Luennot II–III:

Käyttötapauspohjaisen vaatimusmäärittelyn malleista

Tommi Kärkkäinen

JYVÄSKYLÄN YLIOPISTO
Informaatioteknologian tiedekunta
Tietotekniikan laitos
Syksy 2006

1 Johdantoa

1.1 Yleisesti

Monilla ohjelmistokehitysmenetelmillä ja -tekniikoilla on vastineensa muilla elämän alueilla. Tyypillinen esimerkki on samaistaa ohjelmiston rakentaminen talon (tai alkuvaiheessa puutarharakennuksen tai halkovajan ;-) rakennusprojektin kanssa. Ohjelmistotekniikan olemukseen kuuluu, että asioita pyritään jäsentämään selkeässä ja ymmärrettävässä muodossa. Tätä varten on kehitetty ja tullaan kehittämään erilaisia malleja, menetelmiä, notaatioita ja näiden käyttöä tukevia järjestelmiä. Tarkastellaan aluksi nyt meneillään olevan opintojakson, *Johdatus ohjelmistotekniikkaan*, sisällön organisoinnin haasteita (osin huumorimielellä ;-) saman tyylisesti aihetta jäsentäen, mitä tulemme käyttämään käyttötapaustuotannon organisoinnin kuvaamisessa.

(HUOM: Tässä ja jatkossakin suluissa esitetyt asiat ovat kirjoittajan omia (h)ajatelmia...)

Ongelma Ohjelmistotekniikan johdantokurssin sisällön suunnittelu on haastavaa ja toteutunut kurssi menee helposti päällekäin muiden opintojaksojen kanssa jääden liian kauaksi opiskelijoiden osaamiskontekstista, alan käytännöistä ja akateemisesta teoriasta.

Ajurit

- Yliopisto-opetuksessa on vaikeaa tietää mitä opiskelijat oikeasti osaavat tullessaan opintojaksolle.
- Ohjelmistokehityksellä ei ole olemassa selkeää perusteoriaa.
- Kaikkien osapuolien - käyttäjät, managerit, kehittäjät - tyytyväisyyttä on mahdotonta maksimoida, sillä ohjelmistokehitys muodostaa monitavoitteisen optimointikentän (dynaamisessa toimintaympäristössä), jossa on pakko tehdä kompromisseja.
- Ohjelmistotekniikalla tieteen- tai käytännön alana ei ole näkyvää rajausta tai yksikäsitteistä määritelmää.
- Ilman ohjelman varsinaista toteuttamista emme voi puhua ohjelmisto(teknisestä)projektista, mutta tässä suhteessa kurssille osallistuvien opiskelijoiden tieto- ja taitotaso vaihtelee suuresti.
- Opintojakson (tai opinnäytetyön, projektin ym.) todellinen menestys pohjautuu siihen, että löydetään toiminnan _PUNAINEN LANKA_ (so. tarkoitus, päämäärä) eikä hukata sitä työnteon aikana.

Ratkaisu Opintojakson sisällössä painotetaan pääosin aktiviteetteja ja työvaiheita, jotka liittyvät (laadukkaaseen) ohjelmistokehitykseen ja sen hallintaan.

- Teknisiä taitoja Ohjelmointi 1, 2 ym. kursseilla.
- Elinkaari(malli)ajattelun perusteita ja notaatioasioita "Oliokeskeisessä tietojärjestelmien kehittämisessä".
- Yleisempiä ohjelmistoprojekteihin liittyviä asioita "Ohjelmistotuotannossa".
- Tuntuma konkreettisiin asiakasprojekteihin “Sovellusprojektissa”.

Esimerkkejä Verkossa liikkuu ja sieltä löytyy valtavasti ohjelmistotekniikkaan ja -kehitykseen liittyvää materiaalia, jonka fokusoinnin ja organisoinnin suhteen on syytä olla kriittinen.

1.2 Käyttötapauskehitysmalleista

Erilaisia mallikieliä (engl. pattern language), jotka juontavat juurensa Alexander et al. (1977) kirjaan rakennusarkkitehtuurisista malleista, on käytetty ohjelmistokehityksen rakenteiseen kuvaukseen eri osa-alueilla: Design Patterns (Gamma et al., 1994): sovellusrakenteet, Object-oriented Reengineering Patterns (Demayer et al., 2003): ylläpito ja takaisinmallinnus ym. (Lisää esimerkkejä löytyy werkosta, Amazonista ja esim. osoitteesta *http://hillside.net/patterns/*.)

Seuraavassa käsiteltävä materiaali perustuu kirjaan
Adolph and Bramble: "Patterns for Effective Use Cases", The Agile Software Development Series, Cockburn and Highsmith (Eds.), Addison-Wesley, 2003.
Kirjassa käyttötapausmallit on esitetty muodossa

- mallin nimi
- kuva
- konteksti
- ongelma
- metaforinen tarina
- ajurit
- ratkaisu
- esimerkkejä

Jatkossa yllä oleva esitysmuoto supistetaan muotoon
Nimi x *Ongelma* x *Ajurit* x *Ratkaisu* (+kommentit) (x *Esimerkkejä*)

Yleisesti käyttötapauksen avulla ohjelman toiminnallisuutta jäsennetään neljällä eri tarkkuustasolla - neljäbittinen esitys alan gurun Cockburn:n mukaan:

- aktorit ja tavoitteet
- pääskenaario
- poikkeukset
- poikkeusten käsittely

Tavoitteena ei ole tuottaa välittömästi kattavia ja lopullisia tuotoksia (eli käyttötapauskuvauksia), vaan ongelmaa pyritään jäsentämään asteittain tarkentuen - iteratiivisesti ja inkrementaalisesti. Tätä on havainnollistettu liitteestä löytyvässä yhteenvedossa.

Yleisesti ottaen esitettävät mallit jakaantuvat *kehitysmalleihin* (engl. development) ja *rakenteisiin malleihin* (engl. structure), joiden välistä suhdetta on myös havainnollistettu liitteessä. Jako staattiseen rakenteeseen (esim. luokat, oliot, testitapaukset) ja dynaamiseen käyttäytymiseen (esim. viestinvälitys komponenttien/pakettien/luokkien välillä) on tyypillistä myös muuhun ohjelmistokehitykseen liittyen - näin jakautuvat esim. UML-diagrammit.

Tässä tarkasteltavat mallit esitetään eri järjestyksessä kuin alkuperäisessä lähteessä. Tällä pyrimme havainnollistamaan eri mallien “tärkeyttä” koko ohjelmistoprojektin näkökulmasta. Esitysmuoto perustuu seuraavien kysymysten ympärille:

1. Miltä laadukas tuotos (esim. käyttötapausjoukko joka kuvaa SuD:n (engl. System-under-Development) toiminnallisia vaatimuksia) näyttää?

2. Miten kehitysvaiheen työt tulisi organisoida, jotta laadukas tuotos saadaan aikaan?

Näiden kysymysten vastauksia tarkastellaan seuraavien malliosajoukkojen suhteen:

1. Rakenne: Käyttötapausjoukko: Yleisimmät ja (onnistumisen kannalta) tärkeimmät mallit yhteenveto-tasolla (Cockburn), jotka kuvaavat vaihetuotosta kokonaisuudessaan.

2. Kehitys: Prosessi: Tärkeimmät mallit yhteenveto-tasolla, jotka kuvaavat vaihetuotoksen tuottamisen ydinaktiviteetteja.

3. Rakenne: Käyttötapaus: Mistä käyttötapaus koostuu? Miltä se näyttää?

4. Rakenne: Skenaario: Miltä pääskenaario näyttää?

5. Rakenne: Askel: Miltä skenaarion yksittäinen askel näyttää?

6. Rakenne: Käyttötapaussuhteet: Minkätyyppistä uudelleenmuokkausta käyttötapausjoukkoon voidaan soveltaa?

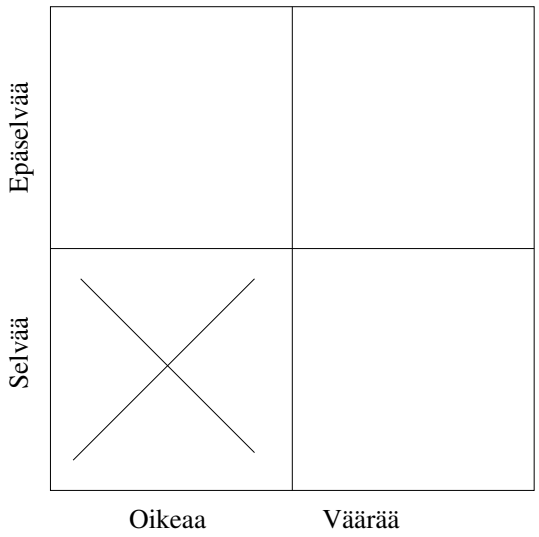
7. Kehitys: Muokkaus: Kuinka uudelleenmuokkaus organisoidaan?

8. Kehitys: Tiimi: Kuinka työ yleisesti organisoidaan?

1.3 Aiheeseen liittyvää (oheis)materiaalia

http://prosessit.it.jyu.fi/ (IT-prosessit)

http://sovellusprojektit.it.jyu.fi/ucot/ (UCOT)



Kuva 1: Ohjelmistokehityksen vaihetuotteiden sisällön nelikenttä.

2 Rakenne: Käyttötapausjoukko

Seuraavat mallit vastaavat melko hyvin ohjelmistoprojektien yleisiä menestystekijöitä (vrt. CHAOS-raporttien listat), joten ne on nostettu tässä mallien joukossa tärkeimpään asemaan.

JaettuSelkeäVisio

Äärimmäinen liiketoimintakäyttötapauksen esivaatimus: Miksi joku maksaa järjestelmän rakentamisesta (ja/tai panostaa siihen resursseja)?

Ongelma Selkeän näkemyksen puuttuminen järjestelmän suhteen voi johtaa päättämättömyyteen ja ristiriitaisiin mielipiteisiin asianosaisten keskuudessa ja nopeasti lamaannuttaa projektin.

Ajurit

- Aikataulupaineet voivat rohkaista ihmisiä aloittamaan järjestelmän kehityksen ennen aikojaan perustaen työnsä virheellisille oletuksille.
- Rakentajilla on luonnollinen taipumus laajentaa järjestelmän vaikutusalaa (esim. kuinka paljon tietoa pidetään päivitettynä tietokannassa jonkun toimesta, eli kuka tarvitsee kentän/taulun nyt ja mitä vaikutuksia, teknologisia ja organisaatiota koskevia, sillä voisi olla jos uudet kentät/taulut lisätään myöhemmin...).
- Asianosaisilla on kilpailevia näkemyksiä.
- Alaiset eivät tiedä mikä on projektin päämäärä.
- Ihmiset eivät kommunikoi.

