OHJELMISTOPROSESSIN PARANTAMINEN ATTRIBUUTTIFOKUSOINNIN AVULLA

Henriikka Ahtiainen ja Olli Günther

8.6.2005
Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

TIIVISTELMÄ

Ohjelmistotuotannossa tarvitaan käyttökelpoisia menetelmiä prosessin kehittämiseen, tavoitteena aika- ja kulukustannusten pienentäminen ja prosessin puutteiden minimoiminen. Erilaisia puutetietoja hyväksi käyttäviä menetelmiä on kehitetty näitä tavoitteita täyttämään. ODC-menetelmä luokittelee puutteiden riippumatonta sekä eri prosessivaiheiden välillä yhtenäistä ja johdonmukaista attribuutti- eli ominaisuustietoa. ODC on tuotos- ja organisaatioriippumaton ja on siten monissa eri tyyppisissä prosesseissa käyttökelpoinen keino edellä mainitun päämäärän saavuttamiseen. Sitä voidaan käyttää prosessinaikaiseen prosessin kehittämiseen tai prosessien väliseen jatkumoon kehitystyössä. Luokitellun puutetiedon sisältämä luokittelijalle merkityksellinen tieto saadaan laajasta aineistosta esille attribuuttifokusointi-menetelmällä tietoa rikastamalla. Tietoa rikastetaan attribuuttien mielenkiintoisuutta mittaavan mielenkiintoisuusfunktion ja tulosten esittämistä suodattavan suodatinfunktion avulla. ODC-luokittelu ja attribuuttifokusointi ovat kustannustehokas keino hyvinkin laajojen ja erilaisten projektien tarpeisiin. Menetelmien käyttöönotto ei vaadi erityisosaamista tai spesialisteja. Projektihenkilöstä tuntee prosessin ja sen tuotoksen ominaispiirteet parhaiten ja voi suorittaa sekä luokittelun että tulosten analysoinnin ja tarkastelun tutusta projektista itse. Tutkielman lopussa esitellään ODC-luokittelua ja attribuuttifokusointia hyödyntävä MiKiFu-järjestelmä ja sen avulla käsiteltyjä esimerkkiaineistoja ja käsittelyn tuloksia.

ACM-luokat (ACM Computing Classification System, 1991 version): ACM D.2.5, D.2.8, H.2.8, K.6.1, K.6.3

Avainsanat: Attribuuttifokusointi (AF, Attribute Focusing), ODC (Orthogonal Defect Classification), ohjelmiston parantaminen, puutteiden luokittelu.

ESIPUHE

Pro gradu –tutkielmamme valitsimme luontevana jatkeena erikoistyöllemme. Aiheena tutkielmassa on ohjelmistotuotantoprosessin parantaminen, puutteiden ominaisuustietoa sisältävän luokitellun aineiston pohjalta tehtävän attribuuttifokusoinnin avulla. Opintoihimme liittyvänä erikoistyönä kehitimme jo aiemmin tätä menetelmää toteuttavan MiKiFu-järjestelmän, jolla kyseistä attribuuttien mielenkiintoisuutta voidaan laskea ja analysoida. Teoriaan olimme jo siis perehtyneet tuota erikoistyötä tehdessämme, joten aiheeseen oli helppo palata ja syventyä. Aiheen mielekkyyttä lisäsi myös tämän tutkielman osaksi liitetty ja MiKiFu-järjestelmällä toteutettu esimerkkiaineiston analysointi. Se osoitti konkreettisesti, mitä kaikkea attribuuttifokusoinnilla voidaan saada aikaan, herättäen myös ajatuksia siitä, mitä parannuksia järjestelmäämme voitaisiin mahdollisesti jatkossa vielä tehdä.

Kiitos ohjaajallemme Raimo Raskille hänen kärsivällisyydestään ja paneutumisestaan aiheeseemme. Yhteistyö kaikkien osapuolien välillä sujui hyvin ja kirjoitustyö oli hauskaa ja antoisaa. Osuutemme tutkielman tekemisessä jakautui tasan, kuitenkin siten, että Olli Günther oli enemmän vastuussa attribuuttifokusointiin ja Henriikka Ahtiainen puutteiden luokitteluun liittyvissä kohdissa. MiKiFu-järjestelmän avulla laskettujen tulosten analysointi ja esittäminen on tehty yhdessä.

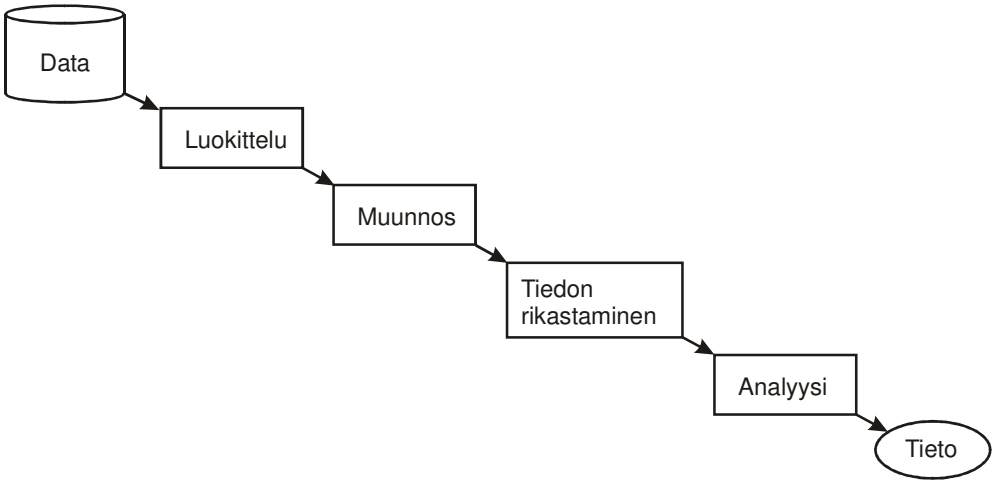
SISÄLLYSLUETTELO

1 JOHDANTO

Tässä tutkielmassa tarkastellaan ohjelmistoprojektien puutetietoja ja niiden teoreettista taustaa sekä erilaisia menetelmiä puutetiedon luokittelemiseen ja puutetiedon projektinaikaiseen analysointiin. Attribuuttifokusointi (Bhandari et al., 1992) on analysointimenetelmä, jossa puuteanalyysia pystytään hyödyntämään ohjelmistoprojektin aikana. Puutetieto kerätään sekä analysoidaan ja tulkitaan parhaiten projektiryhmän itsensä toimesta, mikä säästää aika- ja henkilöstöresursseja. Ongelmakohtiin pystytään myös puuttumaan välittömästi ja projektin etenemisaikataulu tiedostetaan hyvin. Lähtökohtana tutkielmalle on MiKiFu-järjestelmä (Ahtiainen ja Günther, 2003), jonka teoria perustuu attribuuttifokusoinnille ja sitä voidaan käyttää ohjelmistoprosessissa syntyvien puutteiden analysointiin.

Puutetietojen luokitteleminen ja analysointi on tiedonrikastamista eli tiedonlouhintaa (data mining) eli tietokannasta, tässä tapauksessa projektitietokannasta, etsitään kiinnostavaa tietoa algoritmisesti tehokkaasti (Han ja Kamber, 2001). Kuvassa 1.1 esitetään tiedon rikastamisprosessi yleisellä tasolla. Tiedon rikastamisessa tietokannan datasta valitaan halutut tiedot, esimerkiksi tietyllä aikavälillä kirjatut prosessissa syntyneet puutetiedot. Prosessi voi olla valmis tai keskeneräinen. Esimerkiksi MiKiFu-järjestelmässä voidaan tarkastella puutetietoja ohjelmistoprosessin eri vaiheissa, joten tärkeää tietoa voidaan saada analyysin avulla jo keskeneräisestäkin prosessista. Näin puutteiden syntyyn voidaan puuttua jo prosessin edetessä ja välttää toistamasta samoja virheitä myöhemmässä prosessinvaiheessa.

Valitut tiedot esikäsitellään eli luokitellaan määrättyihin ehtoihin perustuen. Tässä tutkielmassa puutteita luokitellaan ominaisuus- eli attribuuttitiedon pohjalta. Attribuuttitieto on puutteiden eri piirteitä ja erilaisia ominaisuuksia kuvailevaa tietoa. Tavoitteena attribuuttitietoon perustuvalla puutteiden luokittelulla on prosessin kehittäminen havaitsemalla juuri oleelliset ja tärkeät puutteet. MiKiFu-järjestelmässä puutetiedot järjestetään luokkiin esimerkiksi ODC-luokittelun mukaisesti. Puuteluokitteluprosessista, puutteen ominaisuuksista ja eri puuteluokittelumalleista kerrotaan tarkemmin luvussa 2.



Kuva 1.1. Tiedon rikastaminen

Esikäsittelyn jälkeen tieto muunnetaan esimerkiksi tilastollisia menetelmiä käyttäen. MiKiFu-järjestelmässä käytetään yksi- ja kaksisuuntaista data-analyysia. Puutetietojen analysoinnissa olemme kiinnostuneita lähinnä sellaisista puutteista, jotka eroavat oletetusta puutejakaumasta. Attribuuttifokusoinnissa mielenkiintoisuusfunktion avulla muodostetaan kiinnostavien tietojen joukko. Yksisuuntaisella data-analyysilla voidaan tutkia odotetusta jakaumasta poikkeavaa attribuuttiarvoa. Kaksisuuntaisella data-analyysillä puolestaan voidaan tutkia onko attribuuttiparien välillä korrelaatiota. (Bhandari ja Roth, 1993; Mendonça ja Basili, 2000).

Suodatinfunktio rikastaa tiedon attribuuttifokusoinnissa analysoijan kannalta järkevään muotoon. Olennaiset tiedot voidaan esittää kuvina, kaavioina ja tilastoina. Lopuksi saatu tulos tulkitaan. Mitkä attribuutit sitten ovat analysoijalle mielenkiintoisia? Yleisesti ottaen mielenkiintoisia ovat tapaukset, jotka ovat analysoijalle ymmärrettäviä, päteviä/toistettavissa, käytännöllisiä ja uusia. MiKiFu-järjestelmässä lasketaan attribuuttien ja attribuuttiparien mielenkiintoisuus oletettujen löydösten ja löydettyjen välisenä erotuksena. MiKiFu järjestää tapaukset järjestykseen erotusprosentin itseisarvon perusteella. Tapaus on mielenkiintoinen, jos erotusitseisarvo on suuri. Erotusitseisarvo on suuri tapauksissa, jossa puutteita löydetään huomattavasti enemmän kuin on oletettu ja toisaalta, jos puutteita ei ole löydetty montaakaan ja oletus löydölle on suhteellisen suuri. Analyysista voidaan tulkita, mitkä puutteet ovat prosessissa yleisimpiä ja parivertailun avulla, mitkä puutteet esiintyvät yhdessä. MiKiFu:n suorittamasta analyysista

voidaan myös todeta, mitä puutteita ei juurikaan esiinny, mikä on myös mielenkiintoinen ja huomionarvoinen seikka (Han ja Kamber, 2001).

Ohjelmistoprosessissa syntyviä puutteita tutkimalla ja niihin johtaneiden syiden ymmärtämisellä voidaan prosessia kehittää jo sen elinkaaren aikana. Saavutettavaa tietoa voidaan hyödyntää myös tulevissa projekteissa, koska menetelmällä havaitut puutteet koskevat nimenomaan prosessia eivätkä niinkään prosessin kohdetta.

Tutkielman alkuosa keskittyy puutetiedon teoreettiseen esittelyyn ja loppuosassa esitellään erilaisia käytännön sovelluksia puutetiedon hyödyntämisestä.

Luvussa 2 esitellään puutetiedon määrittelyä ja sen ominaisuudet sekä syyt puutetiedon luokitteluun. Lisäksi esitellään erilaisia puutetiedon luokitteluun kehitettyjä menetelmiä ja näiden käyttöönottoa ohjelmistoprosessissa.

Luvussa 3 käydään läpi puutteiden ominaisuustietoon perustuvaa attribuuttifokusointia, jota käyttäen pyritään löytämään prosessin kannalta oleellisia ongelmakohtia. Lisäksi tutustutaan attribuuttifokusoinnissa käytettävän datan keräämiseen ja oleellisen, kiinnostavan tiedon suodattamiseen laajasta puuteaineistosta mielenkiintoisuusfunktion ja suodatinfunktion avulla. Luvussa esitellään myös tapaustutkimuksia attribuuttifokusoinnista, millä pyritään havainnollistamaan menetelmän käyttöä todellisissa olosuhteissa.

ODC–menetelmää ja attribuuttifokusointia käyttäviä ohjelmistovälineitä on kehitetty eri elämän aloille, kuten esimerkiksi urheilun, liiketoiminnan tai tieteen pariin (Bhandari & al., 1997; Harris & al., 1998; Nazeri & al., 2001). Luvussa 4 esittelemme erikoistyönä tekemämme MiKiFu-järjestelmän (Ahtiainen ja Günther, 2003), jonka avulla ohjelmistoprosessissa löytyneitä puutteita voidaan kirjata sekä laskea ja suodattaa mielenkiintoisuustietoa näistä puutteista.

Luvussa 5 esitetään yhteenveto ODC-luokittelusta ja attribuuttifokusoinnista sekä MikiFu-järjestelmän käytöstä luokitellun aineiston tarkasteluun.

2 PUUTTEIDEN LUOKITTELU

Laatumittareita tarvitaan mittaamaan ohjelmistojen laatua, koska ohjelmistosuunnittelijat eivät pysty tekemään täydellistä ja virheetöntä tuotetta. Laatumittarit auttavat selvittämään, kuinka tuotteesta löydetään ongelmakohdat, kuinka tehokasta työskentely on ja onko tuote valmis seuraavaan prosessivaiheeseen (Fenton ja Pfleeger, 1997). Eräänä laadun mittarina voidaan käyttää ohjelmistoprosessin kaikissa vaiheissa löytyviä puutteita. Puutteet sisältävät paljon tietoa kehitystyöstä, joten ne soveltuvat hyvin laadun mittaamiseen.

2.1 Puuteluokitteluprosessi

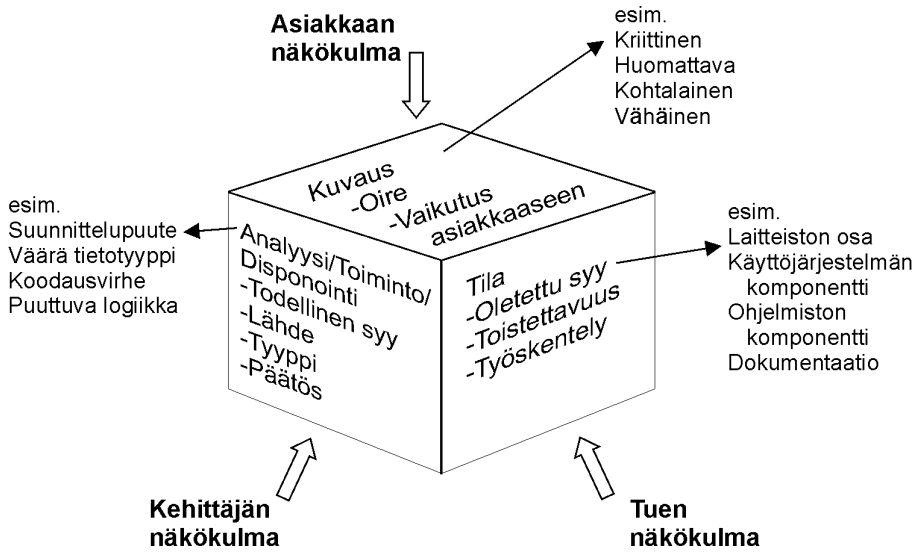
Puute on epäkohta määrityksissä, suunnittelussa tai tuotteen toteutuksessa. Epäkohtien vuoksi on vaikeaa arvioida tuotteen kehityksen ja ylläpidon valmistumista. Siksi ohjelmistojen kehittäjät ja projektipäälliköt keräävät puutedataa ja analysoivat sitä. Ymmärtämällä miksi puutteita syntyy ja miten niitä voidaan poistaa, voidaan parantaa tulevaisuuden ohjelmistoprosesseja (Grady 1992).

Yrityksen, organisaation tai projektin kannattaa miettiä puuteluokittelua valitessaan oman projektinsa tavoitteita. Myös käytettävissä olevat resurssit tulee ottaa huomioon. Puuteluokittelun toteuttaminen tai kehittäminen vaatii seuraavan prosessin (Fredericks ja Basili, 1998):

1. Muodostetaan hyvin perustellut ja valitut tavoitteet.
2. Hankitaan johdon tuki tavoitteille ja kehitystyön vaatimille muutoksille prosessissa.
3. Jaetaan tavoitteet pienemmiksi osatavoitteiksi.
4. Määritellään mittausmenetelmä datan keräämiseksi. Menetelmän antamien arvojen avulla kehittäjät ja johto voivat vastata osatavoitteiden asettamiin kysymyksiin.
5. Opastetaan datan keräämistä ja mahdollisia työvälineitä. Työkalut auttavat datan keräämisessä, jäljittämisessä ja analysoinnissa.
6. Kerätään data.

7. Tarkistetaan datan luotettavuus.
8. Analysoidaan data.
9. Julkistetaan tulokset ja annetaan palaute. Palaute voi seurata myös askelta 1, koska edellisten projektien palaute auttaa asettamaan uusia tavoitteita.

Kuvassa 2.1 nähdään kolme erilaista näkemystä puutteesta: asiakkaan näkemys, tuen näkemys ja kehittäjien näkemys. Jokaisella näistä näkemyksistä on oma terminologiansa ja tarkoituksensa. Kun asiakkaalla on ongelmia ohjelmistotuotteen kanssa, hän tietää, että hän ei voi tehdä työtään valmiiksi, koska tuote ei toimi halutulla tavalla. Asiakkaalle tuotteen puutteet aiheuttavat työn viivästymistä. Hän laskee aiheutuneet menetykset rahassa ja ajassa, jotka hän menettää töiden seisoessa. Tuen on paikannettava asiakkaan virheellinen komponentti ja kerrottava asiakkaalle puutteen korjausaikataulu. Puutteesta on saatava riittävästi tietoja asiakkaalta, jotta pysyvä korjaus voidaan suorittaa. Kehittäjät etsivät koodista vian, priorisoivat puutteen korjaustarpeen, analysoivat kuinka vaikeaa puute on korjata ja määrittelevät odotetun korjausajan (Grady, 1992). Puutteen ilmeneminen vaikuttaa siis ainakin näihin kolmeen tahoon ja kaikkien tahojen toiminnot vaikeutuvat ja hidastuvat.



Kuva 2.1. Puutekuutio (Grady, 1992).

Puutetiedot täytyy kerätä ja luokitella. Puutteiden luokittelu tarjoaa keinon saada ohjelmiston kehitysprosessista nopeaa ja tehokasta palautetta prosessin kehittäjille. Tällä pyritään intuitiivisen päätöksenteon sijasta päätöksentekoon, joka pohjautuu tarkkoihin mittauksiin, analyyseihin ja suunnitelmiin. Puutetiedot sisältävät runsaasti semanttista tietoa kehitystyöstä. Semantiikan muuttaminen tehokkaan luokittelumenetelmän avulla mitattavaan muotoon antaa hyvän pohjan prosessin parantamiseen ja tuotoksen kehittämiseen**.** Luokiteltuja puutteita voidaan edelleen käsitellä erilaisilla menetelmillä ja analyyseillä, kuten esimerkiksi attribuuttifokusointi (Chillarege ja Biyani, 1994), josta kerrotaan tarkemmin luvussa 3.

Puutteiden sisältämää tietoa käytetään useisiin erilaisiin tarkoituksiin, kuten työn loppuunsaattamiseen, resurssienhallintaan, aikataulutukseen, riskianalyyseihin ja prosessien jatkuvaan parantamiseen. Puutteidenluokittelumenetelmiä on kehitetty eri lähtökohdista ja eri tarpeisiin. Useimpien menetelmien tarkoituksena on tunnistaa puutteiden syy korjaavan toimenpiteen määrittämiseksi (Kelly ja Shepard, 2001).

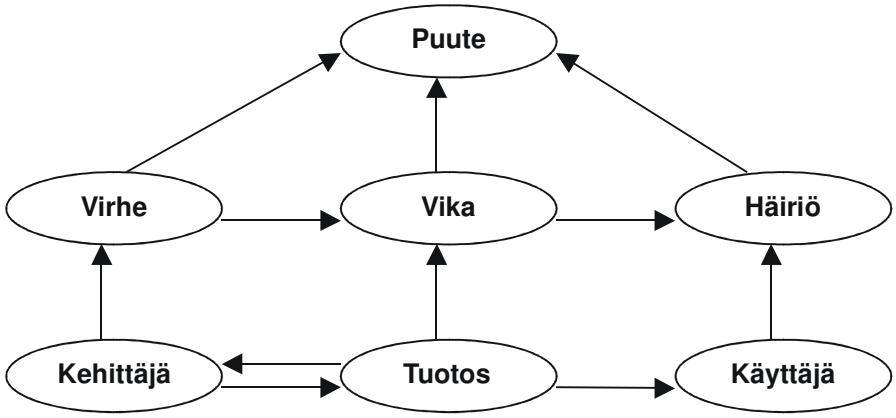
Useat ohjelmistoyritykset käyttävät puuteluokittelumalleja. Puuteluokittelumallia valitessaan yrityksen täytyy miettiä, kuinka hyvä puuteluokittelumalli voidaan määritellä eli räätälöidäänkö uusi, omia tarpeita varten tehty malli vai käytetäänkö jotain aiemmin suunniteltua mallia. Toisaalta puuteluokittelumallia valitsevan yrityksen on harkittava, kuinka puutedataa analysoidaan ja miten tätä dataa esitetään (Freimut, 2001). Luokittelumallin valintaan vaikuttavat myös yrityksen käytettävissä olevat resurssit niin henkilöstön osalta kuin taloudellisestikin.

Erilaisia puutteidenluokittelumenetelmiä kuvataan tarkemmin kohdassa 2.5. Ensisijaisesti tarkastelemme IBM ODC–luokittelumenetelmää. ODC:tä on jatkuvasti kehitetty sen julkaisemisesta (vuonna 1992) lähtien ja sitä pidetään edistyneimpänä luokittelumenetelmänä.

2.2 Puutteen määrittely

Ohjelmistoprosessissa puutteella tarkoitetaan vaadittavaa muutosta, joka on välttämätön ohjelmiston tai tuotoksen korjaamiseksi (Chillarege, 1994). Puute voi olla virhe (error),

vika (fault) tai häiriö (failure). Kehittäjä tekee virheitä, josta aiheutuu vikoja tuotoksiin. Viat tuotoksissa aiheuttavat uusia virheitä sekä synnyttävät häiriötä käyttäjän käyttäessä tuotosta (Fenton ja Pfleeger, 1997). Kuvassa 2.2 on esitetty Krögerin kuvaama puutteiden synty, joka on hahmoteltu Fentonin ja Pfleegerin määritelmän pohjalta.



Kuva 2.2. Puutteiden synty (Kröger, 1999).

Puutteet eivät kuitenkaan aina ole vikoja, koska ne eivät aina johda häiriöön. Esimerkiksi virheellinen koodin kommentointi ei aiheuta häiriötä loppukäyttäjälle. Puutteet eivät myöskään aina ole virheitä, koska puutteet voivat olla tahattomia. Tällöin vika on seurausta huolimattomuudesta eikä varsinaisesta virheestä, esimerkiksi näppäilyvirhe. Puutteettomuus ei aina edes ole tavoite, johon täydellisesti pyrittäisiin. Tällainen on esimerkiksi häiriö tyylioppaan mukaisen koodin kirjoittamisessa. Kaikki häiriöt eivät aiheudu koodissa olevista virheistä. Vika voi olla määrittely- tai suunnitteludokumenteissa. Suunnittelija ei ehkä ymmärrä vaatimuksia oikein ja suunnittelee tuotteen, joka toimii suunnitelman mukaisesti, mutta ei vastaa vaatimuksia (Fredericks ja Basili, 1998; Kelly ja Shepard, 2001).

2.3 Puutteen ominaisuuksia

Puute ilmentää pysähdyskohtaa tuotoksen kehitystyössä. Ennen kuin puute on korjattu, suoritetaan kyseistä puutetta koskevia toimenpiteitä ja tapahtumia. Puute voi siten sisältää suuren määrän tietoa tuotoksesta ja prosessista (Chillarege ja Biyani, 1994). Puutetieto on näin ollen arvokas apuväline prosessin kehittämiseen.

Puutteilla on useita analysoitavia ominaisuuksia. Löydettäessä puute siitä voidaan raportoida löytöajankohta ja sijaintipaikka. Puutteet havaitaan yleensä jonkin merkin tai oireen seurauksena ja ne korjataan tietyllä tähän ongelmakohtaan tyypillisellä tavalla. Puutteista voidaan löytää riippumattomia avainelementtejä, jotka erottavat puutteiden ominaisuuksia. Avainelementit puutteiden luokittelemiseksi ovat (Fenton ja Pfleeger, 1997; Freimut, 2001): `sijainti`, `ajoitus`, `oire`, `lopputulos`, `mekanismi`, `syy`, `vakavuus ja kustannus`.

Puutteen `sijainti` (location) ilmaisee, mistä puute löydettiin. Sijainti voidaan identifioida hyvinkin yksityiskohtaisesti, esimerkiksi kirjaamalla dokumentin nimi ja puutteen paikka dokumentissa tai aliohjelmamoduulin nimi ja aliohjelman nimi, missä puute sijaitsee.

Puutteen `ajoitus` (timing) kertoo milloin puute on syntynyt, milloin se on havaittu ja milloin korjattu. Ajoitukseen voidaan raportoida missä prosessivaiheessa puute on löytynyt, esimerkiksi yksikkötestausvaiheessa. Ajoitukseen voidaan kirjata myös milloin puute on syntynyt eli sen alkuperä, esimerkiksi ao. aliohjelman valmistumispäivä. Puutteen korjaamispäivä on myös raportoitava eli milloin puute on kokonaan korjattu ja raportointi sen osalta voidaan lopettaa.

`Oire` (symptom) kertoo, milloin puute ilmestyi tai toiminnon, joka toi puutteen esiin. Esimerkiksi testausvaiheessa voidaan huomata tietyn näppäimen painamisen aiheuttavan toiminnon, jota ei haluttu.

`Lopputulos` (end result) kuvaa vian aiheuttamat seuraukset. Lopputuloksesta nähdään mikä vaikutus puutteella olisi loppukäyttäjälle, jos se jätettäisiin ohjelmaan.

`Mekanismi` (mechanism) kertoo miten puute syntyi, havaittiin ja korjattiin. Puutteen aiheuttaneet toiminnot ilmenevät mekanismissa eli mitä on tehty, että puute on syntynyt. Mekanismi osoittaa, mitä toimintoja on tehty puutteen havaitsemiseksi, esimerkiksi järjestelmätestaus. Myös puutteen korjaustoiminnot näkyvät mekanismissa.

`Syy` (cause) kuvaa, miten virhe johti vikaan. Syitä voivat olla esimerkiksi käyttäjän tai suunnittelijan tekemät virheet, kuten tiedon puute tai projektin paineet kuten kommunikaation puute tiimissä. Syy voi olla myös katselmusvirhe eli puute on jäänyt aikaisemmin huomaamatta huolimattoman katselmuksen vuoksi.

`Vakavuus` (severity) määritellään käyttäjälle aiheutuvina ongelmina. Vakavuus määrittelee voidaanko vika luokitella häiriöksi ja se kertoo myös käyttäjälle puutteesta aiheutuvan vakavuusasteen, esimerkiksi vähäinen, kohtalainen, huomattava tai kriittinen.

