitsee nostettavaksi summaksi 50 € ja haluaa kuitin paperille. Käyttäjä ottaa kortin, rahat ja kuitin automaatista.**			
Toiminnan päämäärä: **90% nykyisistä pankkiautomaatin käyttäjistä tulisi pystyä suorittamaan tehtävä yhden minuutin sisällä. 70% ensikertaa automaattia käyttävistä tulisi myös suoritua tehtävästä minuutissa.**			
Tehtävän askel	**Mahdollinen toiminta tai ominaisuus**	**Liitä**	**Viite**
1. Asetu automaatti jonoon odottamaan vuoroa.			
2. Syötä kortti.	Kehitä kortin lukija, joka lukee kortin mistä tahansa suunnasta. Korttiin lovi. Automaattiin kuva oikeasta kortin asennosta.		2.4.1 2.4.2 2.4.3
3. Syötä tunnusluku.	Mahdollista tunnusluvun syöttäminen tai sormenjälkitunnistus.		2.4.4
4. Valitse rahan nosto.	Järjestelmä näyttää nostettavan summan maksimimäärän. Järjestelmä tarjoaa mahdollisuuden valita nostettavan summa tietyistä vaihtoehdoista esim. "Nosta 20, 60, 100 tai 120 €"		2.4.5 2.4.6
5. Valitse tai syötä nostettava summa.			
6. Valitse tulostetaanko kuitti.	Tarjoa käyttäjälle mahdollisuus valita "nosto ilman kuittia" tai "nosto ja kuitin tulostus".		2.4.7
7. Ota kortti.			
8. Ota rahat ja kuitti (jos valitsit kuittitulostuksen).			