Ratkaisu Valmistele järjestelmästä tarkoituslausuma, joka kuvailee selkeästi järjestelmän päämäärät ja tukee organisaation tehtävää. Jaa sitä vapaasti kaikille projektin osallisille.

- Järjestelmän päämäärät?
- Ongelmat, jotka järjestelmä tulee ratkaisemaan?
- Ongelmat, joita järjestelmä ei tule ratkaisemaan!
- Ketä ovat asianosaiset?
- Kuinka järjestelmä hyödyttää asianosaisia?

Esimerkkejä UCOT-projektisuunnitelma-0.04.pdf, luku 1.1/eka kappale

NäkyväRaja

Mitä tehdään *tässä projektissa* ja mitä ei? Kuinka paljon mikäkin työvaihe/tuotos maksaa asiakkaalle?

Ongelma Järjestelmän vaikutusala laajenee hallitsemattomasti jos sen rajoja ei tiedetä.

Ajurit

- Eri ihmisillä on eriäviä näkemyksiä järjestelmän rajoista.
- Huonosti määritellyt rajat johtavat järjestelmän vaikutusalan vaivihkaiseen laajenemiseen.
- Epätarkat ja ristiriitaiset päämäärät projektin alussa tekevät usein järjestelmän rajojen määrittämisen vaikeaksi.
- Ihmiset luulevat että rajojen määrittäminen on tarpeetonta.

Ratkaisu Perusta näkyvä raja järjestelmän ja ympäristön välille nimeämällä laitteistot ja ihmiset, jotka ovat järjestelmän kanssa vuorovaikutuksessa.

- **NäkyväRaja** rajoittaa ja tukee mallia **JaettuSelkeäVisio** määrittelemällä ne ulkoiset järjestelmät ja henkilöt joiden kanssa järjestelmän tulee tehdä yhteistyötä, perustamalla mallin **SelkeäRoolijako** ja määrittelemällä mitä resursseja järjestelmällä on saatavilla saavuttaaksen päämääränsä.

Esimerkkejä IT-prosessit/roolit: ulkoiset roolit ("yliopiston *"), yksiköt ja tietojärjestelmät osana prossien kulkuja

SelkeäRoolijako

Mitkä ovat eri asianosaisten roolit ja tarvittavat toimet ohjelmistonkehitysprosessin eri vaiheissa?

Ongelma Jos analysoimme vain järjestelmän käyttäjiä emmekä välitä eri rooleista joita heillä on järjestelmän suhteen voimme jättää huomaamatta tärkeää järjestelmän käyttäytymistä ja luoda turhaa päällekäistä käyttäytymistä.

Ajurit

- Jotta järjestelmä olisi hyödyllinen, on sen tyydytettävä käyttäjiensä tarpeet ja suojeltava asianosaistensa etuja.
- Palvelujen sitominen käyttäjiin voi tehdä järjestelmästä niin jäykän, ettei palveluita voida uudelleenkäyttää.
- Aihepiirin asiantuntijoiden kapea näkemys voi sumentaa käyttäjän tarvitsemat palvelut.
- Käyttäjiin keskittyminen voi saada meidät juuttumaan toteutuksen yksityiskohtiin käyttäjien tarvitsemien palveluiden ymmärtämisen sijaan.

- Useat ihmiset ymmärtävät helpommin konkreettisia kuin abstrakteja käsitteitä (iteratiivinen kehitys).
- Aikataulupaineet ja tarkoituksenmukaisuus rohkaisevat meitä analysoimaan vain käyttäjiä (Huom. todelliset loppukäyttäjät voi olla vaikea löytää...).

Ratkaisu Tunnista aktorit, joiden kanssa järjestelmän pitää olla vuorovaikutuksessa, sekä jokaisen aktorin rooli suhteessa järjestelmään. Kuvaile jokainen näistä selkeästi.

- Tunnista ja luettele palvelut, joita kukin käyttäjä vaatii järjestelmältä. Järjestä ne yhtenäisiin joukkoihin ja nimeä joukot.
- Anna jokaiselle aktorille nimeksi selkeä substantiivi tai substantiivilauseke.

Esimerkkejä IT-prosessit/roolit: alan professori, esimies, opiskelija ym. + prosessit, joissa he ovat mukana

KäyttäjälleArvokkaitaTapahtumia

Miksi järjestelmä kehitetään? Mikä käyttäjien saama toiminnallisuus tukee mallia **JaettuSelkeäVisio**?

Ongelma Järjestelmä on puutteellinen, jos se ei pysty välittämään käyttäjille hyödyllisiä palveluja ja se ei tue järjestelmän vision määrittämiä tavoitteita ja päämääriä.

Ajurit

- Käyttötapauksien joukon tulisi kiteyttää oleelliset lisäarvolliset palvelut, joita käyttäjät ja asianosaiset tarvitsevat järjestelmältä.
- On suhteellisen helppoa tunnistaa matalan tason (liike)toimia, mutta voi olla vaikeaa tunnistaa hyödyllisiä palveluja.
  CRUD ("CREATE, READ, UPDATE, DELETE") eli "Luo, lue, päivitä, poista" -tyyliset käyttötapaukset!!! Löydä syy/tavoite tekemiselle!
- Käyttötapauksien on oltava suhteellisen vakaita, koska ne muodostavat "ankkuripisteet" loppuosalle tuotteen kehitysprosessia (kts. liite Figure 5.1).
- Lukijat haluavat nähdä helposti, kuinka järjestelmä saavuttaa tavoitteensa.
- Ihmiset ovat taipuvaisia työskentelemään abstraktiotasolla, joka on joko liian korkealla tai liian matalalla ("mitä" vs. "kuinka").

Ratkaisu Tunnista hyödylliset palvelut, jotka järjestelmä välittää aktoreille tyydyttääkseen heidän (liike)toimintatarpeensa.

Esimerkkejä Matkatoimiston lähtöpalvelut

- Kirjaa matka
- Etsi lentoja

- Mainosta lomia
- Luo matkasuunnitelma
- Päivitä matkasuunnitelmaa
- Poista matkasuunnitelma

Muutettuna seuraaviksi:

- Kirjaa matka
- Muuta kirjausta
- Peruuta kirjaus
- Etsi lentoja
- Mainosta lomia

Huomaa että yhtälailla (liike)toimintaprosessien (ja ns. Business Use Casejen) tulisi kuvata **(Liike)ToiminnalleArvokkaitaTapahtumia.**

JatkuvastiAvautuvaTarina

Organisoi koodi ja koko dokumentaatio niin, että asianosaiset (liittyen valmisteltaviin tuotoksiin) voivat katsoa vaihetuotteita eri abstraktiotasoilta; **JatkuvastiAvautuvaTarina**isuus on hyvä periaate myös koko kehittämisprosessissa.

Ongelma Järjestelmän käyttäytymisen kuvaamiseen tarvittavien askelten määrä ylittää eri tyyppisten lukijoiden muistin sekä kiinnostuksen.

Ajurit

- Jokaisella käyttötapauksella voi olla useita eri lukijoita sekä useita eri käyttötapoja, jotka kaikki tarvitsevat eri määrän yksityiskohtaisuutta.
- Käyttötapauksien tulee näyttää, kuinka eri toiminnot antavat arvoa (liike)toiminnalle.
- Usealla eri tasolla kirjoittaminen (samanaikaisesti) on hämmentävää.
- Kirjoittajat tarvitsevat toimintaperiaatteen vaatimusten järjestämiseen.

Ratkaisu Järjestä käyttötapausjoukko hierarkkiseksi tarinaksi, joka voidaan joko avata, jotta saadaan lisää yksityiskohtaisuutta, tai taittaa kokoon, jotta yksityiskohdat piiloutuvat ja asiayhteys näkyy paremmin.

Esimerkkejä IT-prosessit: pääikkunan prosessihierarkkia

3 Kehitys: Prosessi

Prosessi määrittelee ohjelmistonkehityksen vaiheet ja toimet. Monet olemassaolevat (raskaat) prosessit (esim. UP, OMT++) eivät useissa tapauksissa ole kaikkein tehokkain tapa kehittää ohjelmistoa sinällään, mutta ne mahdollistavat koko kehittämisen funktionaalisen hallinnan ja esim. eri vaihetuotteiden alihankinnan kustannustehokkuuden lisäämiseksi (vrt. "Ohjelmointi menee Intiaan"). Ohjelmistokehityksessä ei ole olemassa yhtä ainoaa prosessimallia, joka sopisi kaikkeen ja jota käytettäisiin kaikkeen. Kaikilla elinkaarimalleilla on kuitenkin sama allaoleva looginen ongelma-aliongelma-aliratkaisu-ratkaisu rakenne, mutta näiden askelien järjestys ja mahdollisuus liikkua edestakaisin tässä rakenteessa vaihtelee.

Monet allaolevat mallit ovat suorassa suhteessa iteratiivisen ja inkrementaalisen kehityksen hyötyihin verrattuna suoraan yksisuuntaiseen menettelytapaan (joka voi silti olla ainoa mahdollisuus projektihallinnollisista syistä).

LaajuusEnnenSyvyyttä

Ongelma Jos energiaa hukataan yksityiskohtaisten käyttötapausten kirjoittamiseen peräkkäin, ei projekti etene oikea-aikaisesti emmekä pysty luomaan vakiintunutta käyttötapausjoukkoa.

Ajurit

- Vaatimusten kokoaminen on löytämisprosessi.
- Ihmiset tuntevat vetoa kirjoittaa yksityiskohtia aikaisessa vaiheessa.
- Ihmiset hukkaavat energiaa tai jumiutuvat liikoihin yksityiskohtiin.
- On hyödyllistä saada yleiskuva aikaisessa vaiheessa.
- Mitä yksityiskohtaisempia ovat aikaiset ponnistelut sitä enemmän niitä täytyy muuttaa, kun järjestelmää ymmärretään paremmin.

Ratkaisu Säästä energiaa kehittämällä ensin yleiskuva käyttötapauksista ja lisää sitten yksityiskohtia progressiivisesti, työskennellen toisiinsa liittyvien käyttötapausten parissa.