`Kustannus` (cost) kuvaa puutteen paikallistamisesta ja korjaamisesta aiheutuvia aika- ja resurssitappioita. Tämä tieto saadaan yleensä vasta, kun puute on korjattu.

2.4 Syyt puutteiden luokittelemiseksi

Puutteita voidaan luokitella useista eri syistä esimerkiksi siksi, että luokittelu helpottaa päätösten tekoa ohjelmistoprosessin aikana. Luokittelu voi auttaa prosessia jo sen suunnitteluvaiheessa, koska aikaisempien projektien perusteella voidaan valita projektin prioriteetteja ja allokoida resursseja. Prosessin aikana projektia on helpompi analysoida, parantaa ja seurata, kun käytössä on luokitellut puutetiedot. Puutteita voidaan myös paremmin havaita ja ennalta ehkäistä, jos puutteet on hyvin luokiteltu, koska silloin osataan kohdistaa huomio puutteiden löytymisen kannalta tärkeisiin asioihin. Projektin testiaineisto voidaan luoda puuteluokittelua hyväksikäyttäen (Kelly ja Shepard, 2001).

Erilaisten projektitekniikoiden tehokkuutta ja sopivuutta voidaan tutkia luokiteltua puutetietomateriaalia käyttäen. Puutteiden löytäminen voi olla tehokkaampaa ja nopeampaa, jos puutteet on luokiteltu selkeästi. Luokitellut puutetiedot ovat ensiarvoisen tärkeitä uusien puutteenetsintämenetelmien tekemiseen ja suunnitteluun, koska uusia menetelmiä ei voida kehittää, jos puutteiden syitä ja vaikutuksia ei tiedetä riittävän hyvin (Kelly ja Shepard, 2001; Huber, 1999).

Esimerkiksi ODC–luokittelussa ohjelmistoprojektin potentiaaliset ongelmat voidaan selvittää `puutetyypin` ja `laukaisimen` yhteisesiintymisen avulla. Puutetyyppijakauman avulla voidaan selvittää tuotteen sekä prosessin ongelmia ja sitä voidaan

käyttää myös projektin suunnittelun ja tarkkailun välineenä. `Puutetyyppi` ja siihen liittyvät muut muuttujat (esim. uusi/muutettu moduuli, puutteen korjauskustannukset) voivat aikaansaada muutoksia prosessin kehittämiseen (El Emam ja Wieczorek, 1998).

Tärkein asia puutteiden luokittelussa on kuitenkin se, että puuteluokittelu on toistettavissa. On todennäköisempää, että ODC–luokittelun kaltaisella semanttisella puuteluokittelulla saadaan virheettömämpiä tuloksia kuin mielipiteeseen perustuvilla luokitteluilla. Jos puuteluokittelu on subjektiivista, on todennäköistä, että eri ihmiset luokittelevat puutteen eri tavalla. Jos puuteluokittelusta ollaan yleisesti eri mieltä, ei tuloksia voida analysoida oikein eikä tehdä prosessin parantamista koskevia oikeita päätöksiä (El Emam ja Wieczorek, 1998).

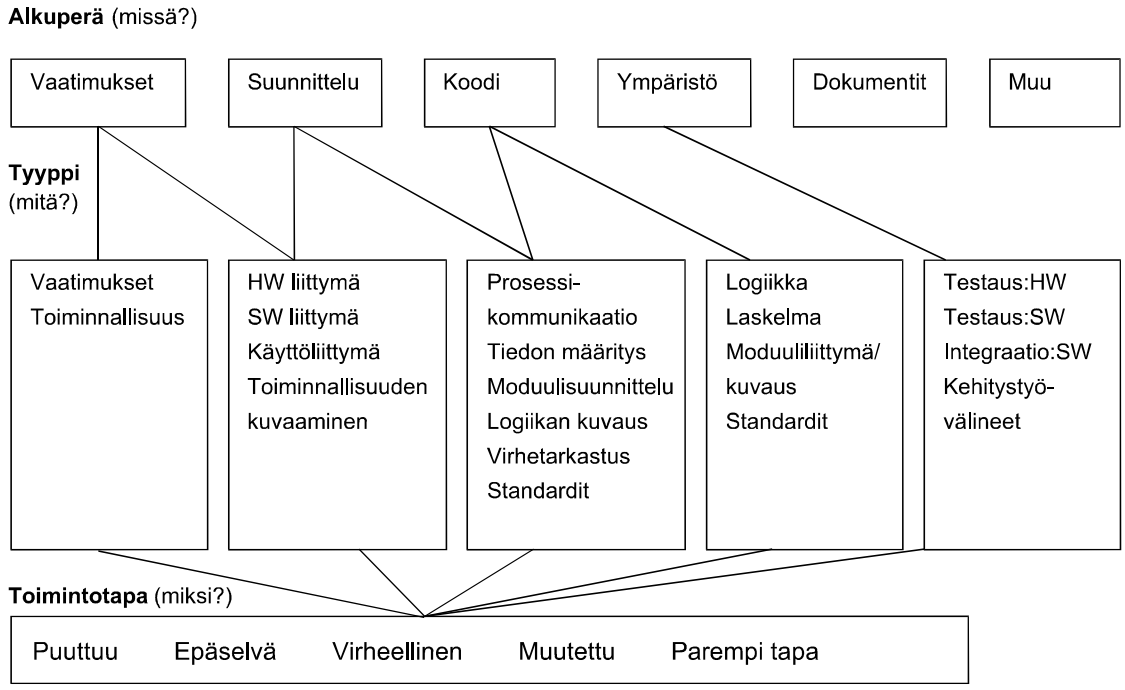
2.5 Puuteluokittelumalleja

IBM:llä Boeblingenissä Saksassa työskennellyt Albert Endres, teki ensimmäisen suuremman työn puutteiden luokittelemiseksi vuonna 1973 (Fredericks ja Basili, 1998). Endresin data sisälsi 432 puutetietoa, jotka oli kerätty testausvaiheessa kehitettäessä DOS/VS -käyttöjärjestelmää. Endres luokitteli datan kuuteen kategoriaan ja määritteli jokaiselle puutteelle tyypin. Hän suunnitteli puutteelle kysymyksiä, joiden avulla puutteesta saatiin lisää informaatiota. Tällaisia kysymyksiä olivat esimerkiksi missä, milloin ja miksi puute oli tullut ohjelmaan. Analyysin aikana Endres keräsi tietoa puutteen tyypistä ja niiden lukumäärästä moduulikohtaisesti sekä teki virhetyyppijakaumaa. Vaikka Endres ei saanut riittävästi tietoa puutteista eikä vastauksia kaikkiin kysymyksiinsä, hän sai kuitenkin keräämänsä datan perusteella pääteltyä, että 85 % tämän projektin puutteista korjaantuisi tekemällä muutoksia yhteen ainoaan ohjelman moduuliin.

Muut tutkijat jatkoivat siitä mihin Endres jäi. Vuonna 1976 Marylandin yliopiston tutkijat yhteistyössä Navalin tutkimuslaboratorion kanssa alkoivat tutkia puutteiden analysointia keinona parantaa ohjelmistoprosessia. Kehitystyötä on jatkettu 1980– ja 1990–luvulla (Fredericks ja Basili, 1998). Hewlett-Packard on kehittänyt oman puuteluokittelumallin ja IBM on kehittänyt kaksi erityyppistä puutetietoja käyttävää menetelmää ohjelmistoprojektin parantamiseksi. Seuraavaksi esittelemme IBM ODC–puuteluokittelumallin, IEEE–standardiin perustuvan mallin ja Hewlett-Packardin mallin.

2.5.1 Hewlett-Packardin luokittelumalli

Hewlett-Packardin luokittelumalli kehitettiin vuonna 1986 parantamaan ohjelmistojen kehitysprosessia. Luokittelumallin lähtökohta on selvittää puutteen alkuperä ja syy sen ilmaantumiseen. Hewlett-Packardin mallin kehittäjät pitivät tärkeänä, että yrityksellä tai projektiryhmällä on selkeä tavoite puutetietojen analysointiin eli on tiedettävä mitä halutaan mitata ja miksi, ennen kuin mittaustehtävät aloitetaan. Tällöin osataan mitata oikeita ja projektille tärkeitä asioita. Esimerkiksi jos tavoitteena on selvittää puutteiden alkuperä, niin täytyy selvittää milloin puute löydettiin ja milloin se korjattiin. Hyvin asetetut tavoitteet pienentävät kustannuksia, koska puutetietojen kerääjillä ja analysoijilla ei mene aikaa turhien tietojen keräämiseen (Fredericks ja Basili, 1998).



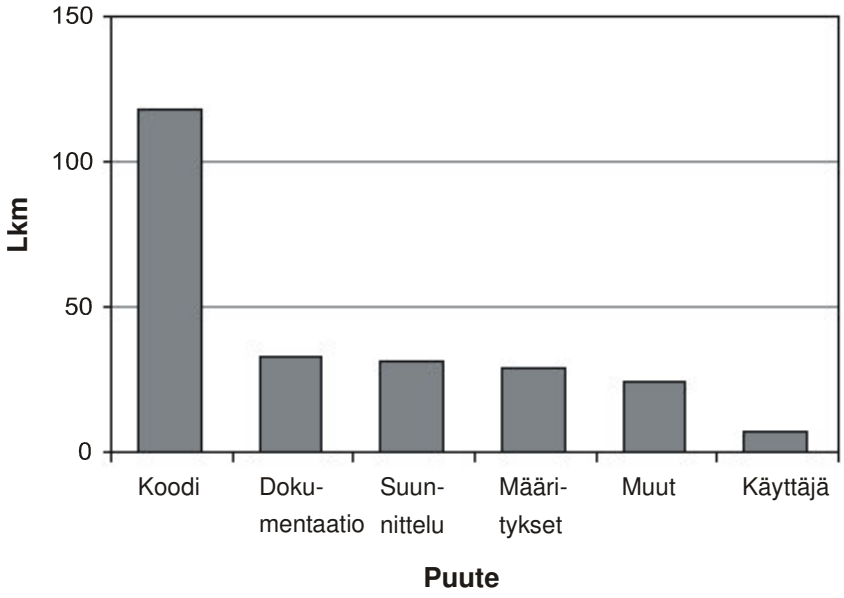
Kuva 2.3. Hewlett-Packardin puuteluokittelumalli
(Freimut, 2001; Fredericks ja Basili, 1998).

Jokainen puute luokitellaan kolmeen tasoon: *alkuperään* (origin), *tyyppiin* (type) ja *toimintotapaan* (mode) (kuva 2.3). Ensimmäinen taso eli alkuperä kertoo ensimmäisen elinkaaren toiminnon, jossa puutteen synty olisi voitu ehkäistä. Alkuperästä riippuen määräytyy puutteelle tyyppi, joka kertoo puutteen paikan. Toimintotapa ilmaisee toi-

minnon, jonka vuoksi puute on syntynyt eli miksi puute on syntynyt. Esimerkiksi puute voi olla suunnittelupuute, joka on käyttöliittymässä ja se on kuvattu sisäisessä määrityksessä puuttuvaksi (Freimut, 2001; Fredericks ja Basili, 1998; Huber 1999).

Hewlett-Packardin mallin avulla kehittäjäryhmä kirjaa, minkä tyyppiset puutteet esiintyvät useimmin ja missä moduuleissa. Tietojen pohjalta on helppoa tunnistaa ne moduulit, jotka tarvitsevat enemmän testausta tai uudelleen kirjoitettua koodia. Puutteista voidaan kirjata myös prosessivaihe, jossa puute on esiintynyt. Edellisten projektien perusteella voidaan ennakoida myös uusien projektien trendejä (Fredericks ja Basili, 1998).

Grady (1992) esittää tapaustutkimuksen, jossa aineisto on kerätty eräästä Hewlett-Packardin projektista ennen testauksen alkua. Tarkoituksena oli löytää puutteet, joiden korjaaminen vie aikaa eniten eli kustannukset puutteiden korjaamiseksi maksavat eniten. Kuvassa 2.4 on esitetty datasta löytyneet puutteet lajiteltuna puutteen alkuperän mukaan.



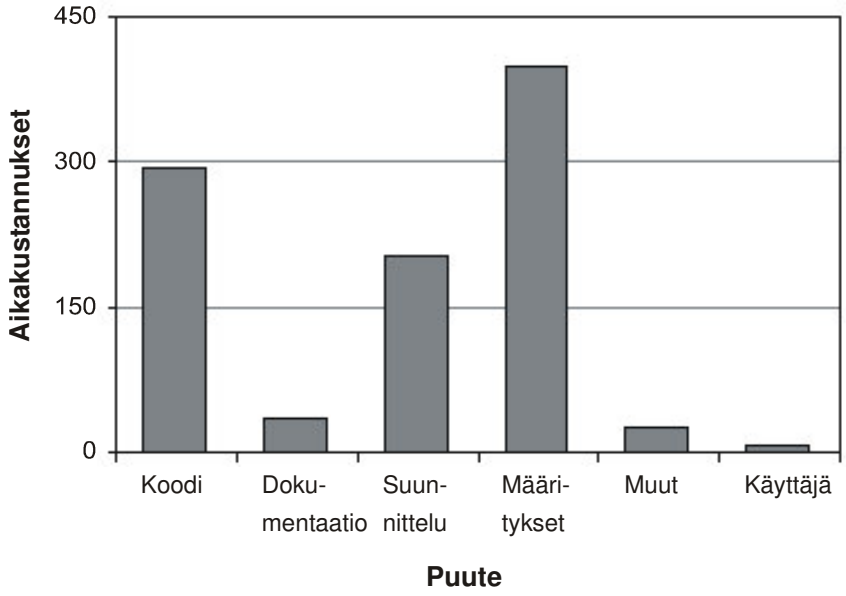
Kuva 2.4. Esimerkkiaineiston puutteiden lukumäärä[1].

Toisessa vaiheessa Gradyn tutkimuksessa keskitytään puutteen aiheuttamiin korjauskustannuksiin. Nettokustannus jokaiselle luokalle (alkuperä) lasketaan kaavalla (2.1).

[1] Kuvaajat 2.4-2.6 on piirretty MS Excel-ohjelmistolla Gradyn (1992) aineistosta

Puutteiden määrä * keskimääräinen kustannus puutteiden korjaamiseksi **(2.1)**

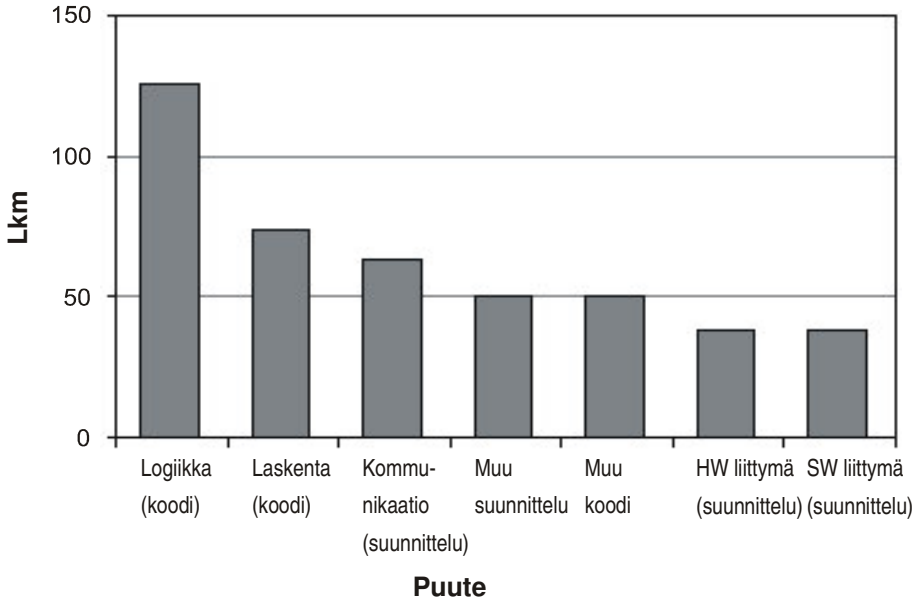
Esimerkkitapauksessa keskimääräisinä korjauskustannuksina käytetään keskimääräistä aikaa, joka korjaukseen kuluu laskettuna muista projekteista, koska tässä projektissa ei näitä tietoja ole saatavilla. Jos puute löytyy aikaisessa kehitysvaiheessa, esimerkiksi määritys- tai suunnitteluvaiheessa, tulevat korjauskustannukset huomattavasti pienemmiksi kuin jos puute löytyy esimerkiksi vasta testausvaiheessa. Keskimääräinen korjaus puutteelle, joka löydetään vasta testauksessa maksaa Gradyn mukaan 2.5 kertaa niin paljon kuin jos puute olisi löydetty jo koodausvaiheessa. Vastaavasti suunnittelun painoarvoksi tulee 6.25 ja määritysten painoarvoksi 14.25. Puutteita olisi siis helpompi ja edullisempi korjata määritys– ja suunnitteluvaiheessa kuin koodaus- ja testausvaiheissa. Gradyn tutkimuksessa dokumentaation, käyttäjän ja muun alkuperän painoarvoksi tulee 1. Kuvassa 2.5 näemme miten painotuksen jälkeen puutteet alkuperän mukaan sijoittuvat asteikolle. Määrityksessä tehtyjen puutteiden korjaukseen kuluu eniten aikaa, kun puute havaitaan testausvaiheessa.



Kuva 2.5. Puutteiden korjaukseen kuluva aika (kustannukset) painotettuna jakaumana.

Korjausajat normalisoidaan siten, että koodipuutteen korjaamiseen aikaa kuluu yksi tunti. Painotetut ajat muuttuvat tällöin varsinaisiksi korjausajoiksi jokaiselle puutekategorialle. Kuvassa 2.6 esitetään koodin ja suunnittelun painotetut jakaumat puute-

tyypeittäin. Siitä huomataan, että huomiota pitäisi kiinnittää logiikka-, laskentavirhe- ja prosessin kommunikaatiopuutteiden poistamiseen jo ennen testausvaihetta.



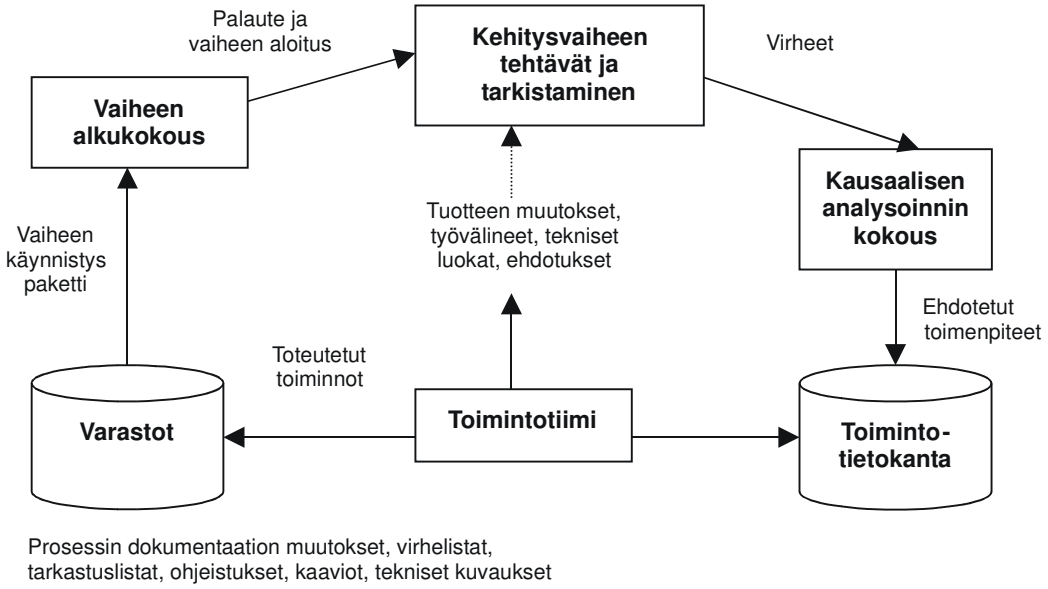
Kuva 2.6. Koodi- ja suunnittelupuutteiden painotettu jakauma.

Jo tämä pieni esimerkkitapaus osoittaa, että puutetietoja analysoimalla voidaan prosessista löytää parannuskohteita.

2.5.2 IBM DPP

DPP (Defect Prevention Process) puutteiden ehkäisymallin tarkoitus on nimensä mukaisesti ohjelmistopuutteiden ehkäisy. Puutteiden esiintyminen pyritään eliminoimaan analysoimalla vikoja eli pyritään ymmärtämään puutteiden syntytapa. Tätä kautta pyritään estämään samanlaisten puutteiden synty uusissa projekteissa (Fredericks ja Basili, 1998).

DPP:ssä on neljä päätoimintoa: kausaalisen analysoinnin kokous, toimintotiimi, alkukokous sekä datan keräys ja jäljitys. Kuvassa 2.7 on esitetty päätoimintojen väliset yhteydet.



Kuva 2.7. DPP–prosessi (Mays, 1990; Fredericks ja Basili, 1998)

Projektiryhmä pitää *kausaalisen analysoinnin kokouksen* jokaisen kehitysvaiheen lopuksi, kun kaikki vaiheen puutteet on havaittu ja korjattu. Kokouksessa abstrahoidaan puute ja luokitellaan se. Puuteluokkia on neljä: huolimattomuus, koulutus, kommunikaatio ja kopiointi. Huolimattomuudesta on kyse, jos kehittäjä ei ole täysin keskittynyt ongelman yksityiskohtiin ja puute on aiheutunut tästä. Koulutuspuute johtuu kohdealuekoulutuksen puutteesta eli kehittäjä ei täysin ymmärrä mihin tarkoitukseen tuote on tulossa. Kommunikaatiopuute on puute, joka johtuu kommunikaation puutteesta kehittäjäryhmän sisällä tai on annettu vääränlaista informaatiota asioista. Kopiointipuute puolestaan on kehittäjän tekemä kirjaus tai muu virhe. Häiriön aiheuttaja ja häiriön esille tuoja sekä prosessivaihe häiriön syntyessä kirjataan myös kokouksessa. Kokouksessa keskustellaan myös miten tulevaisuudessa vastaavanlaiset puutteet olisi vältettävissä ja miten samantyyppiset puutteet löydetään ja poistetaan tästä tuotteesta. Samoin keskustellaan puutteiden trendeistä ja yleisyydestä. Keskusteluissa käydään läpi myös ryhmän kohtaamia ongelmia ja miten niitä voidaan helpottaa sekä annetaan vinkkejä seuraavan vaiheeseen, jos ongelmakohtia on jo osattu selvittää. Vastaukset kaikkiin näihin kysymyksiin toimivat toimintoehdotuksina (Mays, 1990; Frederics ja Basili 1998).

Toimintotiimi pitää säännöllisiä kokouksia arvioidakseen kaikki kausaalisista kokouksista tulevat ehdotukset. Toimintotiimiin kuuluu kehittäjiä, jotka osa-aikaisesti varmistavat, että ehdotetut toiminnot tapahtuvat. Jokaisella jäsenellä on oma toimialueensa eli joku vastaa dokumentoinnista ja tarkastuslistoista, toinen koulutuksesta jne. (Mays, 1990; Frederics ja Basili 1998).

Palautteen anto on tärkeä osa DPP–prosessia. Kehittäjille on tärkeää, että johto ottaa heidän ehdotuksensa vakavasti ja että he voivat tätä kautta vaikuttaa kehitysympäristöön. Toisekseen palautteen avulla kehittäjät saavat tietoa mahdollisista muutoksista prosessissa ja vaatimuksista sekä tietoa muiden löytämistä virheistä, etteivät itse tekisi samanlaisia virheitä (Mays, 1990; Frederics ja Basili 1998).

Vaiheiden alkukokouksissa ryhmää valmistellaan uuteen prosessivaiheeseen. Niissä annetaan palautetta edellisestä vaiheesta ja evästetään kehittäjiä prosessin päivityksillä, jotka toimintotiimi on tehnyt. Alkukokouksessa johto selvittää kehittäjille vaiheen tehtävät ja vaatimukset. Tämän jälkeen jaetaan tehtävät ja tarkastuskohteet kehittäjille eli kehittäjät keräävät dataa ja jäljittävät puutteita (Mays, 1990; Frederics ja Basili 1998).

Toimintotietokantaa ja avustavia työkaluja tarvitaan jäljittämään ehdotettuja toimintoja, jotka toimintotiimi on käsitellyt. Toiminto käy läpi tutkimus-, toteuttamis- ja lopettamisvaiheet, jotka ovat toimintotiimin vastuulla (Mays, 1990).

2.5.3 IBM ODC

IBM:n tutkimusyksikkö julkaisi vuonna 1992 ortogonaalisen puutteiden luokittelumenetelmän (Orthogonal Defect Classification, ODC). Se on prosessin sisäinen laadunvalvontamenetelmä, jonka alkuperäisenä tavoitteena oli luoda perusta puutetietojen analysointiin ja palautteen saamiseen ohjelmistoprosessista. Näiden avulla pyrittiin suunnittelun ja ohjelmoinnin laadun parantamiseen proseduraalisessa ohjelmointiympäristössä (IBM, 2002b). Luokittelua on kehitetty sittemmin edelleen muun muassa huomioimaan olio-ohjelmoinnin tarpeita. Menetelmää käyttävät IBM:n ohella myös mm. Bellcore, Lucent, Motorola, Nortel ja Tandem (Freimut, 2001; IBM, 2002a).

Ohjelmistoprosessin parantamiseen tähtäävän luokittelun täytyy soveltua erilaisiin prosesseihin. Luokittelun kehittämisessä tehtävä yleinen virhe on luokittelun kehittäminen tietyn prosessin tarpeisiin ja sen jatkokäytön jääminen toissijaiseksi ja puutteelliseksi. ODC on luokittelumenetelmä, joka on käyttökelpoinen erilaisissa prosesseissa, sillä se on tuotos- ja organisaatioriippumaton. ODC ei ole jäykkä luokittelumenetelmä, vaan se sallii prosessin muuttumisen mittausjärjestelmää muuttamatta.

