TAMPEREEN TEKNILLINEN YLIOPISTO

Otto Hylli

Ohjelmistoarkkitehtuurien tietämyskannan kehittäminen

Diplomityö

Tarkastaja: Kai Koskimies
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-
neuvoston kokouksessa 07.09.2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Otto Hylli: Ohjelmistoarkkitehtuurien tietämyskannan kehittäminen
Diplomityö, 49 sivua, 1 liitesivu
Maaliskuu 2012
Pääaine: ohjelmistotuotanto
Tarkastajat: Kai Koskimies
Avainsanat: ohjelmistoarkkitehtuuri, arkkitehtuuritietämys, arkkitehtuuritietämyksen hallinta

Viime vuosina ohjelmistoarkkitehtuurien tutkimuksessa on nostettu esiin arkkitehtuuritietämyksen käsite. Arkkitehtuuritietämys kattaa ohjelmiston arkkitehtuurin ja siihen vaikuttavat asiat laajasti, käsittäen perinteisen arkkitehtuurin kuten ohjelmakomponenttien ja niiden suhteiden lisäksi muun muassa suunnittelupäätökset ja arkkitehtuurin pohjalla olevat vaatimukset. Järjestelmää koskevan arkkitehtuuritietämyksen kerääminen auttaa järjestelmän kehitystä ja ylläpitoa, mutta sen kerääminen perinteisiin dokumentteihin voi olla ongelmallista. Tämän vuoksi tässä työssä kehitettiin selaimella käytettävä arkkitehtuuritietämyskanta tietämyksen tallentamiseen ja tarkasteluun. Tietämyskannan idea on, että sinne tallennetaan kaikki järjestelmän elinkaaren aikana syntyvä tieto sitä mukaa, kun se syntyy. Tallennettua tietoa voidaan sitten tarkastella eri sidosryhmien tarpeisiin toteutettujen näkymien avulla.

Tietämyskanta rakennettiin Polarion ALM -ohjelmiston elinkaaren hallintajärjestelmän päälle. Kantaan tallennettavan tiedon rakenteen määritti tätä varten kehitetty arkkitehtuuritietämyksen metamalli, joka yhdistää perinteisen arkkitehtuurin arkkitehtuuriarvioinnissa syntyvään tietoon ja yleiseen arkkitehtuuritietoon. Polarionin tiedon syöttö- ja tarkasteluominaisuuksia täydentämään tietämyskantaan toteutettiin arkkitehtuuriarvioinnissa käytettävä tiedon syöttötyökalu ja kolme erilaista ja eri tarkoituksiin sopivaa näkymää tallennettuun tietoon.

Tietämyskantaa arvioitiin tallentamalla sinne erään oppimisympäristön arkkitehtuuritietämystä ja pyytämällä järjestelmää tuntevia henkilöitä arvioimaan tiedon syöttötyökalua sekä tietämyksestä luotuja näkymiä. Tämä arviointi ei ollut kaikilta osin kovin kattavaa, mutta sen pohjalta voidaan sanoa, että tietämyskanta ja siihen toteutetut ominaisuudet ovat periaatteessa toimivia. Tietämyskantaa ei myöskään voi sanoa tämän työn perusteella valmiiksi, ja se vaatiikin vielä jatkokehitystä, jolle tämä työ tarjoaa hyvän pohjan.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Otto Hylli: Design and Implementation of a Software Architecture Knowledge Base
Master of Science Thesis, 49 pages, 1 Appendix page
March 2012
Major: software engineering
Examiner: Kai Koskimies
Keywords: software architecture, architecture knowledge, architecture knowledge management

The concept of architecture knowledge has gained attention in software architecture research during the last few years. Architecture knowledge encompasses broadly the architecture of a system and things affecting it. In addition to traditional architecture such as components and their relationships architecture knowledge includes, among other things, design decisions and architecturally significant requirements. Collecting the architecture knowledge of a system helps its development and evolution but storing it into normal documents can be problematic. That is why the aim of this master's thesis was to design and implement an architecture knowledge base for storing and viewing architectural knowledge. All knowledge, created during the system's lifecycle, will be stored directly into the knowledge base and it can be viewed using different views designed to meet the requirements of a particular stakeholder.

The architecture knowledge base was build using Polarion application lifecycle management system. The structure of the knowledge base was defined by an architecture knowledge metamodel, which combines traditional architecture to general architecture knowledge and knowledge from architecture evaluations. Polarion's build-in features were enhanced by developing a tool for inputting knowledge created in architecture evaluation and three different views for viewing the knowledge.

The knowledge base was evaluated by using it to store the architecture knowledge of a learning environment and asking persons familiar with the system to evaluate the knowledge input tool and the views generated from the knowledge. This evaluation was not, in all respects, very extensive but based on the evaluation it can be said that the architecture knowledge base and the features added to it are essentially functional. The knowledge base, however, is not finished and it requires further development for which this master's thesis offers a good basis.

ALKUSANAT

Tämä diplomityö tehtiin TTY:n ohjelmistotekniikan laitokselle osana Sulava-projektia. Projektin tarkoituksena oli kehittää ohjelmistoarkkitehtuurikäytäntöjä sulautettujen koneenohjausjärjestelmien kehityksessä. Tavoitteena oli kehittää menetelmiä ja tekniikoita, jotka auttavat laadukkaiden arkkitehtuurien kehittämisessä koneenohjausjärjestelmille. Yksi projektin teemoista oli arkkitehtuuritietämys ja selaimella käytettävän arkkitehtuuritietämyskannan kehittäminen, johon tämä työ liittyy.

Haluan kiittää työni ohjaajaa Kai Koskimiestä sekä Veli-Pekka Elorantaa neuvoista ja diplomityön lukemisesta sekä korjauksien ja parannusehdotusten antamisesta. Samoin kiitokset Peter Saloselle tekstin lukemisesta ja kommentoinnista sekä avusta tässä työssä esiintyvien kuvien kanssa. Kiitokset myös Jari Peltoselle ja Rami Laineelle osallistumisesta ja avusta arkkitehtuuritietämyskannan arvioinnissa IDLE-järjestelmällä.

Tampereella 16. helmikuuta 2012

Otto Hylli

SISÄLLYS

TERMIT JA LYHENTEET

Lyhenne tai termi:	**Selitys:**
API	Application programming interface, ohjelmointirajapinta mahdollistaa ohjelmien välisen vuorovaikutuksen
HTML	Hyper Text Markup Language, käytetään www-sivujen tekemisessä sivujen rakenteen esittämiseen
HTTP	Hypertext Transfer Protocol, protokolla jota selaimet ja WWW-palvelimet käyttävät tiedon siirtoon
Sulava-projekti	Projekti arkkitehtuurikäytäntöjen kehittämiseen sulautetuissa koneenohjausjärjestelmissä, jonka osa tämä työ on.
TTY	Tampereen teknillinen yliopisto
UML	Unified Modelling Language on graafinen mallinnuskieli, joka sisältää eri kaaviotyyppejä ohjelmistojen rakenteen, käyttäytymisen ja vuorovaikutuksen kuvaamiseen.
XHTML	HTML-kielen versio, joka täyttää XML-kielen vaatimukset.
XML	Extensible Markup Language on merkintäkieli, jolla voidaan kuvata tiedon merkitys tai rakenne tiedon seassa.

1. JOHDANTO

Ohjelmistoarkkitehtuurilla on tärkeä asema nykyaikaisessa ohjelmistokehityksessä. Arkkitehtuuria ja siihen liittyviä käsitteitä on myös käsitelty ohjelmistoalan tutkijayhteisössä aktiivisesti viimeisen parinkymmenen vuoden ajan siitä lähtien, kun aihe tuotiin esiin [15]. Ohjelmistoarkkitehtuurilla tarkoitetaan IEEE 1471 -standardin määritelmän mukaan järjestelmän perusrakennetta, joka sisältää järjestelmän komponentit, niiden suhteet toisiinsa ja ympäristöön sekä suunnittelua ja evoluutiota ohjaavat periaatteet [1].

Ohjelmistokehityksessä ohjelmistoarkkitehtuuri ja siihen liittyvät tiedot ovat tärkeässä roolissa koko ohjelmiston elinkaaren ajan ohjelmiston suunnittelusta sen ylläpitoon. Arkkitehtuurilla on lisäksi monia tehtäviä ja arkkitehtuuriin liittyvää tietoa käyttävät monet osapuolet. Ohjelmiston suunnitteluvaiheessa tehdyt arkkitehtuuripäätökset lyövät jo lukkoon monia ohjelmiston ominaisuuksia ja vaikuttavat sen laatuominaisuuksiin kuten suorituskykyyn ja ylläpidettävyyteen. Arkkitehtuuri myös ohjaa ohjelman yksityiskohtaista suunnittelua ja toteutusta. Lisäksi sillä on suuri merkitys ylläpitovaiheessa. Ohjelmistoon ylläpidossa tehtävien muutosten tulisi olla arkkitehtuuriin sopivia, eikä niillä saa olla haitallisia vaikutuksia ohjelman muihin osiin. Ylläpitäjän tulee siis tuntea ohjelman arkkitehtuuri, ja hänen tulee pystyä analysoimaan sitä.

Arkkitehtuuri toimii myös sidosryhmien välisenä kommunikaatiovälineenä, ja se auttaa eri osapuolia ohjelmiston ymmärtämisessä. Se myös mahdollistaa ohjelman analyysin ja arvioinnin aikaisessa vaiheessa. [2] Analyysin ja arvioinnin avulla voidaan välttää myöhempien vaiheiden ongelmia sekä saada parempi ymmärrys arkkitehtuurista.

Arkkitehtuurista puhuttaessa ja sitä dokumentoitaessa keskitytään yleensä pelkkään lopputulokseen: ohjelmakomponentteihin ja niiden suhteisiin eli itse ohjelman rakenteeseen. Tämän takia menetetään kuitenkin paljon tietoa arkkitehtuurista. Tästä tiedosta olisi myöhemmin hyötyä, ja sen puutteesta voi seurata ongelmia. [3] Tämän vuoksi ohjelmistoarkkitehtuurien tutkijayhteisössä on nostettu esiin arkkitehtuuritietämyksen käsite. Arkkitehtuuritietämys kattaa järjestelmän arkkitehtuurin ja siihen vaikuttavat asiat laajasti. Varsinaisen arkkitehtuurin kuvauksen lisäksi arkkitehtuuritietämyksen osia ovat esimerkiksi Tang et al. [17] mukaan arkkitehtuuripäätökset, perustelut ja vaatimukset.

Järjestelmän arkkitehtuuritietämyksen kerääminen mahdollisimman kattavasti edesauttaa järjestelmän kehitystä, ylläpitoa sekä myöskin uusien järjestelmien kehitystä. Kerättyä arkkitehtuuritietämystä voidaan tallentaa perinteisiin dokumentteihin, mutta tähän liittyy useita ongelmia etenkin suurten järjestelmien tapauksessa [9]. Tästä syystä tämän työn tarkoituksena on kehittää internet-selaimella käytettävä arkkitehtuuritietämyskanta. Tietämyskannan idea on, että sinne talletetaan kaikki ohjelmiston elinkaaren aikana syntyvä tietämys sitä mukaa, kun tietoa syntyy. Tietoa tarkastellaan erilaisten näkymien kautta, joita tietämyskanta generoi eri käyttäjien tarpeisiin, jolloin tietoa tarvitseva saa juuri haluamansa tiedon tarkoitukseen sopivassa muodossa.

Ensiksi tässä työssä esitellään tarkemmin arkkitehtuuritietämystä koskevaa teoriaa, joka selittää tarkemmin, mitä arkkitehtuuritietämys on ja mitä hyötyä siitä on. Seuraavaksi luvussa kolme esitellään arkkitehtuuritietämyskannan teknisenä toteutusalustana käytetty Polarion ALM -ohjelmiston elinkaaren hallintajärjestelmä. Luvussa neljä esitellään puolestaan tietämyskannan rakenteen pohjaksi kehitettyä arkkitehtuuritietämyksen metamallia, joka selittää millaista tietoa tietämyskantaan tarkalleen tallennetaan. Luvuissa viisi ja kuusi esitellään tietämyskantaan toteutettuja tiedon syöttö- ja tarkasteluominaisuuksia. Luvussa seitsemän havainnollistetaan tietämyskannan toimintaperiaatetta käyttämällä sitä TTY:llä kehitetyn oppimisympäristön arkkitehtuuritietämyksen tallentamiseen. Luvussa kahdeksan arvioidaan tietämyskantaan toteutettuja ominaisuuksia, ja luvussa yhdeksän esitetään yhteenveto sekä johtopäätöksiä tehdystä työstä.

2. ARKKITEHTUURITIETÄMYS JA SEN HALLINTA

2.1 Arkkitehtuuritietämys

Arkkitehtuuritietämykselle ei ole olemassa yhtä määritelmää. Avgerioun [2] mukaan arkkitehtuuritietämys on integroitu representaatio ohjelmistointensiivisen järjestelmän tai järjestelmä perheen ohjelmistoarkkitehtuurista, johon kuuluu arkkitehtuuripäätökset ja niiden perustelut sekä ulkoiset vaikutteet ja kehitysympäristö. De Boerin et al. arkkitehtuuritietämyksen ydinmalli korostaa puolestaan suhteita. Sen mukaan arkkitehtuuritietämys on joukko suhteita päätösten, ihmisten, arkkitehtuurikuvauksien ja prosessien välillä. [4]

Yksi arkkitehtuuritietämyksen keräämisen tarkoitus on säilyttää tieto siitä, miten ja miksi tiettyyn arkkitehtuuriin on päädytty eli miksi järjestelmä on sellainen kuin se on. Tällä on suuri merkitys ohjelman evoluutiovaiheessa. Jos tehdyt arkkitehtuuria koskevat päätökset ja niiden syyt sekä muu oleellinen tieto eivät ole näkyvillä, vaikeutuu järjestelmän kehittäminen. Tehdyt muutokset voivat rikkoa aiemmin tehtyjen päätösten seurauksena syntyneitä suunnittelusääntöjä ja -rajoituksia, joita ei ole kirjattu ylös. Päätösten muuttaminen tai vanhentuneiden päätösten seurausten poistaminen voi jäädä tekemättä, kun muutosten seurauksia ei tiedon puutteen vuoksi pystytä selvittämään. Tästä seuraa järjestelmän eroosio, joka aiheuttaa suuret ylläpitokustannukset ja voi johtaa järjestelmästä luopumiseen. [3] Arkkitehtuuritietämyksen kerääminen edesauttaa myös tiedon uudelleenkäyttöä, kun vanhoja ratkaisuja ja päätöksiä voidaan tutkia ja soveltaa uusiin tilanteisiin.

2.1.1 Arkkitehtuuripäätökset

Arkkitehtuuritietämyksessä keskeinen käsite on arkkitehtuurinen suunnittelupäätös. Vaikka arkkitehtuuritietämyksestä on monia eri näkökulmia ja määritelmiä, ovat arkkitehtuuripäätökset useita näkökulmia yhdistävä tekijä [5]. Erään määritelmän mukaan arkkitehtuuri on joukko päätöksiä [10]. Jokainen arkkitehtuuripäätös aiheuttaa muutoksen arkkitehtuurissa, jolloin lopullinen arkkitehtuuri on kaikkien näiden muutosten summa. Arkkitehtuuritietämyksen yksi määritelmä on puolestaan tietämys = päätökset + arkkitehtuurin kuvaus [13].

Jansen ja Bosch [10] määrittelevät arkkitehtuurisen suunnittelupäätöksen

seuraavasti:

> Kuvaus joukosta arkkitehtuurisista lisäyksistä, poistoista ja muutoksista ohjelmistoarkkitehtuuriin, perustelut, suunnittelusäännöt ja -rajoitteet sekä lisävaatimukset, jotka (osittain) toteuttavat yhden tai useamman tietylle arkkitehtuurille asetetuista vaatimuksista.

Päätöksen tärkeä osa ovat perustelut eli miksi tietty ratkaisu on tehty. Ratkaisu voi puolestaan olla esimerkiksi komponentin lisäys ohjelmaan tai lisätoiminnallisuuden lisääminen johonkin komponenttiin. Päätös voi käsitellä esimerkiksi arkkitehtuurityylin tai suunnittelumallin valitsemista, jonka avulla täytetään vaatimuksia. Päätöksen seurauksena voi syntyä myöhempiin päätöksiin vaikuttavia suunnittelua ohjaavia suunnittelusääntöjä, rajoja asettavia suunnitteluraajoitteita sekä lisävaatimuksia, jotka on toteutettava. Päätöksessä valitun ratkaisun lisäksi siihen voi liittyä vaihtoehtoisia ratkaisuja, niiden hyviä ja huonoja puolia sekä selitys: miksi päädyttiin tiettyyn ratkaisuun [11].