- Tunnista kandidaatit käyttötapauksiksi liittämällä merkityksellinen tavoite jokaiselle aktorille. Lopputuloksena seuraava tavoitteiden joukko antaa oleellisen ja jaetun ymmärtämyksen asianosaisille ja auttaa vähentämään myöhemmin vaadittavan refaktoroinnin määrää.
- Tunnista tärkeimmät käyttötapaukset (esim. tärkeimpään loppukäyttäjään, suurimpaan kehitysriskiin, avain (liike)toimintaprosessiin tai ohjelmistoarkkitehtuuriin liittyvät), jotka ajavat kehitystä eteenpäin ja jotka tulisi viimeistellä ensimmäisenä.
- "Valmistaudu heittämään yksi pois... koska heität joka tapauksessa" (Brooks, 1995) (Pätee kaikille vaihetuotteiden ja tuotoksien osille).
- Tikku-ukot ja ellipsit, eli UML:n käyttötapauskaavion malli, ei ole käyttötapauksien tarkoitus, mutta se antaa asianosaisille hyvän yleiskuvan perustoiminnallisuudesta/-palveluista. Lisäksi se jäsentää selkeästi järjestelmän toiminnallisuutta eri käyttäjäroolien perspektiiveistä (tätä tietoa tarvitaan käyttöliittymiä suunniteltaessa).

Esimerkkejä Joskus syvyyttä ei tarvita lainkaan, vaan sovellus voidaan kehittää jo yleisellä tasolla kuvattujen käyttötapausten perusteella (vrt. UCOT).

SyklinenKehitys

Ongelma Käyttötapauksien kehittäminen yhdellä kertaa on vaikeaa ja uuden tiedon lisääminen niihin voi tulla kalliiksi. Mikä pahempaa, tämä voi viivästyttää projektin riskitekijöiden löytämistä.

Ajurit

- Järjestelmän käyttäytymisen ymmärtäminen voi viedä pitkän ajan.
- Viivästykset (missä tahansa vaiheessa tai toimessa) ovat kalliita.
- *Vaatimukset (todennäköisesti) muuttuvat* kun niitä tutkitaan (samoin arkkitehtuuri, koodi, testitapaukset, ylläpitotarpeet ja ...).
- Vaatimusvirheiden hinta on korkea.
- Uppoutumalla liian nopeasti masennat muut (Jos tuotostasi ei jaeta ja ymmärretä muiden asianosaisten kesken, joiden se pitäisi ymmärtää (kehitysrajapinnat), niin punainen lanka katkeaa).

Ratkaisu Kehitä käyttötapaukset iteratiivisella laajuus-ensin tavalla, jolloin jokainen iteraatio parantaa käyttötapausjoukon tarkkuutta ja paikkansapitävyyttä[1] progressiivisesti.

- Mallia **LaajuusEnnenSyvyyttä** sovellettaessa yritä vakauttaa aktorien ja näiden tavoitteiden lista, jota voidaan käyttää esim. projektisuunnitelman perustamiseen, työmäärän arviointiin, kehitysryhmien perustamiseen, jne.
- Mallin **LaajuusEnnenSyvyyttä** kanssa valitse työstettävä osajoukko käyttötapauksista (tai vain yksi) laajennettavaksi ja lisää pääskenaariot järjestelmän tarkoituksen tarkastelua varten. Monissa tapauksessa tämä luo uusia käyttötapauksia.
- Käyttötapauksien joukko voidaan ryhmitellä esim. aktorien, samankaltaisen toiminnallisuuden tai abstraktiotason mukaan.

Esimerkkejä IT-prosessit: prosessikuvauksien tilat

MoniaLomakkeita

Seuraava malli yleistyy parhaiten kun vaihtaa sanan "lomake" sanaksi "tekniikka" ja sanan "pohja" sanaksi "työkalu"!

Ongelma Eri projektit tarvitsevat eri määrän formaaliutta työssään ja useilla henkilöillä on eri mieltymykset pohjan suhteen. Kaikkien vaatiminen käyttämään samaa käyttötapauspohjaa on haitallista.

[1] Huom! Tarkkuus eli *precision* tarkoittaa "kuinka paljon asiasta sanotaan", kun taas paikkansapitävyys eli *accuracy* tarkoittaa "Kuinka oikeassa ollaan kun asiasta sanotaan niin paljon" (vrt. kuva 1).

Ajurit

- Ihmiset ja projektit ovat erilaisia.
- Eri tiimit haluavat (ja tarvitsevat) eri verran formaaliutta.
- Yhteisen lomakkeen käyttäminen organisaatiossa edistää kommunikaatiota.

Ratkaisu Valitse formaatti, joka perustuu projektin riskeihin ja mukana olevien ihmisten mieltymyksiin.

- Yksi pohja (= valitut kentät eri mahdollisuuksien luettelosta, jonka määrittelee organisaatio) yhdelle projektille.

Esimerkkejä *http://alistair.cockburn.us/index.php/Basic_use_case_template*

KaksikerrosKatselmukset

Kaikkien projektitapaamisten, katselmointien jne. pitäisi olla merkityksellisiä kaikille osallistujille ja niiden tulisi edistää projektia saavuttamalla tapaamisen ennaltamäärätty tavoite.

Ongelma Useilla henkilöillä saattaa olla tarve katselmoida käyttötapauksia. Tämä on kallis ja aikaavievä toimenpide.

- Katselmointeja tarvitaan kirjoituksen ja sisällön verifioimiseksi ja validoimiseksi.
- Asianosaisilla on (erilaisia) näkökantoja käyttötapauksiin.
- Kaikkien mukaan ottaminen kirjoittamisprosessiin on kallista ja väsyttävää.
- Jos vain pieni kirjoitusryhmä tekee katselmoinnin, niin kaikkien asianosaisten näkökantoja ei oteta huomioon.

Ratkaisu Pidä kahdentyyppisiä katselmointeja: sisäisiä läpikäyntejä mahdollisesti monta kertaa, koko asianosaisryhmällä lähtökohtaisesti vain kerran.

- Katselmoi sisäisesti luettavuutta, toteutettavuutta, tarkkuutta ja paikkansapitävyyttä (Huom! tarkkuus $\neq$ paikkansapitävyys).
- "Ulompien" katselmointien päätarkoitus on harkita
  - Onko tämä oikea asia, jonka rakentamiseen kehittäjien tulisi käyttää aikansa? ((liike)toiminta-arvon tarkistus)
  - Onko tämä oikein määrityksenä? ((liike)toimintasäännöt, avoimuus eri toteutuksille (ja (G)UI:t), tärkeiden päätösten selkeys (esim. tehtäväkuvan muuttuminen organisaation sisällä), avoimien kysymysten dokumentointi, jne.)
  - Pystyvätkö kehittäjät todella rakentamaan sen? (riippuen tilanteesta (esim. sopimus), annetulla ajalla ja resursseilla)

Esimerkkejä IT-prosessit: työmuodot ja esim. virantäyttöprosessit (Hallinnon ja johtamisen prosessit)

AikaLopettaa

Koska lopetat tämänhetkisen vaihetuotteen tai tuotoksen ja siirryt seuraavaan tehtävään projektin viemiseksi eteenpäin? Tiedätkö ja uskallatko kertoa muille, että et pysty toimittamaan omaa osaasi projektista ajoissa? (Jotta voit jättää tai muuttaa sen ajoissa, vrt. CHAOS-raportit)

"Projektin hallitseminen yli asianosaisten ja kehittäjien tarpeiden hukkaa resursseja ja viivästyttää projektia."

Ongelma Käyttötapausmallien kehittäminen yli asianosaisten ja kehittäjien tarpeiden hukkaa resursseja ja viivästyttää projektia.

Ajurit

- Tärkeiden vaatimusten huomaamatta jättämisen pelko rohkaisee kehittäjiä ja asianosaisia pidentämään vaatimusten keräämiseen liittyviä toimenpiteitä.
- Monet tekniset ammattilaiset asettavat tarpeettoman korkean prioriteetin mallin formaaliudelle.
- Moniselitteisyys voi tuhota projektin.
- Useimmat ihmiset pystyvät työskentelemään kohtuullisen moniselitteisyyden kanssa.
- Vaatimusten liian tarkka määrittely voi vakuuttaa asianosaiset siitä, että vaatimukset ovat tarkempia kuin oikeasti ovatkaan (vrt. validointi ja verifiointi)
- Virheen kustannukset ovat kohtuullisen pienet, jos se havaitaan aikaisin. Paikalleen jäämisen kustannukset ovat kohtuuttomat.

Ratkaisu Lopeta käyttötapausten kehittäminen, kun ne ovat valmiita ja täyttävät yleisön vaatimukset.

- Oletko tunnistanut ja dokumentoinut kaikki aktorit ja tavoitteet?
- Onko asiakas tai asiakkaan edustaja myöntänyt, että käyttötapausjoukko on valmis ja että jokainen käyttötapaus on luettava ja oikeellinen.
- Pystyvätkö suunnittelijat ja kehittäjät toteuttamaan nämä käyttötapaukset?

Esimerkkejä UCOT-kayttotapauksia-0.02.pdf.

KirjoittajanLisenssi

"Liiallinen tyylikysymysten painotus - koskien kaikkia vaihetuotteita - haittaa tarpeettomasti projektin viemistä eteenpäin."

Ongelma Liiallinen tyylikysymysten painotus haittaa tarpeettomasti käyttötapausten kirjoittamista.

Ajurit

- Haluaisimme yhtenäisen kirjoitustyylin lukemisen helpottamiseksi.
- On kallista ja epäkäytännöllistä saada kaikki kirjoittamaan juuri samalla tyylillä.
- (Oikeiden) Käyttötapausten saaminen aikaisemmin kehitykseen on arvokasta.
- Käyttötapauksien on silti oltava luettavia, loogisesti oikeita ja tarpeeksi yksiyiskohtaisia kehittäjille.

Ratkaisu Kirjoitustyyleissä on väistämättäkin pieniä eroja. Kun käyttötapaus läpäiseen mallin **AikaLopettaa** testin, kirjoittaja voi vaatia "kirjoittajan lisenssin" koskien pieniä tyylillisiä eroja.

- Jokainen käyttötapaus noudattaa organisaation kirjoittamispohjaa ja perustyyliä, on loogisesti oikea, on luettava loppuarvostelijoille ja on tarpeeksi tarkka toteutusta varten.