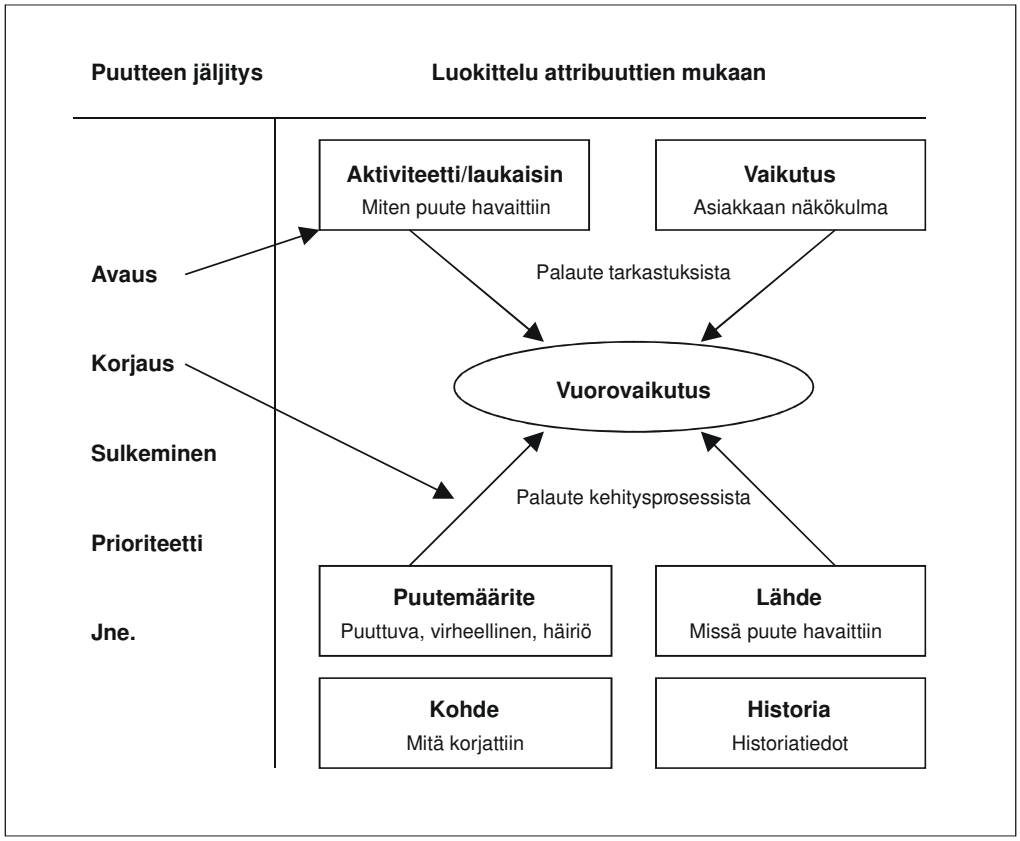
ODC:n perusajatuksena on luokitella puute attribuuteilla, jotka yhdessä osoittavat tiettyyn, huomiota vaativaan prosessivaiheeseen. Menetelmän kehittämisen pääarkkitehti Chillarege (1992) vertaa tätä karteesiseen xyz-koordinaatistoon, jossa eri koordinaatit osoittavat pisteen paikan kolmiulotteisessa tilassa. ODC:ssä puutetyypit ovat siis erillisiä ja toisistaan riippumattomia.

Puutetyyppien arvot ovat käyttökelpoisia kaikissa prosessin vaiheissa ja ne ovat prosessista riippumattomia, mutta yleisiä käytetylle toiminnolle. Puutteiden kautta saadaan tietoa sekä prosessista että tuotoksesta ja näiden ongelmakohdista. Tietoa voidaan hyödyntää projektinhallinnassa, ennusteiden laadinnassa, analyyseissä ja arvioinneissa. Saatava palaute on tärkeää myös laaduntarkkailussa ja ohjelmointimenetelmien tutkimuksessa (Chillarege, 1994).

2.5.3.1 Ominaisuuksia

ODC–luokittelu on yhtenäinen ja riippumaton. Se on yksinkertainen ja hyödyntää puutteiden sisältämää semantiikkaa. Luokittelun yhtenäisyys prosessivaiheissa mahdollistaa trendien havaitsemisen koko prosessissa ja sen vaiheissa. ODC on tuotoksen ja organisaation ominaisuuksista riippumaton ja on näin ollen laajalti käytettävissä. Luokittelu on käyttökelpoinen erilaisissa prosesseissa em. piirteistä (yhtenäisyys ja riippumattomuus) johtuen (Chillarege, 1992, Chillarege & al., 1992).

Luokittelu on yksinkertainen, millä pyritään minimoimaan yleiset luokitteluongelmat, kuten inhimilliset virheet ja valinnan vaikeus eri luokkien välillä tai valmiin aineiston tulkinta. Luokkien vähäinen määrä helpottaa ja täsmentää oikean luokan valintaa (Chillarege, 1992). Kuvassa 2.8 esitetään puutteen jäljitystä ja luokittelua.



Kuva 2.8. ODC-luokittelu (Chillarege, 2000)

Luokittelumenetelmän luokkien on oltava luonteeltaan prosessin eri vaiheissa riittävästi jaottelevia ja samanaikaisesti vaiheen kattavia siten, että eri puutetilanteet voidaan täsmällisesti luokitella. Liian tarkka jako on luokittelijalle turhauttava ja liian epätarkka jako ei vastaa tarkoitustaan puutteista halutun tiedon kannalta. Luokittelun edellytykset ovat siis riippumattomuus (orthogonality), johdonmukaisuus prosessin vaiheiden välillä (consistency across phases) ja yhtenäisyys prosessin vaiheiden välillä (uniformity across phases).

Puutteista saadaan merkityksellistä tietoa niiden semantiikkaa tarkastelemalla. Etenkin attribuutit `laukaisin`, `puutetyyppi` ja `puutemäärite` ovat hyvin semanttisia. Semanttinen tieto on luonteeltaan kvalitatiivista ja ODC:n luokittelukeinoin se saadaan kvantitatiiviseen, mitattavissa olevaan ja jatkokäsittelyyn (analyysit, menetelmät) soveltuvaan muotoon. Yksinkertainen luokitus helpottaa tarkastelua. Ohjelmointimalliin tiukasti kytköksissä oleva luokittelun semanttinen luonne ohjaa prosessin parantamiseen.

Puutetyyppien jakauma eri prosessin vaiheissa indikoi tuotteen kypsyystasoa. Koska prosessin tarkoituksena on tuotoksen kypsyminen prosessin edetessä, niin puutetyyppien jakauman tulisi muuttua etenemisen myötä. Jos ohjelmisto on järjestelmätestivaiheessa, mutta puutteiden jakauma on samankaltainen kuin aiemmissa prosessivaiheissa ja puutteiden määrä on merkittävä, niin se viittaa ongelmaan. Odotetusta jakaumasta poikkeavan puutetyypin perusteella voidaan tunnistaa myös ongelman mahdollinen syy. Jakaumaa voidaan tarkastella minkä tahansa kahden kehitysvaiheen jälkeen, jolloin siitä saatava palaute on nopeasti tuotteen kehittäjillä.

2.5.3.2 Attribuutit

ODC–luokittelussa attribuutteja on keskimäärin kahdeksan, riippuen projektista ja sen tarpeista. Myös attribuuttien arvot ovat määriteltävissä projektista riippuen. Kahdeksan ODC–luokiteltua attribuuttia ovat: `aktiviteetti` (activity), `laukaisin` (trigger), `vaikutus` (impact), `kohde` (target), `lähde` (source), `historia` (age), `puutetyyppi` (defect type) ja `puutemäärite` (defect qualifier). ODC–luokittelussa attribuutit jaetaan kahteen ryhmään prosessivaiheesta riippuen. Vaiheet ovat `havaitsemisvaihe` (phase found) ja `korjausvaihe` (phase fixed). Taulukoissa 2.1 ja 2.2 on esitelty ODC–mallia tarkemmin.

Taulukko 2.1. Havaitsemisvaiheen ODC–luokittelumalli (IBM, 1995).

Attribuutti	**Attribuutin merkitys**	**Attribuutin arvot**
Aktiviteetti (activity)	Todellinen aktiviteetti, jota suoritetaan, kun puute havaitaan.	Suunnitelman tarkastus (design inspection), koodin tarkastus (code inspection), yksikkötesti (unit test), toimintotestaus (function test), integraatiotesti (integration test), järjestelmätesti (system test)
Laukaisin (trigger)	Ympäristö tai ehto, jonka täytyy olla olemassa puutteen paljastumiseksi (vaaditaan myös puutteen uudelleen ilmestymiseen).	Tarkastuslaukaisimet (inspection triggers), yksikkötestauslaukaisimet (unit test triggers), toimintotestauslaukaisimet (function test triggers) ja järjestelmätestauslaukaisimet (system test triggers) ks. Liite 1.
Vaikutus (impact)	Vaikutus, jonka luullaan aiheutuvan käyttäjälle, jos puute jää ohjelmistoon.	Asennettavuus (installability), palvelevuus (serviceability), standardi (standard), eheys/turvallisuus (integrity/security), muutos (migration), luotettavuus (reliability), suorituskyky (performance), dokumentaatio (documentation), vaatimukset (requirements), ylläpito (maintenance), käytettävyys (usability), kyvykkyys (capability) ks. Liite 1.

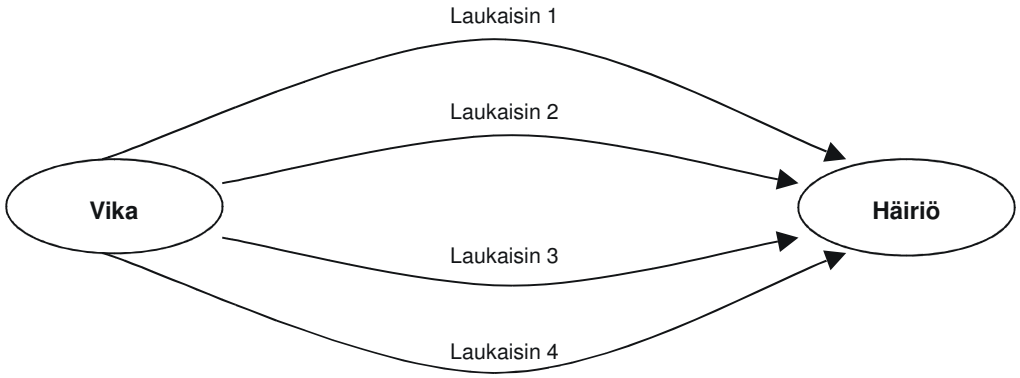
Taulukko 2.2. Korjausvaiheen ODC–luokittelumalli (IBM, 1995).

Attribuutti	Attribuutin merkitys	Attribuutin arvot
Kohde (target)	Korjattu kohde.	Vaatimukset (requirements), suunnittelu (design), koodi (code)
Lähde (source)	Puutteen sisältävän suunnitelman tai koodin alkuperä	Itse kehitetty (developed in-house), uudelleen käytettävästä kirjastosta (reused from library), ulkoistettu (outsourced), siirretty (ported) ks. Liite 1.
Historia (age)	Puutteen sisältävän suunnitelman tai koodin historia	Pohja (base), uusi (new), uudelleenkirjoitettu (rewritten), uudelleenkorjattu (refixed) ks. Liite 1.
Puutetyyppi (defect type)	Tehdyn korjauksen luonne	Sijoitus/alustus (assignment/initialisation), tarkistus (checking), algoritmi/metodi (algorithm/method), funktio/luokka/olio (function/class/object), ajoitus/sarjallistaminen (timing/serialization), rajapinta/oo-viestit (interface/OO messages), yhteys (relationship) ks. Liite 1.
Puutemäärite (defect qualifier)	Kertoo, johtuuko puute: Jonkin tärkeän osan puuttumisesta Virheellisestä toiminnasta Ulkoisesta häiriöstä	Puuttuva (missing), virheellinen (incorrect), häiriö (extraneous) ks. Liite 1.

Aktiviteetti havaitsemisvaiheen attribuuttina viittaa ohjelmistonkehitysvaiheeseen, jossa puute havaittiin. Tällaisia vaiheita ovat taulukon 2.1 mukaisesti esimerkiksi tuotteen tarkastus, yksikkötestaus, toimintotestaus, integraatiotestaus ja järjestelmätestaus. Esimerkiksi järjestelmätestauksen aikana puute voi ilmetä kun painetaan tulostuspainiketta. Vaihe, jossa puute havaitaan on järjestelmätestaus, mutta aktiviteetti on toimintotestaus, koska puute on toimintotestaustyyppiä (Kan, 2003).

Jokaisen yrityksen olisi luotava omanlaisensa aktiviteettilistaus omia toimintatapoja vastaamaan. Joillakin tuotteilla testaus- ja tarkastusvaiheita määritellään enemmän, esimerkiksi turvallisuuskriittisillä ohjelmistotuotteilla. Pienet ja yksinkertaisemmat ohjelmistotuotteet selvinnevät vähemmilläkin tarkastus- ja testausmenettelyillä.

Laukaisin havaitsemisvaiheen attribuuttina ilmaisee, mikä ympäristö tai tila on oltava puutteen ilmaantumiseksi. Laukaisimien avulla voidaan siis paikallistaa todellinen aktiviteetti, jota suoritettiin, kun puute havaittiin. Kuva 2.9 havainnollistaa eri laukaisimia, jotka pakottavat vian häiriöksi (Chillarege, 1992).



Kuva 2.9. Laukaisimet (Chillarege, 2000)

Laukaisimen arvo kuvastaa tarkastusprosessin täydellisyyttä. Tarkastusprosessi voi olla koodin testausta tai tuotteen tarkastusta ja katselmusta. Esimerkiksi jos koko puute-alueen laukaisinjakauma ei ole samantyyppinen kuin järjestelmätestauksen laukaisinjakauma, viestii se ongelmista testauksessa (Chillarege, 1992). Todennäköisesti tällöin ei ole huomioitu kaikkia mahdollisia testitapauksia.

Laukaisimet voidaan taulukon 2.1 mukaisesti jakaa neljään ryhmään sillä perusteella missä prosessivaiheessa laukaisin on havaittu (IBM, 2002a): tarkastus-, yksikkötestaus-, toimintotestaus- ja järjestelmätestauslaukaisimet. Laukaisimet ja niiden merkitykset on esitelty tarkemmin liitteessä 1.

Butcherin & al. (2002) mukaan ohjelmiston testauksen parantamisesta ODC:n avulla on saatu hyviä kokemuksia. He esittävät esimerkkitapauksen ohjelmistotuotteesta, jonka kehittämisprojektissa ODC:tä käyttämällä löydettiin erilaisia puutteita ja parannuskohteita prosessiin. Suurin osa puutteista löytyi toimintojen testausvaiheessa, jolloin `testivaihtelevuus laukaisimena` oli yleisin puute. Tämän perusteella projektiryhmä koulutti testaajia lisää testivaihtelevuuden osalta. Myös testien määrään kiinnitettiin huomiota eli varmistettiin, että testivaihtelevuutta mittaavia testejä oli riittävästi.

Vaikutus havaitsemisvaiheen attribuuttina kuvaa, mitä puute mahdollisesti aiheuttaa tuotteen loppukäyttäjälle, jos se jää tuotteeseen. Vaikutuksia ovat taulukon 2.1 mukaisesti esimerkiksi asennettavuus, käytettävyys ja kyvykkyys. Tarkemmin vaikutuksen

merkitykset on selitetty liitteessä 1. Esimerkiksi asennettavuus tarkoittaa, kuinka helposti asiakas pystyy asentamaan ja ottamaan ohjelmiston käyttöön.

Kohde korjausvaiheen attribuuttina kertoo korjatun kohteen eli korjatun puutteen. Kohteena oleva puute voi taulukon 2.1 mukaisesti liittyä vaatimuksiin, suunnittelun tuotoksiin tai ohjelmakoodiin.

Lähde korjausvaiheen attribuuttina on puutteen sisältävän suunnitelman tai koodin alkuperä. Lähteitä ovat taulukon 2.2 mukaisesti itse kehittäminen, uudelleen käytettävä kirjasto, ulkoistaminen ja toisesta ympäristöstä siirtäminen. Tarkemmin lähteiden merkityksistä on kerrottu liitteessä 1. Esimerkiksi itse kehittäminen tarkoittaa puutteen löytyneen alueelta, joka oli organisaation oman kehitysosaston kehittelemää ja ulkoistaminen tarkoittaa puutteen löytyneen osiosta, joka on saatu ulkopuoliselta toimittajalta.

Historia korjausvaiheen attribuuttina kertoo nimensä mukaisesti puutteen historian. Taulukon 2.2 mukaisesti pohja, uusi, uudelleen kirjoitettu ja uudelleen korjattu ovat historia-attribuutin arvoja. Tarkemmin attribuuttiarvojen merkityksistä on kerrottu liitteessä 1. Esimerkiksi pohja tarkoittaa, että puute on osa tuotetta, jota ei ole muutettu ko. projektissa eikä se ole osa uudelleen käytettyä kirjastoa.

Puutetyyppi korjausvaiheen attribuuttina voi Chillaregen (1994) mukaan taulukon 2.2 mukaisesti kuvata korjauksen tarkoitusta. Puutetyyppi ei ole puutteen aiheuttaja, vaan toimenpide, joka vaaditaan ongelman korjaamiseen. Ohjelmoijan kannalta katsoen puutetyypit yhdessä kuvaavat työn luonteen.

Puutetyyppejä oli alkujaan ODC:ssä viisi ja menetelmän kehittymisen myötä lukumäärä on lisääntynyt kahdeksaan (IBM, 2002a). Lisätyt luokat eivät ole vaikuttaneet alkuperäisin luokkiin vaan ne laajentavat luokittelun kattamaan alkuperäistä pilottitutkimusta laajempien prosessien mekanismeja. Puutetyypit on esitelty tarkemmin liitteessä 1.

Puutetyypit ovat riittävän yleisiä eri ohjelmistojen kehitystyöhön eli ne ovat tuotoksesta riippumattomia. Jaottelu on sopivan yleinen ja kattava kaikkiin prosessivaiheisiin, vastaten kuitenkin riittävästi eri vaiheiden spesifisiä vaatimuksia (Chillarege, 1992). Eri

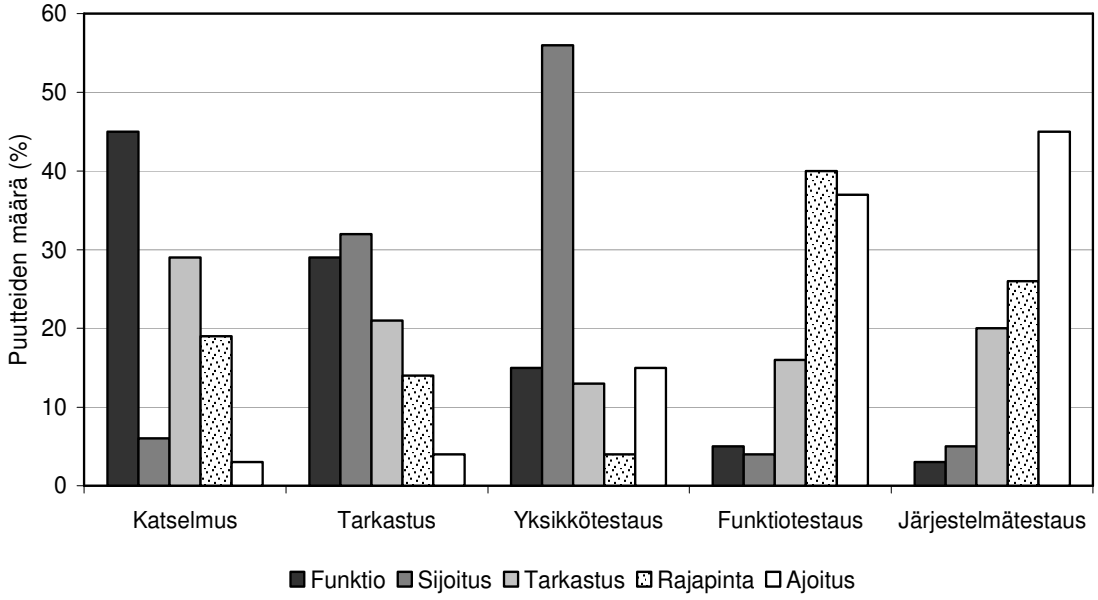
puutetyypit osoittavat löydettäessä eri vaiheisiin kehitystyössä. Puutetyyppi `funktio` assosioituu korkean tason suunnitteluvaiheeseen riippumatta siitä löydetäänkö se järjestelmä- vai yksikkötestivaiheessa. Puutetyyppi `rajapinta` puolestaan yhdistyy matalan tason suunnitteluun ja puutetyyppi `tarkastus` matalan tason suunnitteluun tai koodaukseen. Puutetyyppi `sijoitus` assosioituu koodaukseen, ja puutetyyppi `ajoitus/sarjallistaminen` puolestaan matalan tason suunnitteluun. Puutetyyppi `rakenne/paketti/lomitus` liittyy kirjastovälineisiin, puutetyyppi `dokumentaatio` julkaisuihin ja puutetyyppi `algoritmi` matalan tason suunnitteluvaiheeseen (Kan, 2003). Ohjelmointiprosessissa, jossa koodaus- ja testausvaiheet seuraavat suunnitteluvaihetta, on oletettavaa, että esimerkiksi puutetyyppiä `funktio` löydetään prosessin aikaisissa vaiheissa ja ihannetapauksessa hyvin vähäisiä määriä järjestelmätestauksessa. Toisaalta puutetyyppejä `ajoitus` ja `sarjallistaminen` löytyy oletettavasti enemmän järjestelmätestauksessa kuin aiemmissa vaiheissa. Puutetyyppien jakauma on edellä mainituista syistä johtuen erilainen ja ennustettavissa eri prosessivaiheissa (Chillarege, 1992).

Puutemäärite korjausvaiheen attribuuttina kertoo taulukon 2.2 mukaisesti, johtuuko puute jonkin tärkeän osan puuttumisesta, virheellisestä toiminnasta vai ulkoisesta häiriöstä. Tarkemmin puutemääritteen attribuuttien arvoista on kerrottu liitteessä 1. Esimerkiksi puuttuva tarkoittaa, että puute johtui laiminlyönnistä ja häiriö tarkoittaa, että puute johtui jostain dokumentin tai koodin kannalta epäolennaisesta tai asiaankuulumattomasta asiasta.

2.5.3.3 Tulkinta

Bassin & al. (1998) esittävät erään tulkinnan ODC-luokitellulle datalle käyttäen ns. Butterfly-mallia (Butterfly Model). Malli on nimetty kaaosteoriassa esitellystä ajatuksesta, joka linkittää Shanghaissa tapahtuvan perhosen siivenlyönnin Pohjois-Amerikassa syntyvään pyörremyrskyyn. Analogia ohjelmistotuotantoon pohjautuu monen ohjelmistoprosessien kaoottiseen luonteeseen. Kaoottisissa järjestelmissä pienetkin muutokset aloitusehdoissa voivat johtaa suuriin muutoksiin evoluution myötä.

Keskeisenä piirteenä Butterfly-mallissa ovat prosessin kuvaajat, joita Bassin & al. kutsuu signatuureiksi (signatures). Esimerkiksi puutetyyppi-signatuuri ilmentää prosessin stabiliteettia suhteessa prosessin etenemiseen. Edellistä täydentävä laukaisin-signatuuri auttaa arvioimaan prosessin toimenpiteiden tehokkuutta puutteiden paljastamisessa. Bassinin & al. (1998) mukainen esimerkkikuva 3.9 esittää keskuskoneen käyttöjärjestelmän osan puutetyyppi-signatuuria. Kyseinen järjestelmä on korkean kypsyystason tuotos ja sen laadunvalvonta on hyvin johdettua. Käsiteltävässä prosessissa on viisi aktiviteettia: suunnittelukatselmus, koodin tarkastus, yksikkötestaus, funktiotestaus ja järjestelmätestaus.



Kuva 3.9. Keskuskoneen käyttöjärjestelmän puutetyyppi-signatuuri[2].

Kuvasta 3.9 havaitaan, että suunnittelukatselmuksessa havaituista puutteista suurin osa on tyyppiä funktio ja toisaalta järjestelmätestauksen aikana sen osuus on pieni. Tilanne on toivotunlainen koska puutetyypin funktio (ilmentäen korkean tason suunnittelua ja vaatien yleensä laajoja muutoksia koodiin) osuuden tulisi laskea siirryttäessä prosessissa eteenpäin. Järjestelmätestausta ei ole kovin toivottavaa tehdä tuotokselle, joka tarvitsee laajoja suunnittelumuutoksia.

[2] Kuvaaja on piirretty MS Excel-ohjelmistolla Bassinin & al. (1998) aineistosta

`Ajoitus`-puutetyypin suhteellinen osuus on puolestaan suurin järjestelmätestauksessa, jossa koko järjestelmä on kokeiltavana jäljiteltäessä todellisia olosuhteita ja työkuormaa. Sen osuus on minimaalinen koodin tarkastuksessa, jolloin on hyvin hankalaa tarkastella kyseisen tyyppisiä virheitä. Esimerkin prosessin puutejakaumat ovat pitkälle odotetunlaiset ja Bassin & al. (1998) havaitsivat tuotoksen suunnittelun stabiliteetin kohenevan prosessin edetessä.

Vaikka signatuurit määritellään tavallisesti aiemman, historiallisen datan perusteella, niin ne edustavat odotettua suhteellista jakaumaa tulevissa tuotoksissa, kun käytetään samaa prosessia ja aktiviteetteja. Odotetun signatuurin avulla voidaan kvantitatiivisesti arvioida poikkeamat ja niihin liittyvät riskit projektissa sekä tunnistaa poikkeuksen luonne nopeasti. Muuttumattomasta prosessista signatuurien vangitseminen on yksinkertaista ja helppoa kunhan historiallista dataa on saatavilla.

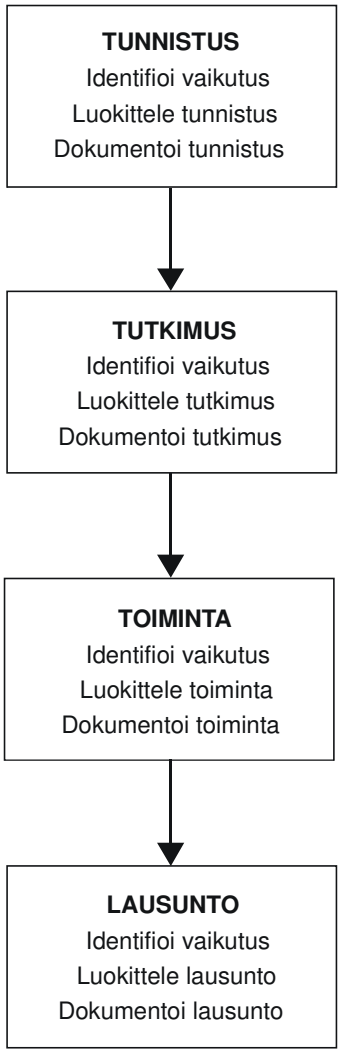
Verrattaessa alkuperäistä aiottua signatuuria todelliseen voidaan Bassinin & al. (1998) mukaan nopeasti arvioida ja määrittää kustannukset, jotka liittyvät tuotoksen laatuun, tuottavuuteen ja poikkeavuuksista johtuviin aikataulumuutoksiin. Syntynyttä todellista signatuuria voidaan lopuksi käyttää arvioitaessa prosessin jatkon tehokkuutta ja syntyvän tuotoksen stabiilisuutta.

2.5.4 IEEE–standardin mukainen puuteluokittelu

IEEE–malli on tarkoitettu kahteen tarkoitukseen. Se on tarkoitettu henkilöille, jotka työskentelevät projekteissa, joissa IEEE–standardia käytetään. Toisaalta sitä voivat hyödyntää henkilöt tai projektit, jotka haluavat laajentaa puutteiden tarkkailua tai puuteluokittelumallia. Mallia voidaan käyttää myös todistamaan joidenkin menetelmien toimivuutta (Freimut, 2001; IEEE, 1993; IEEE, 1995).

IEEE–standardissa puuteprosessilla on neljä askelmaa. Jokaisella askelmalla täytyy tehdä kolme toimintoa: merkata tieto, luokitella puute ja identifioida puutteen vaikutus. Kuvassa 2.10 on havainnollistettu tätä prosessia. Ensimmäinen askelma on *tunnistus* (recognition), joka tapahtuu kun puute on löydetty. *Tunnistuksen vaikutukset* voidaan kirjata tunnistus vaiheessa. Toisessa vaiheessa eli *tutkimuksessa* (investigation) puute

tutkitaan identifioimalla kaikki ne asiat, joita ei tiedetä ja ehdottamalla niihin ratkaisuja tai kirjataan, että puute ei vaadi toimia. *Tutkimuksen vaikutukset* voidaan kirjata tutkimusvaiheessa. *Toiminnan* (action) tarkoituksena on löytää ratkaisu puutteen poistamiseksi ja ehkäistä myös sen myöhemmät esiintymiset. Neljäs vaihe eli *lausunto* (disposition) tehdään, kun kaikki vaaditut tehtävät puutteen poistamiseksi on tehty tai vähintäänkin pitkän aikavälin korjaavat toimenpiteet on identifioitu puutteen poistamiseksi (Freimut, 2001; IEEE, 1995).