2.3 Oliomalli

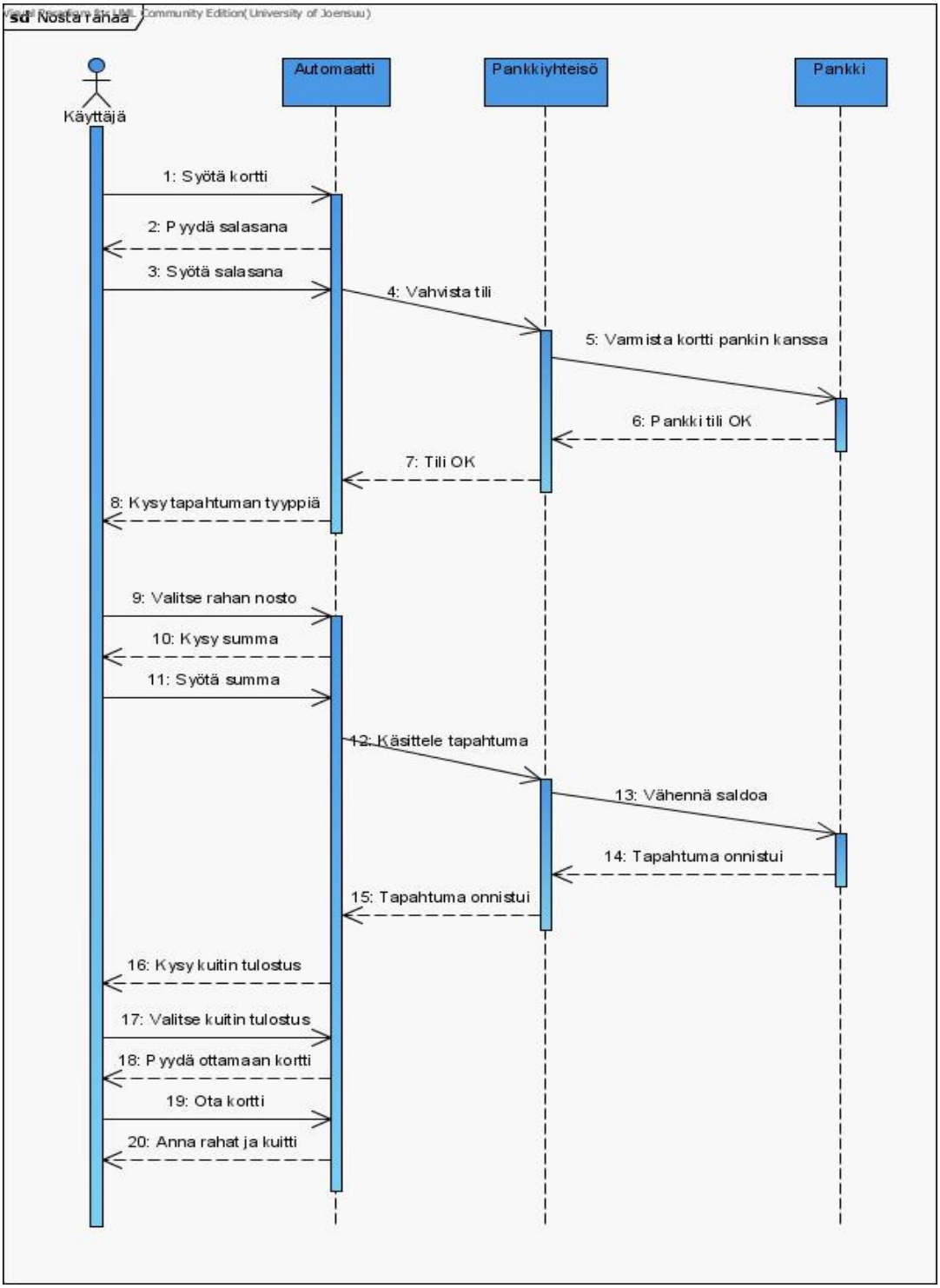


Kuva 37: Alustava luokkakaavio pankkiautomaatti järjestelmälle (Rumbaugh et al., 1991).