Esimerkkejä IT-prosessit: eri aikaan ja eri henkilöiden toimesta tuotetut prosessikuvaukset

4 Rakenne: Käyttötapaus

SaavutaYksiTavoite

Seuraava havainto ohjaa myös ammattimaista lähestymistapaa koskien yksittäistä kehitysvaihetta tai tointa: sillä tulisi olla esim. selkeä ja testattava tavoite, jotta tiedetään koska pitää lopettaa.

Ongelma Sopimattomat tavoitteet jättävät kirjoittajat epävarmoiksi siitä, mihin yksi käyttötapaus loppuu ja mistä toinen alkaa.

- Oikean järjestelmän rakentamiseksi pitää ymmärtää, kuinka aktori pääsee lopputulokseen jolla on arvoa.
- Haluamme hallita monimutkaisuutta jakamalla käyttötapaukset osiin, jotka ovat asianosaisille järkeviä.
- Aktorin tavoitteiden kesto saattaa vaihdella (vrt. laadulliset vaatimukset).
- Liian suuret käyttötapaukset saattavat haudata asianosaiset yksityiskohtiin ja hämärtää käyttötapauksen tarkoituksen.
- Suuret käyttötapaukset voivat haitata uudelleenkäyttöä (vrt. jäljitettävyys).
- Liian pienet käyttötapaukset kuvaavat vain osan jostakin arvokkaasta tavoitteesta.
- Aihepiirin asiantuntijoiden kapea tietämys voi johtaa kohti pieniä käyttötapauksia.

Ratkaisu Kirjoita yksi käyttötapaus kutakin hyvin määriteltyä tavoitetta kohti. Tavoite voi olla millä tahansa tasolla mallissa **JatkuvastiAvautuvaTarina**.

- Tavoite tulisi liittää hyvin määriteltyyn aktoriin.

- Tavoitteen pitäisi olla arvokas aktorille tai asianosaiselle, jonka puolesta aktori työskentelee.
- Tavoitteen pitäisi olla johdonmukainen muiden samalla abstraktiotasolla olevien järjestelmän tavoitteiden kanssa.

Esimerkkejä IT-prosessit: Hallinnon ja johtamisen prosessit: virantäyttö
(hmm... ei ehkä niin helppoa saavuttaa käytännössä ("tavoitteen"+"abstraktiotason" täsmällinen ymmärtäminen yhdessä)

VerbiFraasiNimenä

Hyvät nimeämistavat koskien kaikkia artefakteja vaatimusdokumenteista luokkiin ja muuttujien nimiin itse koodissa (Huom. koskien niiden roolia osana ratkaisua, ei ominaisuuksina jotka seuraavat suoraan kontekstista (esim. tyyppi)) parantavat ja selventävät kehitystiimin keskinäistä ja asianosaisten välistä kommunikaatiota.

Ongelma Merkityksettömät geneeriset nimet eivät aseta lukijalle odotuksia eivätkä anna tarkoituksenmukaista referenssipistettä.

Ajurit

- Nimi asettaa sävyn ja mielleyhtymän yleisölle ja antaa kirjoittajalle lähtökohdan.
- Sopiva nimi luo kahvan käyttötapaukseen.
- Sopivat käyttötapauksien nimet auttavat näkemään kokonaisuuden ja työskentelemään koko joukon parissa.

Ratkaisu Nimeä käyttötapaus aktiiviverbilauseella joka edustaa pääaktorin tavoitetta.

Esimerkkejä Pelottavia esimerkkejä

- Main Use Case = "Pääkäyttötapaus"
- Claim Process = "Hakemusprosessi"
- Use Case 2 = "Käyttötapaus 2"
- Process stuff = "Prosessoi jotain"

...ja parempia

- File Accident Claim = "Lähetä vahinkoilmoitus"
- Approve Property Damage Claim = "Hyväksy vahingonkorvausvaatimus"
- Report Fraudulent Claim to Police = "Ilmoita vilpillinen vaatimus poliisille"

SkenaarioJaTäydennykset

Keskity nykyiseen iteraatioon suorittamalla (vain) ne toiminnot ja valmistelemalla (vain) ne vaihetuotteet, jotka on osoitettu kyseiseen iteraatioon. Validoi ja verifioi työ tarkoituksenmukaisesti. Suunnittele ja yritä vakauttaa ohjelmiston perusarkkitehtuuri (järjestelmä, alijärjestelmä, moduuli, paketti, jne.) mahdollisimman nopeasti. Lisää ominaisuuksia tämän rungon päälle myöhemmissä iteraatioissa (ja/tai järjestä uudelleen kussakin iteraatiossa kuten ketterissä menetelmissä...).

Ongelma Lukijoiden pitää pystyä seuraamaan helposti läpi koko skenaarion tai tarinan, joka heitä kiinnostaa. Muussa tapauksessa he turhautuvat helposti tai tärkeää tietoa jää huomaamatta.

Ajurit

- Mielenkiintoisen käyttötapauksen pitää esittää vaihtoehtoja pääskenaariolle.
- Jokaisen vaihtoehdon kirjoittaminen kokonaisena tarinana hämärtää tarinamuunnelmien välisiä eroja.
- Jokaisen muunnelman erotteleminen vaikeuttaa kirjoittajan elämää.
- Suuri määrä *if*-lauseita sekoittaa tarinan.
- Ihmiset selviävät hyvin inkrementaalisesta monimutkaisuudesta.
- Pääskenaario täytyy tunnistaa selkeästi.

Ratkaisu Kirjoita onnistumistarinan juoni yksinkertaisena skenaariona harkitsematta epäonnistumista. Sijoita tämän alle tarinan katkelmia, jotka kertovat mitä eri vaihtoehtoja voi tapahtua.

- Pääskenaario kuvailee, kuinka pääaktori saavuttaa tavoitteen suoraviivaisella tavalla.
- Mallin **LaajuusEnnenSyvyyttä** (kts. myös liitteen "The Writing Process") mukaisesti pysähtymispisteet käyttötapauksen valmistuksessa ovat: pääskenaario, ehtojen nimeäminen (jotka voivat aiheuttaa poikkeuksen) ja joidenkin tarinan katkelmien viimeistely (poikkeuksien käsittelyä varten).

Esimerkkejä IT-prosessit: prosessikuvausten rakenteet

EsitäVaihtoehdot

Seuraava ei ole välttämättä käytännössä mahdollista, mutta yritä tehdä parhaasi joka tapauksessa (ajan, resurssien ja terveen järjen rajoissa). Esim. liian ennaltaehkäisevä koodi ei ole muuta kuin sisäkkäistä poikkeusten hallintaa, josta logiikka on vaikeasti löydettävissä.

Ongelma Käyttötapauksella voi olla monia vaihtoehtoja. Joidenkin vaihtoehtojen huomaamatta jättämisen takia kehittäjät ymmärtävät järjestelmän käyttäytymisen väärin ja järjestelmästä tulee puutteellinen.

Ajurit

- Kehittäjien pitää tietää kuinka käsitellä (sekä käyttö- (käyttäjä) että teknologisia (järjestelmä-)) virheitä.
- Aikataulupaineet rajoittavat aikaa, jonka kehittäjät voivat käyttää eri muunnelmien tunnistamiseen.
- Jotkut muunnelmien käsittelyn menettelytavat vaativat huomattavaa tutkimusta.
- Tieto eri muunnelmista auttaa kehittäjiä rakentamaan vankan järjestelmän.

Ratkaisu Etsi kaikki vaihtoehdot ja epäonnistumiset, jotka pitää käsitellä käyttötapauksessa.

- Käytä mallia **TodennettavatEhdot** löytääksesi vaihtoehdot, jotka johtuvat käyttäjän tai järjestelmän vaihtoehtoisesta käyttäytymisestä.
- Asteittainen tarkennus: vaihtoehtoisten tilojen läpikäynti ensin ja vasta sen jälkeen (loppuunsaatettu) poikkeusten käsittely. (vrt. tarkoituksenmukaiset ohjelmoinnin menettelytavat...)

Esimerkkejä IT-prosessit: Opintoasioiden prosessit: opiskelijavalinnat

Täydennykset

Vaihetuotteiden iteratiiviseen ja inkrementaaliseen ohjaamiseen tarvitaan laajennettavia rakenteellisia ajureita.

Ongelma Ei-toiminnallisten vaatimusten ottaminen mukaan käyttötapauksiin voi nopeasti hämärtää ja sekoittaa käyttötapauksen.

Ajurit

- Käyttötapauksien tarkoitus on esittää selkeästi järjestelmän toiminnalliset vaatimukset.
- Toiminnallisia vaatimuksia tutkittaessa löytää useasti käyttäytymisestä riippumatonta tietoa.
- Ei-toiminnallisten vaatimuksien ottaminen mukaan käyttötapaukseen on harhaanjohtavaa (esim. käyttöliittymän yksityiskohdat, jotka ovat jo "miten"-maailmassa...)
- Emme halua kuitenkaan menettää tietoa, joka auttaa käyttötapauksen ymmärtämisessä tai on tarpeellista kehittäjille.

Ratkaisu Luo ylimääräisiä skenaariotekstin ulkopuolisia kenttiä käyttötapauspohjaan, jotka sisältävät täydentävää tietoa joka on hyödyllistä liittää käyttötapaukseen.

Esimerkkejä IT-prosessit: esim. Rajoitteet-kenttä (+ delegointi)

TarkkaJaLuettava

Jos yksi vaihe tai toimi iteratiivisessa kehittämisessä hallitsee prosessia (esim. hämmentävä pohja ilman mallia **SelkeäRoolijako** tai liian innokas kirjoittaja, joka tuottaa satoja toiminnallisia kuvauksia ilman abstraktiotasoja, jaettua rakennetta tai assosiaatioita), vakaata kehitystä on mahdoton saavuttaa ja laadun ja aikataulun hallinta on mahdotonta ("kehittäminen osissa").

Projektitiimillä pitäisi olla selkeä näkemys siitä, ketkä asianosaiset ovat kiinnostuneita mistäkin vaihetuotteesta (vrt. käytetty notaatio, esim. UML vs. asiakas, käyttäjä, johtaja, kehittäjä... Kenelle piirrät kaavioita? Kuka kustantaa ne? Miksi?)

Ongelma Käyttötapaukset, jotka ovat liian monimutkaisia ei-teknisille lukijoille tai liian epätarkkoja kehittäjille, ovat puuttellisia ja johtavat todennäköisesti huonosti rakennettuun ja puutteelliseen järjestelmään.