Kuva 2.10. IEEE–luokitteluprosessi (IEEE, 1995; IEEE, 1993).

IEEE–standardissa puuteprosessit on jaettu pakollisiin kategorioihin. Niitä tarvitaan yleisten määritysten luomiseksi, yleisen terminologian luomiseksi ja kommunikaatio-

käsitteiksi projektien, yrityskulttuurien ja ihmisten välillä. Taulukossa 2.3 on esitelty puuteprosessit, kategoriat ja niiden merkitykset.

Taulukko 2.3. IEEE–puuteluokittelu (IEEE, 1995).

Puuteprosessi	**Kategoria**	**Merkitys**
Tunnistus (recognition)	Projektistatus (project status)	Mikä on tuotteen käytettävyys, jos puutetta ei korjata?
	Projektiaktiviteetti (project activity)	Mitä tehtiin, kun puute ilmeni?
	Projektivaihe (project phase)	Missä vaiheessa elinkaarta oltiin projektissa?
	Oletettu syy (suspected cause)	Mistä puute saattaisi aiheutua?
	Toistuvuus (repeatability)	Voisiko puute esiintyä useammin kuin kerran?
	Oire/merkki (symptom)	Miten puute itse tuli ilmi?
Tunnistuksen vaikutus (recognition impact)	Käyttäjäarvo (customer value)	Kuinka tärkeää käyttäjän kannalta on korjata puute?
	Turvallisuus (mission safety)	Kuinka paha puute on projektin vrt. yleisen hyvinvoinnin kannalta?
	Vakavuus (severity)	Kuinka paha puute on?
Tutkimus (investigation)	Todellinen syy (actual cause)	Mikä aiheutti esiintymän?
	Lähde (source)	Missä on puutteen alkuperä?
	Tyyppi (type)	Minkä tyyppinen puute on kooditasolla?
Tutkimuksen vaikutus (investigation impact)	Prioriteetti (priority)	Puutteen tärkeys?
	Käyttäjäarvo (customer value)	Kuinka tärkeää käyttäjän kannalta on korjata puute?
	Turvallisuus (mission safety)	Kuinka paha puute on projektin vrt. yleisen hyvinvoinnin kannalta?
	Projektin aikataulu (project schedule)	Korjaamisen vaikutus aikatauluun
	Projektin kustannukset (project cost)	Korjaamisen aiheuttamat kustannukset
	Projektin laatu/luotettavuus (project risk)	Kuinka vaikuttaa projektin laatuun/luotettavuuteen?
Toiminta (action)	Ratkaisu (resolution)	Miten puutteen uudelleen tapahtuminen voidaan estää?
	Korjaava toimenpide (corrective action)	Miten korjataan puute?
Lausunto (disposition)	Lausunto (disposition)	Mitä todella tapahtui ?

IEEE–standardissa kategoriat täytyy luokitella. Jokaisella organisaatiolla voi olla omanlainen luokitus. Yksinkertaisimmillaan luokitus voisi olla esimerkiksi puutteen vakavuudelle kriittinen, keskinkertainen ja vähäinen. Taulukossa 2.4 on esitelty erään kaupallisen tuotteen käyttäjäarvon luokitus.

Taulukko 2.4. Esimerkki käyttäjäarvon luokittelusta (IEEE, 1995).

Luokitus	**Merkitys**
Korvaamaton (priceless)	Asiakas ei voi jatkaa ohjelmiston käyttämistä tai hän ei osta tuotetta ennen kuin puute on korjattu. (Käyttäjäarvo = 1)
Korkea (high)	Asiakas voi käyttää ohjelmistoa, mutta ei sen kriittisiä osia tai asiakas vakavasti harkitsee ostopäätöstä, ellei puutetta korjata. (Käyttäjäarvo = 2)
Keskinkertainen (medium)	Asiakas voi käyttää ohjelmistoa, mutta puute/puutteet ovat ärsyttäviä tai asiakas vaatii lupausta, että puute poistetaan seuraavasta versiosta. (Käyttäjäarvo = 3)
Alhainen (low)	Asiakas on hieman kiinnostunut puutteesta tai asiakas tekee ostopäätöksen puutteesta välittämättä (Käyttäjäarvo = 4)
Ei mitään (none)	Useat asiakkaat ovat huomanneet puutteen ja ilmoittaneet tästä mielipiteensä (Käyttäjäarvo = 5)
Haitallinen (detrimental)	Puute voi aiheuttaa negatiivisia vaikutuksia asiakkaalle tai rajoittaa myyntiä. (Käyttäjäarvo = 6)

Kategoriat voidaan myös luokitella tarkempiin toisen tason kategorioihin kategorian sisällä. Esimerkiksi prosessi voidaan jakaa kuvan 2.11 mukaisesti vaatimuksiin, suunnitteluun, toteutukseen, testaukseen, käyttöönottoon ja ylläpitoon sekä käytön lopetukseen (IEEE, 1995). Toisella tasolla voidaan määritellä luokkia tarkemmin, esimerkiksi vaatimuksiin kuuluvat toisen asteen luokat käsitteiden luominen, järjestelmävaatimukset, ohjelmistovaatimukset ja prototyypin vaatimukset. Näin saadaan tarkempaa tietoa puutteen sijainnista elinkaaressa ja voidaan paremmin tehdä korjaavia toimenpiteitä puutteen poistamiseksi. Alikategorioihin luokittelu ei kuitenkaan ole välttämätöntä.

Vaatimukset
Käsitteiden luominen
Järjestelmävaatimukset
Ohjelmistovaatimukset
Prototyypin vaatimukset
Suunnittelu
Järjestelmäsuunnittelu
Alustava suunnittelu
Tarkka suunnittelu
Toteutus
Koodi
Yksikkötestaus
Integraatio
Prototyyppi
Testaus
Integraatiotestaus
Järjestelmätestaus
Beetatestaus
Prototyyppitestaus
Hyväksymistestaus
Asennus ja tarkastus
Käyttö ja ylläpito
Käytönlopetus

Kuva 2.11. Prosessivaiheiden luokittelu (IEEE, 1995).

2.6 Puuteluokittelun käyttöönotto

Organisaatio voi ottaa käyttöönsä jo käytössä olevan puuteluokittelun ja mukauttaa sitä omien tavoitteidensa mukaiseksi tai sitten se voi luoda täysin uuden puuteluokittelu-mallin. Yleisimmät uudelleenkäytettävät mallit ovat IEEE-standardiluokittelu ja ODC–luokittelu. Käyttöönottoprosessi on muotoa (Freimut, 2001):

1. Valitaan attribuutit perustuen haluttuun standardiin, datan käyttöön ja organisaation arvoihin.

2. Määritellään jokaiselle attribuutille arvo/arvot. Tarkistetaan, että kaikki attribuutit ovat järkeviä. Luodaan uusia attribuuttiarvoja, jos se on tarpeellista ja poistetaan tarpeettomat.
3. Dokumentoidaan attribuutit. Kuvaillaan jokaisen attribuutin määritelmä ja tarkoitus käyttäen organisaation termejä ja viittauksia.
4. Dokumentoidaan attribuuttien arvot. Kirjataan tarkoin, mitä attribuutin arvo tarkoittaa.
5. Määritellään jokaiselle attribuutin arvolle, kuka sen kerää ja milloin.
6. Määritellään kuinka data kerätään.
7. Suunnitellaan käytettävät analysointimenetelmät.
8. Järjestetään luokittelumallin opastusta käyttäjille ja johdolle.

Chillaregen (1994) mukaan ODC:n käyttöönottokustannukset ovat pienet, etenkin jos käytössä on jo joku muutostenhallintajärjestelmä. Kustannukset koostuvat tällöin työvälineohjelmien modifioinnista ja henkilöstön koulutuksesta. ODC:n käyttöönotto vaatii yleensä vain pieniä laajennuksia jo olemassa olevaan järjestelmään. Käyttöönotto organisaatioissa, joissa ei ole aiempia muutostenhallinnan työkaluja käytössä, on työläämpää ja kustannuksiltaan suurempaa. Suurin muutos on tarvittava työkulttuurin muutos entisestä eli mitattavan puutetiedon ja sen tuomien etujen tarpeellisuuden tunnistaminen ja hyväksyminen osaksi organisaation työskentelytapoja.

Käytettäessä ODC:tä DPP-puutteiden ehkäisyprosessin tukena saadaan Chillaregen (1994) mukaan huomattavia säästöjä työaikakustannuksissa. DPP:ssä yksityiskohtaiseen perussyyanalyysiin kuluu noin yksi miestyötunti puutetta kohden, jolloin työhön sisältyy kvalitatiivinen analyysi ja mahdollinen puutteen ratkaisu ja ratkaisun dokumentointi. Suurien puutemäärien (tuhansia) kohdalla kustannukset tulevat suuriksi, jos kaikki puutteet käsitellään, ja näin ollen osa puutteista jää usein analysoimatta. ODC–luokitteluun kuluva aika on edelliseen verrattuna vähäinen, vain yhdestä neljään minuuttia puutetta kohden. Kustannukset puutetta kohden ovat näin ollen hyvin pienet. Prosessin puutteiden luokittelutyö saadaan siten kaikki puutteet kattavaksi suhteellisen pienin kokonaiskustannuksin. Analysointia voidaan kohdistaa yksittäisen puutteen si-

jaan puutejoukkoon ja tehdä kausaalianalyysiä syyn ja vaikutuksen suhteen. Tämä analyysi kattaa eri kehitystyön vaiheet ja pienentää näin kokonaiskustannuksia. Näin saavutettava säästö on huomattava perinteiseen perussyyanalyysiin verrattuna. Kustannukset ovat yleensä vain noin kymmenesosa jälkimmäisestä.

3 ATTRIBUUTTIFOKUSOINTI

Attribuuttifokusointi (AF, Attribute Focusing) on menetelmä, jolla voidaan esimerkiksi ODC:llä luokitellusta aineistosta havaita hyödyllistä tietoa prosessin parantamiseksi. Bhandarin & al. (1992) mukaan menetelmällä on seuraavia etuja:

- Mittaukset, puuteanalyysi ja prosessin korjaaminen tapahtuvat prosessin aikana. Prosessin muokkaaminen on näin ollen ajantasaista ja validoitavissa.

- Menetelmän soveltuvuus on hyvä, se perustuu kerätyn datan mielenkiintoisuuden (interestingness) mittaamiseen. Mittaukset voidaan määritellä ilman aikaisempia todennäköisyyksiä kertovia aineistoja, toisin kuin monissa muissa menetelmissä.

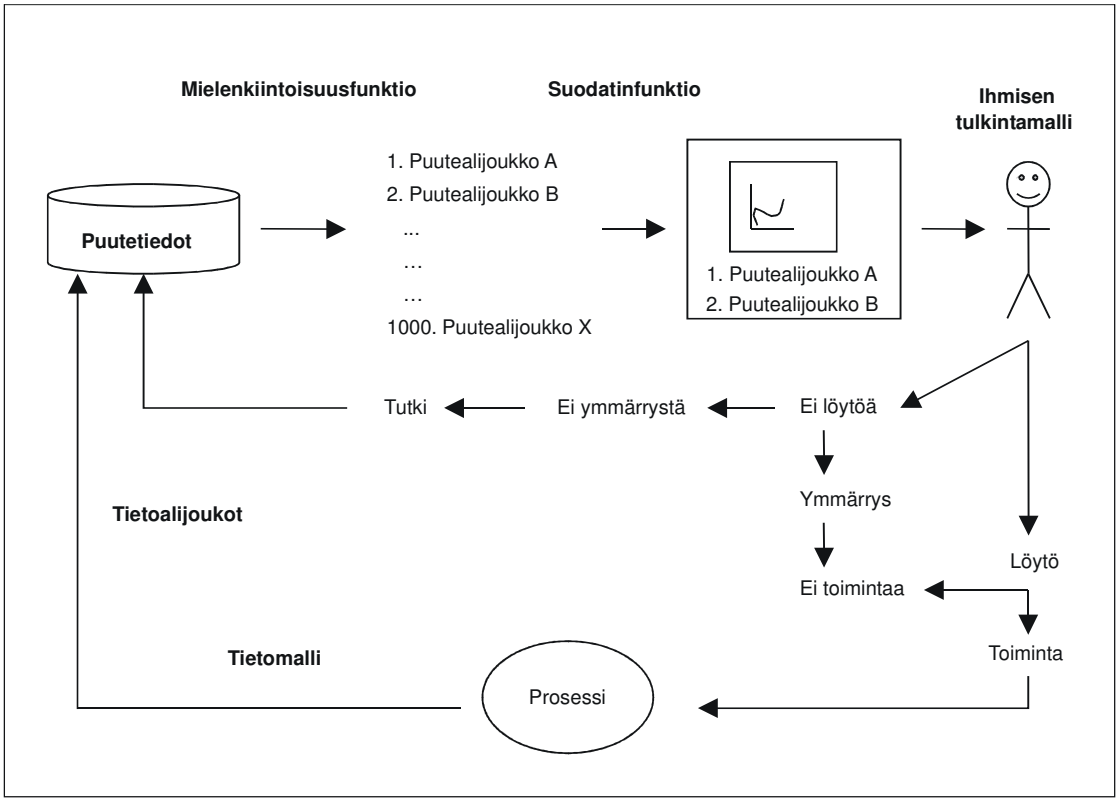
- Menetelmä on tehokas, se analysoi järjestelmällisesti dataa, joka on luokiteltu monen eri attribuutin suhteen.

- Menetelmä on automaation johdosta kustannustehokas verrattuna aiempiin prosessinaikaisiin puuteanalysointeihin ja korjaamisiin. Se ei edellytä erillistä projektiryhmään kuulumatonta eksperttiä datan analysointiin, vaan työ tehdään ryhmän sisällä. Käsiteltävän datan ymmärtäminen ja validointi helpottuu kun sen tekee datan tuottanut ryhmä, jolla on yksityiskohtaista tietoa prosessista ja tuotoksesta. Edellytykset prosessin ja tuotoksen korjaamiseen ovat näin olemassa ja ongelmien tunnistaminen, muutosten teko ja muutosten vaikutus ovat nopeasti nähtävissä. Prosessin parantaminen saadaan menetelmän keinoin osaksi projekti-ryhmän normaalia toimintaa.

Attribuuttifokusoinnin päämääränä on siis tuottaa järjestelmällistä tietoa usean attribuutin suhteen luokitellusta aineistosta. Laajan aineiston joukosta voidaan sen avulla poimia käyttäjän kannalta oleelliset tosiasiat. Analysoijalta ei edellytetä erityistaitoja aineistojen analysoinnista, mikä tekee menetelmästä helposti käytettävän (Bhandari & al., 1994).

3.1 Periaate

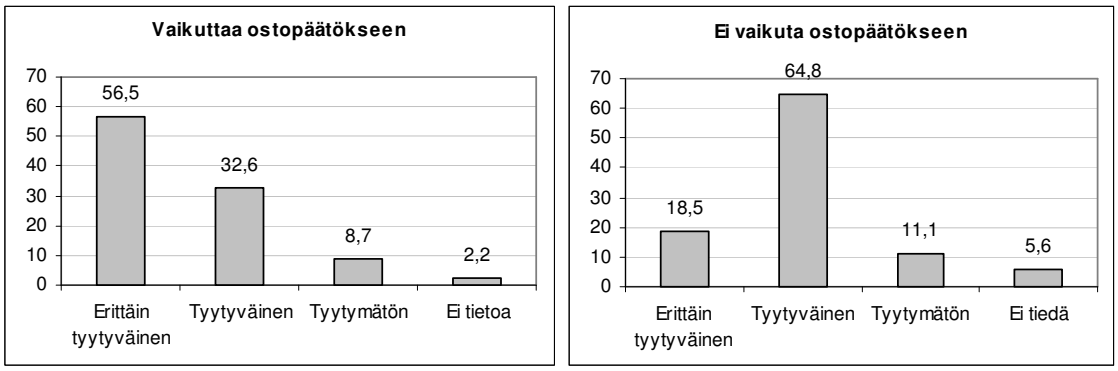
Attribuuttifokusointi voidaan tehdä koko prosessin ajalta tai vain osasta prosessia ja se voidaan aloittaa missä tahansa prosessin vaiheessa. Kuvassa 3.1 on esitetty attribuuttifokusoinnin pääkohdat. Attribuuttifokusointi pohjautuu kahteen tärkeään käsitteeseen: mielenkiintoisuusfunktioon ja suodatinfunktioon. Mielenkiintoisuusfunktio perustuu datan arvojen eli attribuuttien suhteiden käsittelyyn. Suodatinfunktio puolestaan muuttaa mielenkiintoisuusfunktion tuottaman tiedon ihmisen ymmärtämään muotoon.



Kuva 3.1. Attribuuttifokusointi ratkaisumalli (Bhandari & al., 1992)

Prosessista kerätään puutetietoja jonkun tietyn attribuuttiluokittelun mukaisesti. Puutetietojen tietomalliksi oletetaan relaatiomalli. Tietoalijoukon tietue edustaa prosessin aikana löydettyä puutetta, joka on luokiteltu eri attribuuttien suhteen. Esimerkiksi ODC–luokittelussa puutteesta tulisi tietää puutteen `aktiviteetti`, `laukaisin`, `vaikutus`, `lähde`, `historia`, `puutetyyppi` ja `puutemäärite`. Tietoalijoukko sisältää kaiken tiedon kaikista vaiheen tai alivaiheen aikana löytyneistä puutteista (Bhandari & al., 1994).

Luokitellut puutteet käsitellään automaattisesti mielenkiintoisuusfunktiolla, joka järjestää attribuuttiarvot kuvaamaan niiden potentiaalista mielenkiintoisuutta analysoijia (ihminen) varten. Suodatinfunktio käsittelee mielenkiintoisuusfunktion tuottaman datan ja esittää tuloksen analysoijalle sopivalla tavalla. Attribuuttiarvojen jakaumaa voidaan kuvata esimerkiksi pylväsdiagrammilla. Diagrammit on jaettu mielenkiintoisuusasteen mukaan. Se on numeerinen, laskettu arvo, joka kertoo kuinka mielenkiintoinen kukin diagrammi olisi analysoijasta. Kuva 3.2 kertoo, että jos asiakas on mukana tekemässä ostopäätöstä tuotteesta, on hän myös tyytyväisempi tuotteen ominaisuuksiin kuin asiakas, joka ei ole vaikuttamassa ostopäätökseen (Bhandari & al., 1994; Bhandari & al., 1992; Mendonça ja Basili, 2000).



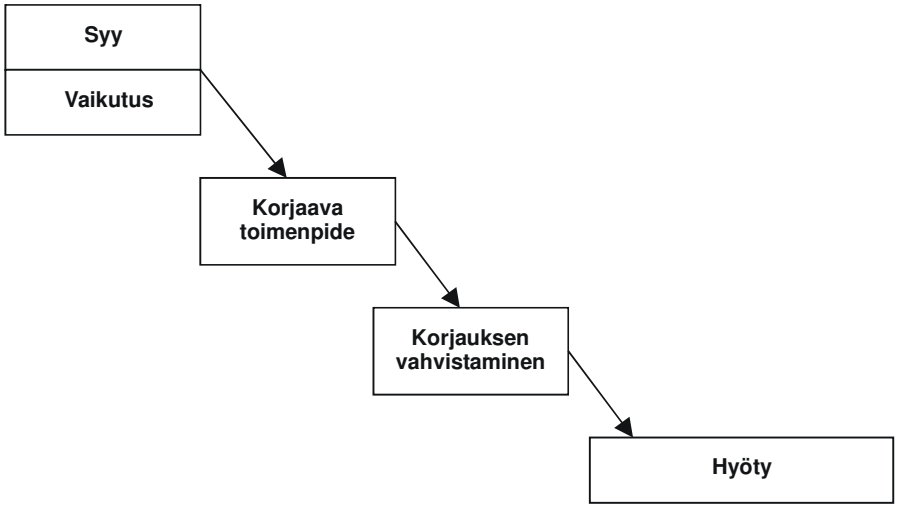
Kuva 3.2. Esimerkki suodatinfunktion suodattamista tuloksista

Jos mielenkiintoista puutetta selitettäessä löytyy uutta tietämystä, pystytään identifioimaan prosessin ongelma-alue ja määrittämään korjaavat toimenpiteet puutteen poistamiseksi. Jos uutta tietämystä ei löydy, voi se johtua kahdesta syystä. Mielenkiintoisuuden selitys voi olla jo analysoijan tiedossa, jolloin se ei enää kiinnosta häntä tai sitten analysoija ei ymmärrä tietoa mielenkiintoiseksi. Jälkimmäisessä tapauksessa on kyseessä niin sanottu *tutkinta-askel*, jolloin analysoijan täytyy tutkia mielenkiintoisuus myöhemmin. Mielenkiintoisuus perustuu attribuutin arvon suuruuteen tai assosiaatioon attribuuttiparin arvojen välillä. Analysoijan täytyy ymmärtää mitä nämä arvot tarkoittavat, jotta hän voi analysoida mielenkiintoisuuden (Bhandari & al., 1994).

Analysoija voi käyttää mielenkiintoisuuden tulkinnassaan apuna jotain tulkintamallia. Kuvassa 3.3 on esitetty eräs tulkintamalli, jossa korjaava toimenpide puutteen poista-

miseksi voidaan suunnitella, kun tiedetään puutteen syy ja vaikutus. Korjauksen vahvistaminen kertoo, onko korjaava toimenpide tuonut halutun tuloksen eli ovatko puutteet vähentyneet tai hävinneet kokonaan. Hyötyjä voidaan analysoida puutteen poistumisena, puutteen leviämisen ehkäisynä, projektin suunnan muutoksena tai prosessin parantumisena (Bhandari & al., 1994).

Projektin hyvä tuntemus ja kokemus edesauttavat ongelmien tunnistamista ja korjaamista. Kokemus edesauttaa piilevien puutteiden poistamista, ehkäisee puutteiden syntyä, ohjaa projektia välttämään ongelmia ja paljastaa mahdollisia ongelmia sekä paljastaa ja korjaa prosessin epäkohtia. Vaikka epäkohtia ei voitaisikaan nykyisen projektin puitteissa korjata, niin projektin ja prosessin ongelmien tunnistaminen hyödyntää tulevia projekteja, näiden voidessa varautua jo etukäteen kyseisiin ongelmakohtiin (Bhandari & al., 1993).



Kuva 3.3. Tulkintamalli (Bhandari & al., 1994).

3.1.1 Datan kerääminen

Puutteet luokitellaan käyttämällä attribuutteja, joiden arvot osittavat prosessin toimintojen joukon. Attribuutin mahdolliset arvot siis eroavat prosessin toimintojen tai tuotteen osien suhteen. Jokainen puute luokitellaan valitsemalla arvo jokaisesta luokittelujärjestelmän attribuutista. Datan oletetaan olevan relaatiomallin mukaisesti taulukko-

muodossa, jossa tietueet ovat riveillä ja attribuutit ovat sarakkeissa. Attribuuttiarvot voivat olla diskreettejä tai numeerisia (Bhandari & al., 1992; Bhandari, 1995).

Ohjelmistoprosessien attribuuttifokusoinnissa käytetään yleensä ODC–luokittelua attribuuttien luokitteluun. Lisäksi kerätään attribuutit milloin puute on löytynyt (phase found), missä prosessivaiheessa se on syntynyt (phase introduced) ja missä komponentissa se esiintyy (component). Puutteen löytymispaikka kirjataan prosessivaiheena tai kehitysvaiheena, joka on menossa puutteen löytyessä. Puutteen esiintymispaikaksi puolestaan kirjataan se prosessivaihe, jossa puute on syntynyt. Komponentti kuvaa komponenttia, jossa puute on. Se voi olla joko ohjelmistokomponentti tai dokumentti (Bhandari & al., 1994).

3.1.2 Mielenkiintoisuusfunktio

Prosesseissa ollaan yleensä kiinnostuneita vain olennaisista puutteista eli sellaisista tapauksista, jotka merkitsevästi eroavat oletetuista tapahtumista. Mielenkiintoisuusfunktio on yleiskäsite, johon on olemassa erilaisia algoritmeja. Algoritmi voidaan kehittää sen mukaan, minkä tyyppisestä puutetiedosta ollaan kiinnostuneita.

Attribuuttifokusoinnissa mielenkiintoisuusfunktiolla lasketaan puutteista mielenkiintoisuusarvot. Mielenkiintoisuusfunktio järjestää datan osajoukot niiden suhteellisen mielenkiintoisuuden mukaan. Mielenkiintoisuustieto, johon mielenkiintoisuusfunktio attribuuttifokusoinnissa yleensä perustuu, voi olla attribuuttiarvon poikkeama odotetusta jakaumasta tai odottamaton korrelaatio attribuuttijoukon arvojen välillä. Yksisuuntaisella data-analyysillä tutkitaan attribuuttiarvon poikkeamaa ja kaksisuuntaisella analyysillä attribuuttiparien välistä korrelaatiota (Bhandari ja Roth, 1993; Mendonça ja Basili, 2000).

Attribuuttijakauman odotusarvo voidaan laskea tasajakaumasta, jolloin kaikilla tapauksilla on sama odotusarvo. Se voidaan laskea myös edellisten projektien perusteella ns. historiatietoon perustuen. Tällöin puute, joka on esiintynyt jossain valitussa edellisessä projektissa useammin kuin toinen, saa suuremman odotusarvon. Samantyyppisissä projekteissa voidaan olettaa olevan suurin piirtein samanlainen puutejakauma, jolloin

mielenkiintoisuustietokin olisi realistisempi kuin tasajakaumalla laskettuna. Valittuja projekteja voi olla useitakin ja ne voivat olla pitkältäkin aikaväliltä. Vertailun vuoksi voidaan laskea jakaumaa myös hyvin erilaisten projektien historiatietoon verraten, jolloin selviää onko puutejakaumilla projektikohtaisia eroja ollenkaan.

3.1.2.1 Yksisuuntainen data-analyysi

Yksisuuntainen analyysi tarkastaa, kuinka jokaisen attribuutin tietojoukon arvojakauma eroaa odotetusta arvojakaumasta (Bhandari & al., 1995). Toisin sanoen annetulle attribuutille etsitään mielenkiintoisia arvoja vertaamalla arvon havaittua frekvenssiä odotettuun frekvenssiin. Odotettu arvo lasketaan olettaen, että todennäköisyydet arvojen esiintymisille ovat tasaisesti jakautuneita (Freitas, 2001).

$$In_1(A = v) = Max([O_A(v) - E_A(v)]) \qquad \textbf{(3.1)}$$

Kaavassa (3.1) attribuuttijoukon jokaisen attribuutin A mielenkiintoisuusarvo v saadaan vähentämällä havaintoarvosta $O_A(v)$ odotusarvo $E_A(v)$. Suurin löydetty ero yksittäisen havaintoarvon ja odotusarvon välillä attribuuttijoukosta A on yksisuuntainen attribuuttijoukon A mielenkiintoisuus. Eron suuruus kertoo kuinka mielenkiintoinen attribuuttiarvojakauma on verrattuna odotettuun jakaumaan. Havaintoarvo $O_A(v)$ lasketaan löydetyistä havainnoista. Attribuuttien oletetaan esiintyvän tasaisesti jakautuneena ja $E_A(v)$ lasketaan tähän perustuen, esimerkiksi jos attribuutilla on viisi attribuuttimääritettä niin $E_A(v)$ = 20 % (Bhandari & al., 1995). Mielenkiintoisuus on pienin eli nolla, kun havaintoarvo ja odotusarvo ovat samoja.