Arkkitehtuuripäätökset ovat usein kiinteästi sidoksissa toisiinsa vaikuttaen samoihin ohjelman osiin. Yksittäinen päätös voi toisaalta vaikuttaa moneen eri ohjelman osaan. Jos päätöksiä ei dokumentoida erikseen, on niiden löytäminen ja muuttaminen vaikeaa, sillä ne ja niiden väliset suhteet eivät näy selvästi pelkässä suunnittelun lopputuloksessa. [10] Yksittäisten päätösten ja niiden ominaisuuksien lisäksi on tärkeää myös dokumentoida päätösten väliset suhteet, jolloin erilaiset päätösten väliset riippuvuudet ovat jäljitettävissä. Päätösten välillä voi olla monia erilaisia suhteita. Kruchten et al. ovat kehittäneet päätöksiä ja niiden suhteita kuvaavan ontologian, joka määrittää kymmenen erilaista suhdetta. Päätös voi muun muassa koostua useasta tarkentavasta päätöksestä, mahdollistaa jonkin päätöksen, olla riippuvainen jostain toisesta päätöksestä, olla jonkin päätöksen vaihtoehto tai syrjäyttää jonkin yleisen päätöksen jossain tietyssä erikoistapauksessa. [13]

2.1.2 Arkkitehtuuritietämyksen luokittelua

Arkkitehtuuritietämyksen piiriin kuuluvien asioiden ymmärtämisessä auttaa tietämyksen luokittelu, joka auttaa myös tietämyksen lähteiden ja käyttötarkoitusten ymmärtämisessä. Farenhorst ja Boer esittävät kaksi merkittävää arkkitehtuuritietämyksen luokittelutapaa, jotka yhdistämällä saadaan neljä eri kategoriaa. Tietämys voi olla hiljaista tai eksplisiittistä ja se voi olla yleistä tai sovelluskohtaista [5] Hiljaista tietämystä ei ole kirjattuna minnekään, vaan se sijaitsee vain ihmisten pään sisässä. Eksplisiittinen tieto on puolestaan ulkoistettu esimerkiksi dokumentteihin tai malleihin. Yleinen tietämys ei ole sidottu mihinkään yksittäiseen järjestelmään, vaan se käsittelee arkkitehtuuria yleisellä tasolla. Sovelluskohtainen tietämys puolestaan koskee vain jonkin tietyn järjestelmän arkkitehtuuria.

Nämä luokittelut yhdistämällä saadaan neljä erilaista tietämyskategoriaa, jotka sisältävät eri tyyppistä tietoa [5]:

- Hiljainen yleinen tietämys: Arkkitehdin yleinen tietämys ohljemistoarkkitehtuureista. Omien kokemusten kautta saadut ja sisäistetyt taidot kuten menetelmät sekä ratkaisut ja tiedot kuten erilaiset käsitteet.
- Sovelluskohtainen hiljainen tietämys: Sovellusalueesta riippuvaa tietämystä tulevaan arkkitehtuuriratkaisuun vaikuttavista voimista kuten bisnes-tavoitteet, sidosryhmien tarpeet ja ohjelman ympäristö yleisesti.
- Yleinen eksplisiittinen tietämys: Yleistä suunnittelutietoa joka on tuotu esiin esimerkiksi kirjoissa tai artikkeleissa. Tällaista tietoa ovat yleiset suunnittelumallit, arkkitehtuurityylit ja -taktiikat sekä arkkitehtuurikuvauskielet, referenssiarkkitehtuurit ja prosessimallit.
- Sovelluskohtainen eksplisiittinen tietämys: Kaikki tietyn järjestelmän arkkitehtuurista ulkoistettu tieto kuten erilaiset arkkitehtuurinäkymät, mallit ja arkkitehtuurin vaatimukset sekä kirjatut suunnittelupäätökset ja perustelut.

Tang et al. puolestaan luokittelevat arkkitehtuuritietämyksen seuraavasti: kontekstitietämys, yleistietämys, perustelutietämys ja suunnittelutietämys. Kontekstitietämys on yleistä tietoa projektin lähtökohdista kuten arkkitehtuurisesti merkittävät vaatimukset ja projektin konteksti. Yleinen tietämys puolestaan auttaa järjestelmän suunnittelussa. Tällaista tietoa ovat esimerkiksi suunnittelumallit ja arkkitehtuurityylit. Perustelutietämys sisältää asioita, jotka selittävät syntynyttä arkkitehtuuria kuten suunnittelupäätökset, harkitut vaihtoehdot ja tehdyt kompromissit. Suunnittelutietämys puolestaan sisältää kaikki järjestelmän suunnitelmat kuten komponentit ja arkkitehtuurimallit. [17]

Molemmat luokittelut kattavat arkkitehtuuritietämyksen laajasti ja ne sisältävät paljolti samat asiat vain eri tavoin luokiteltuna. Molemmat erottavat yleisen ja tiettyä sovellusta koskevan tietämyksen. Molemmat huomioivat myös arkkitehtuuriin vaikuttavat lähtökohdat kuten vaatimukset ja arkkitehtuurin tuloksen sekä siihen liittyvät perustelut.

2.2 Näkökulmia arkkitehtuuritietämyksen hallinnasta

Erilaisten määritelmien lisäksi arkkitehtuuritietämyksestä ja sen hallinnasta on olemassa erilaisia näkökulmia, jotka korostavat tietämyksen eri osia. Farenhorst et al. ovat tunnistaneet arkkitehtuuritietämystä käsittelevässä kirjallisuuskatsauksessaan neljä erilaista näkökulmaa. Nämä näkökulmat ovat: mallikeskeinen, dynamiikkakeskeinen, vaatimuskeskeinen ja päätöskeskeinen. Keskeisenä osana kaikkiin näkökulmiin pohjautuvassa tietämyksen hallinnassa on tiedon muuntaminen kohdassa 2.1.2

esitellyn hiljainen-eksplisiittinen ja yleinen-sovelluskeskeinen jaon määrittelemien luokkien välillä ja myös niiden sisällä. [5]

Mallikeskeinen näkökulma keskittyy yleistä tietämystä sisältäviin malleihin, joiden avulla tietämyksen uudelleenkäyttö ja arkkitehtuureista keskustelu onnistuu, mutta jotka eivät sovellu automatisoituun käsittelyyn. Tällaisen tietämyksen hallinnassa keskeistä on yleisten mallien kehittäminen eksplisiittisestä sovelluskeskeisestä tiedosta, näiden mallien sisäistäminen hiljaiseksi yleiseksi tiedoksi sekä mallien käyttäminen eli muuntaminen sovelluskohtaiseksi tiedoksi.

Dynamiikka-keskeinen näkökulma keskittyy ohjelmiin, jotka voivat muokata omaa arkkitehtuuriaan ajoaikana, ja formaaleihin esityksiin arkkitehtuurista, jollainen tarvitaan, jotta ohjelma voisi käsitellä omaa arkkitehtuuriaan. Tässä tietämyksenhallinta perustuu lähinnä ohjelmakeskeisen eksplisiittisen tietämyksen eli formaalien mallien kehittämiseen ja niiden hyödyntämiseen ja automaattiseen analysointiin jatkokehityksessä. Tässä käytetään yleistä eksplisiittistä tietoa erilaisista mallinnustavoista kuten arkkitehtuurikuvauskielistä.

Vaatimuskeskeinen näkökulma keskittyy arkkitehtuurin pohjalla oleviin vaatimuksiin ja näiden vaatimusten jäljitettävyyteen arkkitehtuurista sekä vaatimusten ja arkkitehtuurin yhtäaikaiseen kehittämiseen. Tässä pääosassa ovat tietämys ongelmasta ja ratkaisuista sekä näiden yhdistäminen. Tämä keskittyy sovelluskeskeiseen tietämykseen. Hiljaista tietoa jaetaan ja kehitetään eri sidosryhmien vuorovaikutuksessa. Eksplisiittistä tietoa kehitetään yhdistämällä ongelma ja ratkaisutietämystä eli varmistamalla jäljitettävyys arkkitehtuurin ja vaatimusten välillä. Tämän lisäksi tarvitaan tietysti hiljaisen tiedon kuten tavoitteiden muuttamista eksplisiittiseksi ja myös eksplisiittisen tiedon sisäistämistä.

Päätöskeskeinen näkökulma korostaa arkkitehtuurisuunnitelmien taustalla olevia syitä. Tietämyksen hallinnassa keskeistä on päätösten syiden muuttaminen sovelluskohtaiseksi eksplisiittiseksi tiedoksi sekä tämän tiedon käyttäminen. Tiedon muuntamisessa voidaan käyttää apuna yleistä eksplisiittistä tietoa kuten suunnittelumalleja. Arkkitehti voi myös sisäistää erilaisia ratkaisutapoja ja käyttää niitä uusien ongelmien ratkaisemiseen.

2.3 Arkkitehtuuritietämyksen hallinnan vaiheet

Arkkitehtuuritietämyksen hallinta kattaa järjestelmän ja siis sen arkkitehtuurin koko elinkaaren. Tietämystä sekä tuotetaan että käytetään kaikissa vaiheissa. Hofmeister et al. [8] esittävät kolmivaiheisen mallin arkkitehtuurin kehittämiselle. Vaiheet ovat arkkitehtuurin analyysi, synteesi ja arviointi. Selittääkseen, miten arkkitehtuuritietämystä käytetään koko ohjelman elinkaaren ajan, Tang et al. [17] laajensivat mallia vielä toteutus- ja ylläpitovaiheilla.

Analyysivaiheessa arkkitehti selvittää arkkitehtuurin kannalta merkittävät vaati-

mukset perehtymällä arkkitehtuuriin vaikuttaviin tekijöihin ja arkkitehtuurin ympäristöön. Arkkitehti tuottaa uutta tietämystä integroimalla vaatimuksia ja yleistä tietämystä järjestelmän kontekstista. Arkkitehti myös etsii olemassa olevasta tietämyksestä analyysiin vaikuttavaa tietoa. Synteesivaiheessa arkkitehti suunnittelee arkkitehtuurivaatimukset täyttäviä arkkitehtuurisia ratkaisuja, joiden tuottamisen apuna hän voi käyttää yleistä arkkitehtuuritietämystä kuten suunnittelumalleja. Tässä vaiheessa syntyvää tietämystä ovat muun muassa ratkaisuihin liittyvät päätökset ja niiden syyt sekä itse ratkaisut sekä näiden eri osien väliset suhteet kuten vaatimusten ja ratkaisujen välillä ja ratkaisujen ja suunnittelumallien välillä. Arviointivaiheessa arvioijat käyttävät tietämystä selvittääkseen ovatko ratkaisut oikeita. Arvioijat tutkivat ratkaisujen perusteita ja jäljittävät esimerkiksi ratkaisuihin liittyviä vaatimuksia. Tässä vaiheessa voidaan myös mahdollisesti luoda uutta yleistä tietoa, jos arkkitehtuurista löydetään jokin uudelleen käytettävä osa kuten suunnittelumalli. Toteutusvaiheessa suunnittelijat lisäävät suunnittelutietämystä yksityiskohtaisella suunnittelulla. Suunnittelijat ja kehittäjät myös käyttävät tietoa kuten päätöksiä ja niiden perusteluita ymmärtääkseen arkkitehtuuria, jotta he voisivat toteuttaa omat tehtävänsä. Ylläpitovaiheessa arkkitehtuuriin voi vielä tulla muutoksia. Ylläpitäjien pitää lisäksi pystyä tutkimaan suunnittelutietämystä ja selvittämään ratkaisujen syitä, jotta he voisivat tehdä luotettavaa muutosanalyysiä. [17]

2.4 Arkkitehtuuritietämyksen käyttäjiä

Arkkitehtuuritietämyksellä on useita eri käyttäjiä, jotka joko tuottavat tai käyttävät tietämystä tai tekevät molempia. Kaikilla näillä käyttäjillä on omat tehtävänsä, jota varten he arkkitehtuuritietämystä tarvitsevat.

Kruchten et al. [13] sekä Van Der Ven et al. [18] ovat tunnistaneet seuraavia käyttäjäryhmiä:

- Arkkitehdit, jotka suunnittelevat isoa järjestelmää tai sen osaa, dokumentoivat suurimman osan arkkitehtuurista. Tärkeää on, että he pystyvät dokumentoimaan tekemänsä päätökset ja niihin liittyvät oletukset [13]. Arkkitehti tarvitsee näkymiä, jotka antavat yleiskuvan arkkitehtuurin tilasta kuten miten hyvin vaatimukset on katettu ja onko arkkitehtuuri yhdenmukainen [18].

- Järjestelmään integroituvia osia suunnittelevat arkkitehdit tarvitsevat tietoa siitä, miten järjestelmän osat vaikuttavat heidän päätöksiinsä [13].

- Kehittäjät, jotka ovat mukana suunnitelmien ja päätösten toteuttamisessa, tarvitsevat toteutustoimintaa tukevaa tietoa [13].

- Arvioijat tarvitsevat tietoa järjestelmän suunnittelun laadun tai edistymisen

arviointiin [13]. He haluavat ymmärtää arkkitehtuurin nopeasti ja heidän tavoitteenaan on ongelmien kuten huonojen päätösten ja epäjohdonmukaisuuden löytäminen [18].

- Analyytikot ovat tekemisissä heitä kiinnostavien vaatimusten kanssa [13].
- Projektipäällikköä kiinnostaa projektin eteneminen ja hän tarvitsee tietoa arkkitehtuurin tilasta, mahdollisista ongelmista, tulevista riskeistä ja niiden välttämismahdollisuuksista sekä kehitykseen vaikuttavista sidosryhmistä esimerkiksi, mikä taho on suurin riski. [18]
- Ylläpitäjät tarvitsevat järjestelmää kehittäessään tai korjatessaan tietoa siitä, miten heidän tekemänsä päätökset sopivat yhteen jo tehtyjen kanssa [13]. Erityisen tärkeää tietoa ylläpitäjille ovat vaihtoehtoiset tai epäonnistuneet ratkaisut, jotka auttavat järjestelmän ymmärtämisessä ja auttavat välttämään aiemmin tehtyjä virheitä [2]. Lisäksi ylläpitäjän pitää pystyä selvittämään suunnittelemansa muutoksen seuraukset [18].
- Käyttäjät käyttävät arkkitehtuuritietoa esimerkiksi suunnitellun järjestelmän kanssa vuorovaikuttavan järjestelmän kehittämiseen tai kehitetyn järjestelmän dokumentointiin [13].
- Uudelleenkäyttäjät haluavat käyttää olemassa olevaa arkkitehtuuritietämystä uuden järjestelmän suunnittelussa [13].
- Opiskelijat haluavat opiskella ohjelmistoarkkitehtuureja tutkimalla arkkitehtuuritietämystä eri näkökulmista [13].
- Tutkijat käyttävät tietämystä tutkimusaineistonaan etsiessään esimerkiksi uusia suunnittelumalleja tai muuta uutta tietoa [13].
- Ohjelmalliset työkalut voivat tuottaa tietoa sekä analysoida jo olemassa olevaa tietoa. Työkalut voivat esimerkiksi tarkistaa suunnitelman johdonmukaisuuden, etsiä malleja tai generoida raportteja. [13]

2.5 Arkkitehtuuritietämyksen hallintatyökalut

Yksinkertaisimmillaan arkkitehtuuritietämystä voidaan kerätä perinteisellä dokumentoinnilla. Perinteisiin dokumentteihin liittyy kuitenkin monia ongelmia, jotka hankaloittavat tietämyksen tallentamista ja käyttöä. Jansen et al. [9] ovat listanneet useita perinteisen dokumentoinnin ongelmia, jotka tulevat esiin etenkin isompien järjestelmien tapauksessa, kun arkkitehtuuridokumentaatio jakautuu useaan isoon dokumenttiin:

- Ymmärrettävyys: Dokumenttia luettaessa lukija ei välttämättä saa siitä kirjoittajan tarkoittamaa kuvaa. Tilannetta hankaloittaa se, että eri sidosryhmillä on erilainen tausta ja erilaisia käsityksiä asioista, jolloin dokumentointiin voi tulla moniselitteisyyttä ja väärinkäsityksiä.

- Oleellisen tiedon löytäminen: Tiedon löytäminen useammasta isosta dokumentista on hankalaa ja tieto voi lisäksi olla jakautunut useampaan dokumenttiin. Vaikka dokumentit olisivatkin rakenteeltaan selkeitä eivät ne ole niin tarkkaan jäsenneltyjä, että tietyn tiedon löytäminen olisi helppoa.

- Jäljitettävyys eri entiteettien välillä: Erilaisten suhteiden esittäminen tekstillä tai taulukoilla ei ole erityisen toimivaa. Kuvien ja mallien ongelmana on näiden semantiikan eksplisiittisen selityksen puute, josta seuraa huonompi ymmärrettävyys. Erityisen hankalaa jäljitettävyys on eri dokumenttien kuten vaatimus- ja arkkitehtuuridokumenttien välillä.

- Muutosten vaikutusanalyysin suorittaminen: Muutosta tehtäessä arkkitehtuurin johonkin osaan tarvitsee selvittää, mikä on muutoksen vaikutus muuhun järjestelmään. Dokumentaatio ei usein tuo tällaisia suhteita kuitenkaan esiin, joten luotettavan analyysin tekeminen on vaikeaa.

- Arkkitehtuurin kypsyyden arviointi: Dokumentaatiosta ei ole saatavilla yleiskatsausta arkkitehtuurin käsitteellisen yhtenäisyyden, virheettömyyden ja valmiusasteen tilasta. Nämä ovat melko monimutkaisia ominaisuuksia, joten niiden arvioiminen esimerkiksi skenaariopohjaisilla menetelmillä on vaikeaa.