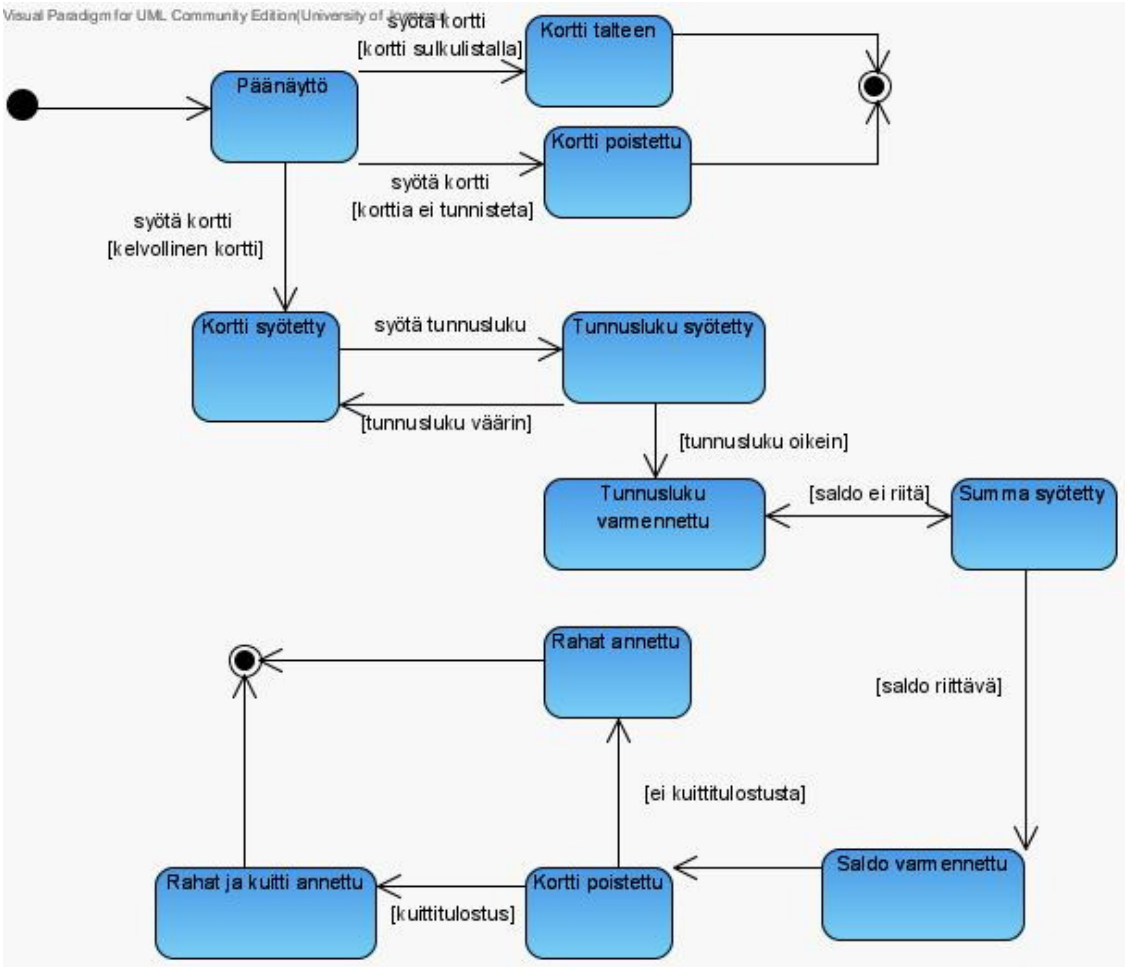


Kuva 38: Pankkiautomaatin luokkakaavio (Rumbaugh et al., 1991).

2.4 Dynaaminen malli

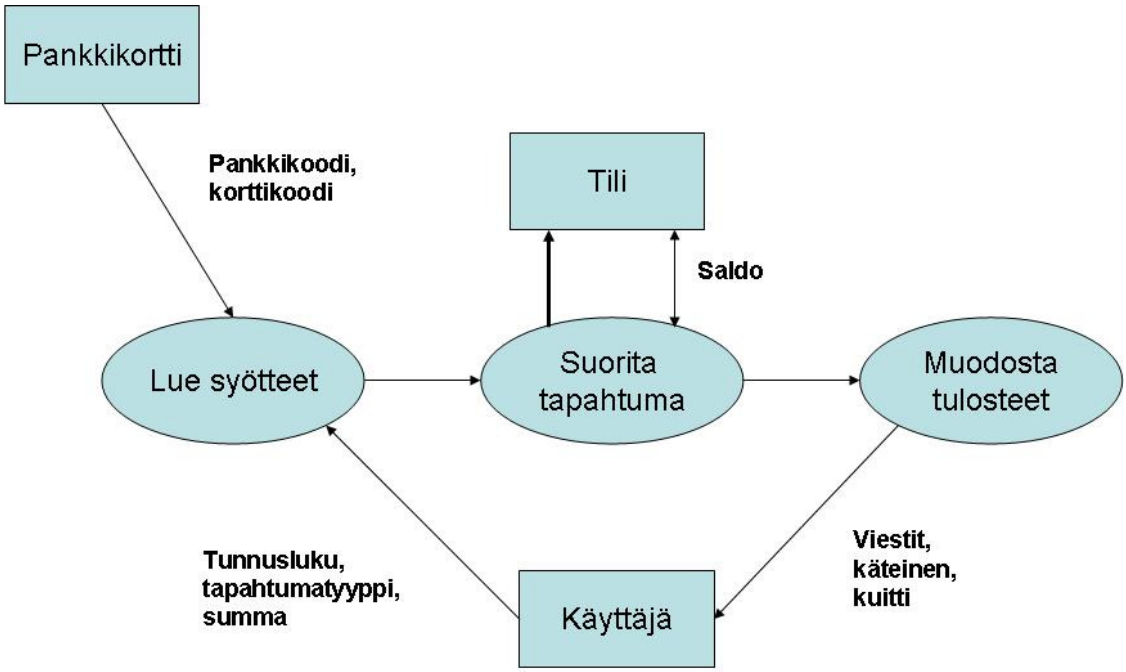


Kuva 39: Rahan nostaminen sekvenssikaaviona (Rumbaugh et al., 1991).