Attribuutilla `lähde` on neljä eri attribuuttiluokkaa eli luokat `itse kehitetty`, `uudelleen käytettävästä kirjastosta`, `ulkoistettu` ja `siirretty` (ks. liite 1). Jokaisen attribuuttiluokan oletetaan esiintyvän tasaisesti eli jokaisen luokan oletettu esiintymistiheys on 25 % kaikista luokan arvoista. Arvojen oletettu esiintymistiheys on esitetty taulukon 3.1 sarakkeessa odotettu frekvenssi. Todellisuudessa luokat voivat kuitenkin olla epätasaisemmin jakautuneita. Taulukossa 3.1, sarakkeessa *Havaittu frekvenssi* näemme esimerkkitapauksemme havaitut frekvenssit. Taulukon viimeinen sarake eli *Mielenkiintoisuusarvo* kertoo, miten mielenkiintoinen attribuuttiluokka on. Suurin arvo

itseisarvona katsottuna on mielenkiintoisin eli esimerkkitapauksessamme attribuuttiluokka siirretty (|25| = 25) on mielenkiintoisin, ulkoistettu (|-15| = 15) toiseksi mielenkiintoisin jne.

Taulukko 3.1. Esimerkki yksisuuntaisesta data-analyysista attribuutille `lähde`

Attribuuttiluokka	**Havaittu frekvenssi (%)**	**Odotettu frekvenssi (%)**	**Mielenkiintoisuusarvo (%)**
Itse kehitetty	27	25	2
Uudelleen käytettävästä Kirjastosta	13	25	-12
Ulkoistettu	10	25	-15
Siirretty	50	25	25

3.1.2.2 Kaksisuuntainen data-analyysi

Kaksisuuntaisessa analyysissa mielenkiintoisuusfunktio mittaa kuinka paljon attribuuttiparin arvojen yhteinen havaittu frekvenssi eroaa odotetusta frekvenssistä, olettaen että nämä kaksi attribuuttia ovat tilastollisesti riippumattomia (Freitas, 2001).

$$In_2(A_x, A_y) = Max(|O(v,u) - O_{A_x}(v) \times O_{A_y}(u)|) \qquad \textbf{(3.2)}$$

Kaavassa (3.2) A_X on attribuuttijoukon A x:s arvo ja A_Y on attribuuttijoukon A y:s arvo. Mielenkiintoisuusfunktio lasketaan näiden kahden attribuutin yhteiselle esiintymiselle. $O(v,u)$ on todennäköisyys attribuuttiparin yhteiselle esiintymiselle. Se lasketaan niiden tietueiden prosenttiosuutena, jossa kummallakin attribuutilla A_x ja A_y on arvo eli ne esiintyvät yhdessä. Kaksisuuntainen analyysi ei käytä mitään oletettuja arvojakaumia, vaan se perustuu assosiaatioon attribuuttien A_x ja A_y välillä. Laskenta suoritetaan näiden attribuuttien erillisesiintymisprosenttien tulona $O_{A_x}(v) \times O_{A_y}(u)$. Tämä todennäköisyysarvo vähennetään attribuuttien arvojen yhteisesiintymisprosentista $O(v,u)$ (Bhandari & al., 1995; Freitas, 2001). Kun attribuuttien arvoja ei esiinny tai kun erillisesiintymisprosenttien tulo on sama kuin yhteisesiintymisprosentti, on assosiaation mielenkiintoisuusarvo nolla. Mielenkiintoisuus voidaan määrittää erilaisilla tarpeen ja

kohteen mukaan valitulla mielenkiintoisuusfunktiolla, joita esimerkiksi Hildermann & al. (1999, 2001) sekä Tan & al. (2002) on käsitellyt artikkeleissaan. Kaavan (3.2) mielenkiintoisuusfunktio vastaa Piatetsky-Shapiron (1991) esittelemää mielenkiintoisuusfunktiota (Tan & al., 2002).

Esimerkkitaulukossa 3.2 esitetään kahdessa ensimmäisessä sarakkeessa löydetyt attribuuttiluokat attribuuteille `lähde` ja `laukaisin`. Kolmannessa sarakkeessa on attribuutin `lähde` luokkien todellisten esiintymien frekvenssit ja `laukaisimen` vastaavat frekvenssit ovat sarakkeessa neljä. Sarakkeen *Odotettu (%)* arvot saadaan kertomalla nämä frekvenssit keskenään. Sarakkeen *Lähde ja laukaisin (%)* arvot saadaan laskemalla prosenttiosuus tapauksista, joissa ao. `lähde` ja `laukaisin` esiintyvät yhdessä. Esimerkiksi attribuutin `lähde` attribuuttiluokka `uusi funktio` ja attribuutin `laukaisin` attribuuttiluokka `testikattavuus` esiintyvät yhtä aikaa 9 % kaikista tapauksista. Sarake *Mielenkiintoisuus (%)* katsotaan sarakkeiden *Lähde ja laukaisin (%)* ja *Odotettu (%)* erotuksena itseisarvon suuruuden mukaan, jolloin tässä tapauksessa `lähteen` ja `laukaisimen` attribuuttiluokkien uusi funktio ja testikattavuus ovat mielenkiintoisin attribuuttipari.

Taulukko 3.2. Esimerkki kaksisuuntaisesta data-analyysista attribuuteille `lähde` ja `laukaisin`

Lähde	**Laukaisin**	**Lähde (%)**	**Laukaisin (%)**	**Lähde ja laukaisin (%)**	**Odotettu (%)**	**Mielen-kiintoisuus (%)**
Uusi funktio	Testikattavuus	37	52	9	19	-10
Uusi funktio	Yksinkertainen polku	37	15	13	5	7
Uudelleen kirjoitettu koodi	Testikattavuus	52	52	33	27	6
Uudelleen kirjoitettu koodi	Yksinkertainen polku	52	15	2	8	-6
Siirretty	Testi-vaihtelevuus	11	31	1	3	-2

3.1.3 Suodatinfunktio

Suodatinfunktio käsittelee ja esittää mielenkiintoisuusfunktioiden antamat tulokset analysoijille. Suodatinfunktio päättää, mitkä mielenkiintoisuusarvot ovat riittävän mielenkiintoisia herättääkseen projektiryhmän huomion. Valittujen puutteiden tiedot voidaan esittää suodatinfunktiolla taulukoiden ja kaavioiden avulla. Esimerkiksi Bhandarin & al. (1993) artikkelissaan kuvaaman projektin puutetyypit ja niiden esiintyminen on havainnollistettu taulukossa 3.4 ja kuvassa 3.4.

Bhandarin & al. (1992; 1994) mukaan kaavioiden esityksessä tulee ottaa huomioon ihmisen työmuistin rajoitukset. Kaavion eri kohteiden (esim. piirakan viipaleet) lukumäärä on siten esityksessä yleensä rajattu korkeintaan kahdeksaan, perustuen Millerin (1956) tutkimukseen. Vaihetta kohden tuotettujen kaaviokuvien lukumäärä on rajattu korkeintaan 20:een, mikä on katsottu kokemuksen mukaan sopivaksi määräksi yhdellä kertaa tulkittavaksi (Bhandari, 1995). Näiden tulkitsemiseen kuluva aika on noin kaksi tuntia.

3.2 Tapaustutkimus

Bhandari & al. (1994) esittelevät edellä kuvatun attribuuttifokusoinnin käyttöä erilaisissa projekteissa. Esimerkkiprojektin A luonnehdinta on esitetty taulukossa 3.3. Projektiryhmät luokittelivat kaikki tuotoksien (dokumentit, käyttöohjeet, koodi) teon ja testauksen aikana löytyneet puutteet kohdassa 3.1.1 esitetyn datan keruun mukaan. Projektiryhmillä ei ollut aiempaa kokemusta attribuuttifokusoinnista, joten analysointityö tehtiin ilmeisimmällä tavalla eli heti jokaisen vaiheen päättymisen jälkeen. Prosesseja muokattiin analysoinnin tuloksen pohjalta ennen uuden vaiheen aloitusta.

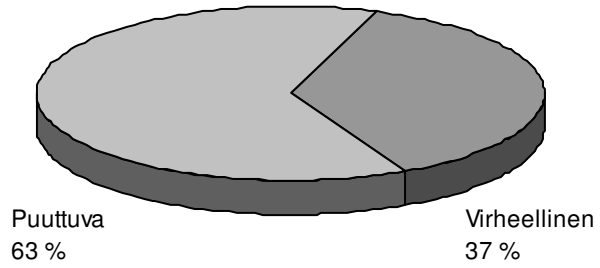
Taulukko 3.3. Projektin A tiedot.

Ohjelmisto-ympäristö	Uuden ja muutetun koodin määrä	Henkilöstö	Prosessimalli	Rinnakkaista kehitystyötä
Käyttö-järjestelmä	Keskisuuri	15	Vesiputousmalli yhdistettynä iteratiiviseen malliin	Kyllä

Kaavan (3.1) mukainen mielenkiintoisuusfunktio laskee jokaiselle attribuutille erotuksen ja järjestää ne laskevaan järjestykseen erotusprosentin itseisarvon (mielenkiintoisuusarvo) mukaan. Ollessaan itseisarvoltaan yhtä suuria, erotusprosentin positiiviset ja negatiiviset arvot ovat näin ollen yhtä mielenkiintoisia. Suodatinfunktio valitsee järjestyksen perusteella attribuuttiarvot kaaviossa esitettäväksi. Funktio suodattaa suurimmat 20 mielenkiintoisuusarvoa kuvaajina esitettäväksi, jättäen loput huomioimatta. Menetelmä on nopea tuottaen kyseiset 20 kuvaajaa vaiheen analysointia varten muutamassa minuutissa. Kaikkiaan menetelmän käyttö vei projekteissa noin kaksi tuntia vaihetta kohden, mikä aika kului pääosin palautekokoukseen. Tulokset on kuvattu taulukoissa 3.4 ja 3.5 sekä kuvissa 3.4-3.7.[3]

Taulukko 3.4. `Puuttuva` tai `virheellinen` puutetyyppi projektissa A.

Puutetyyppi	Havaittu %)	Odotettu (%)	Erotus (%)
Virheellinen	37	50	-13
Puuttuva	63	50	13



Kuva 3.4. `Puuttuva` ja `virheellinen` puutetyyppi projektissa A.

Kuva 3.4 ja taulukko 3.4 esittävät yksittäisen attribuutin puutetyyppi mielenkiintoisuutta. Taulukon ensimmäiseltä riviltä nähdään, että 37% puutteista on luokiteltu kohtaan `virheellinen`. Tasaisesti jakautuneiden puutetyyppien odotusarvo esitetään seuraavassa sarakkeessa. Tässä kahden puutetyypin (`virheellinen`, `puuttuva`) tapauksessa odotusarvo on 50 %. Erotus esittää edellisten sarakkeiden erotuksen (37% – 50% = -13%).

[3] Kuvaajat 3.4-3.7 on piirretty MS Excel-ohjelmistolla Bhandari & al. (1998) aineiston pohjalta.

AF-kuvaajassa näkyvä puutetyypin `puuttuva` suuri osuus ja sen oikea tulkinta johti prosessin ongelman löytämiseen ja korjaamiseen. Tulkintamallin (kuva 3.3) mukainen kuvaajien tulkinta oli seuraava.

Syy: Puutetyypin `puuttuva` suuri osuus aiheutui puutteellisesta kommunikaatiosta eri aliryhmissä työskentelevien suunnittelijoiden kesken. Tämä ilmeni toiminnallisuuden puutteena suunnittelussa.

Vaikutus: Tuotos olisi voitu toimittaa asiakkaalle toiminnallisesti vajavaisena.

Korjaava toimenpide: Suunniteltiin ja toteutettiin istunnot, joiden aikana ryhmän jäsenet esittelivät omaa työtään muille. Näiden tarkoituksena oli kommunikaatiovirheiden ehkäisy.

Korjauksen vahvistaminen: Puutetyyppi `puuttuva` väheni merkittävästi seuraavassa vaiheessa. Bhandari & al. (1994) tähdentävät, että aineisto ei kertonut tunteesta, että projektin osaryhmien kommunikaatio on puutteellista. Se ei myöskään paljastanut, että puutteet löytyivät eri ryhmien tarkastustenaikaisissa keskusteluissa. `Puuttuva` puutteiden luonne ei myöskään ilmennyt aineistosta. Aineiston kuvaajien nopea ja oikea tulkinta edellyttää projektin hyvää tuntemista, jota projektiryhmällä on. Käytettäessä projektiryhmän ulkopuolista ja sitä tuntematonta laatueksperttiä, on tällä vaara päätyä mahdolliseen väärään tulkintaan edellisenkaltaisia asioita tuntematta.

Taulukko 3.5. Projektin A attribuuttiparin esiintyminen (Bhandari & al., 1994).

Puutetyyppi	**Syntyvaihe**	**Tyyppi (%)**	**Vaihe (%)**	**Tyyppi ja vaihe (%)**	**Odotettu (%)**	**Erotus (%)**
1. Dokumentti	KS	14	16	5	2	3
2. Dokumentti	MS	14	84	8	11	-3
3.Sijoitus/alustus	MS	16	84	16	13	3
4.Sijoitus/alustus	KS	16	16	0	3	-3

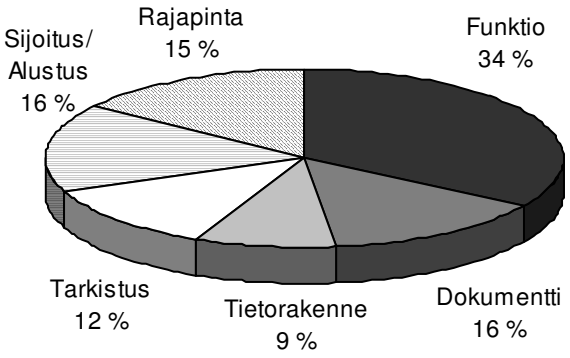
Tyyppi = Sarakkeen 1 puutetyypin esiintymisprosentti

Vaihe = Sarakkeen 2 vaiheessa syntyneiden puutteiden määrä (prosenttia kaikissa vaiheissa syntyneistä)

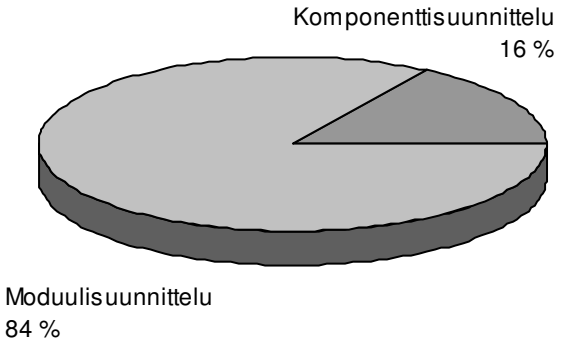
Tyyppi ja vaihe = Puutetyypin (sarake 1) kaikista puutteista esiintymisvaiheessa (sarake 2) syntyneet prosentteina

Hyödyt: Esimerkkiprojektin saama hyöty oli, että korjaava toimenpide ehkäisee puutteiden syntymistä sekä tunnistaa ja korjaa projektin käyttämän prosessin epäkohdan.

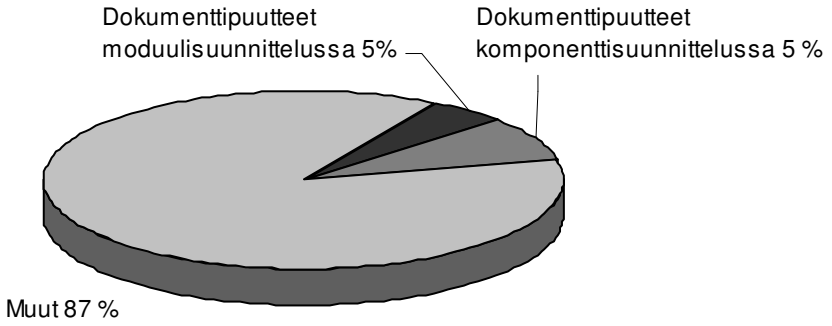
Taulukko 3.5 esittää Projektin A moduulisuunnittelun tarkastusvaiheessa luotua attribuuttien `puutetyyppi` ja `syntyvaihe` ristituloa. Taulukon tulokset on saatu kaavan (3.2) mukaisella kaksisuuntaisella data-analyysillä. Taulukossa esitetään kyseisten attribuuttien mielenkiintoisuusarvot pohjautuen niiden assosiaatioasteeseen. Ensimmäisen rivin sarakkeessa *Tyyppi (%)* nähdään, että 14% puutteista on puutetyyppiä `dokumentti` (ks. kuva 3.5). Seuraava sarake kertoo 16% puutteista syntyneen `Komponenttisuunnittelu` (`KS`) –vaiheessa (ks. kuva 3.6). Sarakkeesta *Tyyppi ja vaihe(%)* ilmenee viiden prosentin kaikista puutteista syntyneen `KS`–vaiheessa ja olevan tyyppiä `dokumentti` (ks. kuva 3.7). Odotettavissa oleva yhteisesiintymisprosentti on sarakkeen *Odotettu (%)* mukainen eli 2 %. Todellisen ja odotetun erotus (5% - 2% = 3%) näkyy *Erotus (%)* -sarakkeessa. Se ilmentää kyseisten attribuuttiarvojen (`dokumentti` ja `KS`) välistä assosiaatiota.



Kuva 3.5. Puutetyyppi projektissa A.



Kuva 3.6. Puutteen löytöpaikka projektissa A.



Kuva 3.7. Puutteen esiintyminen projektissa A.

Kuvaajia tulkitakseen projektiryhmän piti selittää attribuuttiarvojen assosiaatio tai odotetun assosiaation puuttuminen määrittämällä syy ja arvioimalla mahdollista vaikutusta. Jos tulkinta johti puutteen tunnistamiseen, ryhmän piti edelleen määritellä korjaava toimenpide. Projektiryhmä pystyi helposti selittämään taulukossa 3.5 olevat assosiaatiot. Esimerkiksi rivillä kolme näkyvä assosiaatio voitiin selittää puutteilla, jotka korjattiin `sijoituslauseita` ja `moduulisuunnitteluvaihetta` korjaamalla. Rivillä neljä näkyvä assosiaation puuttuminen kyseisten puutteiden ja `komponenttisuunnittelun` välillä on odotettu, sillä komponenttisuunnitteluun ei työn luonteen vuoksi sisälly sijoituslauseiden määrittelyjä, toisin kuin alemman tason moduulisuunnittelussa.

Tulkintamallin (kuva 3.3) mukainen kuvaajien (kuvat 3.5-3.7) tulkinta oli seuraava.

Syy: Komponenttisuunnittelua jatkettiin vielä moduulisuunnittelun aloittamisen jälkeen. Prosessin ongelman vahvistus osoitti: 1) Vaatimusmäärittelyn puutteellisuus aiheutti epätäydellisen komponenttisuunnittelun ja puute löydettiin ja korjattiin komponenttisuunnittelussa. 2) Formaalia vaatimusmäärittelydokumenttia ei ollut, sen sijaan ryhmien vetäjät ylläpitivät omia listojaan vaatimuksista. 3) Ryhmien jäsenet olivat yksimielisiä siitä, että osa vaatimuksista oli määrityksellisesti hankalia tuotoksen erilaisista asiakaspohjista johtuen.

Vaikutus: Moduulisuunnittelu tulisi tehdä vasta, kun komponenttisuunnittelu on saatu päätökseen. Muutoin prosessi ei etene järjestyksessä, mistä voi olla vakavia seurauksia, jotka näkyvät aikatauluongelmina tai tuotoksen huonona laatuna.

Korjaava toimenpide: Vaatimukset ja lopullinen ohjelmointispesifikaatio tarkastettiin kattaviksi.

Korjauksen vahvistaminen: Puuttuvia vaatimuksia ei enää löytynyt testeissä.

Hyödyt: Korjaava toimenpide poistaa puutteen ja ehkäisee puutteellisen tuotoksen toimittamisen asiakkaille. Se paljastaa myös ongelman prosessissa eli tuotos on liian monimutkainen tuotettavaksi ilman formaalia vaatimusprosessia. Uudet julkistukset edellyttävät, että prosessia on muutettava tältä osin.

Bhandari & al. (1994) esittelivät tutkimuksessaan myös muita erityyppisiä projekteja. Näiden osalta olennainen havainto oli, että menetelmä sopi erilaisiin projekteihin. Menetelmällä saavutettiin tutkimuksen projekteissa erilaisia hyötyjä. Sen käyttö auttoi poistamaan projektien puutteita, ehkäisemään puutteita, tunnistamaan tarvittavat korjaustoimenpiteet sekä tunnistamaan ongelmakohdat käytetyissä prosesseissa.

4 MIKIFU-JÄRJESTELMÄ

Attribuuttifokusoinnin toteuttaminen käytännössä edellyttää siihen suunniteltujen työvälineiden käyttöä. Tässä luvussa esittelemme MiKiFu-järjestelmän ja MiKiFu-järjestelmällä toteutettua esimerkkiaineiston attribuuttifokusointia.

4.1 Järjestelmän esittely

MiKiFu on Joensuun yliopistossa, tietojenkäsittelytieteen laitoksella erikoistyönä tehty attribuuttifokusointiin perustuva puutteiden analysointiväline (Ahtiainen & Günther, 2003). Järjestelmä tukee ODC-luokittelua. Siihen voidaan projektikohtaisesti määritellä luokitteluperusteet puutteille ja tallentaa puutetietoja, joista ohjelma laskee mielenkiintoisuusfunktiolla arvot ja suodattaa nämä suodatinfunktiolla analysoijalle tarkasteltaviksi. Puutteista saadaan lukumäärä-, oletusprosentti- ja löytöprosenttitietoa sekä löytö- ja oletusprosentin erotus sekä erotusitseisarvo, jonka perusteella mielenkiintoisuus määräytyy. Mitä suurempi on oletetun ja löydetyn puuteprosentin ero, niin sitä mielenkiintoisempi puute on.

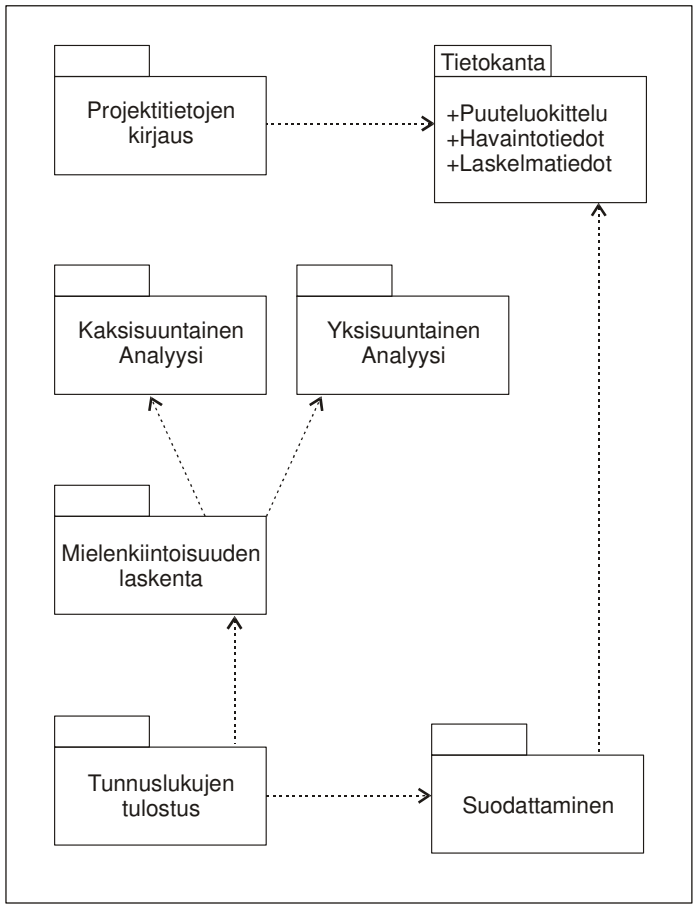
MiKiFu esittää mielenkiintoisuuden taulukoituna laskevassa järjestyksessä. Ensimmäisenä on mielenkiintoisin tapaus eli se, jolla erotusitseisarvo on suurin. MiKiFu:lla voidaan suodattaa kuusi erilaista suodatusta (ks. tarkemmin kohta 4.2) eli attribuutille, komponentille, prosessivaiheelle, attribuutille ja kumppaniattribuutille, attribuutille ja komponentille sekä attribuutille ja prosessivaiheelle omansa. Järjestysnumero kertoo tapauksen mielenkiintoisuuden koko aineiston osalta eli kaikkien suodatusten mielenkiintoisuusjärjestyksen.

Yksisuuntainen data-analyysi, johon kuuluvat suodatukset attribuutti, komponentti ja prosessivaihe antavat yleensä suurempia erotusitseisarvoja kuin kaksisuuntainen data-analyysi, johon suodatukset attribuutti ja kumppaniattribuutti, attribuutti ja komponentti sekä attribuutti ja prosessivaihe kuuluvat. Tämä aiheutunee siitä, että yleensä yhteisesiintymisellä on pienempi todennäköisyys kuin yksittäisellä esiintymisellä.

4.1.1 Yleisrakenne

Ohjelmisto on toteutettu Visual Basic 6.0 -kielellä ja se koostuu kahdesta osasta. Ensimmäinen osa on puutetietojen kirjaamisosa ja toinen osa on järjestelmään kirjattujen tietojen laskenta- ja tulostusosa. Kirjaamisosaan syötetään projektitiedot sekä näiden tunnetut puutetiedot. Laskenta- ja tulostusosassa lasketaan puutetietojen ja mielenkiintoisuusfunktion perusteella mielenkiintoisimmat tunnusluvut. Nämä luvut esitetään näytöllä taulukkona ja myös haluttaessa pylväsdiagrammina. Järjestelmän yleisrakenne on esitetty UML-notaation mukaisesti pakettien ja niiden riippuvuussuhteiden avulla kuvassa 4.1.

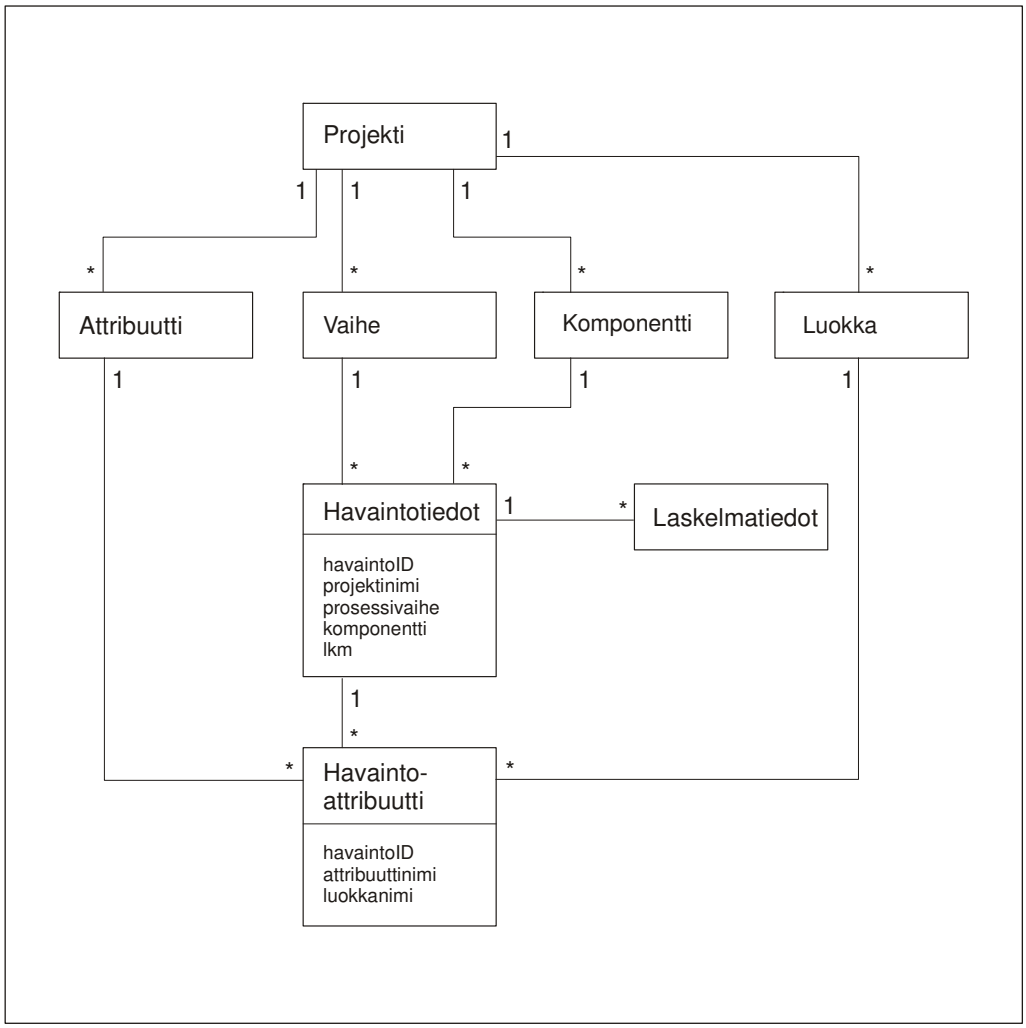
Järjestelmän tietokanta sisältää tiedot järjestelmässä käytetyistä puutetyypeistä sekä järjestelmään kirjatut projektit ja näistä lasketut projektikohtaiset tunnusluvut.