Ajurit

- Käyttötapauksen pitäisi olla luettava sekä asianosaisille että kehittäjille.
- Kehittäjät ovat taipuvaisia lisäämään yksityiskohtia ja ratkaisuja (esim. käyttötapauksiin liittyvät koodin komponenttien välistä yhteistoimintaa kuvaavat sekvenssikaaviot)
- Ei-tekniset asianosaiset eivät todennäköisesti huomaa tarvittavia asioita.
- Haluamme rohkaista asianosaisten ja kehittäjien välistä dialogia vaatimuksien paikkansapitävyyden takaamiseksi (**OsallistuvaYleisö**).
- Saman asian kuvaaminen kahdella eri tavalla, asiakkaille ja kehittäjille, ei ole järkevää, koska tällöin joudumme ylläpitämään useita kuvauksia samalle asialle (vrt. formaalit menetelmät ja tekstit+kaaviot; toisaalta, jos tämä tehdään automaattisesti kuten IT-prosesseissa asia ei ole ongelma).

Ratkaisu Kirjoita käyttötapauksesta tarpeeksi luettava, jotta asianosaiset vaivautuvat lukemaan ja arvioimaan sen, ja tarpeeksi tarkka, jotta kehittäjät ymmärtävät mitä ovat rakentamassa.

- "Tunne yleisösi" (pätee kaikkiin vaihetuotteisiin)

Esimerkkejä IT-prosessit: Hallinnon ja johtamisen prosessit: virantäytöt

5 Rakenne: Skenaario

TodennettavatEhdot

Seuraavia periaatteita voidaan miettiä rajapintoja mietittäessä ja refaktorointia harkitessa.

Ongelma Kehittäjät painivat aina sen kanssa, kuinka monta tilaa toteutukseen tulisi ottaa mukaan ja mitkä nämä ovat.

Ajurit

- Järjestelmä ei voi käsitellä tapahtumia joita se ei havaitse.
- Kehittäjien pitää tietää mitkä tilat tulisi havaita.
- Pelko tärkeän vaihtoehdon huomaamatta jättämisestä rohkaisee kehittäjiä määrittelemään merkityksettömiä tiloja, joita järjestelmä ei voi havaita.
- Monta näennäisesti erilaista tilaa voi johtaa tarpeettomien vaihtoehtojen kirjoittamiseen.

Ratkaisu Ota mukaan vain havaittavat tilat. Yhdistä tilat, joilla on sama nettovaikutus järjestelmään.

- Jokaisen käyttötapauksen skenaarion *on alettava* toiminnalla, jonka järjestelmä pystyy havaitsemaan.

Esimerkkejä IT-prosessit: Hallinnon ja johtamisen prosessit: Dosentuurin myöntäminen

SamantasoisetAskeleet

Seuraava pätee myös hyvälle algoritmille (eli ohjelmalogiikan jako ali- ja ylirakenteen välillä).

Ongelma Tarpeettoman suuri tai pieni käyttötapaus hämärtää tavoitteen ja tekee käyttötapauksesta vaikeasti luettavan ja tajuttavan.

Ajurit

- Tarpeettoman pienet askeleet tekevät käyttötapauksesta pitkän ja vaikeasti luettavan ja hämärtävät “miksi?”-kysymyksen.
- Tarpeettoman isot askeleet voivat piilottaa tärkeää käyttäytymistä.
- Yksityiskohtaisuustason vaihteleminen skenaariossa on harhauttavaa.

Ratkaisu Pidä skenaarioissa kolmesta yhdeksään askelta (7 ± 2-periaate). Ihanteellisesti askeleet ovat kaikki vastaavilla tasoilla ja abstraktiotasolla joka on juuri käyttötapauksen tavoitteen alapuolella.

Esimerkkejä IT-prosessit: IT-tiedekunnan ATK-tuen asiakasrajapinnan prosessit: Atk-hankintojen käsittely

6 Rakenne: Askel

Kuka tekee *Mitä* saavuttaakseen *Minkä tavoitteen*?

KäyttajänTavoiteSaavutettu

Ongelma Sekä lukijat että kehittäjät hämmentyvät järjestelmän käyttäytymisestä, jos ei ole selvää, mikä aktori on vastuussa askeleen suorittamisesta ja mitä aktori yrittää saavuttaa kyseisessä askeleessa.

Ajurit

- Laadukkaiden käyttötapausten kirjoittaminen on kovaa työtä; hyvän (rakenteisen) proosan luominen vie paljon henkistä energiaa.
- Kehittäjien täytyy tietää selkeästi, mitä heidän pitää toteuttaa, minä hetkinä järjestelmän pitäisi odottaa syötettä ja milloin järjestelmän pitäisi tehdä aloite.

Ratkaisu Kirjoita jokainen askel näyttääksesi selkeästi, mikä aktori toimii ja mitä aktori saa aikaiseksi.

Esimerkkejä IT-prosessit: Hallinnon ja johtamisen prosessit: 1.3.1.1 Lomailmoituksen hyväksyminen - TDK

EteenpäinMenevästi

Eliminoi tai yhdistä toimet, jotka eivät edistä projektia. Yksinkertaista toimia, jotka häiritsevät kehityksen kontrollointia ja hallintaa.

Ongelma Kirjoittajien pitää päättää kuinka paljon käyttäytymistä yhteen askeleeseen laitetaan. He kirjoittavat helposti liikaa yksityiskohtia, jolloin käyttötapauksesta tulee pitkä ja väsyttävä luettava.

Ajurit

- Selkeät ja ytimekkäät askeleet alentavat käyttötapauksen ymmärtämiseen ja arvioimiseen tarvittavaa henkistä kustannusta.
- Kokonaisuuteen ja yksityiskohtaisuuteen pyrkiminen voi johtaa käyttötapauksen tavoitteen suhteen sivuavien askelten ottamiseen mukaan.

Ratkaisu Eliminoi tai yhdistä askeleet jotka eivät vie aktoria eteenpäin. Yksinkertaista kohtia, jotka harhauttavat lukijaa tästä kehityksestä.

Esimerkkejä IT-prosessit: Hallinnon ja johtamisen prosessit: 1.3.1.1 Lomailmoituksen hyväksyminen - TDK

TeknologisestiNeutraali

Käyttäjät ovat kiinnostuneita sovelluksista, jotka edistävät ja helpottavat heidän (liike)toimiaan, eivät alla olevasta teknologiasta. Kehityksen aikana käytettävän teknologian on kuitenkin oltava määrätty, mutta tämä ei ole osa toiminnallisia vaatimuksia.

Ongelma Teknologiarajoitteiden ja toteutusyksityiskohtien ottaminen mukaan käyttötapauksen kuvaukseen lisää monimutkaisuutta ja hämärtää käyttötapauksen tavoitteen.

Ajurit

- Monet ihmiset haluavat ajatella konkreettisilla termeillä.
- Teknologia on ailahtelevaa. Tiettyjen teknologioiden yksityiskohtien mukaan ottaminen aiheuttaa muokkausta.
- Teknologiset yksityiskohdat määräävät sopimattomia rajoitteita tuleville toimille.
- Teknologisten yksityiskohtien lisääminen kasvattaa käyttötapausten lukemisen ja kirjoittamisen kustannuksia.
- Joskus teknologia on vaatimus (esim. SuD:lle olemassa olevan teknologisen ympäristön vuoksi)

Ratkaisu Kirjoita jokainen askel tavalla, joka on neutraali teknologian suhteen.

Esimerkkejä "Horror story": IT-prosessit: Opintoasioiden prosessit: Prosessi 1.4.8.4.1 (Kandidaattiopintojen eHOPSin tekeminen - perusvalinnan kautta valitut - TTL)

7 Rakenne: Käyttötapaussuhteet

Seuraavat mallit havainnollistavat hyvin kehitystä, joka on yleensä havaittavissa myös erilaisten ohjelmistonkehitysmallien kehittymisessä. Jossain vaiheessa eri asioita (lähinnä vaihetuotteita) aletaan abstrahoimaan ja rajapintoja parametrisoimaan uudelleenkäytön edistämiseksi. Tässä tätä sovelletaan järjestelmän toiminnallisuuden rakenteisiin kuvauksiin.

YhteinenOsaKäyttäytyminen

Ongelma Saman askeleen kirjoittaminen eri käyttötapauksiin on tuhlaavaista ja tekee käyttötapausmallin yhteisten aliprosessien näkemisen vaikeammaksi.

Ajurit

- Yleisten askelten uudelleenkirjoittaminen on tarpeetonta ja kasvattaa epäjohdonmukaisuuden ja paikkansapitämättömyyden riskiä mallissa.
- Yksittäisten käyttötapauksien jakaminen sirottelee tärkeää käyttäytymistä ja tekee niistä vaikeammin ymmärrettäviä.
- *includes*-suhteen väärin ymmärtäminen johtaa väärinkäyttöön.

Ratkaisu Ilmaise jaetut toiminnot alempitasoisilla, sisällytetyillä käyttötapauksilla.

- Ideaalitapauksessa, luo *includes*-suhde käyttötapauksien välille kun kaksi tai useampi käyttötapaus jakaa yhteisen askeljoukon (esim. käyttäjän validointi).
- UML:n mukaan *included* käyttäytyminen suoritetaan *aina*.

Esimerkkejä Liite: Use Case 7.1&7.2 ja Figure 7.1

KeskeytyksetLaajennuksina

Ongelma Vaihtoehto, joka vaikuttaa useampaan kuin yhteen askeleeseen skenaariossa, voi sirotella yksityiskohtia ympäri käyttötapausta, aiheuttaen hämmennystä lukijalle ja tärkeän tiedon hukkaamista.

Ajurit

- Useat poikkeamat ja vaihtoehdot, jotka keskeyttävät useita askeleita skenaariossa, voivat saada lukijat hukkaamaan polun jota he yrittävät seurata ja tämä tarkoittaa ongelmia itse perusskenaarion ymmärtämisessä.
- Laajennuskäyttötapauksien luomisella on taipumusta sirotella tärkeää käyttäytymistä ja tehdä siitä vaikeammin hahmotettavaa.
- *extends*-suhteen väärin ymmärtäminen johtaa väärinkäyttöön (historiallisesti *extends* on yhdistetty perintään ja täten erikoistamiseen, kuitenkin UML:ssä *extends* tarkoittaa riippuvuutta).

Ratkaisu Luo laajennuskäyttötapaus kun vaihtoehtoinen toiminto keskeyttää usean askeleen skenaariossa.

Esimerkkejä Liite: Use Case 7.3 ja 7.4.