- Tiedon uskottavuus: Suuren järjestelmän suunnittelun ja kehityksen aikana tapahtuvien lukuisten muutosten päivittäminen dokumentteihin voi vaatia paljon resursseja, jolloin sitä ei aina tehdä. Tästä seuraa se, että dokumentaatio ei enää pidä yhtä järjestelmän kanssa, jolloin eri sidosryhmät eivät enää luota siihen.

Näiden ongelmien vuoksi tarvitaan erilaisia vartavasten suunniteltuja työkaluja ja järjestelmiä arkkitehtuuritietämyksen keräämiseen ja tarkasteluun. Näiden tulee skaalautua suurien järjestelmien kehitystyöhön sopivaksi ja tarjota käyttäjilleen helposti löydettävää ja uskottavaa tietoa, joiden suhteiden ja riippuvuuksien selvittäminen on helppoa. Jotta tämä onnistuisi tulee arkkitehtuuritiedon olla formaalia, jolloin sitä voidaan käsitellä automaattisesti. Arkkitehtuurissa olevia ongelmia voidaan osoittaa käyttäjille ja halutut asiat ja yhteydet näyttäviä näkymiä voidaan rakentaa käyttäjän tarpeiden mukaan.

Arkkitehdit eivät ole halukkaita panostamaan paljoa aikaa arkkitehtuuripäätösten ja muun tietämyksen tallentamiseen, sillä siitä ei saa välitöntä hyötyä. Tietämyksen keräämisen hyöty tulee esille vasta paljon myöhemmin, ja hyötyjinä voivat lisäksi olla muut henkilöt. [13] Tämän vuoksi arkkitehtuuritietämyksen keräämiseen käytettävien työkalujen tulee olla helppokäyttöisiä ja niiden tulee sopia arkkitehdin työnkulkuun.

3. POLARION

3.1 Yleiskuvaus

Tässä työssä kehitetty arkkitehtuuritietämyskanta rakennettiin Polarion Softwaren kehittämän Polarion ALM -ohjelmiston elinkaaren hallintajärjestelmän avulla. Polarion on selaimella käytettävä järjestelmä, joka kattaa ohjelmiston kehitysprosessin eri osa-alueet kuten vaatimusten, lähdekoodin ja testitapausten hallinnan [16]. Aluksi tässä työssä käytetty Polarion-asennus oli ohjelmistoalan palveluyritys Cybercomin palvelimilla ylläpidetty, mutta työn aikana TTY:llä otettiin käyttöön oma Polarion-palvelin, jota työssä myös käytettiin.

Polarion pohjautuu Subversion-versionhallintaan, jonne Polarioniin luotu sisältö tallennetaan. Varsinaisen Polarion sisällön lisäksi versionhallintaan voidaan tallentaa lähdekoodia, jota Polarionilla voi sitten tarkastella ja jota voi yhdistää Polarion-sisältöön. Työ jakautuu Polarionissa projekteihin, joiden alle kuhunkin projektiin liittyvä sisältö tallennetaan. Projektit voivat puolestaan muodostaa projektiryhmiä. Näin ollen Polarionin käytössä on kolme eri tasoa: projektitaso, ryhmätaso ja globaali koko versionhallinnan kattava taso. Käyttäjä voi valita haluamansa tason, jolloin hänelle näytetään vain tällä tasolla tai sen alapuolella olevaa sisältöä.

3.2 Työalkiot

3.2.1 Työalkioiden kentät, linkitys ja konfigurointi

Polarionin toiminnassa keskeisessä osassa ovat työalkiot (work item), jotka edustavat ohjelmistokehityksen eri asioita kuten vaatimuksia, tehtäviä, muutosvaatimuksia ja testitapauksia. Työalkiot ovat erityyppisiä, ja nämä tyypit ovat konfiguroitavissa. Erityyppisillä työalkioilla on omat kentät työalkioon liittyvälle tiedolle. Yleisiä kenttiä ovat muun muassa työalkion otsikko eli nimi, tekijä, kuvaus sekä tila, joka kuvaa työalkion asemaa työnkulussa. Arvoina voivat olla esimerkiksi avoin, työnalla ja valmis. Lisäksi näihin tilasiirtymiin voidaan konfiguroida rajoittavia sääntöjä ja toimintoja, jotka suoritetaan siirtymän tapahtuessa. Jokaisella työalkiolla on myös id, jonka avulla työalkio yksilöidään ja jonka järjestelmä generoi työalkion luomisen yhteydessä. Vaatimuksen kenttä voi olla esimerkiksi vaatimuksen tärkeys. Bugilla voisi puolestaan olla kenttä vakavuutta varten ja tehtävällä arvioitu aika tehtävän

suorittamiseen. Myös nämä kentät ovat käyttäjän konfiguroitavissa. Työalkioihin voidaan liittää kommentteja, joiden avulla niistä voidaan keskustella. Lisäksi työalkioihin voidaan liittää hyperlinkkejä sekä liitetiedostoja kuten työalkioon jotenkin liittyviä kuvia tai dokumentteja.

Työalkioita voi linkittää toisiinsa, jolloin saadaan luotua jäljitettävyyttä eri asioiden välille. Vaatimus voi esimerkiksi tarkentaa toista vaatimustyöalkiota tai testitapaustyöalkio voi testata jonkin vaatimustyöalkion toteutumista. Myös nämä linkityssuhteet ovat konfiguroitavissa. Linkin nimen lisäksi voidaan määrittää sääntöjä, jotka rajoittavat sitä, minkä tyyppisten työalkioiden välillä tietyn niminen linkki voi olla.

Teknisesti työalkiot ovat XML-tiedostoja, joita säilytetään versionhallinnassa. Myöskin työalkioiden ja niiden suhteiden konfigurointitiedot ovat versionhallinnassa sijaitsevia XML-tiedostoja. Konfiguraatiotiedostoja voi muokata suoraan käsin tai Polarionin tarjoamalla graafisella asetusten muokkaus -käyttöliittymällä.

Muun Polarioniin talletetun sisällön tapaan myös työalkiot jakautuvat projekteihin. Työalkioiden linkittäminen onnistuu projektista toiseen. Työalkioiden ja niiden suhteiden konfiguraatiot voivat toimia monella eri tasolla. Ne voivat olla täysin globaaleja eli kaikissa projekteissa on samanlaiset työalkiot. Jokaisessa projektissa voi myöskin olla täysin erilaiset työalkiot tai osittain samanlaiset. Esimerkiksi tietyn projektin tarpeisiin voidaan helposti lisätä tietyn tyyppiseen työalkioon jokin uusi kenttä. Käytännössä tämä tehdään versionhallinnan hakemistohierarkian eri tasoilla olevilla tietyllä tavalla nimetyillä konfiguraatiotiedostoilla.

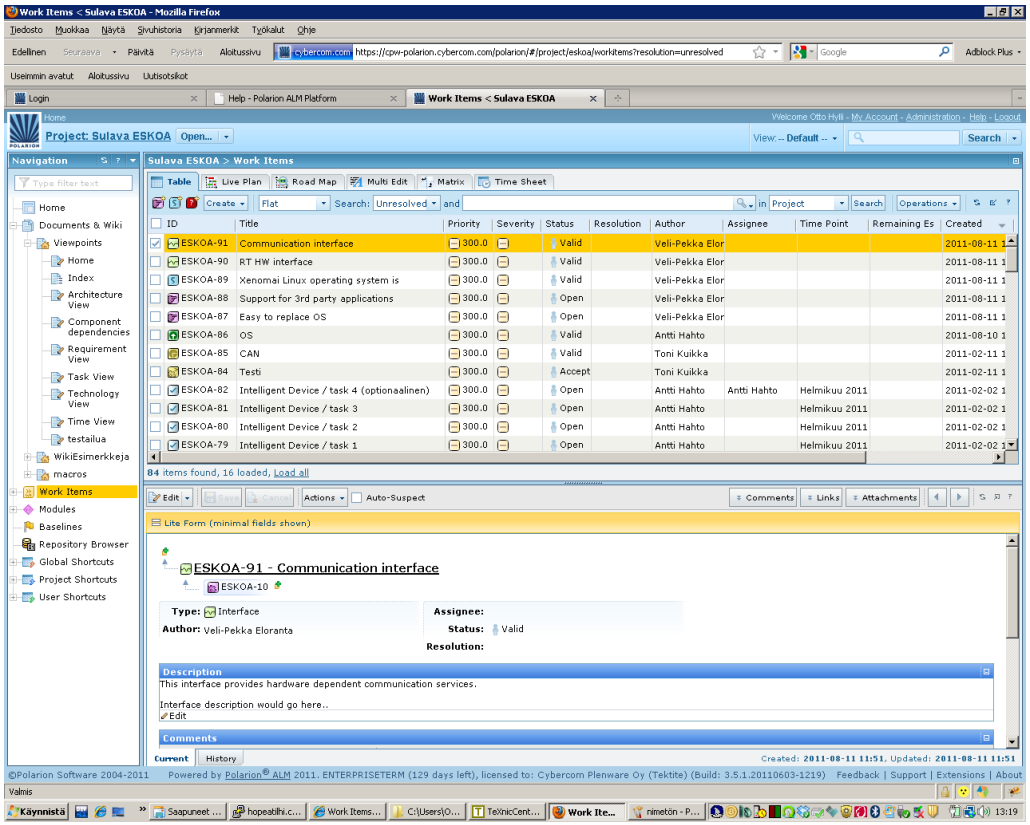
3.2.2 Työalkioiden käsittely

Työalkioita pääsee käsittelemään valitsemalla työalkiot-kohdan Polarionin päävalikosta. Työalkiot osiosta onnistuu työalkioiden luominen. Käyttöliittymästä valitaan työalkion luominen, jonka jälkeen työalkion vaatimat tiedot syötetään niille varattuihin kenttiin.

Polarion tarjoaa työalkioiden selaamiseen useita erilaisia näkymiä, jotka sopivat eri tilanteisiin kuten linkkisuhteiden jäljittämiseen. Näitä näkymiä ovat esimerkiksi taulukko- ja matriisinäkymä. Taulukkonäkymässä työalkiot näytetään tavallisessa taulukossa ja ne voi lajitella eri sarakkeiden otsikoiden mukaan kuten tilan. Lisäksi taulukkonäkymässä voidaan käyttää puunäkymää, jossa työalkioon linkitetyt työalkiot ovat myös tarkasteltavissa. Matriisinäkymä näyttää työalkiot ruudukossa, jonka avulla on helppoa katsoa millaisia linkkisuhteita eri työalkioiden välillä on. Lisäksi Polarion tarjoaa näkymiä, jotka auttavat projektin etenemisen seuraamisessa ja projektin hallinnassa. Työalkion yksityiskohtaista tarkastelua varten työalkio valitaan näkymästä, jolloin sen tiedot avautuvat tiedoille varatulle sivun alueelle. Tässä näkymässä näkyvät kaikki työalkion kentät sekä lista linkitetyistä työalkioista, kom-

menteista, hyperlinkeistä ja liitetiedostoista. Tästä näkymästä myös pääsee muokkaamaan kaikkia edellä mainittuja kohtia. Kuvassa 3.1 nähdään taulukkonäkymässä listattuna työalkioita, joista yksi on valittu tarkemmin tarkasteltavaksi.

Näkymissä näytettäviä työalkioita voi rajata käyttämällä Polarionin hakuominaisuutta. Polarionin työalkiohaku pohjautuu Apache Luceneen, joka on java-pohjainen tekstihakumoottori [6]. Lucene-kyselykielellä työalkioita voi hakea eri kenttien kuten vaikka tyyppi ja tila perusteella. Myös linkitettyjen tai linkittävien työalkioiden haku onnistuu. Lisäksi hakutermejä voi yhdistellä loogisilla operaattoreilla kuten ja sekä tai.



Kuva 3.1: Työalkioiden tarkastelu taulukkonäkymässä.

3.3 Wiki

Tämän työn kannalta toinen keskeinen Polarionin ominaisuus on sen tarjoama wiki. Wiki-sivuja kirjoitetaan wikeille ominaiseen tyyliin yksinkertaisella wiki-syntaksilla, jolla onnistuu muun muassa eri tasoisten otsikoiden, luetteloiden ja taulukoiden tekeminen sekä linkkien luonti muihin sivuihin ja kuvien lisääminen. Wiki-syntaksi tarjoaa lisäksi erilaisia makroja työalkioiden tietojen näyttämiseen ja niihin linkittämiseen. Makron avulla voidaan näyttää tietyn työalkion, jonka id tiedetään, tiedot.

Käyttäjä voi määrittää mitkä tiedot työalkiosta näytetään. Toinen makro puolestaan mahdollistaa tietyn Lucene-hakukyselyn löytämien työalkioiden listaamisen.

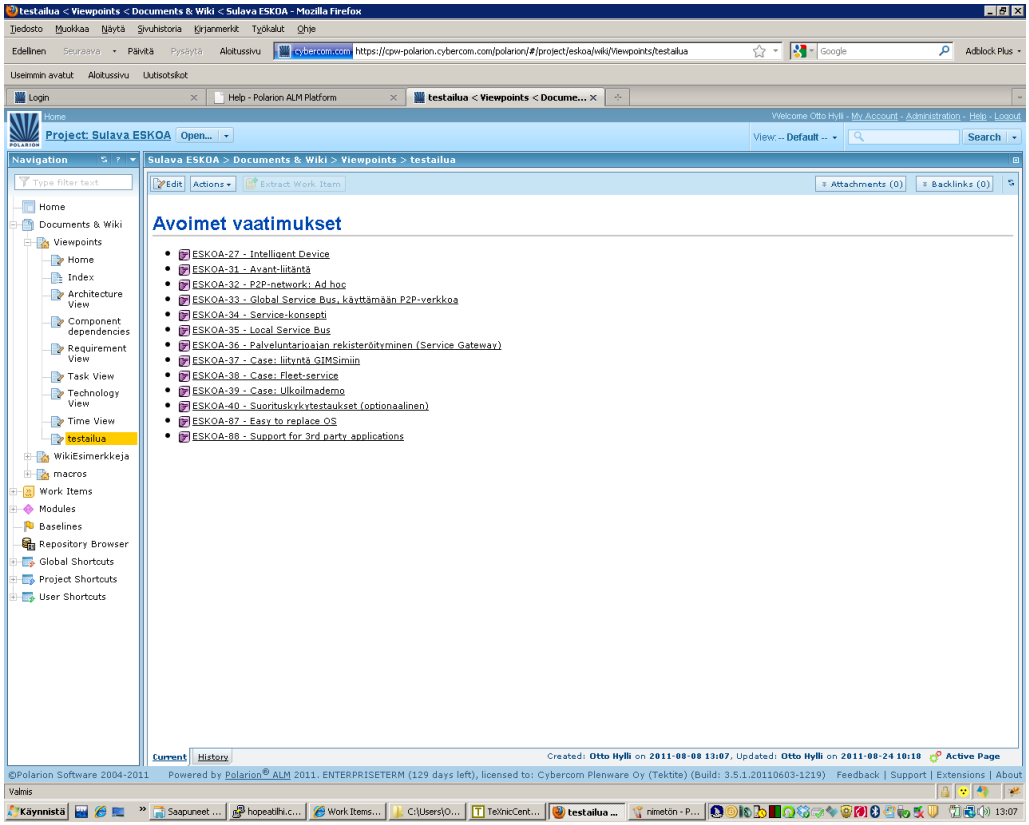
Wiki-syntaksin lisäksi sivujen kirjoittamisessa voidaan käyttää HTML:ää (XHTML 1.1), joka mahdollistaa monipuolisemmat wiki-sivut, joilla käytetään esimerkiksi tyylimäärityksiä sivun ulkonäön muokkaamiseen ja HTML:n lomake-elementtejä toiminnallisuuden lisäämiseen. Toiminnallisuutta voidaan myös lisätä käyttämällä JavaScriptiä, jolla voidaan esimerkiksi tarkistaa käyttäjän syötteitä ennen niiden lähettämistä.

Wiki-sivut hyödyntävät Java-pohjaista Velocity-sivumoottoria, jonka tarjoama Velocity-sivunkuvauskieli mahdollistaa muun muassa työalkioiden makroja monipuolisemman hakemisen ja esittämisen sekä työalkioiden muokkaamisen, luomisen ja poistamisen käyttäjän syötteiden pohjalta. Velocityn toiminta-ajatuksena on Javalla tehtävässä web-ohjelmoinnissa erottaa sivuston toimintalogiikka ja sivun ulkoasu. Se mahdollistaa malli-näkymä-ohjain (MVC) arkkitehtuurin mukaisten web-ohjelmien toteutuksen. Velocity-kieltä upotetaan HTML-koodin lomaan ja sen avulla voidaan käyttää Java-koodissa Velocityn käyttöön asetettuja olioita. [7] Polarion tarjoaa wiki-sivun kirjoittajalle kahdeksan oliota, joiden kautta onnistuu muun muassa tiedon saaminen kyseisestä sivusta, projektien käsittely ja työalkioiden hakeminen. Nämä oliot kuuluvat Polarion Java API:iin, jonka avulla ulkopuoliset kehittäjät voivat laajentaa Polarionin toimintaa wikin lisäksi myös muin keinoin.