Kuva 4.1. MiKiFu-järjestelmän yleisrakenne.

4.1.2 Tietokannan rakenne

Järjestelmän tietokannan looginen rakenne esitetään kuvassa 4.2 UML-mallinnuskielen mukaisena luokkakaaviona, jota voidaan tulkita kohdan 2.5.3 ODC-luokittelun mukaisesti.



Kuva 4.2. Järjestelmän tietojen rakenne luokkakaaviona.

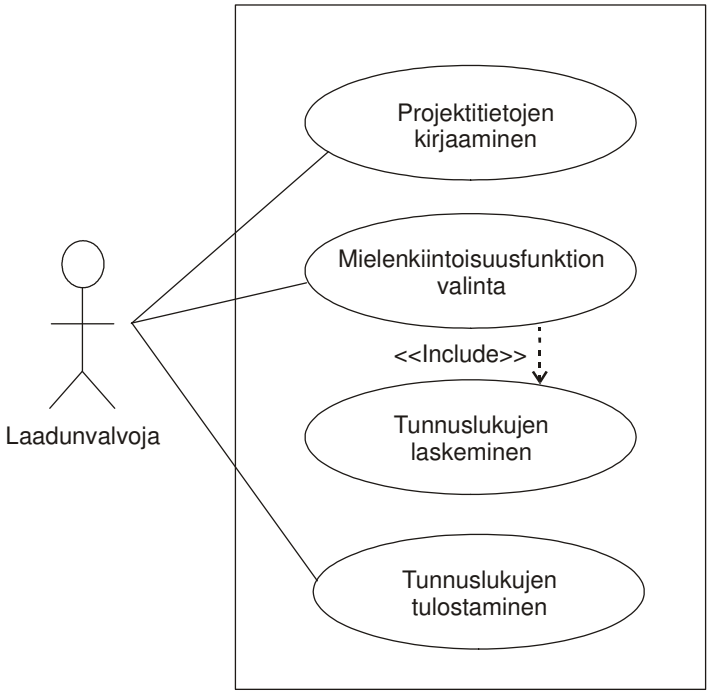
Kutakin kuvan 4.2 luokkaa vastaa tietokantataulu. Projektista kerättävät puutetiedot tallennetaan Havainto-tauluun ja Havaintoattribuutti-tauluun kuvan 4.2 attribuuttimäärittelyjen mukaisesti.

Laskelmatiedot-taulu muodostetaan tallennettavista puutetiedoista kuvan 4.1 Mielenkiintoisuuden laskenta –paketin ja sen käyttämien analyysipakettien avulla, kuten kohdassa 4.1.3 on esitetty.

Lisäksi tietokannassa on prosessitietona järjestelmän käyttämiä prosessikohtaisia aputauluja, jotka määrittävät kaikki mahdolliset projektivaiheet, attribuutit, luokat ja komponentit.

4.1.3 Toiminnallisuus

Järjestelmän toiminnallisuus esitetään neljään osaan ositettuna käyttötapauskaavioina kuvissa 4.3 - 4.5 UML-mallinnuskielen mukaisesti.

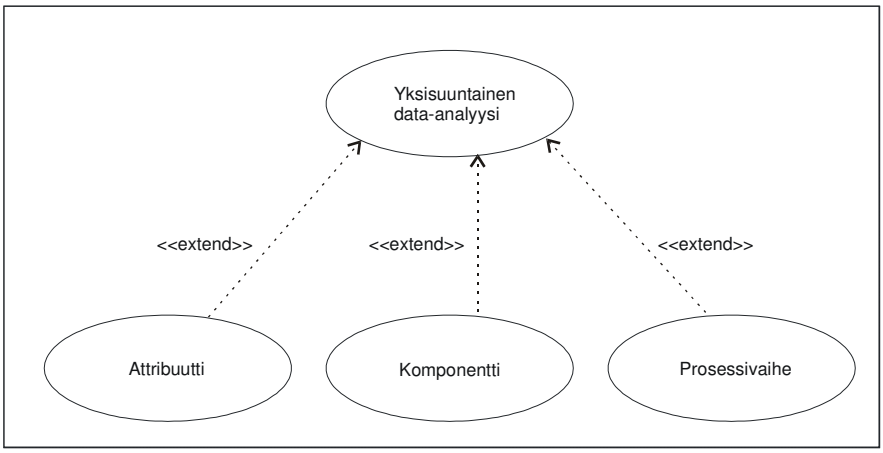


Kuva 4.3. Puutteiden analysointijärjestelmän käyttötapauskaavio.

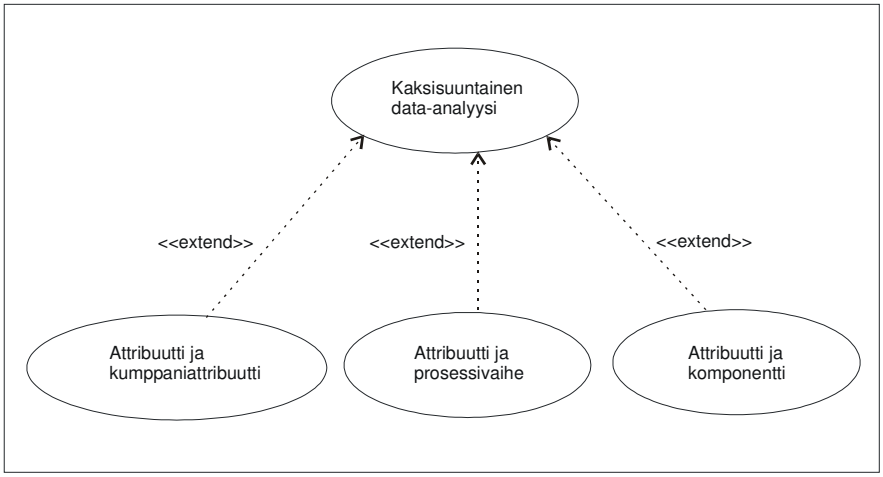
Projektin alussa MiKiFu–järjestelmään syötetään projektikohtaiset prosessivaihetiedot, komponenttitiedot ja muut attribuuttitiedot, jotka voivat olla esimerkiksi ODC–luokiteltuja. MiKiFu:n attribuuttiluokittelu on suunniteltu mahdollistamaan ODC–luokittelun mukaisen puuteluokittelun, mutta siihen voidaan syöttää myös muulla tavalla luokiteltuja attribuutteja, kunhan niillä on myös yksi tai useampi attribuutti-

luokka. Prosessivaiheella tarkoitetaan prosessivaihetta, jossa puute on syntynyt. Komponenttitietoihin laitetaan kaikki ohjelman komponentit ja myös dokumentit. Attribuuttitietoihin syötetään attribuutit ja niiden luokat.

Luokittelu vaikuttaa puutteiden analysointiin ja niistä saataviin tietoihin. Luokittelun tulisikin olla mahdollisimman kattava ja selkeä, että luokittelija osaisi luokitella puutteet oikein. Jos puutteista jää tietoa saamatta eli tulee niin sanottua ”ei tietoa” –tietoa, niin analysointi vaikeutuu, koska ei tiedetä mihin luokkaan, komponenttiin tai prosessivaiheeseen puute oikeasti kuuluu. Luokittelu on tällöin ollut puutteellinen.



Kuva 4.4. Yksisuuntaisen data-analyysin käyttötapauksen tarkennus.



Kuva 4.5. Kaksisuuntaisen data-analyysin käyttötapauksen tarkennus.

Kun kaikki puutetiedot on syötetty, ohjelma laskee mielenkiintoisuuden ja suodatinfunktio suodattaa mielenkiintoisimmat tapaukset analysoijan nähtäväksi. Mielenkiintoisuus lasketaan attribuuteille, komponentille, prosessivaiheelle, attribuutille ja kumppaniattribuutille, attribuutille ja komponentille sekä attribuutille ja prosessivaiheelle. Mielenkiintoisuustieto lasketaan tasajakaumana eli kaikki tapaukset ovat odotusarvoltaan samoja. Mielenkiintoisuuden laskeminen attribuutille, komponentille ja prosessivaiheelle perustuu kohdassa 3.1.2.1 esitettyyn yksisuuntaiseen data-analyysiin ja muissa tapauksissa se lasketaan kohdassa 3.1.2.2 esitetyn kaksisuuntaisen data-analyysin perusteella. Koko aineistosta muodostetaan mielenkiintoisuuden mukaan laskevassa järjestyksessä oleva lista Laskelmatiedot-tauluun kuvan 4.1 mukaisesti.

Suodatinfunktio näyttää tapaukset Laskelmatiedot-taulusta mielenkiintoisuusjärjestyksessä käyttäjän valinnan mukaan attribuutille, prosessivaiheelle, komponentille, attribuutille ja kumppaniattribuutille, attribuutille ja komponentille sekä attribuutille ja prosessivaiheelle erikseen. Tiedot esitetään taulukkona, jossa mielenkiintoisuusjärjestys perustuu siis erotusitseisarvon suuruuteen. Suodatinfunktio esittää myös kaikille tapauksille järjestysnumeron, joka kertoo kuinka mielenkiintoinen tapaus on koko aineistoon verrattuna. Taulukosta on siis nähtävissä kaikki ao. valinnan tapaukset, seitsemän tapausta kerrallaan. Lisäksi seitsemästä mielenkiintoisimmasta tapauksesta suodatinfunktio esittää taulukon.

Kohdassa 2.5 esitellyt muut luokittelumallit eivät sovellu MiKiFu:n puuteluokittelumalleiksi sellaisenaan. DPP–luokittelu tai paremminkin puutteiden analysointiprosessi on erilainen ODC–luokittelun ja MiKiFu:n kanssa, joten MiKiFu:n avulla ei voida analysoida DPP–puutteita. IEEE-standardia ja Hewlett–Packardin mallia voitaisiin käyttää hieman MiKiFu:a kehittämällä/muuttamalla.

IEEE–standardissa kategorioissa on suuri osa ODC–luokittelun mukaisista attribuuteista, mutta esimerkiksi MiKiFu:ssa käytetyt komponentti ja prosessivaihe puuttuvat. Kategoriat IEEE–standardissa voisivat olla attribuutteja ja niiden alikategoriat attribuuttiluokkia, mutta MiKiFu ei mahdollista puutteen tarkempaa luokittelua esimerkiksi välillä haitallinen puute – ei haitallinen puute. Jotta IEEE–standardin mukainen puuteluokittelu kävisi MiKiFu-järjestelmään ja attribuuttifokusointiin, täytyisi MiKiFu:n

puuteluokittelulomakkeen mahdollistaa attribuuttien muuttaminen kategorioiksi ja prosessivaiheen ja komponentin poistamisen. Myös puutteiden arvottaminen niiden haitallisuuden mukaan tulisi mahdollistaa tietojen syöttölomakkeella. Mielenkiintoisuusfunktion laskentaperusteita tulisi myös muokata huomioimaan haitallisuus.

Hewlett-Packardin mallissa alkuperä on sama kuin prosessivaihe MiKiFu:ssa, eli paikka, jossa puute on. Tyyppi on sama kuin MiKiFu:ssa komponentti. Toimintotapa ei kuitenkaan vastaa ODC–luokittelun attribuuttia, jolla kuvataan puutteen lähdettä eli paikkaa, jossa se on syntynyt, vaikutusta yms., vaan Hewlett-Packardin mallissa toimintotapa kertoo miksi puute on syntynyt eikä toimintotavalle ole alaluokkia eli ODC–luokituksen kaltaisia attribuuttiluokkia. MiKiFu:lla voitaisiin kuitenkin luokitella puutteet Hewlett-Packard mallin mukaisesti, jos prosessivaiheeseen syötetään alkuperä, komponenttiin tyyppi ja attribuuttiin toimintatapa. Attribuuttiluokaksi syötetään aina jotain esimerkiksi `toimintotapa puuttuu` ja sen luokaksi `puuttuva`. Mielenkiintoisuusfunktiota ei tarvitse muuttaa Hewlett-Packard mallia varten, mutta luokittelusta johtuen puutteesta ei saada yhtä paljon tietoa kuin ODC–luokittelun käytössä ollessa.

4.2 Esimerkkiaineiston analysointi MiKiFu-järjestelmällä

Esimerkkitapauksina oleva puuteaineisto on peräisin kirjasta Handbook of Software Reliablity Engineering (Lyu, 1995). Aineisto on kerätty IBM Watson Research Center-tutkimuslaitoksessa viidestä eri projektista ODC–luokittelun soveltamista varten. MiKiFu:lla analysoitiin kaksi projektiaineistoista, ODC2 ja ODC5. Nämä projektit valittiin siksi, että niissä puutetietoja oli kattavimmin kaikissa puuteluokissa, mikä on MiKiFu:lla analysoinnissa tärkeää, jos halutaan saada mahdollisimman paljon informaatiota projektista. Jos attribuutit, komponentit ja prosessivaiheet ovat ns. ”ei tietoa” tyyppisiä, mielenkiintoisuusanalysointi ei anna juuri muuta tietoa kuin sen, että projektissa ei ole osattu kerätä puutetietoja oikein.

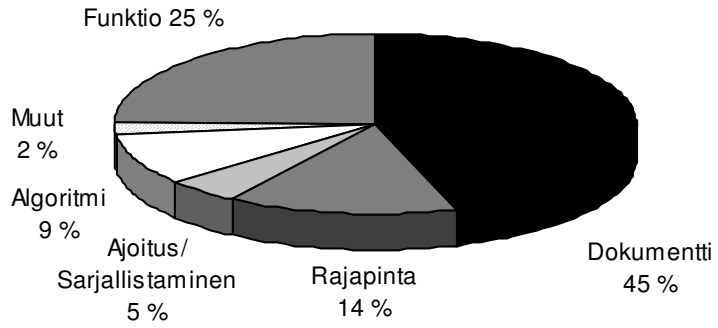
MiKiFu:n antamia tuloksia tulkitaan yleisellä tasolla, mutta projektin ulkopuolisen henkilön on vaikeaa analysoida tuloksia tarkasti, koska analysointi vaatisi projektin tuntemista. Attribuuttifokusoinnin ideahan on, että projektissa toimivat analysoivat itse projektinsa puutteita. Ulkopuolelta voidaan kuitenkin nähdä jotain suuntaviivoja

projekteista ja sen puutteista sekä pohtia mahdollisia eri syitä, jota ovat johtaneet puutteiden syntyyn.

4.2.1 Esimerkkiaineisto 1 (ODC2)

ODC2-aineistossa on vain attribuutit `puutetyyppi`, `puuttuva/virheellinen` ja `laukaisin`. Niiden jakaumat aineistossa on esitetty kuvissa 4.6-4.8[4]. Aineistossa oli puutteita yhteensä 255 kappaletta. Puutteista ei tiedetty komponenttia eikä prosessivaihetta. Tämän takia tulokset analyysista jäävät hieman vajavaisiksi, koska puutteita ei voida paikantaa.

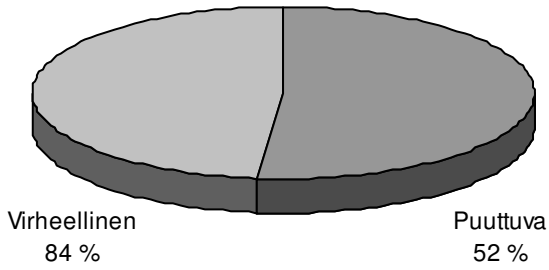
Kuvassa 4.6 on esitetty eri puutetyyppien jakautuminen aineistossa ODC2. Dokumenttipuutteita on eniten, melkein puolet. Tästä voidaan päätellä, että dokumentoinnin tarkastusmenettelyt ovat tarkat ja hyvät ja/tai dokumentoinnissa on ongelmia.



Kuva 4.6. Puutetyyppijakauma.

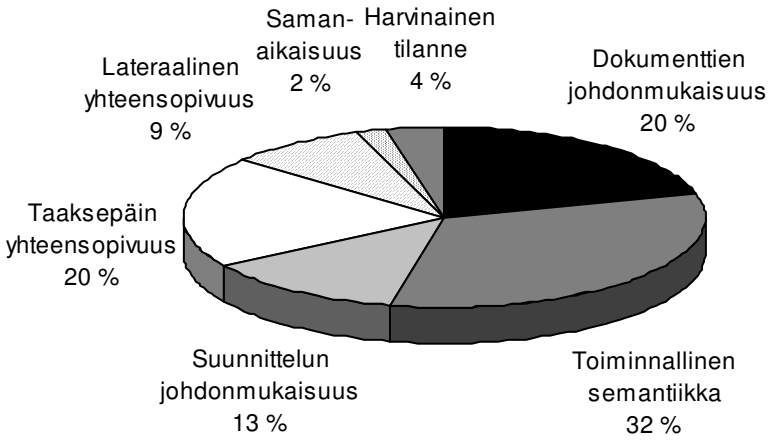
Kuvasta 4.7 huomataan, että attribuutit ovat jakautuneet aika tasaisesti puuttuviin ja virheellisiin. Se onko huolestuttavampaa, että joku puute on puuttuva tai virheellinen on projektin sisäinen asia.

[4] Kuvat 4.6, 4.7 ja 4.8 on tehty puutetietoaineistosta MS Excel -ohjelmalla.



Kuva 4.7. Attribuutin puuttuva/virheellinen jakauma.

Attribuutin laukaisin jakauma on melko tasainen kuvan 4.8 mukaisesti. Toiminnallinen semantiikka, taaksepäin yhteensopivuus ja dokumenttien johdonmukaisuus ovat yleisimpiä.



Kuva 4.8. Attribuutin laukaisin jakauma.

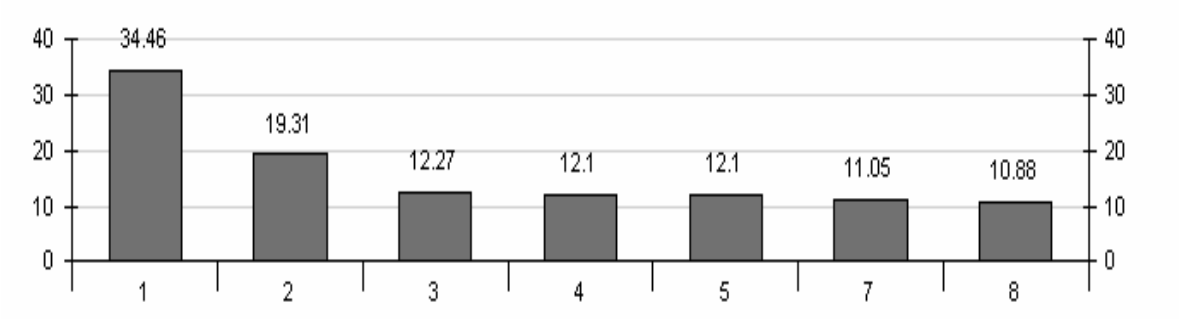
Taulukossa 4.1 attribuuttien mielenkiintoisuus on analysoitu MiKiFu:lla ja huomataan, että attribuutti `puutetyyppi` ja sen luokka `dokumentti` on mielenkiintoisin tapaus. Kuva 4.9 esittää taulukon tiedot kuvaajana ja se havainnollistaa vielä selkeämmin seitsemän mielenkiintoisimman tapauksen erot. Koska dokumenttien erotusprosentti on positiivinen, voidaan todeta, että joko dokumentit ovat huolimattomasti tehtyjä tai tarkastusmenettelyt ovat äärimmäisen hyviä. Suuresta erotusprosentista voimme myös päätellä, että havaintoja on huomattavasti odotettua enemmän.

Huomataan myös, että vaikka puutetyyppejä `funktio` oli 25 % aineistossa (kuva 4.5), niin sen mielenkiintoisuus ei kuitenkaan ole MiKiFu:n mukaan kovin suuri. Taulukossa

4.1 esiintyvissä viidessä viimeisessä tapauksessa oletusprosentti on ollut suurempi kuin puutteiden todellinen esiintyvyys, mikä tekee niistäkin mielenkiintoisia. Voidaan pohtia, johtuuko puutteiden vähyys käytettävästä tasajakaumasta, tarkastusmenettelyiden vähyydestä tai huonoudesta vai yksinkertaisesti siitä että puutteita ei näillä osilla esiinny ja prosessi on tältä osin kunnossa. Edelleen voidaan pohtia sitä, olisiko historiatietoihin eli organisaation aiempiin prosesseihin pohjautuva oletusarvon laskenta tasajakauma-laskentaa parempi menetelmä jatkossa tämänkaltaisen prosessin arviointiin. Taulukon ensimmäisen sarakkeen järjestysnumero kertoo koko aineiston osalta tapauksen mielenkiintoisuusjärjestyksen, joten tapaus numerolla 6 ei näy tässä. Se tulee esille toisessa suodatuksessa, mikä esitetään taulukossa 4.2.

Taulukko 4.1. Attribuuttien mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Attribuutti	Luokka	Luku-määrä	Löytö (%)	Oletus (%)	Erotus (%)	Erotus-itseisarvo
1	Puutetyyppi	Dokumentti	116	46.96	12.50	34.46	34.46
2	Laukaisin	Toiminnallinen semantiikka	83	33.60	14.29	19.31	19.31
3	Laukaisin	Samanaikaisuus	5	2.02	14.29	-12.27	12.27
4	Puutetyyppi	Tarkastus	1	0.40	12.50	-12.10	12.10
5	Puutetyyppi	Rakenne/paketti/ lomitus	1	0.40	12.50	-12.10	12.10
7	Laukaisin	Harvinainen tilanne	8	3.24	14.29	-11.05	11.05
8	Puutetyyppi	Toimeksianto	4	1.62	12.50	-10.88	10.88



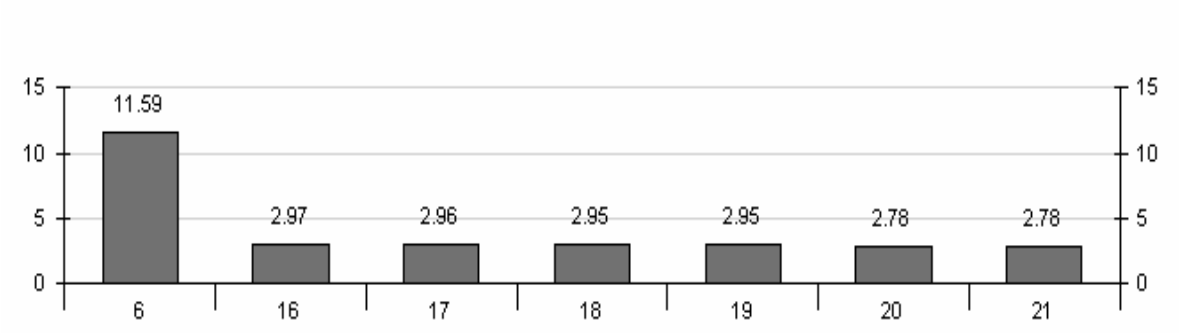
Kuva 4.9. Taulukon 4.1 esitys kuvaajana.

Attribuutin ja kumppaniattribuutin mielenkiintoisuus on analysoitu taulukossa 4.2 ja esitetty kuvaajana kuvassa 4.10. Attribuutti `laukaisin` ja sen luokka `dokumenttien johdonmukaisuus` sekä kumppaniattribuutti `puutetyyppi` ja sen luokka `dokumentti` ovat yhteisesiintymiseltään mielenkiintoisimpia. Ne ovat jopa koko aineiston osalta järjestyksessään kuudenneksi mielenkiintoisin tapaus.

Jo attribuuttien analysoinnissa huomasimme attribuutin `puutetyyppi` ja `dokumentti` olevan mielenkiintoinen, mutta parivertailu antaa siitä lisätietoa sen verran, että yleisimmät dokumentti-virheet on laukaissut `dokumenttien johdonmukaisuus`. Projektissa tulisikin keskittyä dokumenttien kirjoitukseen esimerkiksi laatimalla laatukäsikirja, jos sitä ei vielä ole, tai parantaa/noudattaa paremmin jo olemassa olevaa laatukäsikirjaa.

Taulukko 4.2. Attribuutin ja kumppaniattribuutin mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

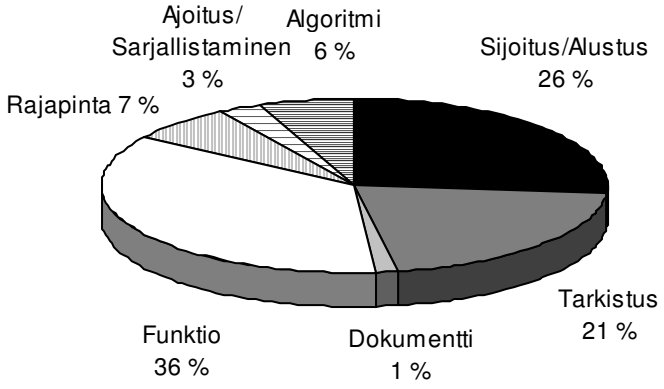
Järjestys	Attribuutti	Luokka	Kumppani-attribuutti	Kumppani-luokka	Löytö (%)	Oletus (%)	Erotus (%)	Erotus-itseis-arvo
6	Laukaisin	Dokumenttien johdon-mukaisuus	Puutetyyppi	Dokumentti	21.68	10.27	11.59	11.59
16	Laukaisin	Dokumenttien johdon-mukaisuus	Puuttuva/ virheellinen	Virheellinen	13.77	10.80	2.97	2.97
17	Laukaisin	Dokumenttien johdon-mukaisuus	Puuttuva/ virheellinen	Puuttuu	8.10	11.06	-2.96	-2.96
18	Puuttuva/ virheellinen	Virheellinen	Puutetyyppi	Dokumentti	20.24	23.19	-2.95	-2.95
19	Puuttuva/ virheellinen	Puuttuu	Puutetyyppi	Dokumentti	26.72	23.77	2.95	2.95
20	Laukaisin	Lateraalinen yhteen-sopivuus	Puuttuva/ virheellinen	Virheellinen	1.62	4.40	-2.78	-2.78
21	Laukaisin	Lateraalinen yhteen-sopivuus	Puuttuva/ virheellinen	Puuttuu	7.29	4.51	2.78	2.78



Kuva 4.10. Taulukon 4.2 esitys kuvaajana.