YlennettyVaihtoehto

Ongelma Pitkät ja monimutkaiset vaihtoehdot voivat hallita käyttötapausta ja ne voivat vaikuttaa tärkeämmiltä kuin oikeasti ovatkaan, koska ne ovat niin silmäänpistäviä.

Ajurit

- Monimutkaiset ja pitkät vaihtoehdot sekoittavat käyttötapauksen ja voivat hämärtää muita vaihtoehtoja.
- Jotkut ongelmat ovat todella monimutkaisia ja niiden riittävään kuvailuun tarvitaan monimutkaisia käyttötapauksia.
- Yksittäisten käyttötapausten jakamisella on tapana sirotella tärkeää käyttäytymistä ja tehdä siitä vaikeammin ymmärrettävää.

Ratkaisu Harkitse monimutkaisten, liikaa käyttötapausta hallitsevien vaihtoehtojen siirtämistä erilliseen käyttötapaukseen.

- **SkenaarioJaTäydennykset**
- vrt. **JatkuvastiAvautuvaTarina**, **SamantasoisetAskeleet**, **SaavutaYksiTavoite** ja **JaaVaurausUudelleen**

Esimerkkejä IT-prosessit: metaprosessit, jotka "peritään" ja tarkennetaan

KaapattuAbstraktio

Ongelma On vaikeaa ja hämmentävää yrittää dokumentoida samassa käyttötapauksessa kahta tai useampaa erillistä vaihtoehtoa, joista kumpikaan ei ole hallitseva.

Ajurit

- Monimutkaiset ja pitkät vaihtoehdot sekoittavat käyttötapauksen ja hämärtävät muita vaihtoehtoja.
- Jotkut ongelmat ovat hyvin monimutkaisia ja niillä on useita erillisiä ratkaisuja (hyvinkin totta ohjelmistoille yleensä...)
- Useiden yksittäisten käyttötapausten pitäminen sirottelee helposti tärkeän käyttäytymisen ja tekee käyttäytymisen vaikeammin ymmärrettäväksi.

Ratkaisu Harkitse abstraktin yleistetyn käyttötapauksen luomista. Laita jokainen erillinen vaihtoehtoskenaario, joka erikoistaa abstraktioita, omaan erikoistuneeseen käyttötapaukseensa.

Esimerkkejä Liite: Figure 7.13.

Huomaa, että kaikilla yo. rakenteilla on vastineensa koodin jäsentämisessä. Tämä havainnollistaa, kuinka formaalimmat ja täsmällisemmät tavat työskennellä vaihetuotteiden kanssa "kuinka"-maailmassa hiipivät (lähemmäksi ja lähemmäksi) kohti vakiintuneita "mitä"-maailman periaatteita (puhumattakaan formaaleista määrittelymenetelmistä).

8 Kehitys: Muokkaus

Käyttötapauksen muodon muokkauksella on hämmästyttävän paljon yhtäläisyyksiä koodin rakenteen uudelleen harkinnan ja refaktoroinnin kanssa. Myös kehitysprojektin dokumentaation sisältö ja muoto yleisesti nostattaa mielenkiintoisia ajatuksia, kun niitä vertaa tämän kappaleen malleihin.

JaaVaurausUudelleen

Ongelma Tarpeettoman pitkä käyttötapaus on hankalasti käsiteltävä ja vaikea käyttää, ja se harhauttaa ja herpaannuttaa käyttäjien keskittymisen.

Ajurit

- Käyttötapauksen tavoitteen on oltava selkeä asianosaisille.
- Uusien käyttötapauksien lisääminen on kallista.
- Liiallinen yksityiskohtaisuus tekee käyttötapauksesta vaikeasti luettavan ja ymmärrettävän.

Ratkaisu Siirrä pitkä vaikeasti käsiteltävä osa tai liian monimutkainen laajennos omaan käyttötapaukseensa.

- **SaavutaYksiTavoite**, tarvittaessa **YhdistäPalaset**, siirrä katkelmat toisiin käyttötapauksiin tai täydentävään spesifikaatioon tai luo uusi (mahdollisesti alemman tason) käyttötapaus.

Esimerkkejä IT-prosessit: metaprosessit

YhdistäPalaset

Problem Käyttötapaukset, jotka kuvaavat vain pikkuriikkisiä tai eristettyjä käyttäytymisen katkelmia, eivät välitä tarpeeksi informaatiota auttaakseen lukijoita ymmärtämään kuinka järjestelmä välittää **KäyttäjälleArvokkaitaTapahtumia**.

Ajurit

- Osittaiset käyttötapaukset ovat epätäydellisiä, eivätkä ne onnistu kertomaan koko tarinaa.
- Informaatio on parasta sijoittaa vain yhteen tiettyyn paikkaan (jos mahdollista).
- (Organisaatiot usein veloittavat ja kehittäjät usein toteuttavat järjestelmän käyttötapauksien perusteella)
- Käyttötapauksien määrä tulisi minimoida.
- Pienemmät käyttötapaukset ovat helpompia ymmärtää ja käyttää.

Ratkaisu Yhdistä toisiinsa liittyvät pikkuriikkiset käyttötapaukset tai käyttötapauksen katkelmat käyttötapauksiin, joilla on sama tavoite.

Esimerkkejä Liite: Use Case 8.4 ja 8.5.

TyhjennäTalo

Seuraavia ohjeita voisi ja pitäisi käyttää myös tarpeettomaan koodiin, käyttämättömään toiminnallisuuteen (käytä logia nähdäksesi kuinka paljon mitäkin toimintoa kukin käyttää), tarpeettomaan dataan tietokannassa, jne.

Ongelma Käyttötapaukset, jotka eivät tuo arvoa kokonaisuudelle, ovat harhauttavia ja voivat johtaa yleisön väärälle polulle.

Ajurit

- Vanhojen käyttötapauksien siivoaminen vie aikaa ja “älyenergiaa”.
- Käyttämättömät käyttötapaukset sekoittavat työtilan ja tuhlaavat “älyenergiaa”.
- Käyttötapauksen poistamisella vältetään valtava kehitystyö ja säästetään “älyenergiaa”.

Ratkaisu Poista ne käyttötapaukset, jotka eivät lisää arvoa järjestelmääsi tai jotka eivät ole enää aktiivisten luettelossa.

- Toteutettavan käyttötapauksen/käyttötapausten (joukon) valinta nykyisessä iteraatiossa.

- Mikä toiminnallisuus (= käyttötapaukset) tuo arvoa käyttäjille?
- Kuinka paljon aikaa ja resursseja kuluu tietyn käyttötapauksen toteuttamiseen? (vrt. käyttäjätarinoiden arvioiminen, XP, Personal Software Process)

Esimerkkejä Ei anneta tässä, mutta tämä malli (kuten muutkin muokkausmallit) ovat selkeitä tapoja parantaa asiakkaan organisaation tehokkuutta yksinkertaistamalla (liike)toimintaprosesseja (engl. Business Process Management and Reengineering).

9 Kehitys: Ryhmä

Niin vaatimusten keräämisessä kuin ohjelmistonkehityksessä yleensäkin eri asianosaisten roolit ja etenkin koko kehitysryhmän organisoinnin tulee olla selkeästi määritelty koskien eri rooleja eri vaiheissa ja toimissa. Ohjelmistonkehitys vaatii onnistuakseen ihmisiä, joilla on erilaisia ominaisuuksia ja jotka tekevät oman osansa työstä kohti yhteistä päämäärää (= käyttäjien tarpeiden muuttaminen ohjelmistoksi). On elintärkeää löytää asiakkaan oma ääni ja kuunnella sitä läpi kehityksen.

PieniKirjoittajaRyhmä

Ongelma Liian monen ihmisen käyttäminen käyttötapauksien kirjoittamiseen ei ole tehokasta ja kompromissit, joita joudutaan tekemään usean eri näkökannan vuoksi, voivat johtaa vähemmän tyydyttävään järjestelmään.

Ajurit

- Käyttötapaukset vaativat monipuolisten ryhmien näkemyksiä ja ammattitaitoa.
- Ison ryhmän kokoaminen yhteen on vaikeata.
- Teoriassa mitä enemmän ihmisiä työskentelee käyttötapausten parissa sitä nopeammin ne saadaan valmiiksi. (Enemmän ihmisiä, enemmän johtajia, kommunikointiaika kasvaa, jaettua ammattitaitoa tarvitaan, jne. Huomaa että useimmissa tapauksissa ihmisten lisääminen projektiin joka on jo myöhässä lisää ongelmia ennestään. Miksi?)
- Liian suurista ryhmistä tulee tehottomia (vrt. N-yhtiön operatiivinen rakenne)
- Suuret ryhmät voivat johtaa komiteapohjaiseen suunnitteluun ja tuotteen liiallisiin toimintoihin.

Ratkaisu Rajoita yhtä työtuotetta hiovien henkilöiden määrä kahteen tai kolmeen henkilöön. Käytä **KaksikerrosKatselmukset** prosessia uusien ihmisten mukaan ottamiseen.

- Pienen ryhmän on helpompi tavata, kommunikoida ja tehdä yhteisiä päätöksiä.
- Kehityksessä voidaan käyttää useita malleja **PieniKirjoittajaRyhmä**, mutta tällöin tarvitaan uusi rooli yleiselle (vision) hallinnalle - käyttötapausarkkitehti.

OsallistuvaYleisö

Ongelma Asianosaisten tarpeita ei voida tyydyttää ilman heidän omaa panostaan ja palautettaan.

Ajurit

- Useilla eri ryhmillä on etuja valvottavana käyttötapausjoukoissa.
- Jos asiakkaat eivät pidä siitä, he eivät osta sitä.
- On yleensä epäkäytännöllistä ottaa koko kohdeyleisö mukaan käyttötapausten kehitysprosessiin.
- Kehitysorganisaatiot eivät edusta loppukäyttäjiä riittävän hyvin.

Ratkaisu Ota asiakkaasi ja sisäiset asianosaiset aktiivisesti mukaan käyttötapausten kehitysprosessiin aina kun mahdollista.

- Sisäisiä asianosaisryhmiä voi olla useita, esim. rahoittaja(t), loppukäyttäjät, dokumentointiryhmä, laadunvarmistusryhmä, johto, kehittäjät, jne., jotka ovat kuin asiakkaita käyttötapausten kehitysryhmälle.
- Useat rajoitteet voivat estää suorien kontaktien saamisen loppukäyttäjiin. Yritä päästä näiden esteiden yli käyttämällä sopivia tekniikoita.