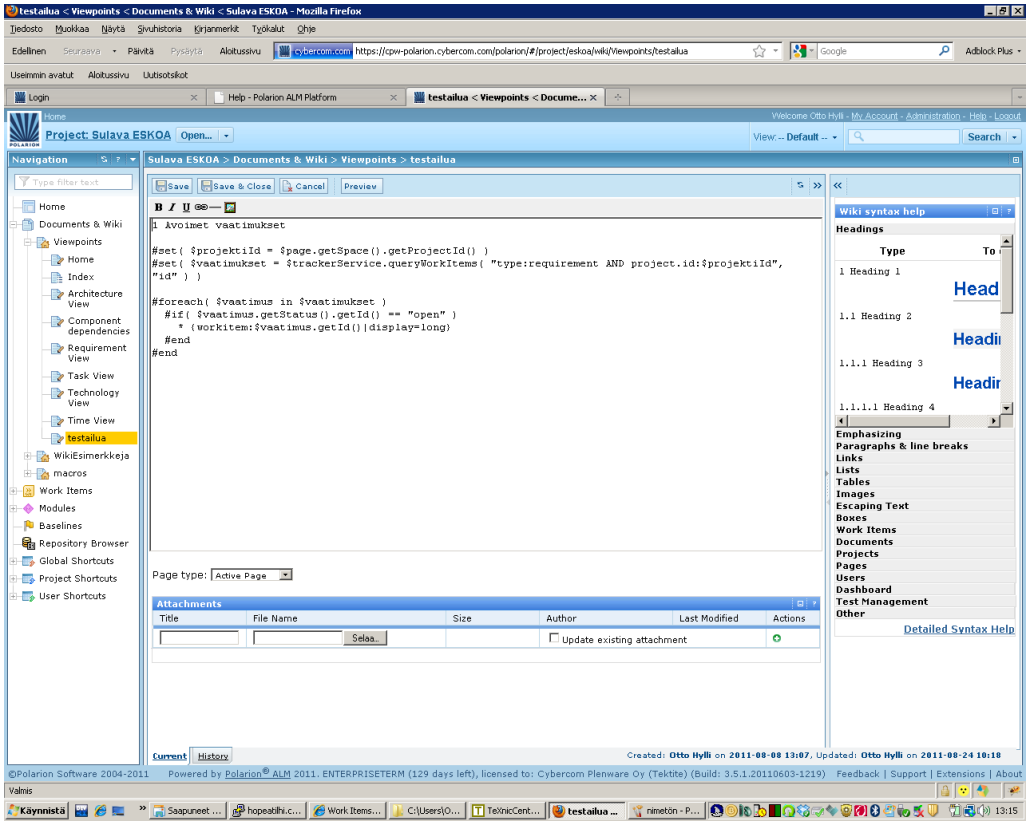
Kuvassa 3.2 on esimerkki wiki-sivusta, joka näyttää kaikki kyseisen projektin vaaatimukset, joiden tila on avoin, listana, jossa on vaatimuksen id ja nimi. Jokainen tällainen lista-alkio on myös linkki, jota klikkaamalla työalkion tiedot avautuvat omaan ikkunaansa tarkasteltavaksi ja muokattavaksi.

Kuvassa 3.3 on tämä sivu muokkaustilassa, jossa näkyy sivun toteuttava koodi. Sama koodi löytyy myös liitteestä A. Aluksi wiki-syntaksilla sivulle määritetään otsikko ja Velocity-koodilla haetaan sivun projektin yksilöivä id. Tämän jälkeen tämän projektin vaatimukset haetaan käyttämällä wiki-kontekstiin asetetun olion metodia ja Lucene-hakukyselyä. Tämän jälkeen haetut vaatimukset käydään läpi silmukassa. Jos vaatimuksen tila (status) on avoin (open), sen otsikko ja id tulostetaan käyttämällä wiki-syntaksin makroa. [1]

[1] Todellisuudessa tällaisen sivun olisi voinut vielä tehdä ilman Velocityä käyttämällä hakutulokset listaavaa makroa ja lisäämälä hakukyselyyn yhdeksi termiksi työalkion tilan.



Kuva 3.2: Avoimet vaatimukset listaava wiki-sivu.



Kuva 3.3: Avoimet vaatimukset näyttävän wiki-sivun muokkaus.

4. ARKKITEHTUURITIEDON ESITTÄMINEN

4.1 Metamallin tausta

Tässä työssä kehitetyn arkkitehtuuritietämyskannan rakenteen pohjana on Sulava-projektissa kehitetty arkkitehtuuritietämyksen metamalli. Malli kuvaa, mitä käsitteitä tietämyskantaan tallennetaan, mitä tietoa niihin liittyy ja millaisia suhteita niiden välillä on. Metamalli kehittyi kuuden workshopin aikana. Osallistujat olivat Tampereen teknillisen yliopiston ohjelmistotekniikan laitokselta. Lisäksi mukana oli teollisuuden edustajia Metsolta sekä Cybercomin Polarion asiantuntija. Malli kehittyi lisäksi myös tämän työn aikana, kun siinä huomattiin ongelmia ja puutteita.

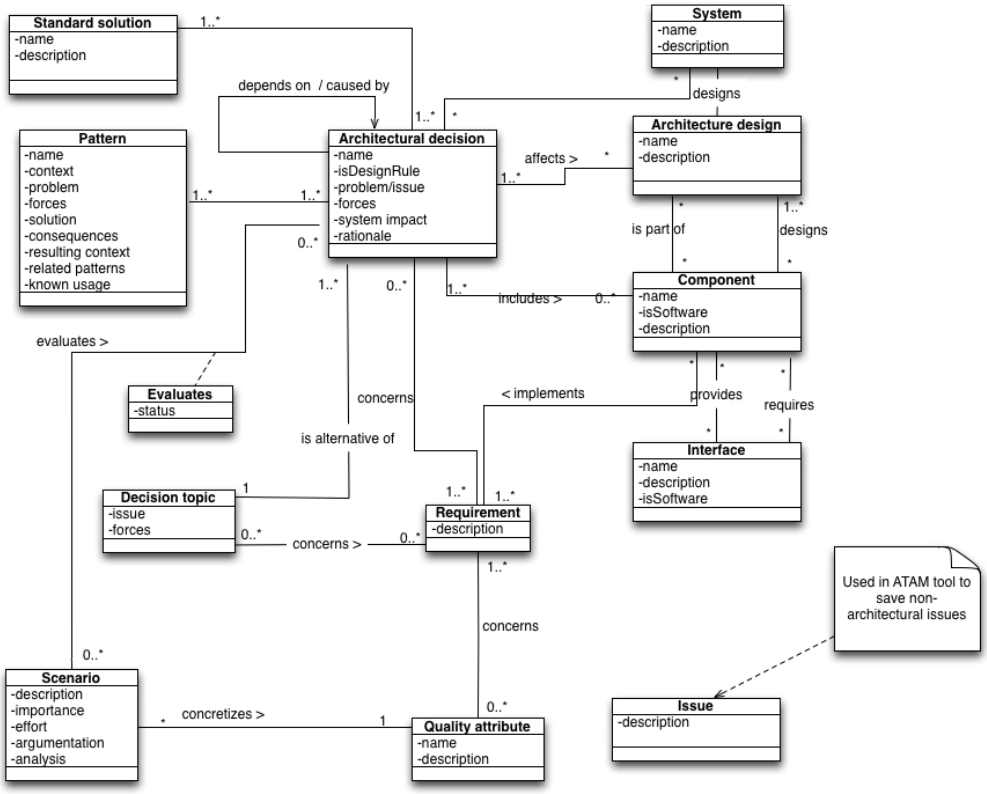
Mallin kehityksen taustalla oli useita lähtöajatuksia ja tavoitteita. Ensinnäkin mallin tuli olla käytännönläheinen ja sen tuli olla yleinen eli sen tulee kattaa useat arkkitehtuurialueet puhtaista ohjelmistoista laitekomponentteja sisältäviin sulautettuihin järjestelmiin. Mallista haluttiin lisäksi yksinkertainen. Mallia olisi voitu tarkentaa yksityiskohtaisemmaksi, mutta tätä ei haluttu tehdä. Malliin sisällytettävien käsitteiden ja suhteiden valintaa puolestaan ohjasi ajatus perinteisen arkkitehtuurin yhdistämisestä arkkitehtuuriarvioinneissa saaatavaan tietoon sekä yleistä tietämystä edustaviin suunnittelumalleihin. Lisäksi kohdassa 2.1.1 mainittu päätösten tärkeä asema arkkitehtuuritietämyksessä haluttiin tuoda esiin asettamalla päätökset keskeiseksi osaksi mallia. Mallin voidaan katsoa siis edustavan kohdassa 2.2 mainittua päätöskeskeistä arkkitehtuuritietämyksen hallinnan näkökulmaa.

4.2 Metamallin rakenne

Syntynyt metamalli on kuvassa 4.1. Seuraavassa käydään metamallin käsitteet ja niihin liittyvät tiedot läpi. Lisäksi selostetaan käsitteiden väliset suhteet.

Järjestelmä (system) edustaa järjestelmää, jonka arkkitehtuuria koskevaa tietoa tietämyskannassa säilytetään. Se sisältää yleistä tietoa järjestelmästä. Järjestelmän yleiskuvauksen lisäksi se sisältää tiedon järjestelmää kehittävästä organisaatiosta sekä kuvaukset järjestelmään liittyvistä bisnes-tavoitteista, järjestelmän sidosryhmistä ja järjestelmän kehitysympäristöstä.

Arkkitehtuuripäätös (architectural decision) edustaa arkkitehtuurin pohjalla olevia ja sitä ohjaavia päätöksiä (kts. kohta 2.1.1). Päätös voi olla suunnittelusääntö. Päätökseen voivat vaikuttaa erilaiset voimat, jotka voivat olla lähtöisin hyvin eri-



Kuva 4.1: Sulava-projektin arkkitehtuuritietämyksen metamalli.

laisista lähteistä kuten muut päätökset, vaatimukset, bisnes-tavoitteet tai organisaatiokulttuuri. Päätöksestä tulee kuvata taustalla oleva ongelma, päätöksen vaikutus järjestelmään sekä perustelut. Päätökset vaikuttavat toisiinsa: päätös voi riippua toisesta päätöksestä tai se voi olla seurausta toisesta päätöksestä. Päätös on yksi mallin keskeisimmistä käsitteistä sillä se vaikuttaa tai siihen vaikuttaa useat muut osat kuten vaatimukset, rajapinnat, komponentit ja arkkitehtuurikuvaukset. Kun arkkitehtuurissa käytetään jotakin suunnittelumallia tai standardiratkaisua, ilmaistaan tämä päätöksellä, jolla on soveltaa-suhde kyseiseen suunnittelumalliin tai standardiratkaisuun.

Päätösaihe (decision topic) edustaa jotakin asiaa, jonka suhteen on tehtävä päätös ja johon liittyy useita mahdollisia vaihtoehtoja. Tällainen asia voi olla esimerkiksi ohjelmointikieli tai korkean tason arkkitehtuurityyli. Päätösaiheeseen liittyy ongelman kuvaus, johon haetaan ratkaisua sekä siihen vaikuttavat voimat. Päätösaihe voi liittyä joihinkin vaatimuksiin. Päätösaiheesta on suhde niihin päätöksiin, jotka ovat sen mahdollisia vaihtoehtoja. Valittu vaihtoehto osoitetaan kyseisen päätöksen tilalla.

Komponentti (component) edustaa osia, joista järjestelmä koostuu. Tässä komponentti voi olla sekä laitteisto- että ohjelmistokomponentti tai se voi olla molempia. Komponentti voi toteuttaa järjestelmälle asetettuja vaatimuksia. Komponent-

tien yhteistoiminta tapahtuu rajapintojen kautta. Komponentti tarjoaa joukon rajapintoja muiden komponenttien käytettäväksi ja vastaavasti se tarvitsee tiettyjä rajapintoja muilta komponenteilta. Komponentilla on tämän lisäksi suoria suhteita muihin komponentteihin. Komponentti voi riippua muista komponenteista ja se voi olla osa jotain muuta komponenttia. Komponentti voi olla myös osa jotain arkkitehtuurikuvausta.

Rajapinnat (interface) edustavat komponenttien välisiä liityntöjä. Nämä voivat olla sekä laitteisto- että ohjelmistorajapintoja tai molempia. Rajapinta on osa jotakin arkkitehtuurikuvausta.

Suunnittelumalli (pattern) edustaaa yleistä ja hyväksi havaittua ratkaisua tietynlaiseen ongelmaan. Mallin osia ovat yleiskuvauksen lisäksi kuvaus ongelmasta, kontekstista, joka kuvaa suunnittelutilanteen, jossa ongelma esiintyy, ongelmatilanteessa vaikuttavat voimat kuten laatuominaisuudet, joita mallin soveltaminen muuttaa, ratkaisun kuvaus, mallin seurausten kuvaus, kuvaus malliin liittyvistä malleista sekä mallista seuraava konteksti ja mahdolliset tunnetut mallin käyttötilanteet. Malli voi täsmentää jotakin korkeamman tason mallia. Suunnittelumalli voi myös olla samankaltainen kuin jokin muu suunnittelumalli tai standardiratkaisu.

Standardiratkaisu on suunnittelumallin kaltainen yleisesti tunnettu ratkaisu. Se eroaa mallista siinä, että sen kuvausta ei ole mallin tapaan jaettu useaan osaan kuten konteksti, ongelma ja ratkaisu eli sen käyttötilanteita ja seurauksia ei välttämättä ole analysoitu yhtä tarkkaan. Standardiratkaisulla on samanlaiset suhteet muihin käsitteisiin kuin mallilla. Se voi täsmentää toista ratkaisua sekä olla samankaltainen kuin jokin suunnittelumalli ja standardiratkaisu.

Arkkitehtuurikuvaus (architecture design) edustaa tietyn komponentin, koko järjestelmän tai sen osan arkkitehtuurin suunnitelmaa. Kuvauksen tekstikuvauksen lisäksi siihen liittyvät kyseistä suunnitelmaa kuvaavat kuvat kuten UML-kaaviot. Kun kuvaus esittää jonkin komponentin sisäisen rakenteen, on sillä suunnittelee-suhde kyseiseen komponenttiin. Kun kuvaus puolestaan esittää koko järjestelmän arkkitehtuurin eli se on korkeimman tason arkkitehtuurikuvaus, on sillä suunnittelee-suhde järjestelmään. Kuvaukset voivat muodostaa hierarkian, kun alemman tason kuvauksilla on yhteys korkeamman tason kuvaukseen. Kuvaus on yhteydessä vaatimuksiin siten, että se toteuttaa jonkin vaatimuksen.

Vaatimus (requirement) edustaa suunniteltavalle järjestelmälle asetettuja vaatimuksia. Vaatimukset voivat olla hierarkisia. Vaatimus voi liittyä johonkin laatuominaisuuteen.

Skenaario (scenario) edustaa ATAM-arvioinnissa syntyviä laatuominaisuuksia konkretisoivia skenaarioita. ATAM:ssa skenaarioiden on tarkoitus arvioida miten hyvin tehdyt päätökset edesauttavat skenaarion toteutumista. Tätä kuvaa skenaarion ja päätöksen välillä olevan arvioi-suhteen tila-attribuutti. Mahdollisia tiloja ovat riski,

ei-riski, kompromissi ja ei vielä arvioitu. Skenaarioon liittyviä tietoja ovat ATAM:ssa annettava priorisointi, päätösten argumentointi, skenaarion analyysi sekä tieto, missä ATAM:in vaiheessa skenaario on syntynyt, vai onko se syntynyt ATAM:in ulkopuolella. ATAM:sta voi lukea lisää kohdasta 5.2.1.

Laatuominaisuus (quality attribute) kuvaa järjestelmältä vaadittuja laatutekijöitä. Tällaisia voivat olla esimerkiksi joustavuus ja suorituskyky. Laatuominaisuudet voivat liittyä niitä koskeviin vaatimuksiin. Laatuominaisuuteen voi liittyä sitä tarkentavia laatuominaisuuksia.

4.3 Polarion ja metamalli

Arkkitehtuuritietämyskannan toteuttamiseksi Polarioniin tehtiin metamallia vastaava konfiguraatio. Jokaista metamallin käsitettä varten luotiin oma työalkiotyyppi, joille luotiin kyseisen käsitteen attribuutteja vastaavat kentät. Vastaavasti käsitteiden välisiä suhteita varten konfiguroitiin vastaavat työalkioiden linkityssuhteet. Tässä jouduttiin hieman poikkeamaan metamallista skenaarion ja päätöksen välisissä suhteissa. Polarion ei mahdollista minkäänlaista lisätiedon lisäämistä linkkisuhteeseen, joten yksi arvioi-suhde ei riittänyt. Sen sijaan jokaista tilaa varten piti tehdä oma linkkityyppinsä. Muuten metamallin mukaisen Polarion konfiguraation tekeminen oli melko suoraviivaista.

5. TIETÄMYKSEN TUOTTAMINEN

5.1 Tietämyksen tuottamisen periaatteet

Kun Polarion on konfiguroitu arkkitehtuuritietämyskannaksi, onnistuu arkkitehtuuritietämyksen lisääminen yksinkertaisimmillaan luomalla työalkioita kohdassa 3.2.2 kuvatulla tavalla. Tämä ei kuitenkaan ole paras mahdollinen tapa, sillä se ei kunnolla tue ohjelmistokehitysprosessia vaan lähinnä hidastaa sitä, jolloin sen käyttö jäisi todennäköisesti vähäiseksi (kts. kohta 2.5). Tämän vuoksi Polarioniin on tarkoitus kehittää erilaisia tiedon tuottamisessa auttavia korkeamman tason työkaluja. Työkalujen on tarkoitus tukea erilaisten ohjelmistotuotannon yleisten menetelmien käyttöä. Työkalu tukee menetelmän työnkulkua eli eri vaiheita tarjoamalla joka vaiheessa tarvittavat tiedot sopivassa muodossa näyttävän käyttöliittymän, johon voi nopeasti syöttää vaiheessa syntyvän tiedon. Tavoitteena on, että syntyvä tieto menee samantien Polarioniin eikä vasta jälkikäteen.