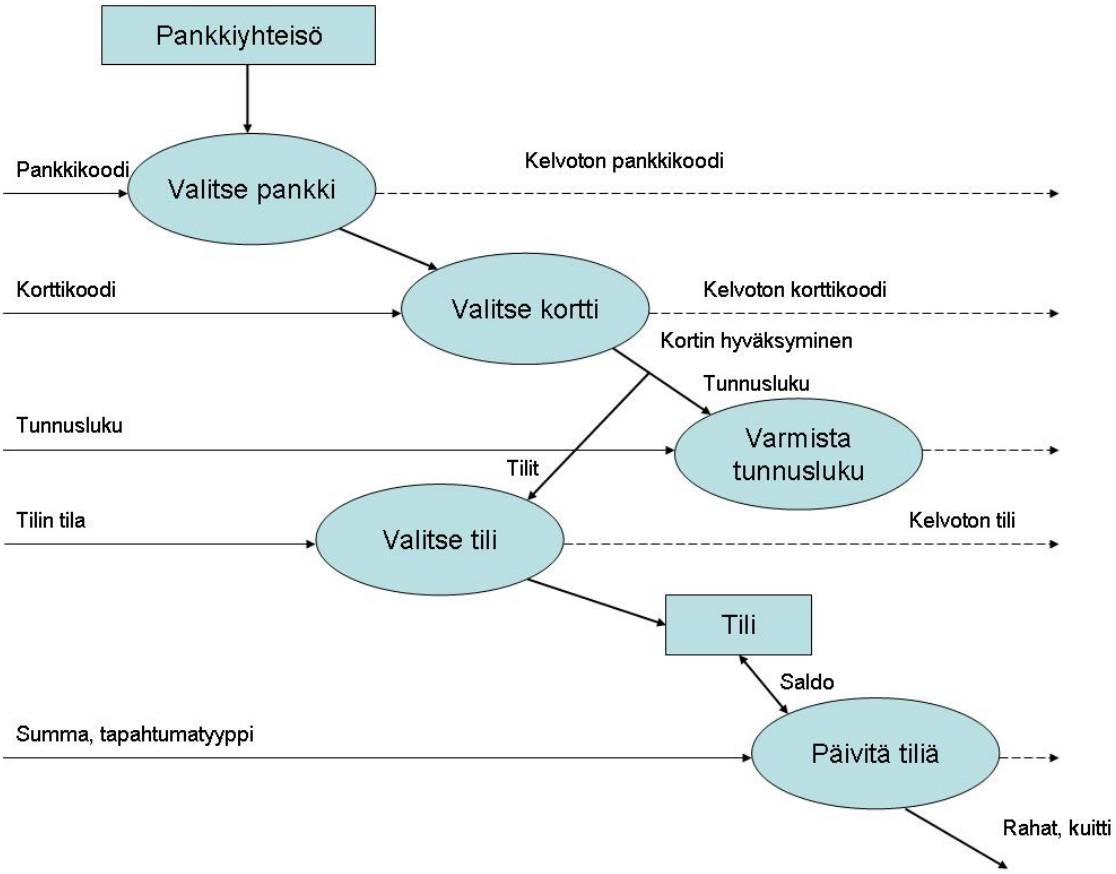


Kuva 40: Pankkiautomaatin tilakaavio (Rumbaugh et al., 1991).

2.5 Toiminnallinen malli



Kuva 41: Korkean tason tietovuokaavio pankkiautomaattijärjestelmälle (Rumbaugh et al., 1991).



Kuva 42: Tietovuokaavio tapahtumalle *Nosta rahaa* (Rumbaugh et al., 1991).