4.2.2 Esimerkkiaineisto 2 (ODC5)

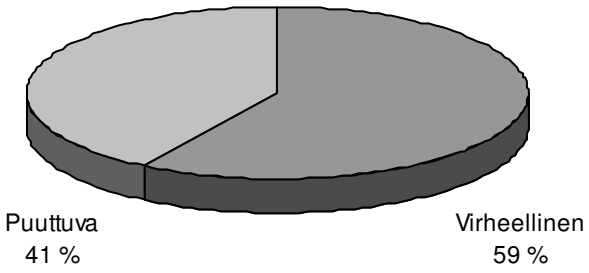
ODC5–aineistossa on attribuutit `puutetyyppi`, `puuttuva/virheellinen`, `laukaisin`, `lähde` ja `vaikutus`. Muita attribuutteja ovat sekä `prosessivaihe`, jossa puutteen löytyessä oltiin menossa, että `puutteen sijainti` eli paikka, jossa puute fyysisesti sijaitsi. Niiden jakaumat aineistossa on esitetty kuvissa 4.11- 4.17[5]. Aineistossa oli puutteita yhteensä 378 kappaletta.



Kuva 4.11. Esimerkkiaineiston puutetyyppijakauma.

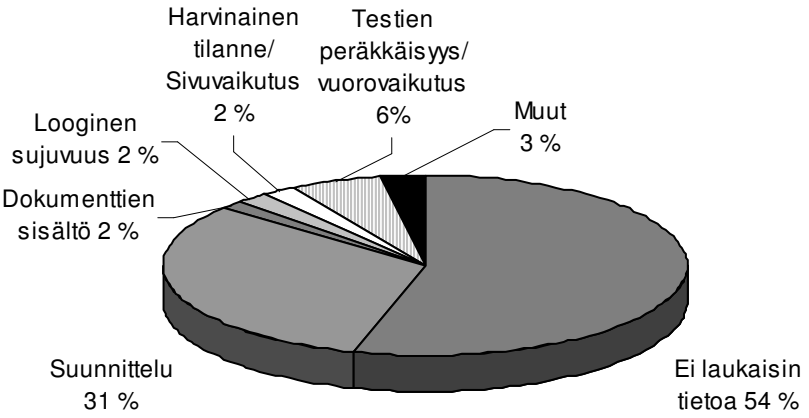
Kuvasta 4.11 nähdään, että aineistossa ODC5 puutetyyppejä `funktio`, `sijoitus/alustus` ja `tarkistus` on suurin osa puutteista. Muiden neljän puutetyypin osuus on yhteensä 17 %.

[5] Kuvat 4.11 - 4.17 on tehty puutetietoaineistosta MS Excel -ohjelmalla



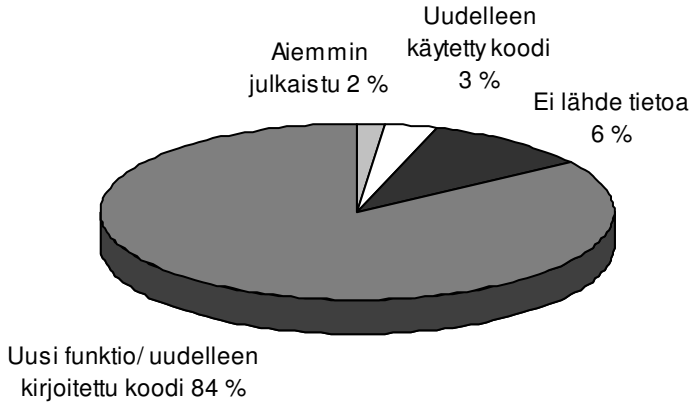
Kuva 4.12. Esimerkkiaineiston puuttuva/virheellinen jakauma.

Kuvan 4.12 mukaisesti `puuttuvia` on attribuuteista vain 41 % ja `virheellisiä` jopa 59 % attribuuteista. Koska projektista ei tiedetä enempää on vaikea sanoa, onko jakauma hyvä vai huono.



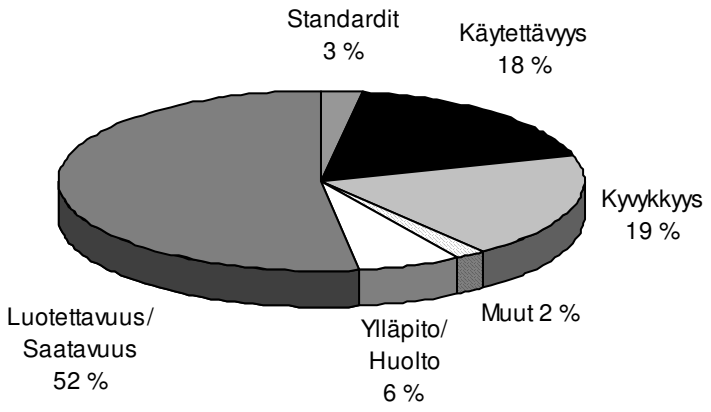
Kuva 4.13. Esimerkkiaineiston laukaisinjakauma.

Kuvan 4.13 laukaisinjakauma ei kerro paljonkaan laukaisimista, koska 54 % laukaisimista on luokiteltu kohtaan `ei laukaisintietoa`. Joko laukaisimia on ollut hankala luokitella, niitä ei ole osattu luokitella tai ne ovat jääneet muista syistä luokittelematta. Tämä kuitenkin hankaloittaa tulosten tulkintaa, koska ei tiedetä mihin luokkaan laukaisimet todellisuudessa kuuluvat. Esimerkin perusteella voidaankin vahvistaa jo aiemmin esitetty toteamus aineiston huolellisen luokittelun merkityksestä analyysille.



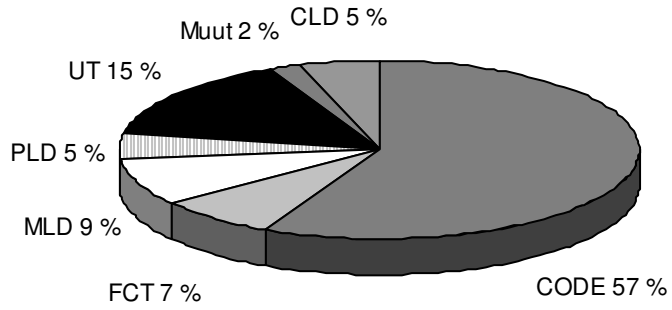
Kuva 4.14. Esimerkkiaineiston lähdejakauma.

Kuvasta 4.14 nähdään, että suurin osa eli 84 % puutteista on lähteestä `uusi funktio/ uudelleen kirjoitettu koodi`. Tästä voidaan päätellä, että suurin osa puutteista löytyy siis koodista.



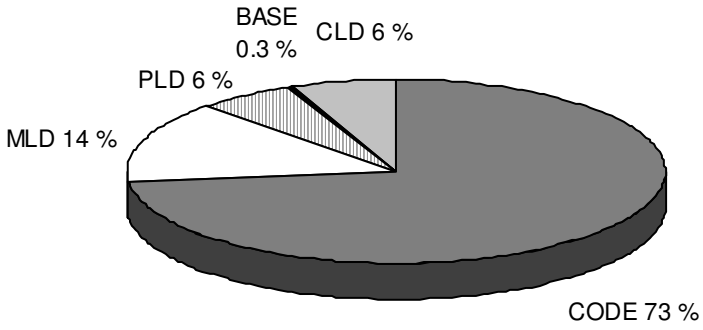
Kuva 4.15. Esimerkkiaineiston vaikutusjakauma.

Kuvan 4.15 vaikutusjakaumasta huomataan, että yli puolet eli 52% puutteista vaikuttaa luotettavuuteen/saatavuuteen. Seuraavaksi suurimmat osuudet ovat käytettävyydelle ja kyvykkyydelle.



Kuva 4.16. Esimerkkiaineiston prosessivaihe puutteen löytyessä.

Kuvan 4.16 mukaan valtaosa (57%) puutteista on löytynyt koodatessa. Kuvasta 4.14 jo havaittiin että suurin osa puutteista on koodilähteestä. Ne ovat myös löytyneet koodausvaiheessa, joten tarkastusmenettelyt koodausvaiheessa ovat oletettavasti aika toimivat.



Kuva 4.17. Esimerkkiaineiston puutteiden sijainti.

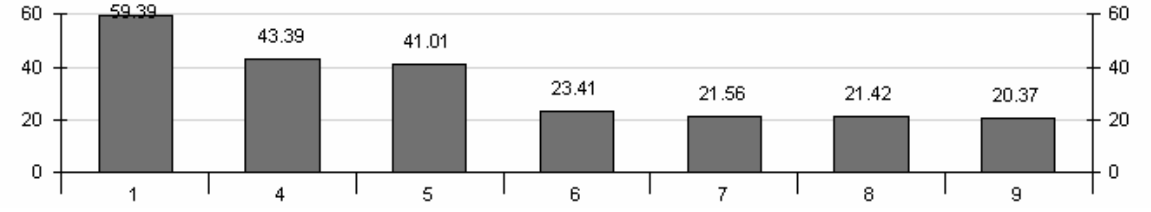
Kuvan 4.17 mukaan puutteet ovat sijainneet koodissa, mikä oli pääteltävissä edellistenkin kuvien perusteella. Koodausta täytyisi selkeästi parantaa, koska puutteita siellä näyttää syntyvän.

Taulukossa 4.3 ja kuvassa 4.18 on esitetty seitsemän mielenkiintoisinta attribuuttitapausta MiKiFu:lla analysoituna. Attribuutti `lähde`, jonka luokka on `uusi funktio/uudelleen kirjoitettu koodi`, on mielenkiintoisin tapaus. Se on myös koko analysoinnin mielenkiintoisin tapaus, koska sen järjestysnumero on yksi. Puutteen `lähde` useimmiten sijaitsee siis uudessa funktiossa tai uudelleen kirjoitetussa koodissa. Tämän

saattoi jo aavistaa, koska `lähde`-attribuutista jopa 84 % (kuva 4.14) löytyi juuri luokasta `uusi funktio/uudelleen kirjoitettu koodi`. Toiseksi mielenkiintoisin tapaus eli `laukaisin` ja sen luokka `ei laukasintietoa` ei sinällään kerro muuta kuin, että laukaisintietoa ei ole osattu tai voitu kirjata.

Taulukko 4.3. Attribuuttien mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Attribuutti	Luokka	Lukumäärä	Löytö (%)	Oletus (%)	Erotus (%)	Erotus-itseisarvo
1	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	319	84.39	25.00	59.39	59.39
4	Laukaisin	Ei laukaisintietoa	206	54.50	11.11	43.39	43.39
5	Vaikutus	Luotettavuus/ Saatavuus	197	52.12	11.11	41.01	41.01
6	Lähde	Aiemmin julkaistu	6	1.59	25.00	-23.41	23.41
7	Lähde	Uudelleen-käytetty koodi	13	3.44	25.00	-21.56	21.56
8	Puutetyyppi	Funktio	135	35.71	14.29	21.42	21.42
9	Laukaisin	Suunnittelun yhdenmukaisuus	119	31.48	11.11	20.37	20.37

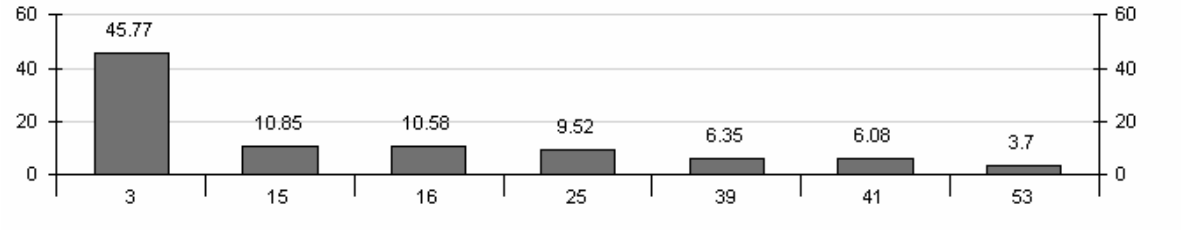


Kuva 4.18. Taulukon 4.3 esitys kuvaajana

Prosessivaiheiden seitsemän mielenkiintoisinta tapausta on esitetty taulukossa 4.4 ja kuvassa 4.19. Koodausvaihe (CODE) on mielenkiintoisin. Koodausvaiheessa löytyy yli 50 % puutteista, mikä on paljon suurempi osuus kuin on oletettu. Tämä voi osaltaan johtua tasajakaumasta, jolla oletusprosentti lasketaan. Todennäköisesti, jos oletusprosentti laskettaisiin vanhojen, samantyyppisten projektien puutetietojen perusteella, mielenkiintoisuus voisi olla toinen. Jos jakauma kuitenkin olisi todenmukainen, niin voitaisiin päätellä, että koodausvaiheen tarkastusmenettelyt toimivat suhteellisen hyvin.

Taulukko 4.4. Prosessivaiheiden mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Prosessi-vaihe	Lukumäärä	Löytö (%)	Oletus (%)	Erotus (%)	Erotusitseisarvo
3	CODE	215	56.88	11.11	45.77	45.77
15	PT	1	0.26	11.11	-10.85	10.85
16	BLD	2	0.53	11.11	-10.58	10.58
25	ST	6	1.59	11.11	-9.52	9.52
39	PLD	18	4.76	11.11	-6.35	6.35
41	CLD	19	5.03	11.11	-6.08	6.08
53	UT	56	14.81	11.11	3.70	3.70

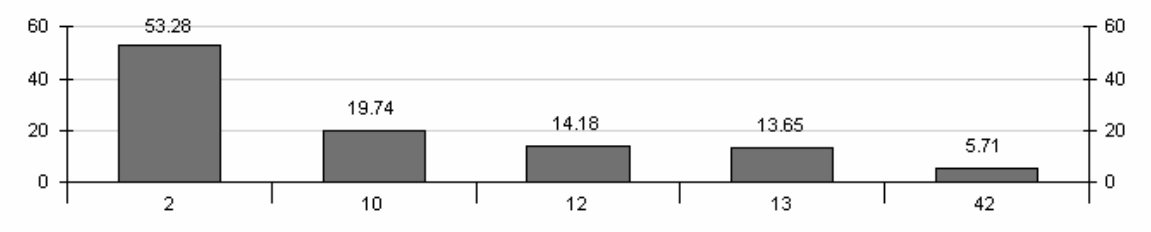


Kuva 4.19. Taulukon 4.4 esitys kuvaajana

Taulukkoon 4.5 ja kuvaan 4.20 on koottu seitsemän mielenkiintoisinta komponenttitapausta. Koodista (CODE) löytyy yli 70 % puutteista. Oletusprosentti on vain 20, joten erotusitseisarvo ja sitä kautta mielenkiintoisuus kasvaa. Tässäkin voidaan arvostella tasajakaumaa tuloksen vääristäjänä. Jos tasajakauma pitäisi kuitenkin paikkaansa, voitaisiin todeta, että koodauksesta puutteita tulisi vähentää koodaajien huolellisuutta lisäämällä ja heitä myös kouluttamalla.

Taulukko 4.5. Komponenttien mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Kompo-nentti	Lukumäärä	Löytö (%)	Oletus (%)	Erotus (%)	Erotusitseisarvo
2	CODE	277	73.28	20.00	53.28	53.28
10	BASE	1	0.26	20.00	-19.74	19.74
12	PLD	22	5.82	20.00	-14.18	14.18
13	CLD	24	6.35	20.00	-13.65	13.65
42	MLD	54	14.29	20.00	-5.71	5.71

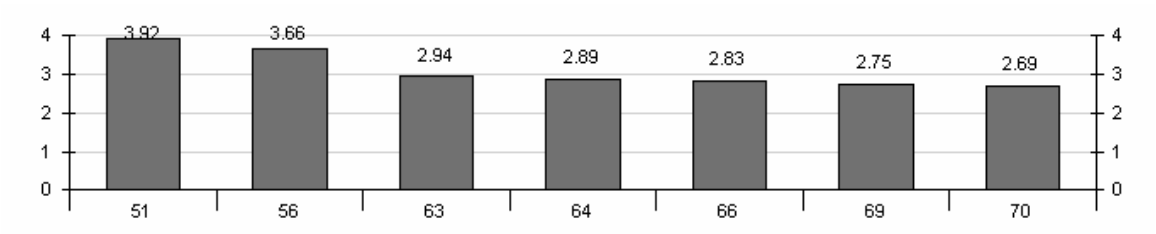


Kuva 4.20. Taulukon 4.5 esitys kuvaajana.

Taulukossa 4.6 ja kuvassa 4.21 on esitetty seitsemän attribuutti- ja kumppaniattribuuttiyhdistelmien mielenkiintoisinta tapausta. Mielenkiintoisin tapaus on attribuutti `lähde` ja sen luokka `ei lähdetietoa` sekä kumppaniattribuutti `laukaisin` ja sen luokka `ei laukaisintietoa`. Tämä ei kerro analysoijalle taaskaan muuta kuin sen, että puutteista ei ole osattu kirjata kaikkea tarvittavaa tietoa.

Taulukko 4.6. Attribuuttien ja kumppaniattribuuttien mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Attribuutti	Luokka	Kumppani-attribuutti	Kumppani-luokka	Löytö (%)	Oletus (%)	Erotus (%)	Erotus-itseis-arvo
51	Lähde	Ei lähdetietoa	Laukaisin	Ei laukaisin-tietoa	1.85	5.77	-3.92	3.92
56	Puutetyyppi	Tarkastus	Vaikutus	Luotettavuus/ Saatavuus	14.55	10.89	3.66	3.66
63	Puuttuva/ virheellinen	Virheellinen	Vaikutus	Ylläpito/Huolto	0.79	3.73	-2.94	2.94
64	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	Puutetyyppi	Funktio	27.25	30.14	-2.89	2.89
66	Lähde	Uudelleen käytetty koodi	Vaikutus	Käytettävyys	3.44	0.61	2.83	2.83
70	Lähde	Ei lähdetietoa	Laukaisin	Suunnittelun johdon-mukaisuus	6.08	3.33	2.75	2.75
21	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	Laukaisin	Ei laukaisin-tietoa	48.68	45.99	2.69	2.69

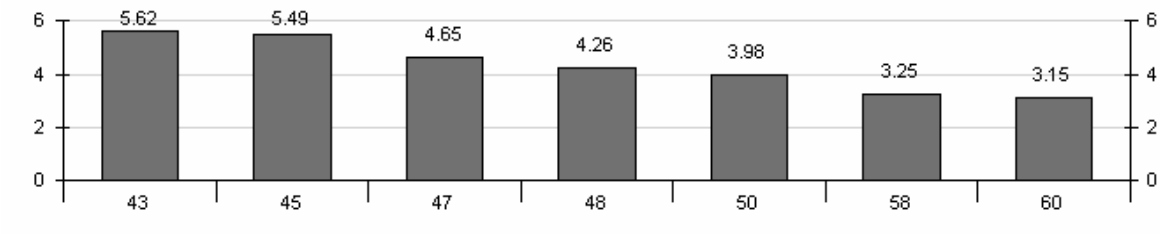


Kuva 4.21. Taulukon 4.6 esitys kuvaajana.

Attribuuttien ja prosessivaiheiden yhteisesiintymisen seitsemän mielenkiintoisinta tapausta on esitetty taulukossa 4.7 ja kuvassa 4.22. Attribuutti `puutetyyppi` ja sen luokka `sijoitus/alustus` sekä prosessivaihe `koodaus` ovat mielenkiintoisia, vaikka eivät suinkaan esiinny useimmiten yhdessä. `Koodaus` sekä attribuutti `lähde` ja sen luokka `uusi funktio/uudelleen kirjoitettu koodi` esiintyvät yleisemmin kuin tämä mielenkiintoisin tapaus yhdessä, mutta koska sitä oletetaankin, niin se ei ole yhtä mielenkiintoista tässä analyysissa.

Taulukko 4.7. Attribuuttien ja prosessivaiheiden mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Attribuutti	Luokka	Prosessi-vaihe	Löytö (%)	Oletus (%)	Erotus (%)	Erotus-itseis-arvo
43	Puutetyyppi	Sijoitus/Alustus	CODE	19.31	13.69	5.62	5.62
45	Lähde	Ei lähdetietoa	CODE	0.53	6.02	-5.49	5.49
47	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	CODE	52.65	48.00	4.65	4.65
48	Lähde	Ei lähdetietoa	PLD	4.76	0.50	4.26	4.26
50	Puutetyyppi	Tarkastus	CODE	15.87	11.89	3.98	3.98
58	Laukaisin	Suunnittelun yhdenmukaisuus	CODE	21.16	17.91	3.25	3.25
60	Vaikutus	Käytettävyys	CODE	13.23	10.08	3.15	3.15

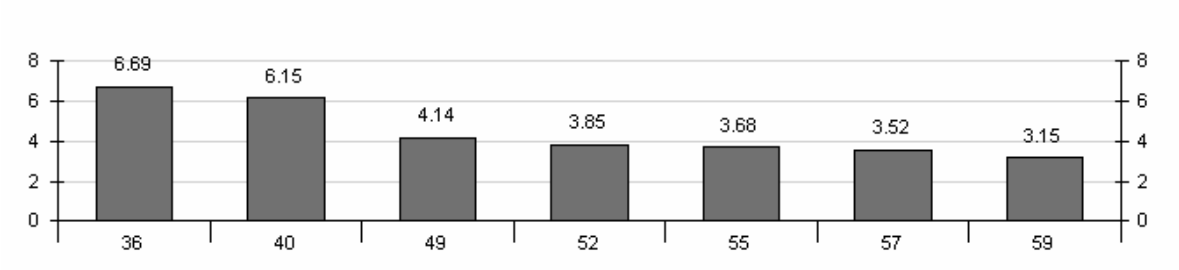


Kuva 4.22. Taulukon 4.7 esitys kuvaajana

Taulukossa 4.8 ja kuvassa 4.23 on esitetty seitsemän mielenkiintoisinta tapausta attribuutin ja komponentin yhteisesiintymisestä. Mielenkiintoisimman tapauksen attribuutti on `lähde` ja sen luokka on `ei lähdetietoa`, joten tapauksesta ei juuri saada tietoa. Toiseksi mielenkiintoisin tapaus vahvistaa entisiä analyyseja eli attribuutti `lähde` ja sen luokka `uusi funktio/uudelleen kirjoitettu koodi` ja puutteen sijaintina `koodi` esiintyvät usein yhdessä ja ovat mielenkiintoisia.

Taulukko 4.8. Attribuuttien ja komponenttien mielenkiintoisuus MiKiFu-järjestelmällä analysoituna.

Järjestys	Attribuutti	Luokka	Komponentti	Löytö (%)	Oletus (%)	Erotus (%)	Erotusitseisarvo
36	Lähde	Ei lähdetietoa	CODE	1.06	7.75	-6.69	6.69
40	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	CODE	67.99	61.84	6.15	6.15
49	Lähde	Ei lähdetietoa	PLD	4.76	0.62	4.14	4.14
52	Lähde	Uusi funktio/ Uudelleen kirjoitettu koodi	PLD	1.06	4.91	-3.85	3.85
55	Puutetyyppi	Funktio	CODE	22.49	26.17	-3.68	3.68
57	Puutetyyppi	Sijoitus/Alustus	CODE	21.16	17.64	3.52	3.52
59	Vaikutus	Käytettävyys	CODE	16.14	12.99	3.15	3.15



Kuva 4.23. Taulukon 4.8 esitys kuvaajana

4.2.3 Historiatietoon perustuva esimerkki

Analysoimme esimerkkiaineistot ODC2 ja ODC5 toistensa historiatietoon perustuen. Koska ODC2-aineisto sisältää vain yhden prosessivaiheen ja yhden komponentin päätimme analysoida vain attribuuttitiedot. Historiatietona käytimme siis oletusprosenttina toisen aineiston löytöprosenttia. Niille attribuuteille ja luokille, joille ei ollut vasti-

netta toisessa aineistossa, laskimme oletusprosentiksi jäävän prosenttiosuuden keskiarvon niiden attribuuttiluokkien kesken, joilla historiatiedoissa ei ole vastinetta. Toisena vaihtoehtona olisi asettaa oletusprosentiksi nolla, eli esiintymiä ei aikaisemmin olisi ollut. Huomattavaa on myös se, että ODC2-aineistossa on attribuutti `laukaisin` ja sen luokka `taaksepäin yhteensopivuus` sekä attribuutti `laukaisin` ja sen luokka `lateraalinen yhteensopivuus` on erikseen luokiteltu, kun taas aineistossa ODC5 ne on luokiteltu samaan luokkaan. Esimerkki osoittaa, että organisaatiossa olisi hyvä noudattaa samaa luokitusta eri projekteihin.

4.2.3.1 ODC2-aineiston attribuutit ODC5-aineisto historiatietona

Taulukossa 4.9 on esitetty tulokset historiatietoon perustuvasta laskelmasta. Attribuutti- ja luokkasarakkeissa ovat ODC2-aineistossa esiintyneet attribuutit ja niiden luokat. Löytö (%) kertoo löydettyjen puutteiden prosenttiosuuden kyseisen attribuutin luokassa. Oletus (%) on historiatietoon perustuva oletusprosentti: jos ODC5-aineistossa on samaa attribuuttia ja sen luokkaa edustava löytöprosentti, on sitä käytetty tässä oletusprosenttina. Erotus (%) on historiatietoon perustuvan laskelman löytöprosentin ja oletusprosentin erotus. Taulukon mielenkiintoisuusjärjestys on säilytetty tasajakaumalaskelmasta saatuna mielenkiintoisuusjärjestyksenä (taulukko 4.1). Tässä esimerkissä attribuutin `laukaisin` ja sen luokkien `lateraalinen yhteensopivuus` ja `taaksepäin yhteensopivuus` oletusprosentti on laskettu jakamalla ODC5-aineiston löytöprosentti kahdella, koska ODC5-aineistossa `lateraalinen yhteensopivuus` ja `taaksepäin yhteensopivuus` on luokiteltu samaan luokkaan.

Taulukosta 4.9 näemme, että mielenkiintoisuus on historiatietoon perustuen erilaista ja erot eri tavalla tulkittavia. Nyt attribuutti `tyyppi` ja sen luokka `dokumentti` ovat entistäkin mielenkiintoisempia, koska oletuksena on ollut hyvin pieni prosentti, mutta löytyneiden tapausten määrä pysynyt samana. Tapauksessa attribuuttina `laukaisin` ja luokkana `samanaikaisuus` taas huomaamme, että aineistot antavat samantyyppisiä tuloksia oletusprosentin laskennasta riippumatta.

Myös erotusprosentin etumerkki on osissa tapauksista vaihtunut. Se ei sinänsä kerro muuta kuin sen, että oletus- ja löytöprosentit ovat olleet eri suhteessa kuin tasa-

jakaumalla laskien. Jos projektit olisivat samantyyppisiä, niin esimerkiksi attribuutin `tyyppi` ja sen luokan `dokumentti` puutteiden määrästä olisi syytä huolestua. Edellisessä projektissa löydettyjä puutteita on ollut vain 1,32 prosenttia ja nyt 45,96 %.