5.2 Esimerkki: ATAM

Esimerkkinä edellä kuvatun kaltaisesta työkalusta toteutettiin työkalu tukemaan ATAM-arkkitehtuurinarviointimenetelmää. Työkalun tarkoitus on auttaa suunniteltavan arkkitehtuurin arvioinnissa sekä tuottaa uutta arkkitehtuuritietämystä arvioinnin pohjalta. Seuraavissa alikohdissa kuvataan ensiksi ATAM, jonka jälkeen kuvataan kehitetty työkalu.

5.2.1 Mikä on ATAM

ATAM (Architecture Tradeoff Analysis Method) on Carnegie Mellon yliopiston Software Engineering Institute -yksikössä kehitetty skenaariopohjainen menetelmä ohjelmistoarkkitehtuurien arviointiin. ATAM:n tarkoitus on arvioida tehtyjen arkkitehtuuripäätösten vaikutusta suunniteltavalta järjestelmältä vaadittuihin laatuominaisuuksiin. ATAM:n tarkoitus on tunnistaa ne päätökset, jotka ovat laatuominaisuuksien kannalta riskejä, ei-riskejä, herkkyyspisteitä tai kompromisseja. Riskipäätös on sellainen, jota ei ole vielä tehty tai jonka seuraukset voivat olla ongelmallisia. Ei-riski on hyvä päätös tietyn laatuominaisuuden kannalta eli se tukee tätä ominaisuutta. Herkkyyspisteeksi määritellyn asian muutoksella on selvä vaikutus johonkin laatuominaisuuteen. Kompromissi on herkkyyspiste usean laatuominaisuuden

kannalta siten, että muutos vaikuttaa laatuominaisuuksiin erisuuntaan. Esimerkiksi muutos voi kasvattaa luotettavuutta, mutta laskea suorituskykyä. [12]

Arviointi tapahtuu skenaarioiden avulla. Skenaarioita laaditaan eri näkökulmista kuten käyttäjän, kehittäjän ja ylläpitäjän. Skenaario on konkreettinen arkkitehtuurin testitapaus, jonka osia ovat ärsyke: mitä tapahtuu, ympäristö: missä tilassa ympäristö ja järjestelmä ovat ja vaste: miten järjestelmä reagoi. Ärsyke voi esimerkiksi olla tietty järjestelmän muutos ja vaste voisi olla, että muutos saadaan tehtyä kahden viikon aikana. Ympäristö voi käyttöskenaariossa olla esimerkiksi käytön ruuhkahuuippu.

Riskien, herkkyyspisteiden ja kompromissien löytämisen lisäksi ATAM:in tavoitteena on selventää järjestelmän laatuominaisuuksien vaatimukset ja tehdyt arkkitehtuuripäätökset. Tämä on välttämätön osa ATAM:ia, sillä laatuominaisuudet ja päätökset täytyy olla mahdollisimman tarkkaan selvillä, jotta niiden välisiä yhteyksiä voitaisiin arvioida. [12]

ATAM koostuu alla olevista yhdeksästä askeleesta [12]. Kaikkien askeleiden aikana voidaan kerätä askeleeseen liittyvän tiedon lisäksi huomioita, jotka ovat arvioinnin aikana esiin tulleita asioita, jotka eivät sinänsä ole suoranaisesti arkkitehtuuriin liittyviä asioita kuten riskejä, mutta jotka kuitenkin kannattaa huomioida.

Esittely

1. ATAM-esittely. Menetelmä selitetään ATAM-tilaisuudessa paikallaolioille, joita voivat olla esimerkiksi arkkitehti, asiakkaan edustajat, käyttäjien edustajat, testaajat sekä johtajat.

2. Liiketoimintatavoitteiden esittely. Projektipäällikkö kertoo, mitkä liiketoiminnan tavoitteet motivoivat kehitystä eli ohjaavat arkkitehtuuria. Näitä voivat olla esimerkiksi järjestelmän saaminen nopeasti markkinoille tai järjestelmän tekeminen erittäin luotettavaksi.

3. Arkkitehtuurin esittely. Arkkitehti esittelee suunnitellun arkkitehtuurin kiinnittäen erityisesti huomiota siihen, miten arkkitehtuuri täyttää liiketoimintatavoitteet.

Tutkimus ja analyysi

4. Arkkitehtuuriratkaisujen tunnistaminen. Arkkitehti nimeää arkkitehtuurista löytyvät ratkaisut, jotka listataan, mutta joita ei vielä analysoida.

5. Laatupuun luominen. Järjestelmältä halutut yleiset laatuominaisuudet, kuten suorituskyky ja luotettavuus, listataan. Tämän jälkeen niitä tarkennetaan järjestelmän kannalta konkreettisimmiksi laatuvaatimuksiksi ja edelleen täysin konkreettisiksi laatuominaisuuksia testaaviksi skenaarioiksi, jotka priorisoidaan.

6. Arkkitehtuuriratkaisujen analysointi. Korkeiten priorisoidut skenaariot analysoidaan. Skenaarioon liittyvät ratkaisut listataan ja analysoidaan niiden vaikutus skenaarioon ja sen kautta skenaarion laatuominaisuuteen. Tuloksena löydetään herkkyyspisteitä, riskejä ja kompromisseja.

Testaus

7. Skenaarioiden luonti ja priorisointi. Laatupuuvaiheessa luotujen skenaarioiden pohjalta kaikkien sidosryhmien edustajat luovat yhdessä isomman joukon skenaarioita. Kaikki osallistujat äänestävät skenaarioiden priorisoinnista.

8. Arkkitehtuuriratkaisujen analyysi. Vaihe kuusi toistetaan tässä edellisessä vaiheessa korkeiten priorisoiduille skenaarioille. Tässä vaiheessa voidaan löytää uusia ratkaisuja, riskejä, herkkyyspisteitä ja kompromisseja.

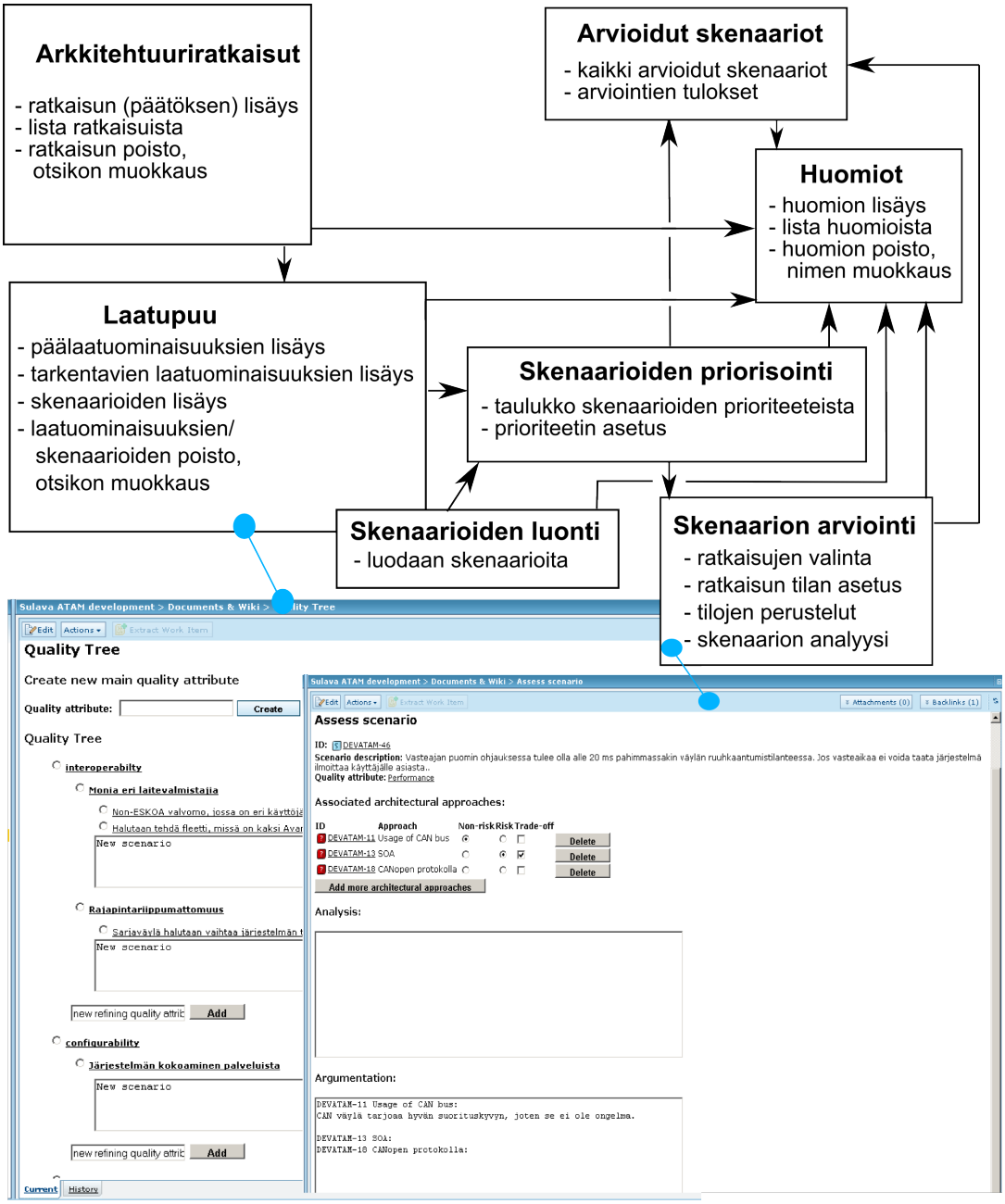
Raportointi

9. Tulosten esittäminen. ATAM-tiimi esittää ATAM:ssa kerättyihin tietoihin pohjautuvat johtopäätökset paikalla oleville sidosryhmien edustajille. Lisäksi tuloksista voidaan kirjoittaa raportti, jossa tulosten lisäksi esitetään keinoja ongelmien korjaamiseksi.

ATAM jakautuu tavallisesti kahteen vaiheeseen eli tilaisuuteen, joilla on eri osanottajat ja erilainen tavoite. Ensimmäisessä vaiheessa mukana on vain pienempi osa arvioijista, arvioitavan järjestelmän kehittäjistä ja muista sidosryhmistä. Tässä vaiheessa tarkoituksena on kerätä arkkitehtuuritietoa sekä analysoida sitä, ja siinä keskitytään ATAM:n askeliin 1-6. Toisessa vaiheessa osanottajia on enemmän ja tarkoituksena on kerätä eri sidosryhmien näkemyksiä sekä vahvistaa edellisen vaiheen tuloksia. [12]

5.2.2 ATAM-työkalu

ATAM-työkalu toteutettiin käyttämällä Polarionin wikiä, HTML:ää, JavaScriptiä sekä Velocityä (katso kohta 3.3). Varsinainen työkalu koostuu seitsemästä wiki-sivusta, jotka ovat: arkkitehtuuriratkaisut (Architectural approaches), laatupuu (Quality Tree), skenaarioiden luonti (brainstorm scenarios), skenaarioiden priorisointi (prioritize scenarios), skenaarion arviointi (Assess scenario), arvioidut skenaariot (Assessed scenarios) ja huomiot (Issues). Lisäksi työkaluun liittyy ATAM-tulosraportti -näkymä, jota käsitellään kohdassa 6.3. Sivujen välillä liikutaan niillä olevien linkkien avulla. Yleisesti tietyltä sivulta on linkit sitä edeltävän vaiheen sivuun ja sitä seuraavan vaiheen sivuun. Sivulla olevat työalkioiden otsikot tai id:t ovat linkkejä, jotka avaavat kyseisen työalkion tarkempaan tarkasteluun Polarionin omassa työalkionäkymässä.



Kuva 5.1: ATAM-työkalun näyttökartta ja ruutukaappaukset laatupuusta ja skenaarion arvioinnista.

Kuvassa 5.1 on ATAM-työkalun näyttökartta. Siitä nähdään miten työkalun sivujen välillä navigoidaan ja mitä kullakin sivulla on tarkoitus tehdä. Lisäksi kuvassa on ruutukaappaukset laatupuusta ja skenaarion arvioinnista.

Arkkitehtuuriratkaisut-sivua käytetään ATAM:in askeleessa 4. Sen avulla kirjataan esiintulleet arkkitehtuuriratkaisut. Ratkaisusta annetaan vain lyhyt kuvaus, jonka pohjalta luodaan päätös -työalkio. Ajatuksena on täydentää päätöksen tiedot ATAM:in jälkeen. Sivulla olevassa taulukossa on listattuna kaikki kyseisen projektin päätökset eli ennen ATAM:ia ja ATAM-työkalulla luodut. Päätösten nimiä voi

muokata ja niitä voi poistaa taulukossa olevien painikkeiden avulla.

Laatupuu-sivua käytetään askeleessa 5. Sivulla on kenttä korkeimman tason laatuominaisuuksien luomiseksi. Luodut laatuominaisuudet ilmestyvät listaan, jossa niiden alle voidaan lisätä tarkentavia laatuominaisuuksia, joiden alle voidaan edelleen lisätä skenaarioita. Työkalu luo vastaavat laatuominaisuuksia ja skenaarioita edustavat työalkiot ja niiden väliset linkkisuhteet. Skenaarion tai laatuominaisuuden voi valita listasta sen kohdalla olevalla valintapainikkeella, jonka jälkeen sen nimeä pääsee muokkaamaan tai sen saa poistaa. Poisto onnistuu kuitenkin vain, jos valitun kohteen alla laatupuussa ei ole mitään. Jos kohteen alla on jotain, näytetään käyttäjälle ilmoitus asiasta.

Skenaarioiden luonti -sivua käytetään askeleessa 7 eli ATAM:n toisen vaiheen alussa, jolloin luodaan uusia skenaarioita ilman laatupuuta. Kuvauksen lisäksi uudelle skenaariolle voidaan valita laatuominaisuus laatupuun korkeimman tason laatuominaisuuksien joukosta. Sivulla on lisäksi taulukko sivulla luoduista skenaarioista. Taulukon kautta skenaarioita voi poistaa ja niiden kuvauksia ja laatuominaisuutta pääsee muokkaamaan.

Skenaarioiden priorisointi -sivua käytetään askeleissa 5 ja 7 kussakin vaiheessa luotujen skenaarioiden priorisointiin. Sivulla käyttäjä voi valita minkä vaiheen skenaarioita priorisoidaan. Valitun vaiheen skenaariot listataan taulukossa, jossa skenaariolle voidaan asettaa prioriteetti ja jo asetettuja prioriteetteja voidaan tarkastella ja muokata. Skenaariot listataan prioriteetin mukaisessa laskevassa järjestyksessä. Työkalussa käytetään eri priorisointitapoja eri vaiheiden skenaarioille. Ensimmäisessä tavassa skenaarion prioriteetti jakautuu kahteen osaan: skenaarion tärkeys ja skenaarion onnistumisen vaatima työmäärä. Molempia arvioidaan kolmiportaisella asteikolla matala, keskimääräinen ja korkea. Tätä menettelyä käytetään ATAM:in ensimmäisessä vaiheessa askeleessa 5 luoduille skenaarioille, kun läsnä on vähemmän ihmisiä. Toinen priorisointimenetelmä pohjautuu skenaarioille annettaviin pisteisiin. ATAM:in toisessa vaiheessa, askeleessa 7, kaikki osallistujat äänestävät skenaarioiden priorisoinnista jakamalla niille pisteitä. Jokaisella osallistujalla on jaettavanaan tietty määrä pisteitä, yleensä kolmannes skenaarioiden määrästä. Työkalu tukee tätä äänestysprosessia eli pisteiden lisäämistä vähitellen eri skenaarioille. Lisäksi se ehdottaa sopivaa pistemäärää osallistujaa kohden.

Skenaarion arviointi -sivua käytetään askeleissa 6 ja 8. Tälle sivulle yksittäistä skenaariota arvioimaan päästään valitsemalla skenaario arvioitavaksi priorisointisivulla. Arvioinnin aluksi sivulla näytetään lista kaikista päätöksistä, joista valitaan arvioitavaan skenaarioon vaikuttavat päätökset. Kun tämä tieto tallennetaan, luodaan näiden päätösten ja skenaarion välille väliaikaiseksi tarkoitettu linkityssuhde (todo) is evaluated by. Tämän jälkeen valitut päätökset näytetään taulukossa, josta jokaisen kohdalta voidaan valita onko päätös riski vai ei-riski, jonka lisäksi päätös

voidaan merkitä kompromissiksi. Skenaarioon liittyvien päätösten valintaan pääsee myöhemmin tarvittaessa takaisin, jolloin sen kautta voi myös poistaa jo valittuja päätöksiä. Kun tiedot skenaarion ja päätösten suhteista tallennetaan, poistetaan väliaikainen linkkisuhde ja se korvataan valintaa vastaavalla suhteella. Sivulla on myös tekstikenttä päätöksiä koskevien valintojen perusteluille ja toinen kenttä skenaarion yleiselle analyysille.