TasapainoinenRyhmä

Ongelma Samanmielisten yksilöiden käyttäminen samanlaisissa ryhmissä käyttötapauksien kehittämiseen voi johtaa rajoittuneeseen ja kapea-alaiseen käyttötapausten joukkoon, joka ei tyydytä kaikkien tarpeita.

Ajurit

- Ryhmät, joiden jäsenillä on yhtenevä tekninen tausta, keskittyvät usein samoihin rajoittuneisiin kysymyksiin.
- Jokaisessa ammatissa on oma erikoistunut sanastonsa, jota ammatin ulkopuoliset ihmiset eivät ymmärrä (vrt. *Domain Engineering*).
- Ryhmät tarvitsevat erilaista osaamista eri hetkinä.
- Hyvät ryhmät ovat tasapainossa niin, että jokaisen ryhmän jäsenen erikoisosaaminen kompensoi jonkun toisen jäsenen heikkoutta (vrt. pelaajien roolit esim. jalkapallossa, koripallossa tai jääkiekossa).

Ratkaisu Aseta ryhmään eri erikoisosaamista omaavat henkilöt puolustaaksesi asianosaisten etua kehitysprosessissa. Varmista, että ryhmässä on sekä suunnittelijoita että loppukäyttäjiä.

- Huomaa, että **OsallistuvaYleisö** ja **TasapainoinenRyhmä** ohjaavat kehitysryhmää eri suuntaan kuin **PieniKirjoittajaRyhmä**. Tämä havainnollistaa taidokkaan ohjelmistonkehityksen liikettä pitkin yksittäinen/yksityinen-ryhmä/jakaminen dimensiota.

- Eri sovellusalueet tarvitsevat erilaisia käyttötapausten kehitysryhmiä, koska sovellusaluetietouden määrä (terminologia, (liike)toimintaprosessit, lainsäädäntö, jne.) vaihtelee projektista toiseen. Nykyään yleinen käytäntö ammattimaisissa ohjelmistonkehitysorganisaatioissa on dokumentoida perustava sovellusaluetietous esim. liiketoimintaarkkitehtuurikuvausten muodossa.

- Ihmisasiat (peopleware) ja näiden järjestely on tärkeä esivaatimus onnistuneelle ohjelmistonkehitykselle, mutta ryhmän pitää myös seurata prosessia pystyäkseen tunnistamaan ja kirjoittamaan laadukkaita vaihetuotoksia (esim. käyttötapaukset).

Esimerkkejä Liite A, käyttötapaukset 2.1 ja 2.2.

Liite

Reminders

Write something readable.

Casual, readable use cases are still useful, whereas unreadable use cases won't get read.

Work breadth-first, from lower precision to higher precision.

Precision Level 1: Primary actor's name and goal
Precision Level 2: The use case brief, or the main success scenario
Precision Level 3: The extension conditions
Precision Level 4: The extension handling steps

For each step:

Show a goal succeeding.
Capture the actor's intention, not the user interface details.
Have an actor pass information, validate a condition, or update state.
Write between-step commentary to indicate step sequencing (or lack of).
Ask "why" to find a next-higher level goal.

For data descriptions (only put Precision Level 1 into the use case text):

Precision Level 1: Data nickname
Precision Level 2: Data fields associated with the nickname
Precision Level 3: Field types, lengths, and validations

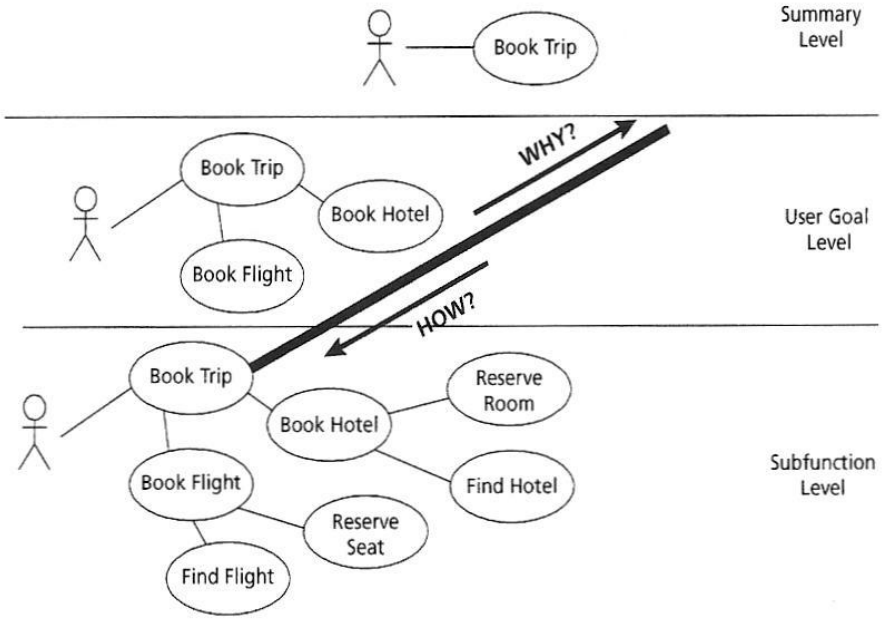


Figure 4.12 *Use case goal levels (Cockburn 2001)*

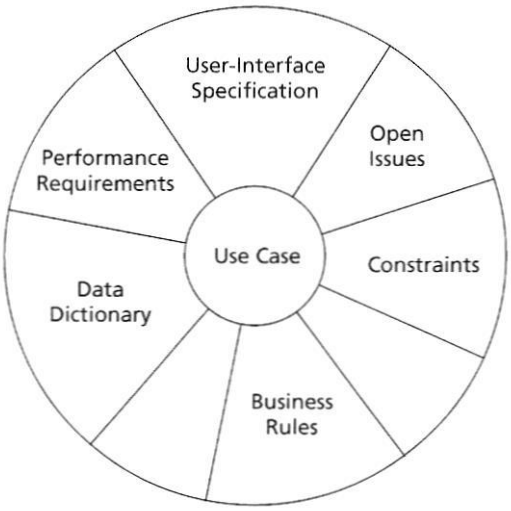
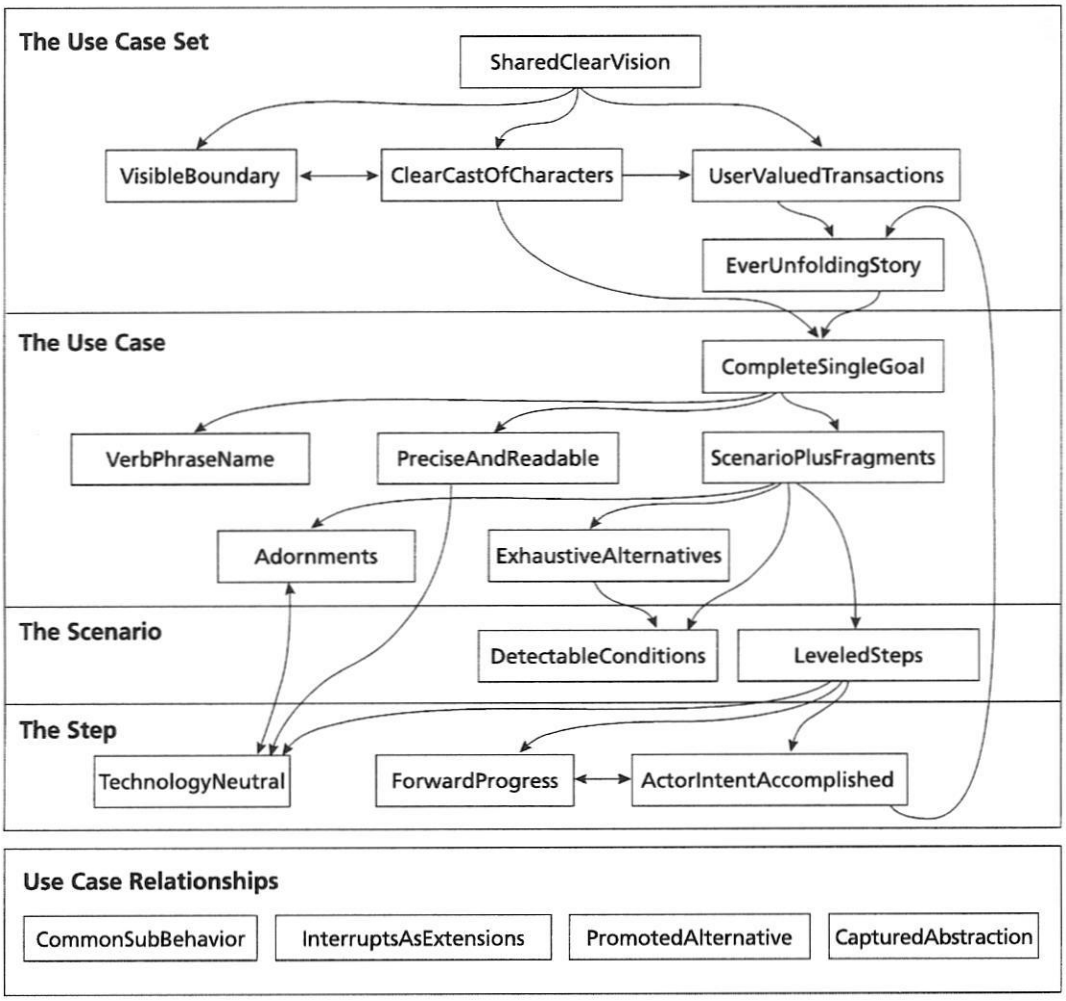


Figure 5.1 *The use case anchors other supplementary requirements (adapted from Cockburn 2001).*