Taulukko 4.9 ODC2-aineiston attribuutit ODC5-aineiston löytöprosenttiin perustuen[6]

Attribuutti	**Luokka**	**Löytö (%)**	**Oletus (%)**	**Erotus (%)**	**|Erotus (%)|**
Tyyppi	*Dokumentti*	*46,96*	*1,32*	*45,64*	*45,64*
Laukaisin	Toiminnallinen semantiikka	33,6	63,21	-29,61	29,61
Laukaisin	*Samanaikaisuus*	*2,02*	*1,06*	*0,96*	*0,96*
Tyyppi	*Tarkastus*	*0,4*	*20,90*	*-20,50*	*20,50*
Tyyppi	Rakennus/pakkaus/	0,4	10,33	-9,93	9,93
Laukaisin	*Harvinainen tilanne*	*3,24*	*2,12*	*1,12*	*1,12*
Tyyppi	Toimeksianto	1,62	10,33	-8,71	8,71
Tyyppi	*Funktio*	*21,86*	*35,71*	*-13,85*	*13,85*
Tyyppi	*Ajoitus/sarjallistaminen*	*4,86*	*5,03*	*-0,17*	*0,17*
Laukaisin	*Dokumenttien johdonmukaisuus*	*21,86*	*1,59*	*20,27*	*20,27*
Laukaisin	*Taaksepäin yhteensopivuus*	*20,24*	*0,27*	*19,98*	*19,98*
Laukaisin	*Lateraalinen yhteensopivuus*	*8,91*	*0,27*	*8,65*	*8,65*
Laukaisin	*Suunnittelun yhdenmukaisuus*	*10,12*	*31,48*	*-21,36*	*21,36*
Tyyppi	*Algoritmi*	*9,31*	*6,08*	*3,23*	*3,23*
Tyyppi	Rajapinta	14,57	10,33	4,24	4,24
Puuttuva/virh.	*Puuttuu*	*50,61*	*41,27*	*9,34*	*9,34*
Puuttuva/virh.	*Virheellinen*	*49,39*	*58,73*	*-9,34*	*9,34*

4.2.3.2 ODC5-aineiston attribuutit ODC2-aineisto historiatietona

Taulukon 4.10 sarakkeet ovat merkitykseltään samoja kuin taulukossa 4.9. Taulukosta 4.3 nähdään tasajakauman antamat mielenkiintoisuusarvot. Tässä esimerkissä oletus (%) on laskettu attribuutille `laukaisin` ja `taaksepäin/lateraalinen yhteensopivuus` ODC2-aineiston attribuutin `laukaisin` ja sen erillisten luokkien `lateraalinen yhteensopivuus` ja `taaksepäin yhteensopivuus` yhteenlaskettuna löytöprosenttina.

[6] Kursivoidulla olevat rivit ovat niitä, joissa historiatietoa on ollut mahdollista käyttää

Aineistossa ODC2 `lateraalinen yhteensopivuus` ja `taaksepäin yhteensopivuus` ovat luokiteltu omiksi luokikseen.

Taulukko 4.10 ODC5-aineiston attribuutit ODC2-aineiston löytöprosenttiin perustuen[7]

Attribuutti	**Luokka**	**Löytö (%)**	**Oletus (%)**	**Erotus (%)**	**|Erotus (%)|**
Lähde	Uusi funktio	84,39	25,00	59,39	59,39
Laukaisin	Ei laukaisin tietoa	54,5	7,20	47,30	47,30
Vaikutus	Luotettavuus	52,12	11,11	41,01	41,01
Lähde	Aiemmin julkaistu	1,59	25,00	-23,41	23,41
Lähde	Uudelleen käytetty koodi	3,44	25,00	-21,56	21,56
Tyyppi	*Funktio*	*35,71*	*21,86*	13,85	*13,85*
Laukaisin	*Suunnittelun yhdenmukaisuus*	*31,48*	*10,12*	21,36	*21,36*
Lähde	Ei lähde tietoa	10,58	25,00	-14,42	14,42
Tyyppi	*Dokumentti*	*1,32*	*46,96*	-45,64	*45,64*
Vaikutus	Muutto	0,53	11,11	-10,58	10,58
Vaikutus	Dokumentointi	0,53	11,11	-10,58	10,58
Vaikutus	Yhteensopivuus	0,53	11,11	-10,58	10,58
Vaikutus	Suorituskyky	0,53	11,11	-10,58	10,58
Laukaisin	*Taaksepäin/lateraalinen yht.sopivuus*	*0,53*	*29,95*	-29,42	*29,42*
Laukaisin	*Samanaikaisuus*	*1,06*	*2,02*	-0,96	*0,96*
Laukaisin	Testien peräkkäisyys	1,32	7,20	-5,88	5,88
Tyyppi	Sijoitus/alustus	24,07	2,04	22,03	22,03
Laukaisin	*Dokumenttien johdonmukaisuus*	*1,59*	*21,86*	-20,27	*20,27*
Tyyppi	*Ajoitus/sarjallistaminen*	*5,03*	*4,86*	0,17	*0,17*
Laukaisin	Looginen sujuvuus	1,85	7,20	-5,35	5,35
Laukaisin	Harvinainen tilanne	2,12	7,20	-5,08	5,08
Puuttuva/virh.	*Puuttuu*	*41,27*	*50,61*	-9,34	*9,34*
Puuttuva/virh.	*Virheellinen*	*58,73*	*49,39*	9,34	*9,34*
Vaikutus	Standardit	2,65	11,11	-8,46	8,46
Tyyppi	*Algoritmi*	*6,08*	*9,31*	-3,23	*3,23*
Vaikutus	Kyvykkyys	19,05	11,11	7,94	7,94
Tyyppi	*Rajapinta*	*6,88*	*14,57*	-7,69	*7,69*
Vaikutus	Käytettävyys	17,72	11,11	6,61	6,61
Tyyppi	*Tarkistus*	*20,9*	*0,4*	20,50	*20,50*
Laukaisin	Testikattavuus	5,56	7,20	-1,64	1,64
Vaikutus	Ylläpito/huolto	6,35	11,11	-4,76	4,76

[7] Kursivoidulla olevat rivit ovat niitä, joissa historiatietoa on ollut mahdollista käyttää

Taulukosta 4.10 nähdään mielenkiintoisuuksien mielenkiintoisimpien puutteiden osalta olevan ennallaan verraten taulukkoon 4.3. Jos projektit olisivat samantyyppisiä, voisi historiatiedosta olla enemmän hyötyä. Nyt voimme vain kuvata miten erilaisia tuloksia historiatietoa käyttämällä voisimme saada.

5 YHTEENVETO

ODC-puuteluokittelua voidaan käyttää eri tyyppisissä projekteissa ja organisaatioissa, koska se on yhtenäinen ja riippumaton. Luokittelu on yksinkertainen ja selkeä, joten inhimilliset virheet puuteluokittelussa ovat vähäisiä. Organisaation on mahdollista muokata luokittelua projektin ja oman toiminnan mukaisesti. Esimerkiksi voidaan käyttää niitä prosessivaiheita, joita yleensäkin organisaatiossa käytetään, vaikka varsinaisessa luokittelumallissa niitä olisi enemmänkin.

ODC-kritiikin osalta voi yhtyä Kan'in (2003) kritiikkiin ODC:stä puutetyyppien osalta, pohtien voidaanko puutetyypit assosioida prosessiin samalla tavoin eri tyyppisissä tuotoksissa ja organisaatioissa. Erot prosessien yksityiskohdissa ja painopisteissä voivat Kan'in mukaan johtaa eroihin puutteiden syissä ja puutetyyppien jakaumissa jopa samantyyppisissä kehitysprosesseissa. Prosessien kypsyystaso vaikuttaa etenkin tuotettujen virheiden määrään. Kan esittää esimerkkinä, että puutetyyppijakauma on erilainen organisaatiolla, jolla tuotettujen virheiden määrä on 60 puutetta/KLOC kuin organisaatiolla, jolla se on 20 puutetta/KLOC. Erot puutteiden syissä vaikuttavat tuotettujen virheiden vähentämiseksi ja puutteiden ehkäisemiseksi tehtävien toimien tehokkuuteen. Toisiin syihin vaikutus on vahvempi kuin toisiin.

Luokiteltua puutetietoa voidaan analysoida attribuuttifokusoinnin avulla. Attribuuttifokusoinnissa etsitään puutetiedoista mielenkiintoisimmat puutteet mielenkiintoisuusfunktion ja suodatinfunktion avulla. Mielenkiintoisuusfunktiolla etsitään mielenkiintoisimmat puutteet. Mielenkiintoisimmat puutteet ovat yleensä niitä, joiden odotettu jakauma eroaa löydetystä jakaumasta. Suodatinfunktio esittää mielenkiintoisimmat tapaukset ihmisen ymmärtämässä muodossa kuvina, kaavioina tai taulukkoina.

Attribuuttifokusoinnilla saadaan reaaliaikaista tietoa ohjelmistoprosessin tilasta. Sen etuna on selkeys ja ohjelmoijan mahdollisuus luokitella ja analysoida puutteet itse. Hän saa tiedot mahdollisista puutteista heti ilman välikäsiä, joten hän pääsee korjaamaan puutteita välittömästi. Projektin päätyttyä puutteet kannattaa myös analysoida. Huomiota kannattaa kiinnittää prosessivaiheisiin, jossa odottamattomia puutteita syntyi ja

analysoida miksi näin pääsi käymään. Seuraavassa projektissa kannattanee käyttää aikaa samantyyppisten puutteiden ennaltaehkäisyyn.

Attribuuttifokusointia helpottamaan kehitetyn MiKiFu- järjestelmän avulla puutteista saadaan sekä lukumäärä- että sijaintitietoa. Tieto saadaan sekä oletetusta että todellisesta esiintymisestä ja se esitetään taulukkomuodossa. Koska mielenkiintoisuuden laskeminen perustuu havaitun ja odotetun puutetietomäärän eroon, saadaan projektista monenlaista palautetta. Jos erotusprosentti on negatiivinen, voidaan pohtia, ovatko tarkastusmenettelyt riittäviä vai jääkö osa puutteista huomaamatta. Jos katsotaan että tarkastusmenettelyt ovat riittäviä, niin projektissa on tapahtunut kehitystä tämän asian kohdalla. Jos taas erotusprosentti on positiivinen eli havaittuja puutteita on enemmän kuin olisi odotettu, voidaan miettiä onko projektissa jotain vialla, vai ovatko tarkastusmenettelyt kehittyneet niin paljon, että puutteita havaitaan odotettua enemmän.

MiKiFu-järjestelmän ja attribuuttifokusoinnin etu on se, että analysointia voidaan tehdä pienistäkin aineistosta, kuten tässäkin tutkielmassa kohdissa 4.2.1 ja 4.2.2 on tehty aineistoilla ODC2 ja ODC5. Attribuuttifokusoinnissa keskitytään etsimään mielenkiintoisia puutetietoja projektista tai jopa projektin osasta, jolloin puutetietoja ei välttämättä ole paljon. Tällöin saadaan tärkeää tietoa projektissa esiintyvien puutteiden tyypistä ja voidaan tehdä korjaavia toimenpiteitä prosessin parantamiseksi.

Jos MiKiFu:a haluttaisiin jatkossa kehittää, voitaisiin siihen lisätä myös tieto siitä missä prosessivaiheessa puute on löytynyt. Tämä tarkentaisi puutteen tietoja ja antaisi informaatiota esimerkiksi projektin tarkastusmenettelyistä.

Toinen parannuskohta MiKiFu-järjestelmään voisi olla historiatietoon perustuva odotusarvon laskeminen attribuuteille, komponenteille ja prosessivaiheille. Siinä odotusarvo laskettaisiin edellisten projektien havaittujen arvojen keskiarvona. Analysoija voisi valita mitkä projektit hän historiatietojen laskuun ottaisi. Silloin hän voisi halutessaan valita samantyyppiset projektit kuin se, joka on analysoitavana. Tällöin saataisiin todellisempaa tietoa mielenkiintoisuudesta, koska harvoinhan puutteet ovat jakautuneet tasajakautuneesti.

MiKiFu:n puutetietojen syöttämisen voisi tehdä automaattiseksi, jos data olisi valmiina, eli ohjelma osaisi ottaa haluamansa tiedot tiedostosta. Tämä helpottaisi ainakin jälkikäteen analysoitavia projekteja, joissa todennäköisesti puutetiedot on jo johonkin kirjattu.

VIITELUETTELO

Ahtiainen, H., Günther, O. (2003) MiKiFu-järjestelmä, Tietojenkäsittelytieteiden erikoistyö, Joensuun yliopisto, Tietojenkäsittelytieteiden laitos.

Bassin, K., Kratschmer, T., Santhanam, P. (1998) Evaluating Software Development Objectively. *IEEE-Software* **1998**(11-12), 66-74.

Bhandari, I. (1995) Attribute Focusing: Data Mining for the layman. *Research Report RC 20136*, IBM Thomas J. Watson Research Center.

Bhandari, I., Roth, N. (1993) Post-process Feedback with and without Attribute Focusing: A Comparative Evaluation. *Proceedings of the 15th international conference on Software Engineering*, Baltimore, Maryland, USA, 89-98.

Bhandari, I., Halliday, M., Tarver, E., Brown D., Chaar J., Chillarege R. (1992) A Case Study of Software Process Improvement During Development. *IEEE Transactions on Software Engineering* **19**(12), 943-956.

Bhandari, I., Halliday, M., Chaar, J., Chillarege, R., Jones, K., Atkinson, J., Lepori-Costello, C., Jasper, P., Tarver, E., Lewis, C., Yonezava, M. (1994) In-process Improvement through Defect Data Interpretation. *IBM Systems Journal* **33**(1), 182-214.

Bhandari, I., Mendonça, M., Dawson, J. (1995) On the Use of Machine-Assisted Knowledge Discovery to Analyze and Reengineer Measurement Frameworks. *Proceedings of the CASCON 1995*, Toronto, Canada, 275-284.

Bhandari, I., Colet, E., Parker, J., Pines, Z., Pratap, R., Ramanujam, K. (1997) Advanced Scout: Data Mining and Knowledge Discovery in NBA Data (Brief Application Description). *Data Mining and Knowledge Discovery* **1**(1), 121-125.

Butcher, M., Munro, H., Kratschmer, T. (2002) Improving software testing via ODC: Three case studies. *IBM Systems Journal*, **41**(1), 31-44.

Chillarege, R. (1992) Orthogonal Defect Classification, *Handbook of Software Reliability Engineering* (toim Lyu, M.) 359-400.

Chillarege, R. (1994) ODC for Process Measurement, Analysis and Control. *Proceedings of the Fourth International Conference on Software Quality*. 1994 McLean, VA, USA, 143-158.

Chillarege, R. (2000) ODC Orthogonal Defect Classification. Center for Software Engineering, IBM Thomas J. Watson Research Center.

Chillarege, R., Biyani, S. (1994) Identifying risk using ODC based growth models. *Software Reliability Engineering*. IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 282-288.

Chillarege, R., Bhandari I., Chaar J., Halliday M., Moebus D., Ray B., Wong, M. (1992) Orthogonal Defect Classification - A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering*, **18**(11) 943-956.

El Emam, K. ja Wieczorek, I. (1998) The Repeatability of Code Defect Classifications. *Proceedings of the Ninth International Symposium on Software Reliability Engineering (ISSRE)*, Paderborn, Germany, 322-333.

Fenton, N., Pfleeger, S. (1997) *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing Company.

Frederics, M., Basili, V. (1998) Using Defect Tracking and Analysis to Improve Software Quality. *A DACS State-of-the-Art Report*. DoD Data & Analysis Center for Software (DACS).

Freimut, B. (2001) Developing and Using Defect Classification Schemes. *IESE-Report No. 072.01/E*, Frauenhofer Institut für Experimentelles Software Engineering.

Freitas, A. (2001) Understanding the Crucial Role of Attribute Interaction in Data Mining. *Artificial Intelligence Review* **16**(3) 177-199.

Grady, R. (1992) *Practical software Metrics for Project Management and Process Improvement*. Prentice-Hall.

Han, J., Kamber, M. (2001) *Data Mining: Consepts and Techniques*. Academic Press.

Harris, E., Bloedorn, E., Rothleder, N. (1998) Recent Experiences with Data Mining in Aviation Safety. *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. Santa Barbara, California, USA, 562-566.

Hilderman, R., Hamilton, H. (1999) Heuristics for Ranking the Interestingness of Discovered Knowledge. *Third Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-99),* Springer Verlag, 204-209.

Hilderman, R., Hamilton, H. (2001) Evaluation of Interestingness Measures for Ranking Discovered Knowledge. *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'01)* (toim. Cheung, D., Williams, G., Li, Q.), Springer-Verlag, 247-259.

Huber, J. (1999) A Comparison of IBM's Orthogonal Defect Classification to Hewlett-Packard's Defect Origins, Types, and Modes. *Proceedings of International Conference on Applications of Software Measurement (ASM)*, San Jose, CA, USA.

IBM Research Group (2002a) Details of ODC, `http://www.research.ibm.com/softeng/ODC/DETODC.HTM`, (27.5.2003) IBM Center for Software Engineering.

IBM Research Group (2002b) Frequently Asked Questions, `http://www.research.ibm.com/softeng/ODC/FAQ.HTM`, (27.5.2003) IBM Center for Software Engineering.

IEEE Computer Society (1993) IEEE Standard Classification for Software Anomalies. IEEE Standard 1044-1993.

IEEE Computer Society (1995) IEEE Guide to Classification for Software Anomalies. IEEE Standard 1044.1-1995.

Kan, S. (2003) *Metrics and Models in Software Quality Engineering*. Addison-Wesley.

Kelly, D., Shepard, T. (2001) A Case Study in the Use of Defect Classification in Inspections. *Proceedings of the CASCON 2001*, Toronto, Canada.

Kröger, E. (2001) Tavoitepohjainen ohjelmistoprosessin kehittäminen. Tietojenkäsittelytieteiden Pro gradu-tutkielma. Joensuun yliopisto, Tietojenkäsittelytieteiden laitos.

Lyu, M. (toim.) (1995) *Handbook of Software Reliability Engineering*, McGraw-Hill, New York.

Mays, R. (1990) Applications of Defect Prevention in Software Development. *IEEE Journal on Selected Areas in Communications*, **8**(2), 164-168.

Mendonça, M., Basili, R. (2000) Validitation of an Approach for Improving Existing Measurement Frameworks, *IEEE Transactions on Software Engineering*, **26**(6), 484-499.

Miller, G. (1956) The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological Review, **63**, 81-97.

Nazeri, Z., Bloedorn, E., Ostwald, P. (2001) Experiences in Mining Aviation Safety Data. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data and Symposium on Principles of Database Systems*.

Piatetsky-Shapiro, G., (1991) Discovery, analysis and presentation of strong rules. teoksessa Piatetsky-Shapiro, G., Frawley, W. (toim.) *Knowledge Discovery in Databases*. MIT Press, Cambridge, MA. 229-248.

Tan, P., Kumar, V., Srivastava, J. *Selecting the Right Interestingness Measure for Association Patterns*. Technical Report 2002-112, Army High Performance Computing Research Center.

Laukaisin-ryhmä	**Laukaisin**	**Merkitys**
Tarkastus-laukaisimet (inspection triggers)	Suunnittelun vastaavuus (design conformance)	Suunnitelman ja koodin tarkastaja huomaa virheen kun vertaa suunnitelmaa tai koodia aikaisemmassa vaiheessa luotuun määritykseen.
	Logiikka/tietovirta (logic/data flow)	Tarkastaja arvioi koodia ohjelmointikäytäntöjen ja standardien avulla.
	Komponenttien yhteensopivuus (lateral compability)	Tarkastaja huomaa ristiriitoja rajapintojen osalta.
	Yhteensopivuus taaksepäin (backward compability)	Tarkastajan tuote/komponenttituntemus auttaa huomaamaan ristiriidan funktion tai koodin yhteen sopivuudessa edeltävään versioon nähden.
	Samanaikaisuus (concurrency)	Tarkastaja tutkii yhteisten resurssien sarjoitettavuutta huomatessaan puutteen (esim. lukitukset).
	Sisäinen dokumentointi (internal documentation)	Tarkastaja huomaa ristiriitoja kommentoinnin ja koodin välillä.
	Kieliriippuvuus (language dependency)	Puutteellisuus löytyy koodista (esim. Pascal koodissa on = vaikka pitäisi olla :=)
	Sivuvaikutukset (side effects)	Asiantuntemuksensa ansiosta tarkastaja pystyy ennakoimaan järjestelmän käyttäytymistä, jota ei ole kirjattu vaatimuksiin (epätavalliset kokoonpanot tai käyttö)
	Harvinaiset tilanteet (rare situations)	Asiantuntemuksensa ansiosta tarkastaja pystyy ennakoimaan järjestelmän käyttäytymistä, jota ei ole kirjattu vaatimuksiin.
Yksikkö-testaus-laukaisimet (unit test triggers)	Yksinkertainen polku (simple path)	Valkoinen/harmaa laatikko-testaus: Testitapaus, jonka avulla puute löydettiin, suoritettiin yksinkertaisella polulla yksinkertaisessa funktiossa.
	Monimutkainen polku (complex path)	Valkoinen/harmaa laatikko-testaus: Testitapaus, jonka avulla puute löydettiin, suoritettiin monimutkaisella polkujen yhdistelmillä joukossa funktioita.
Toiminto-testaus-laukaisimet (function test triggers)	Testikattavuus (test coverage)	Musta laatikko –testaus: Testitapaus, jonka avulla puute löydettiin oli suoraviivainen tapa suorittaa yksittäisen funktion koodia ilman parametreja.
	Testivaihtelevuus (test variation)	Musta laatikko –testaus: Testitapaus, jonka avulla puute löydettiin oli suoraviivainen tapa suorittaa yksittäisen funktion koodia käyttäen vaihtelevia syötteitä ja parametreja.
	Testien peräkkäisyys (test sequencening)	Musta laatikko –testaus: Testitapauksessa, jonka avulla puute löydettiin ajettiin useiden funktioiden koodia tietyssä järjestyksessä. Tämä laukaisin valitaan jos jokainen funktio yksin suoritettuna toimii, mutta tietyssä järjestyksessä ajettuna ei toimi.
	Testien vuorovaikutus (test interaction)	Musta laatikko –testaus: Testitapaus, jonka avulla puute löydettiin aloitti vuorovaikutuksen kahden tai useamman koodirungon välillä. Tämä laukaisin valitaan, jos jokainen funktio yksin suoritettuna toimii, mutta tietyn yhdistelmän suorittaminen ei.
Järjestelmä-testaus-laukaisimet	Työkuorma/stressi (workload/stress)	Järjestelmä toimii resurssirajalla tai sen lähellä. Voidaan kokeilla esim. pienen tai suuren kuormituksen käyttämistä, usean sovelluksen yhtäaikaista ajamista sekä järjestelmän pitkäaikaista käyttöä.

(system test triggers)	Käynnistys/ uudelleen käynnistys (startup/restart)	Järjestelmä tai osajärjestelmä käynnistettiin tai uudelleen käynnistettiin jonkin aikaisemman sammutuksen tai järjestelmän/osajärjestelmän virheen vuoksi.
	Elpyminen/ poikkeus (recovery/exception)	Järjestelmää testataan tarkoituksena herättää poikkeuksien käsittelijä tai jonkin tyyppinen elpymiskoodi.
	Laitteistokokoonpano (hardware configuration)	Järjestelmää testataan, jotta varmistetaan sen toiminta tietyssä laitteistokokoonpanossa.
	Ohjelmistokokoonpano (software configuration)	Järjestelmää testataan, jotta varmistetaan sen toiminta tietyssä ohjelmistokokoonpanossa.
	Normaalitila (blocked tested mode/ normal mode)	Tämä laukaisin valitaan, kun järjestelmätestauksen skeenaarioita ei voida suorittaa, koska esiintyy perusongelmia, jotka estävät sen.

Vaikutus	**Merkitys**
Asennettavuus (installability)	Kuinka helposti asiakas pystyy asentamaan ja ottamaan ohjelmiston käyttöön.
Palveltavuus (servicebility)	Kyky diagnosoida häiriö helposti ja nopeasti, pienin haittavaikutuksin asiakkaalle.
Standardi (standard)	Standardin noudattaminen (esim. käyttöliittymä).
Eheys/turvallisuus (integrity/security)	Järjestelmien ja ohjelmien turvallisuus/eheys ja tiedon suojaus.
Muutos (migration)	Järjestelmän päivittämisen helppous.
Luotettavuus (reliability)	Ohjelmiston kyky suorittaa yhdenmukaisesti sille tarkoitettuja toimintoja ilman yllättäviä keskeytyksiä.
Suorituskyky (performance)	Ohjelmiston nopeus, joka mitataan käyttäjän kykynä toteuttaa tehtäviä.
Dokumentaatio (documentation)	Missä määrin ohjelmiston rakenteen ja käyttötarkoituksen ymmärtämistä helpottavat dokumentit ovat virheettömiä ja täydellisiä.
Vaatimukset (requirements)	Vaatimusmäärittelystä puuttuvat asiakkaan lisäodotukset ohjelmistotuotteelle.
Ylläpito (maintance)	Estävien ja korjaavien päivitysten helppous.
Kyvykkyys (capability)	Ohjelmiston kyky suorittaa sille tarkoitetut tehtävät ja toteuttaa määritellyt vaatimukset.

Lähde	**Merkitys**
Itse kehittäminen (developed in-house)	Puute löytyi alueelta, joka oli organisaation oman kehitysosaston kehittelemää.
Uudelleen käytettävä kirjasto (reused from library)	Puute löytyi käytettäessä osia uudelleenkäytettävästä kirjastosta.
Ulkoistaminen (outsourced)	Puute löytyi osiosta, joka on saatu ulkopuoliselta toimittajalta.
Toisesta ympäristöstä siirtäminen (ported)	Puute löytyi osasta, joka oli kelpuutettu toiseen ympäristöön.

Historia	**Merkitys**
Pohja (base)	Puute on osa tuotetta, jota ei ole muutettu ko. projektissa eikä se ole osa uudelleen käytettyä kirjastoa.
Uusi (new)	Puute on uudessa projektiryhmän luomassa funktiossa.
Uudelleen kirjoitettu (rewritten)	Puute ilmestyi vanhan funktion uudelleen suunnittelussa/kirjoituksessa.
Uudelleen korjattu (refixed)	Puute tuli esiin vanhan puutteen korjauksessa.

Puutetyyppi	**Merkitys**
Sijoitus/alustus (assignment/initialisation)	Arvot sijoitetaan virheellisesti tai ei lainkaan
Tarkastus (cheacking)	Puutteet, jotka johtuvat parametrien tai datan tarkistamattomuudesta ehtorakenteiden yhteydessä
Algoritmi/metodi (algorithm/method)	Tehokkuus-/oikeellisuusongelmat, jotka vaikuttavat tehtävään. Voidaan korjata toteutusta muuttamalla.
Funktio/luokka/olio (function/class/object)	Puutteen korjaus vaatii suunnitelman muutosta, koska se vaikuttaa käyttöliittymään, tuoterajapintoihin tai globaaleihin tietorakenteisiin.
Ajoitus/sarjallistaminen (timing/ serialisation)	Jaetun resurssin sarjoitus puuttuu, väärä resurssi sarjoitettu tai käytetty väärää sarjallistamistekniikkaa.
Rajapinta/OO-viestit (interface/ oo messages)	Kommunikointiongelmia moduulien, komponenttien, laiteajurien, olioiden ja funktioiden välillä, kun kommunikointi on toteutettu makroja, kutsulauseita, ohjauslohkoja ja parametrilistoja käyttäen.
Yhteys (relationship)	Ongelmat, jotka liittyvät yhteyksiin proseduurien, tietorakenteiden ja olioiden välillä.

Puutemäärite	**Merkitys**
Puuttuva (missing)	Puute johtuu laiminlyönnistä.
Virheellinen (incorrect)	Puute johtuu toimeksiannosta.
Häiriö (extraneous)	Puute johtuu jostain dokumentin tai koodin kannalta epäolennaisesta tai asiaankuulumattomasta asiasta.