Arvioidut skenaariot -sivu ei liity suoranaisesti mihinkään vaiheeseen. Se tarjoaa nopean tavan käydä läpi kaikki arvioidut skenaariot. Sivulla listataan jokainen arvioitu skenaario, niihin liittyvät päätökset tiloineen, päätösten perustelut sekä skenaarion analyysi. Sivun kautta pääsee myöskin muokkaamaan kunkin skenaarion analyysia.

Huomiot-sivu ei myöskään liity tiettyihin askeleihin. Tämän takia sinne pääsee jokaiselta muulta sivulta, vaikka käyttöajatus onkin, että se avataan omaan ikkunaansa tai välilehteen, jolloin huomioiden lisääminen koko ATAM-prosessin ajan on helppoa. Toiminnaltaan ja ulkonäöltään sivu vastaa arkkitehtuuriratkaisut -sivua eli se mahdollistaa huomioiden luomisen ja näyttää ne taulukossa.

6. ARKKITEHTUURINÄKYMIEN TOTEUTTAMINEN

6.1 Näkymien toteutuksen periaatteet

Samoin kuin arkkitehtuuritietämyksen syöttäminen onnistuu tietämyksen käyttäminen periaatteessa Polarionin tarjoamilla valmiilla työkaluilla. Näitä ovat kohdassa 3.2.2 esitellyt eri työalkionäkymät sekä Polarionin työalkiohaku. Sopivan näkymän valitseminen, oikean haun kirjoittaminen ja näkymän konfigurointi näyttämään haun löytämät työalkiot voi olla hankalaa. Vaikka Polarion tarjoaakin mahdollisuuden tallentaa hakuja ja näkymäkonfiguraatioita, eivät valmiit näkymät ole paras mahdollinen tapa näyttää tietoa, sillä niiden joustavuuden rajat tulevat nopeasti vastaan. Tämän vuoksi Polarionin wikiä käyttämällä luotiin erilaisia näkymiä arkkitehtuuritietämyksestä. Tällaisen näkymän tarkoituksena on hakea tietämyskannasta tietylle tilanteelle tai käyttäjälle oleellinen tieto ja esittää se tilanteeseen sopivalla tavalla. Näkymissä esiin tulevat työalkioiden nimet tai id:t tehdään linkeiksi, jolloin kiinnostavaa työalkiota pääsee tutkimaan tarkemmin sekä tarvittaessa muokkaamaan Polarionin omassa työalkio-näkymässä. Seuraavissa kohdissa esitellään kolme tällaista näkymää.

6.2 Komponentin teknologianäkymä

Komponentin teknologianäkymä on tarkoitettu ensisijaisesti tietyn komponentin kehittäjän työn tueksi. Sen tarkoitus on antaa kehittäjälle tietoa komponentista ja sen kontekstista. Tämä auttaa kehittäjää ymmärtämään paremmin, mitä hänen tulee tehdä. Näkymästä voi olla apua myös ylläpitäjälle, jonka tarvitsee tehdä muutoksia komponenttiin.

Näkymässä on kaksi osaa. Kun näkymä avataan, listaa se kaikki projektissa olevat komponentit. Listaa voi rajata käyttämällä näkymän hakutoimintoa, johon voi syöttää Lucene-kyselyn. Listasta valitaan komponentti, jonka tiedot näytetään. Alla on listattu näkymän toisen osan eli komponentin tiedot näyttävän osan sisältö.

- Kuvaus
- Liitetiedostot
- Vaatimukset

- Arkkitehtuurikonteksti
- Vaaditut ja tarjotut rajapinnat
- Komponenttiriippuvuudet
- Vaikuttavat päätökset
- Komponentin rakenne

Ensiksi näkymässä on komponentin yleiskuvaus, jonka avulla lukija saa nopeasti kuvan siitä, mistä komponentissa on kyse. Kuvauksen jälkeen on lista komponenttityöalkion liitetiedostoista, jotka voivat tarjota lisätietoa komponentista. Seuraavaksi luetellaan kuvauksineen ne vaatimukset, jotka kyseinen komponentti toteuttaa. Näin kehittäjä ymmärtää, miksi komponenttia tarvitaan, ja hän saa mahdollisesti lisätietoa siitä, mitä komponentilta vaaditaan esimerkiksi suorituskyvyn suhteen.

Seuraavaksi näytetään komponentin laajempi arkkitehtuurikonteksti. Tässä näytetään ne arkkitehtuurikuvaukset, joiden osa komponentti on. Kuvauksesta näytetään tekstin lisäksi työalkion liitetiedostoina olevat kuvat eli komponentin rakenteen kuvaavat kaaviot. Näin kehittäjälle selviää, mikä osa komponentilla on arkkitehtuurissa.

Seuraavat osiot tarkentavat tätä kontekstitietämystä. Ensiksi listataan ne rajapinnat kuvauksineen, jotka komponentin tulee toteuttaa. Jokaisen rajapinnan kohdalla luetellaan myös komponentit, jotka tulevat rajapintaa käyttämään. Sitten listataan puolestaan ne rajapinnat, joita komponentin tulee käyttää. Näistä kerrotaan myös, mitkä komponentit niitä tarjoavat. Tämä tieto auttaa kehittäjää integroimaan komponentin osaksi järjestelmää.

Seuraavaksi listataan komponentin yhteyksiä muihin komponentteihin. Ensiksi listataan kuvauksineen ne komponentit, joista kyseinen komponentti riippuu ja sitten komponentit, jotka ovat riippuvaisia kyseisestä komponentista. Tämä tieto voi olla erityisen hyödyllistä ylläpitäjälle, jonka tarvitsee tehdä komponenttiin muutoksia, joilla voi olla vaikutusta muihin komponentteihin. Riippuvuuksien jälkeen listataan kyseisen komponentin lapsikomponentit sekä komponentit, joiden lapsi kyseinen komponentti on.

Seuraavana listataan ne arkkitehtuuripäätökset, joilla on vaikutus komponenttiin. Päätöksen nimen lisäksi annetaan päätöksen vaikutus järjestelmään, jolla voi olla suoria vaikutuksia komponentin suunnitteluun. Päätöksiin liittyen listataan niiden soveltamat suunnittelumallit ja standardiratkaisut, jotka osaltaan voivat auttaa komponentin merkityksen ja sille asetettujen vaatimusten ymmärtämisessä. Lisäksi, jos päätöstä on arvioitu ATAM-skenaarioilla, listataan myös nämä skenaariot sekä arvioinnin tulos eli oliko päätös riski, ei-riski tai kompromissi. Tämä voi antaa tietoa komponentin vaikutuksesta laatuominaisuuksiin eli mitä komponentilta halutaan kuten suorituskykyä. Viimeisenä näkymässä näytetään komponentin sisäinen arkkitehtuuri eli komponentin suunnitteleva arkkitehtuurikuvaus.

6.3 ATAM-raportti

ATAM-raportti -näkymä liittyy kohdassa 5.2.2 esiteltyyn ATAM-työkaluun. Se auttaa ATAM:in yhdeksänteen askeleeseen eli tulosten esittämiseen kuuluvan tulosraportin luomisessa. Tämä näkymä eroaa käyttötavaltaan komponentin teknologianäkymästä, jota käytetään sellaisenaan. ATAM-raportti -näkymä on puolestaan pohja täydelle ATAM:in tulokset esittävälle dokumentille. Ajatuksena on, että wiki-sivulle kirjoitetaan wiki-editorilla raportin tekstiä kuten johtopäätökset. Tämän tekstin lomaan sijoitetuilla Velocity-makroilla, joiden toteutus on varsinaisen tekstiosuuden jälkeen, haetaan arvioinnissa syntyneitä tietoja kuten ratkaisuja ja skenaarioita. Osa raportin sisällöstä löytyy jo lähes sellaisenaan ATAM-työkalusta ja osa tarjoaa uudenlaisia näkymiä arvioinnin tuloksiin. Alla on listattuna raportin sisältö.

- Kansilehti
- Yhteenveto
- Sisällysluettelo
- Sanasto
- Johdanto
  - Tarkoitus ja kattavuus
  - Arviointiryhmä
  - ATAM-arviointiprosessin kuvaus
  - Huomioita prosessista
- Kohdejärjestelmä
  - Järjestelmän yleiskuvaus
  - Tärkeät arkkitehtuuriratkaisut
- Arvioidut skenaariot
  - Kohta jokaiselle laatuominaisuudelle
    - Kohta jokaiselle tämän laatuominaisuuden arvioidulle skenaariolle.
  - Skenaariot, joilla ei ole laatuominaisuutta
    - Kohta jokaiselle skenaariolle, joka ei ole linkitetty laatuominaisuuteen
- Riskiteemat ja analyysi
  - Riskit
  - Riskiteemat
    - Kohta jokaiselle riskiteemalle.
  - Priorisointiyhteenveto
  - Huomiot
- Johtopäätökset

- Viitteet
- Liite A: kaikki skenaariot
- Liite B: laatupuu

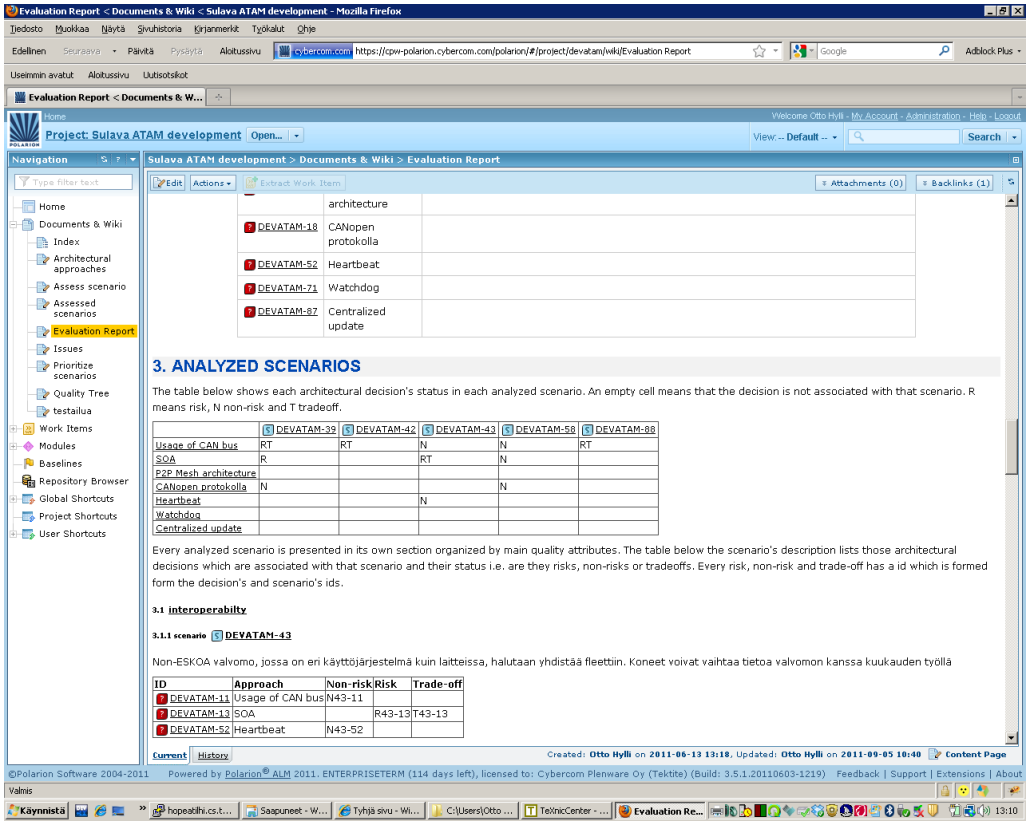
Kansilehdellä kerrotaan, minkä järjestelmän arvioinnista on kyse, mikä organisaatio on kyseessä, dokumentin kirjoittaja, versio ja päiväys. Yhteenveto -kohtaan voidaan kirjoittaa arvioinnin kattava yhteenveto, jossa apuna voidaan käyttää skenaarioiden määrän ja arvioitujen skenaarioiden määrän hakevia makroja. Sanastoon voidaan koota ATAM:iin tai arvioitavaan järjestelmään liittyviä termejä.

Johdanto-luvun tarkoitus on tarjota yleistietoa järjestelmästä ja arviointiprosessista. Tarkoitus ja kattavuus -kohdassa kuvataan dokumentin tarkoitus. Arviointiryhmä-kohdassa listataan arviointiin osallistuneiden henkilöiden nimet ja roolit arvioinnissa. Seuraavissa kohdissa ensiksi kuvataan ATAM yleisesti, jonka jälkeen voidaan kertoa erityisiä huomioita prosessin kulusta kyseisen arvioinnin aikana. Yleinen ATAM-menetelmän kuvaus on dokumentissa jo valmiina, mutta sitä voi halutessaan tietysti muuttaa tai sen voi korvata esimerkiksi paremmin kyseisen dokumentin kohderyhmän huomioonottavalla versiolla.

Kohdejärjestelmä-luku tarjoaa yleistä tietoa arvioitavasta järjestelmästä. Ensiksi kuvataan järjestelmä yleisesti sekä sen arkkitehtuuri. Järjestelmän yleiskuvaus saadaan hakemalla ja näyttämällä järjestelmän yleistiedot sisältävän järjestelmätyöalkion sisältö. Korkean tason arkkitehtuuri saadaan puolestaan järjestelmä-työalkioon suoraan linkitetyistä arkkitehtuurikuvauksista. Seuraava kohta listaa arkkitehtuuri päätökset. Nimen ja id:n lisäksi näytetään päätöksen vaikutus järjestelmään.

Arvioidut skenaariot -luku näyttää varsinaiset arvioinnin tulokset. Ensimmäinen raportin tarjoama uusi näkymä on kuvassa 6.1 näkyvä luvun alussa oleva päätösyhteenvetotaulukko, josta näkee kunkin ratkaisun tilan jokaisessa skenaariossa. Tämä tarjoaa nopean yhteenvedon ATAM:n tuloksista ja auttaa mahdollisten ongelmakohtien löytämisessä, jos esimerkiksi jokin päätös on merkitty riskiksi useassa skenaariossa tai jos huomataan, että jotain oleellista päätöstä ei ole arvioitu missään skenaariossa. Tämän jälkeen näytetään arvioidut skenaariot ATAm-työkalusta tutussa formaatissa. Ainoa ero on se, että skenaariot on luokiteltu laatuominaisuuksien mukaisiin alakohtiin.

Riskiteemat ja analyysi -luku tarjoaa syvällisempää analyysiä arvioinnin tuloksista sekä uusia näkymiä tiedon sisäistämisen tueksi. Riskit -kohdasta löytyy kaksi uutta näkymää. Ensimmäisessä on listattuna laatuominaisuudet, joiden alle on listattu näihin liittyvät riskipäätökset skenaarioineen. Tämä auttaa ongelmallisten tilanteiden löytämisessä, jos esimerkiksi tiettyyn tärkeään laatuominaisuuteen liittyy paljon riskejä. Toinen riskit-kohdasta löytyvä uusi näkymä on taulukko, jossa näytetään jokaisesta päätöksestä kuinka monessa skenaariossa se on katsottu riskiksi, ei-riskiksi sekä kompromissiksi. Oletuksena päätökset on järjestetty riskien



Kuva 6.1: Ruutukaappaus ATAM-raportista, jossa nähdään loppuosa löydetyt ratkaisut -taulukosta sekä arvioidut skenaariot -luvun alkua.

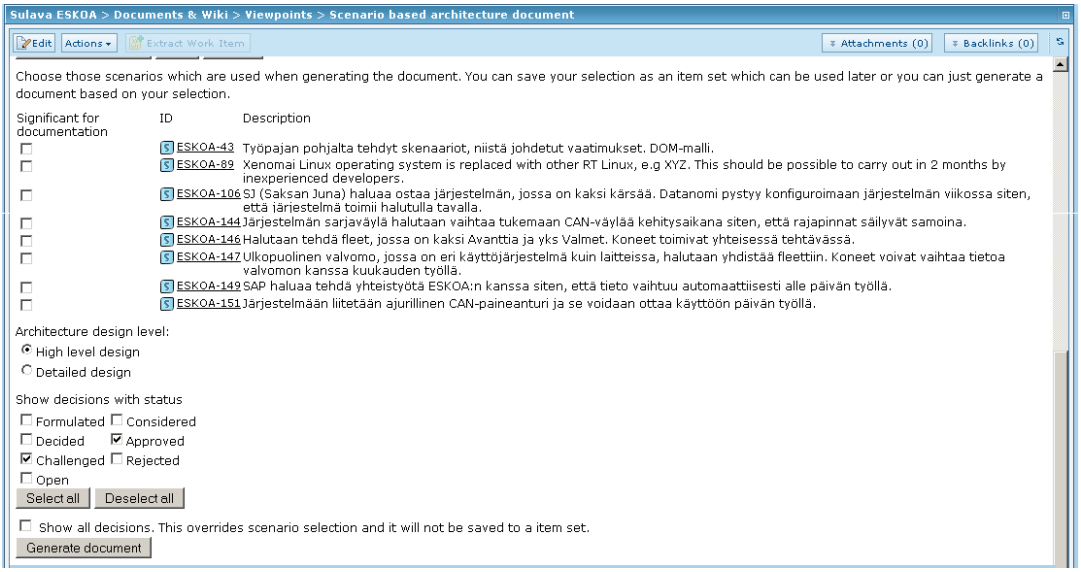
määrän mukaan, mutta sarakkeiden otsikoita klikkaamalla järjestys perustetta voi vaihtaa. Päätösyhteenvedon tapaan tämä näkymä auttaa riskialttiiden tai muuten mielenkiintoisten päätösten löytämisessä. Riskiteemat-kohtaan voidaan kirjata riskiteemoja, joiden avulla voidaan ryhmitellä ja luokitella useasti esiintyviä tai toisiinsa liittyviä riskejä. Huomiot-kohta listaa ATAM:in aikana löytyneet huomiot.