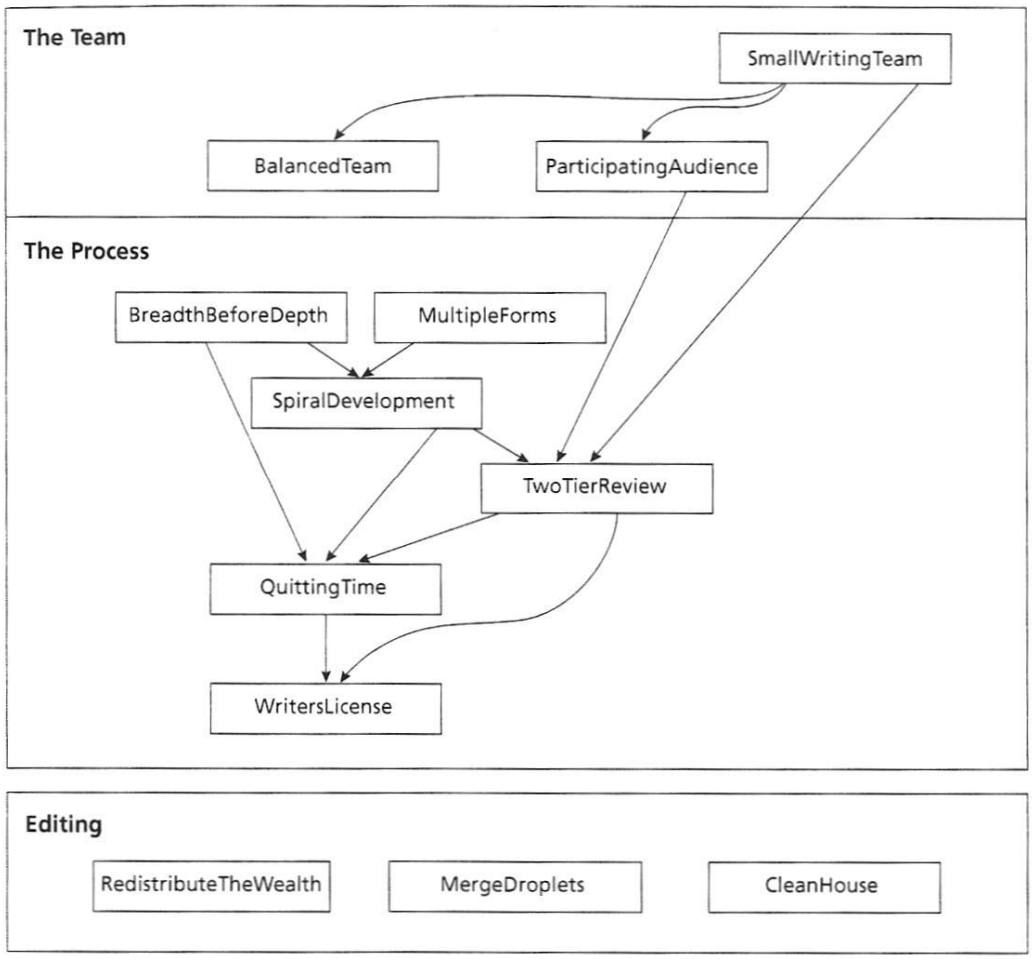
The Writing Process

1. Name the system scope and boundaries.
   Track changes to this initial context diagram with the in/out list.
2. Brainstorm and list the primary actors.
   Find every human and non-human primary actor, over the life of the system.
3. Brainstorm and exhaustively list user goals for the system.
   The initial Actor-Goal List is now available.
4. Capture the outermost summary use cases to see who really cares.
   Check for an outermost use case for each primary actor.
5. Reconsider and revise the summary use cases. Add, subtract, or merge goals.
   Double-check for time-based triggers and other events at the system boundary.
6. Select one use case to expand.
   Consider writing a narrative to learn the material.
7. Capture stakeholders and interests, preconditions and guarantees.
   The system will ensure the preconditions and guarantee the interests.
8. Write the main success scenario (MSS).
   Use 3 to 9 steps to meet all interests and guarantees.
9. Brainstorm and exhaustively list the extension conditions.
   Include all that the system can detect and must handle.
10. Write the extension-handling steps.
    Each will end back in the MSS, at a separate success exit, or in failure.
11. Extract complex flows to sub use cases; merge trivial sub use cases.
    Extracting a sub use case is easy, but it adds cost to the project.
12. Readjust the set: add, subtract, merge, as needed.
    Check for readability, completeness, and meeting stakeholders' interests.

Structural Patterns



Development Patterns



Use Case 7.1 Use Case Horror: Director-style *Book Flight* Use Case That Incorrectly Applies the *Includes* Relationship

Book Flight

Level: User Goal

1. The use case begins when the agent selects Set Travel Itinerary. Call *Capture Flight Information* to get the travel itinerary.
2. The agent chooses Select Flight. Call *Reserve Flight Segments* to reserve a flight segment.
3. The agent chooses Book Flight. Call *Obtain Payment Information* to get payment choice.
4. Call *Issue Ticket* to print ticket.

Use Case 7.2 The Revised *Book Flight* Use Case

Book Flight

Level: User Goal

1. The use case begins when the agent specifies a travel itinerary.
2. The system searches a set of appropriate flights and presents them to the agent.
3. The agent selects the flight.
4. The system verifies that space is available on the flight and reserves the seats on the flight.
5. The agent finalizes the booking by supplying payment information.
6. The system books the seats and issues the ticket.

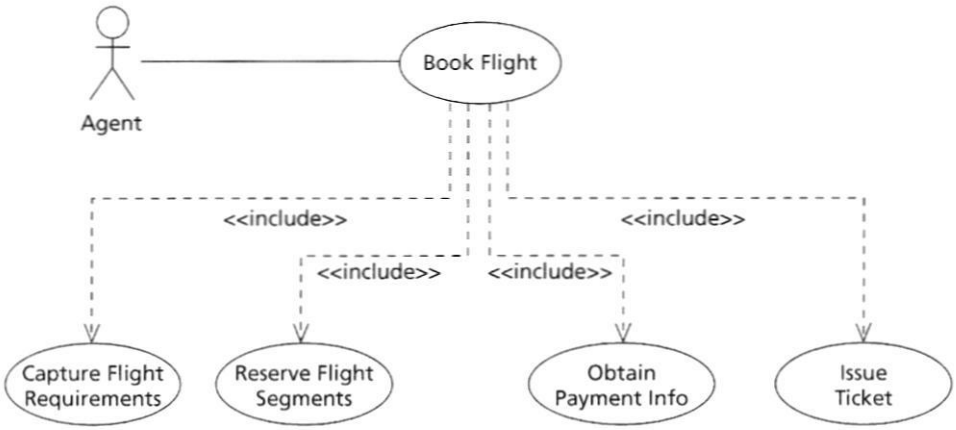


Figure 7.1 *Misuse of* includes *to create a director-style use case*

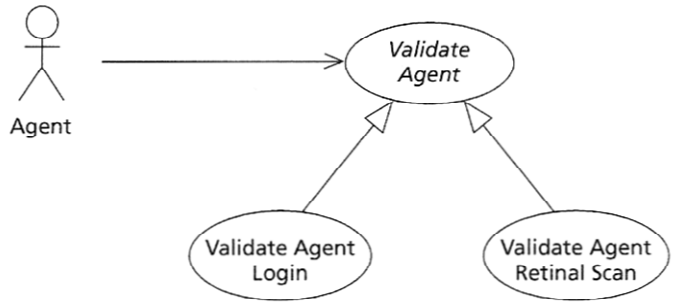


Figure 7.13 *UML generalization relationship*

Use Case 8.4 Use Case Horror: A Set of Small Use Cases for Placing a Telephone Call

Primary Actors: Caller—Person making the call
Called—Person receiving the call
Supporting Actors: Switching Network—Equipment handling calls between telephones
Level: User Goal

Use Case 1: *Place Call*

Preconditions: Caller's and Called's phones are on-hook.
Success Guarantee: System has collected all of Caller's dialed digits.
Main Course:

1. The use case begins when Caller takes phone "off-hook."
2. Switching Network sends dial tone.
3. Caller dials call.
4. Switching Network translates digits.
5. Switching Network connects call to Called's phone.
6. Switching network sends ring signal to Called's phone.
7. Switching Network sends ring signal to Caller's receiver, and the use case terminates.

Use Case 2: *Talk on the Phone*

Preconditions: Use Case *Place Call* has sent ring signal to Called's phone.
Success Guarantee: Users have finished call.
Main Course:

1. The use case begins when Called's phone rings.
2. Called answers phone, taking phone "off-hook."
3. Switching Network completes connecting Caller and Called.
4. Called and Caller talk.
5. Called and Caller hang up, and the use case terminates.

Use Case 3: *Terminate Call*

Preconditions: Call is connected via *Talk on the Phone.*
Success Guarantee: Switching Network has broken down call, and all companies are in the same state they were before call.
Main Course:

1. The use case begins when Caller and Called hang up.
2. The Switching Network frees all of the components that the call required.
3. The Switching Network *Logs Billing Information,* and the use case terminates.

Use Case 2.1 Use Case Horror: *Provision a Cross-Connect* (Engineering-Centric Version)

Provision a Cross-Connect (Engineering-Centric Version)

Primary Actor: Network Administrator: Person provisioning **Network Elements.**

Secondary Actor: **Network Element** Device being provisioned.

Level: User Goal

Preconditions: Network Element is accessible and user has a valid account on it.

Success Guarantee: Cross-connect is "active."

Main Success Scenario:

1. The use case begins when the **Network Administrator** activates account on **Network Element**_(ACT-USER:TID:USERID:::PASSWORD).
2. The **Network Element** establishes a session.
3. The **Network Administrator** enters_ENT-STSnC:TID:FROMAID::::IS; (or ENT-T3:TID:AID::::IS; if it is on a DS3) to establish the source STS.
4. The **Network Element** establishes the "From AID" STS.
5. The **Network Administrator** enters ENT-STSnC:TID:TOAID::::IS; (or ENT-T3:TID:AID::::IS; if it is on a DS3) to establish the sink STS.
6. The **Network Element** establishes the "To AID" STS.
7. The **Network Administrator** enters _ENT-CRS-STSnC:TID::FROMAID,TOAID:::; to establish the cross-connection.
8. The system establishes the cross-connection.
9. The user logs off of the **Network Element** (CANC-USER:TID:userid:;)
10. The use case terminates when the **Network Element** terminates the session.

Use Case 2.2 *Provision a Cross-Connect* (User-Friendly Version)

Provision a Cross-Connect

Primary Actor: Network Administrator: Person provisioning **Network Elements.**

Secondary Actor: **Network Element** Device being provisioned.

Level: User Goal

Preconditions: Network Element is accessible and user has a valid account on it.

Success Guarantee: Cross-connect is provisioned and viable.

Main Success Scenario:

1. The use case begins when the **Network Administrator** logs on to the **Network Element.**
2. The **Network Element** verifies the user and begins a session.
3. The **Network Administrator** determines the size of the cross-connection, and sets up the endpoints accordingly.
4. The **Network Element** establishes the endpoints.
5. The **Network Administrator** defines the cross-connection.
6. The **Network Element** establishes the cross-connection.
7. The user logs off of the **Network Element.**
8. The use case terminates when the **Network Element** terminates the session.