Priorisointiyhteenveto-kohta sisältää puolestaan taulukon, joka näyttää tiivistetysti skenaarioiden tärkeyteen ja työmäärään pohjautuvan priorisoinnin. Taulukko on kolme kertaa kolme taulukko , jossa jokaiselle tärkeyden ja työmäärän yhdistelmälle on oma solunsa, jossa listataan kyseisen prioriteetin omaavien skenaarioiden id:t. Tarkoituksena on, että taulukosta huomataan mahdolliset epäkohdat skenaarioiden valinnassa. Jos esimerkiksi korkean vaikeuden skenaarioita ei ole, voi se kertoa siitä, että skenaarioiden luomisessa on ehkä tiedostamattomastikin vältelty hankalia tilanteita.

Johtopäätökset-lukuun on tarkoitus kirjata arvioinnista vedettävät johtopäätökset. Raportin lopussa liitteinä ovat ATAM-työkalusta tutut laatupuu sekä lista kaikista skenaarioista priorisoituna. Mukana on makrot sekä tärkeyden ja työmäärän että priorisointipisteiden mukaisen priorisoinnin näyttävistä taulukoista.

6.4 Skenaariopohjainen arkkitehtuuridokumentti

Skenaariopohjainen arkkitehtuuridokumentti -näkymä on tarkoitettu tietyn käyttäjän tarpeisiin sopivan arkkitehtuuripäätöksiin keskittyvän arkkitehtuuridokumentin luomiseen. Ajatuksena on, että erilaiset päätöstiedon käyttäjät ovat kiinnostuneita eri asioihin liittyvistä päätöksistä. Tällöin kaikki päätökset sisältävästä dokumentista tiettyyn aiheeseen liittyvien päätösten löytäminen on hankalaa. Näkymässä tiettyyn aiheeseen liittyvien päätösten löytämisessä käytetään skenaarioita, joihin on liitetty päätöksiä. Päätöksiä ja skenaarioita yhdistetään ATAM-arvioinnissa, mutta näkymän käyttö ei edellytä ATAM-arvioinnin suorittamista, sillä skenaarioita ja niiden linkkejä päätöksiin voidaan luoda ATAM:in ulkopuolellakin. Jos skenaariot ovat syntyneet ATAM:in tuloksena, on niiden luomiseen osallistunut useiden sidosryhmien edustajia, jolloin ne ovat monipuolisia ja käsittelevät järjestelmää ja sen arkkitehtuuria eri näkökulmista, jotka todennäköisesti edustavat hyvin erilaisten päätöstiedon käyttäjien tarpeita. Näin ollen skenaariot soveltuvat hyvin kiinnostavien päätösten löytämiseen.



Kuva 6.2: Skenaariopohjaisen arkkitehtuuri dokumentti -näkymän luontivalinnat.

Näkymä jakautuu kahteen osaan. Näkymän avattuaan käyttäjän tulee valita listasta ne skenaariot, joiden pohjalta dokumentti rakennetaan. Vaihtoehtoisesti käyttäjä voi valita kaikkien päätösten näyttämisen, jolloin mukaan saadaan myös päätöksiä, jotka eivät liity mihinkään skenaarioon. Käyttäjän tulee myös valita, millä tarkkuudella järjestelmän arkkitehtuuri esitetään. Vaihtoehtoja ovat korkea ja yksityiskohtainen. Käyttäjä voi myös suodattaa näytettäviä päätöksiä päätöksen tilan mukaan. Näkymässä listataan kaikki mahdolliset päätösten tilat, joista käyttäjä voi valita haluamansa. Tehtyään haluamansa valinnat käyttäjä painaa generoi dokumentti -painiketta. Esimerkki tästä näkymän osasta löytyy kuvasta 6.2.

Käyttäjä voi tallentaa tekemänsä skenaariovalinnat, jolloin ne ovat käytettävissä myöhemminkin. Tallennettavalle valinnalle annetaan nimi ja kuvaus. Ajatuksena on, että eri käyttäjät voivat luoda erilaisiin tilanteisiin sopivia skenaariovalintoja. Nämä luodut valinnat listataan näkymän aloitussivulla ennen skenaarioiden valintaa. Käyttäjä voi valita haluamansa valmiin valinnan ja generoida siitä dokumentin. Valintojen poistaminen ja muokkaaminen jälkikäteen on myös mahdollista. Skenaariovalintojen tallennus on toteutettu uudella alkiojoukko-tyyppisellä työalkiolla, joka ei ole osa metamallia. Skenaarion liittäminen alkiojoukkoon tapahtuu on jäsen -linkkisuhteen avulla, jollainen voi olla kaikkien eri tyyppisten työalkioiden ja työalkiojoukon välillä.

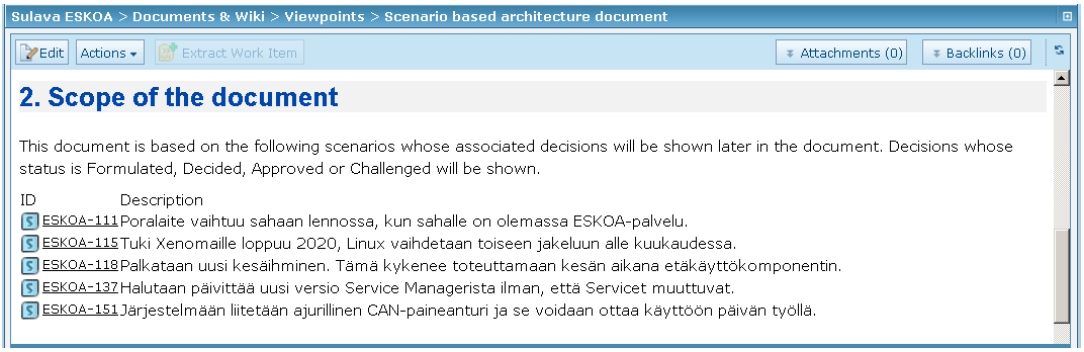
Alla on listattu näkymän toisen osan eri varsinaisen dokumentin sisältö.

- Johdanto
- Dokumentin laajuus
- Arkkitehtuurisesti merkittävät vaatimukset
- Arkkitehtuurin yleiskuvaus
- Luku jokaiselle toisen tason arkkitehtuurikuvaukselle
  - Kohta jokaiselle kuvauksen lapsikuvaukselle ja näiden lapsikuvauksille. (yksityiskohtainen arkkitehtuurikuvaus)
- Arkkitehtuuriarvioinnin yhteenveto

Johdanto-luvussa esitetään järjestelmän yleiskuvaus, joka koostuu järjestelmää edustavan järjestelmä -työalkion sisällöstä. Tämän luvun avulla myös järjestelmää huonosti tunteva henkilö saa yleiskuvan siitä mistä on kyse. Dokumentin laajuus -luvussa listataan ne skenaariot, joiden pohjalta dokumentti rakennetaan eli ne skenaariot, joita arvioivat päätökset dokumentissa näytetään. Näin esimerkiksi tietyn valmiin skenaariovalinnan valinnut käyttäjä voi nähdä mihin skenaarioihin dokumentti tarkalleen perustuu. Tässä luetellaan myös ne tilat, joissa olevat päätökset näytetään. Jos käyttäjä on valinnut kaikkien päätösten näyttämisen, ei tätä lukua näytetä ollenkaan ja lista tiloista näytetään johdannossa. Esimerkki dokumentin laajuus -luvusta on kuvassa 6.3.

Arkkitehtuurisesti merkittävät vaatimukset -luvussa listataan ne vaatimukset, jotka liittyvät dokumentissa esitettäviin päätöksiin. Näin dokumentin lukija saa käsityksen arkkitehtuurin ja päätösten taustalla vaikuttavista tekijöistä. Arkkitehtuurin yleiskuvaus -luku antaa lukijalle arkkitehtuurista kokonaiskuvan näyttämällä korkeimman tason arkkitehtuurikuvaukset ja näihin liittyvät päätökset. Korkeimman tason kuvauksia ovat ne arkkitehtuurikuvaukset, joilla on suunnittelee -linkkisuhde järjestelmään.

Loput alemman tason arkkitehtuurikuvaukset ja päätökset esitetään seuraavissa



Kuva 6.3: Esimerkki dokumentin laajuus -luvusta.

luvuissa. Jokaista toisen tason arkkitehtuurikuvausta eli korkeimman tason arkkitehtuurikuvauksen lapsikuvausta kohden on oma lukunsa. Luvussa näytetään ensiksi kyseinen arkkitehtuurikuvaus, jonka jälkeen listataan siihen ja sekä kaikkiin sen alla oleviin kuvauksiin liittyvät päätökset, mikäli on valittu korkea arkkitehtuurin esittämisen taso. Jos tasoksi on valittu yksityiskohtainen, näytetään kaikki arkkitehtuurikuvaukset päätöksineen hierarkisesti omissa kohdissaan. Näin päätökset saadaan järjestettyä järkevästi ja lukija saa kuvan järjestelmän arkkitehtuurista ja ennen kaikkea siitä, mihin päätökset vaikuttavat arkkitehtuurissa. Yksittäisestä päätöksestä näytetään kaikki tiedot: id, nimi, kuvaus, ongelma, voimien kuvaus sekä perustelut. Tämän lisäksi listataan päätöksen lapsipäätökset, vaihtoehtoiset päätökset sekä päätöksen soveltamat suunnittelumallit ja standardiratkaisut. Jokainen dokumentin laajuuteen kuuluva päätös näytetään vain kerran, vaikka se liittyisikin useampaan arkkitehtuurikuvaukseen. Näin vältytään turhalta toistolta. Ennen tietyn päätöksen näyttämistä selvitetään, onko päätös jonkin muun päätöksen lapsi. Jos näin on eikä tätä korkeamman tason päätöstä ole jo näytetty, näytetään se ennen sen lapsipäätöstä. Tämä tehdään vaikka kyseinen päätös ei liittyisikään mihinkään pohjalla olevista skenaarioista. Näin varmistetaan, että lukija ymmärtää lapsipäätöksen täysin.

Viimeisessä luvussa esitetään yhteenveto arkkitehtuurille tehdystä ATAM-arvioinnista. Yhteenvetona käytetään ATAM-raportista tuttua taulukkoa, jossa on listattu päätökset ja kuinka monessa skenaariossa ne katsottiin riskeiksi, ei-riskeiksi ja kompromisseiksi. Taulukosta lukija näkee, millaiseksi jokin häntä kiinnostava päätös on arvioitu tai miltä arvioinnin tulokset yleisesti näyttävät: löytyikö esimerkiksi riskejä tai kompromissejä paljon.

7. TIETÄMYSKANNAN SOVELTAMINEN: IDLE

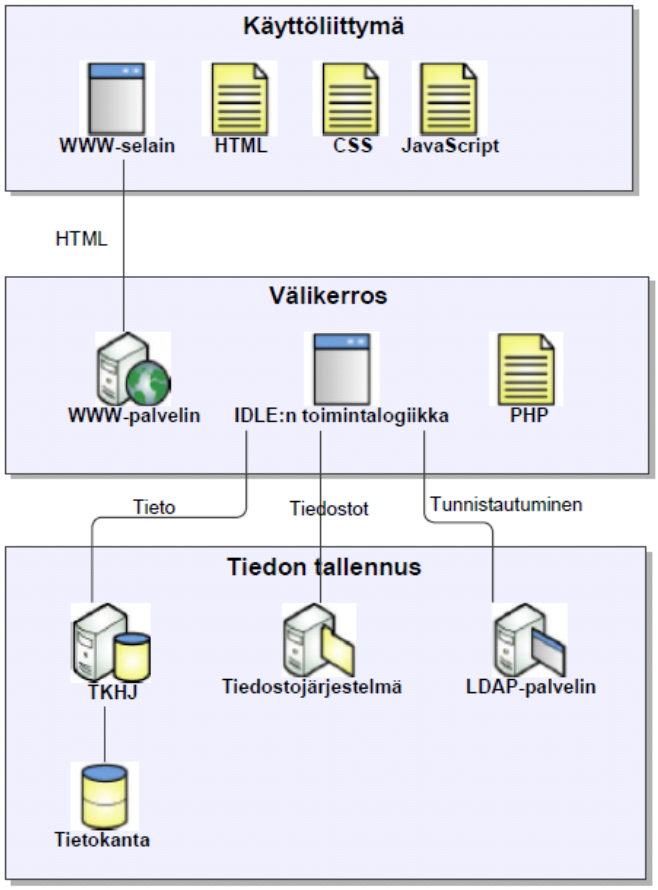
7.1 IDLE

Arkkitehtuuritietämyskannan testaamiseksi sitä sovellettiin TTY:n ohjelmistotekniikan laitoksella kehitettyyn laitoksen kursseilla käytettävään IDLE-oppimisympäristöön. IDLE on perinteinen asiakas-palvelin-mallin mukainen web-sovellus eli sitä käytetään internet-selaimella ja varsinainen toimintalogiikka sijaitsee palvelimella. IDLE:n tarkoitus on vähentää kurssien hallintotyöhön käytettävää aikaa sekä helpottaa viikkoharjoitusten pitämistä. IDLE:n avulla opiskelijat voivat ilmoittautua kurssien viikkoharjoituksiin ja muihin tilaisuuksiin sekä tehdä ja palauttaa harjoitustehtäviä niin kotona kuin viikkoharjoituksissa. Opiskelijat voivat myös muodostaa IDLE:n avulla ryhmiä harjoitustöiden tekemiseen ja viikkoharjoitusten tekemiseen. [14].

Korkean tason arkkitehtuuriltaan IDLE noudattaa kolmikerrosarkkitehtuuria, jossa ohjelma on jaettu erillisiin kerroksiin, jotka kommunikoivat vain seuraavan ja edellisen kerroksen kanssa. Käyttöliittymänä toimii internet-selain, toimintalogiikka sijaitsee web-palvelimella ja tietojen talletuksesta huolehtii SQL-pohjainen tietokanta. Kuva 7.1 esittää tämän IDLE:n korkean tason arkkitehtuurin. IDLE:n toimintalogiikan arkkitehtuuri noudattaa puolestaan malli-näkymä-ohjain eli MVC -arkkitehtuuria. Ohjaimet käsittelevät selaimelta tulevat HTTP-pyynnöt hakemalla tai tallettamalla tarvittavat tiedot. Tietojen käsittelyyn ohjaimet käyttävät tietosisältöä edustavia malleja, jotka huolehtivat tietokannan käsittelystä. Käyttäjille näytettävien web-sivujen rakentamiseen käytetään puolestaan näkymiä.

7.2 Arkkitehtuuritietämyksen lähteet

IDLE:n arkkitehtuuritietämyksen tallentaminen Polarioniin aloitettiin IDLE:lle pidetyllä ATAM-arvioinnilla, jossa käytettiin kohdassa 5.2.2 esiteltyä ATAM-työkalua. ATAM:in avulla saatiin kerättyä 28 arkkitehtuuripäätöstä, seitsemän ylimmän tason laatuominaisuuttta ja 34 skenaariota, joista 7 arvioitiin. ATAM:in jälkeen pidettiin erillinen tilaisuus, jossa täydennettiin päätösten tiedot. ATAM:issa ja päätösten täydennystilaisuudessa mukana oli IDLE:ä hyvin tuntevia henkilöitä. Muu Polarioniin tallennettu IDLE:n arkkitehtuuritietämys kerättiin IDLE:ä käsittelevästä dokumentaatiosta IDLE-projektin ulkopuolisten henkilöiden toimesta. Esimerkiksi arkkitehtuurikuvaukset olivat IDLE:ä käsittelevästä diplomityöstä [14] ja vaatimukset



Kuva 7.1: Korkean tason esitys IDLE:n arkkitehtuurista [14].

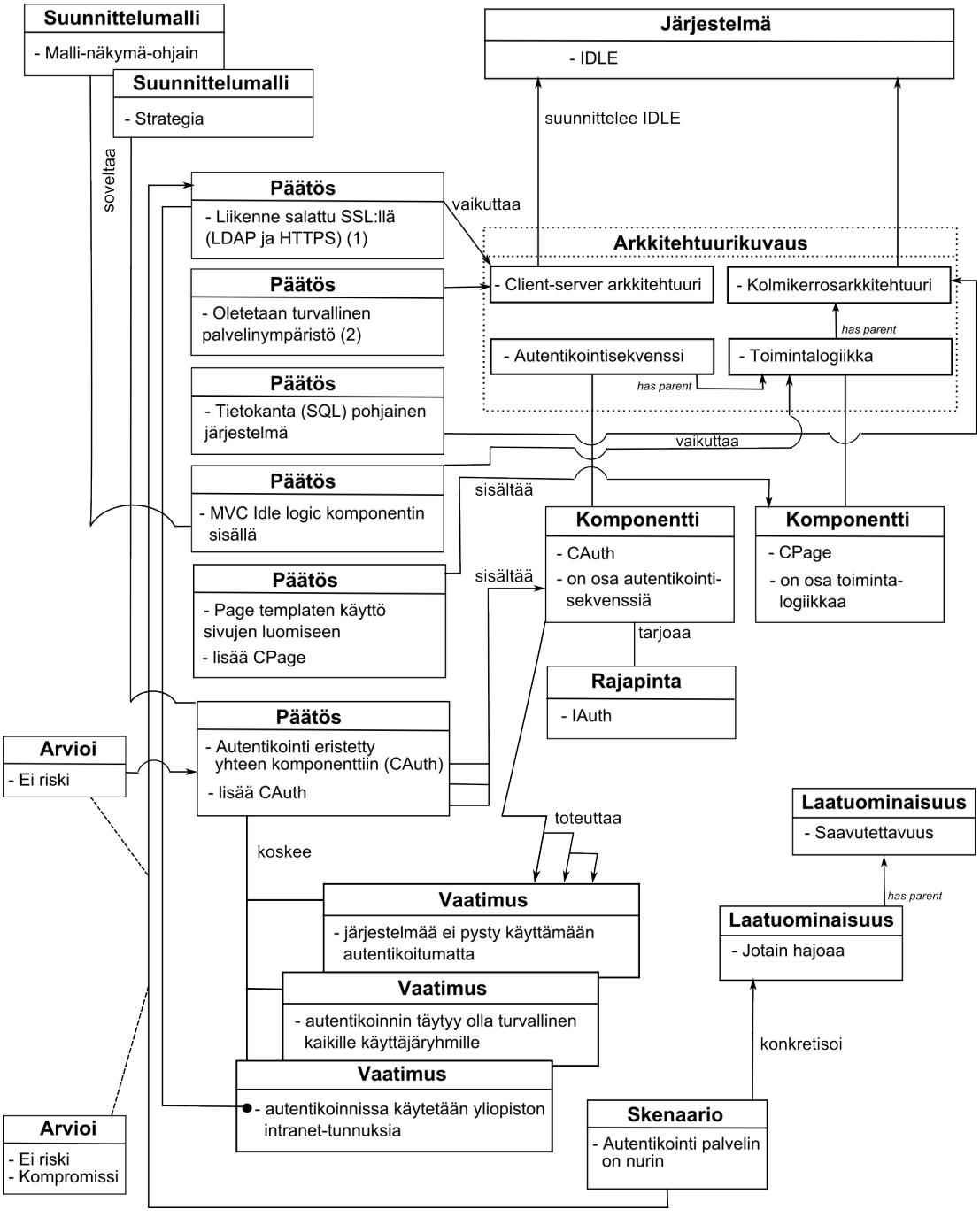
IDLE:n kehityksen aluksi tehdystä esitutkimuksesta.

7.3 Esimerkkiotos IDLE:n arkkitehtuuritietämyksestä

Tämän kohdan tarkoitus on antaa konkreettisia esimerkkejä siitä, millaista tietämyskantaan tallennettava arkkitehtuuritietämys voi olla ja minkälaisia hyötyjä se tarjoaa tiedon käyttäjille. Kuva 7.2 esittää esimerkkiotoksen IDLE:n arkkitehtuuritietämyksestä. Se esittää joukon tietämystyöalkioita, joista kerrotaan tyyppi ja nimi sekä näiden alkioiden väliset suhteet. Kuva sisältää ainakin yhden esimerkin jokaisesta eri tietämystyypistä standardiratkaisua ja päätösaihetta lukuunottamatta.

Järjestelmä-työalkio nimeltä *IDLE* antaa yleiskuvan koko järjestelmästä. Siinä selitetään, mikä IDLE on, mitä sillä voi tehdä ja miksi sitä tarvitaan. Työalkion tarkoitus on tarjota järjestelmää huonosti tuntevalle henkilölle yleiskuva järjestelmästä, joka auttaa häntä muun arkkitehtuuritietämyksen ymmärtämisessä.

Järjestelmään suunnittelee-suhteella liittyvät kaksi arkkitehtuurikuvausta *Client-server arkkkitehtuuri* ja *kolmikerrosarkkitehtuuri* esittävät IDLE:n korkeimman tason arkkitehtuuri ratkaisuja. Ne auttavat IDLE:n perusrakenteen ymmärtämistä,



Kuva 7.2: Esimerkkiotos IDLE:n arkkitehtuuritietämyksestä

Otsikko	Kolmikerrosarkkitehtuuri
Kuvaus	WWW-sovellukselle ominaisesti sovellus on toteutettu käyttäen kolmikerrosmallia. Sovelluksessa käyttöliittymä, toimintalogiikka ja tiedon tallennus on toteutettu erillisinä kokonaisuuksina, jotka kommunikoivat keskenään asiakas-palvelin-mallin mukaisesti. Sovelluksen käyttöliittymä toimii käyttäjän WWW-selaimessa. WWW-selaimen ja tiedon tallennukseen käytettävän tietokantapalvelimen välissä on WWW-palvelimella sijaitseva välikerros, joka toteuttaa sovelluksen kaiken toiminnallisuuden. Välikerros toimii siten asiakkaana tietokantapalvelimelle ja palvelimena WWW-selaimelle.

Taulukko 7.1: Kolmikerrosarkkitehtuuri -arkkitehtuurikuvaustyöalkion sisältö. Kuvaukseen kuuluu tekstin lisäksi kuva 7.1

jota tarvitaan ennen kuin perehdytään yksityiskohtaisempaan arkkitehtuuriin. Arkkitehtuurikuvaukset kertovat millainen arkkitehtuuri on kuten taulukossa 7.1 oleva *kolmikerrosarkkitehtuuri*- työalkion sisältö. Tarkempaa tietoa kuvaukseen liittyvistä ratkaisuista ja niiden taustoista kuten syistä löytyy kuvaukseen vaikuttavista arkkitehtuuripäätöksistä. Esimerkiksi päätös *tietokanta (SQL) pohjainen järjestelmä* selittää, miksi IDLE rakennettiin juuri SQL-tietokantapohjaiseksi.

Kolmikerrosarkkitehtuuri -kuvauksen eräs lapsi kuvaus on *toimintalogiikka*, jonka tarkoitus on tarkentaa isä-kuvauksen jotain osaa eli tässä tapauksessa WWW-palvelimella sijaitsevaa toimintalogiikkaa. Kuvauksessa kerrotaan, miten kohdassa 7.1 mainitut ohjaimet, mallit ja näkymät toimivat eli miten IDLE:ssä sovelletaan MVC-mallia. Esimerkkinä tähän arkkitehtuurikuvaukseen liittyvistä komponenteista on sivumoottoriluokka *CPage*, jota näkymät käyttävät käyttäjille näytettävien sivujen luomisessa. *CPage* sisältää muun muassa kaikille sivuille yhteiset elementit kuten IDLE:n päävalikon. Komponentin kuvaus kertoo, mitä luokka tekee, ja komponentin järjestelmään lisännyt *Page templaten käyttö sivujen luomiseen* päätös kertoo, miksi tällaisen komponentin lisäämiseen päädyttiin. Koska *CPage* on osa *toimintalogiikka* -arkkitehtuuri kuvausta, on sen lisäämistä käsittelevästä päätöksestä myös yhteys tähän kuvaukseen.

MVC-mallin valinnan taustoista antaa puolestaan lisätietoja päätös *MVC Idle logic komponentin sisällä*. Tämä päätös kertoo, miksi MVC:tä käytetään juuri IDLE:ssä. Päätös on taulukossa 7.2. Päätökseen soveltaa-suhteella liitetty *mallinäkymä-ohjain* -suunnittelumalli tarjoaa yleiskuvan suunnittelumallista, sen rakenteesta, sen käyttöön johtavista syistä sekä sen hyvistä ja huonoista puolista. Tämän kaltainen päätösten ja suunnittelumallien käyttö helpottaa päätösten dokumentointia, kun tietyn suunnittelumallin mukaisen päätöksen voi kuvata lyhyesti kyseisen järjestelmän kannalta ja liittää suunnittelumalliin, joka sisältää tarkemman

Otsikko	MVC IDLE logic komponentin sisällä
Ongelma	Miten eriyttää käyttöliittymä, toimintalogiikka ja tietokanta?
Voimat	• Bisnes-tavoitteet: Ylläpito pitäisi olla mahdollisimman helppoa: Esim. jos tietomalli muuttuu taustalla tai tietokanta vaihdetaan, pitäisi muutos olla helppo tehdä. • Prosessit: Järjestelmän rakenne pitäisi olla ymmärrettävä • Muut päätökset: Järjestelmässä kolmikerrosarkkitehtuuri korkealla tasolla.
Vaikutus järjestelmään	toimintalogiikka-komponentin sisällä järjestelmä on jaettu MVC:n mukaisesti model, view ja controller osiin. Eri osat näkyvät kooditiedostoissa alkukirjaimilla.
Perustelut	• Mahdollistaa useat näkymät samaan tietoon • Helppo muokata: Eri osat, logiikka, tietokanta, käli helposti löydettävissä eri tiedostoissa. • Laajasti käytetty ja helppotajuinen ratkaisu
On suunnittelusääntö	Ei

Taulukko 7.2: MVC IDLE logic -komponentin sisällä -päätöstyöalkion sisältö

kuvauksen ratkaisusta. Samaa suunnittelumallin tai standardiratkaisun kuvausta voidaan käyttää useassa eri projektissa. Itseasiassa Sulava-projektissa luotiin Polarioniin suunnittelumallikirjasto -projekti, johon talletettiin sulautetuissa koneenohjausjärjestelmissä käytettyjä suunnittelumalleja.

Toimintalogiikka -arkkitehtuurikuvauksen lapsikuvauksessa *autentikointisekvenssi* mennään jo melko yksityiskohtaiseen arkkitehtuurikuvaukseen. Se sisältää tapahtumasekvenssikaavion ja selityksen siitä, miten IDLE:n autentikointi toimii tilanteessa, jossa kirjautumaton käyttäjä yrittää mennä sivulle, johon ei kirjautumatta pääse. Tilanteessa käyttäjä ohjataan kirjautumissivulle, josta hänet onnistuneen kirjautumisen jälkeen ohjataan hänen alunperin pyytämälleen sivulle. Tässä sekvenssissä mukana ovat muun muassa *CPage*-luokka, autentikoinnin yksityiskohdat toteuttava *CAuth*-luokka ja autentikoinnin rajapinnan määrittävä *IAuth*-rajapinta, jonka *CAuth*-luokka tarjoaa ja joka mahdollistaa erilaisten autentikointitapojen toteuttamisen ja käytön. *CAuth* ja *IAuth* -työalkioiden sisällöt löytyvät taulukosta 7.3. Autentikointiratkaisua selittää *strategia*-suunnittelumallia soveltava *autentikointi eristetty yhteen komponenttiin (CAuth)* -päätös, jolla on vaikutusta arkkitehtuurikuvaukseen, rajapintaan sekä komponenttiin. Samoin kaikkiin edellä mainittuihin autentikointi-työalkioihin vaikuttavat kolme autentikointiin liittyvää vaatimusta kuten *järjestelmää ei pysty käyttämään autentikoitumatta*. Näin saadaan

Otsikko	CAuth
Kuvaus	Tunnistautuminen on erotettu muusta järjestelmästä selvästi erilleen, jotta siirtyminen erilaiseen tunnistautumismenetelmään olisi mahdollisimman helppoa. Tunnistautumisen hoitaa erillinen luokka CAuth. CAuth-luokka toteuttaa IAuth-rajapinnan, joka määrittää metodit, joita käytetään tunnistautumiseen. Tunnistautumismenetelmä voidaan valita IDLE:n asetustiedostossa. Asetustiedostossa määritetään PHP-tiedosto, joka sisältää tunnistautumisesta huolehtivan CAuth-luokan.
On ohjelmisto	Kyllä

Otsikko	IAuth
Kuvaus	IAuth-rajapinta määrittää metodit, joita käytetään tunnistautumiseen. Rajapinta sisältää kaksi julkista metodia, joista toinen tunnistaa käyttäjän tunnus-salasanaparilla ja toinen hakee käyttäjän tiedot.
On ohjelmisto	Kyllä

Taulukko 7.3: CAuth komponentin ja IAuth rajapinnan sisällöt.

jäljitettävyyttä autentikoinnin toteutustavan ja siihen liittyvien vaatimusten välillä. Jos esimerkiksi huomataan, että jokin vaatimus ei täysin täyty, voidaan helposti löytää tätä vaatimusta toteuttavat järjestelmän osat ja siihen vaikuttavat päätökset.

Esimerkkinä ATAM-arvioinnissa syntyneestä skenaariosta on *Autentikointipalvelin on nurin*. Skenaarion analyysi tarjoaa lisätietoa tehtyjen ratkaisujen seurauksista. Skenaarion ja päätösten välisistä suhteistä nähdään, että autentikoinnin eristys yhteen komponenttiin ja SSL:n käyttö on katsottu hyviksi ratkaisuiksi, vaikka SSL:n käyttöön liittyykin kompromissi. Skenaario-työalkion sisältö on taulukossa 7.4. Skenaariosta nähdään myös, että se liittyy laatuominaisuuteen *jotain hajoaa*, joka puolestaan liittyy päälaatuominaisuuteen *saavutettavuus*.

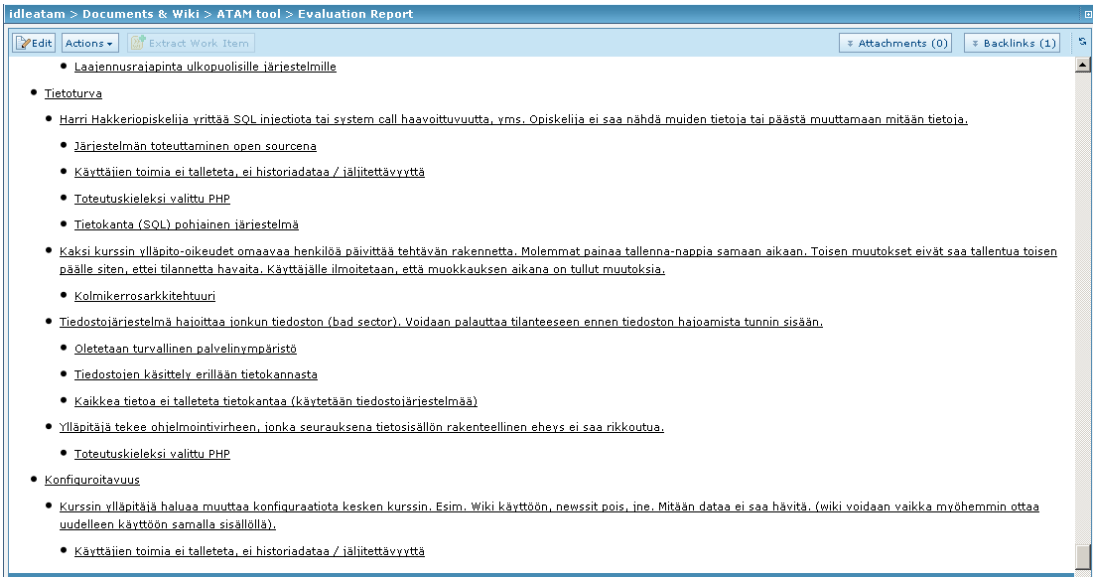
7.4 Näkymäesimerkki: ATAM-raportti

Seuraavassa esitellään esimerkkinä IDLE:n arkkitehtuuritietämyksestä rakennetuista näkymistä sisältöä IDLE:n ATAM-raportista. Tarkoituksena on sisältöesimerkin lisäksi näyttää, miten näkymää voi käyttää eli miten esitetyn tiedon pohjalta voi tehdä johtopäätöksiä ATAM:in tuloksista.

Kuvassa 7.3 on raportin kohdasta 4.1 löytyvä lista riskeistä jaoteltuna skenaarioiden ja laatuominaisuuksien mukaan. Siitä nähdään, että tietoturvallisuuteen liittyy selvästi enemmän riskejä sisältäviä skenaarioita kuin muihin laatuominaisuuksiin. Laajennettavuuteen ja konfiguroitavuuteen liittyy kuhunkin yksi riskipäätös ja skenaario, kun taas tietoturvallisuuteen liittyy 8 eri riskipäätöstä ja neljä skenaariota. Tästä ei kuitenkaan voi vetää suoraan johtopäätöstä, että IDLE:n tietoturva olisi erityisen huono, vaikka siihen selvästikin liittyy useita mahdollisia ongelmia, jotka

Otsikko	Autentikointipalvelin on nurin.
Kuvaus	Autentikointipalvelin on nurin, eikä siihen saada yhteyttä. Tilanne ei saa vaikuttaa käyttäjiin mitenkään. Pitäisi voida käyttää LDAP:n varapalvelinta.
Tärkeys	Korkea
Työmäärä	Keskimääräinen
argumentaatio	WI-28 Autentikointi eristetty yhteen komponenttiin (CAuth): Mahdollistaa erilaisten autentikointien käytön ja helpottaa ylläpitoa. WI-31 Liikenne salattu SSL:llä (LDAP ja HTTPS): Muukin voi mennä pieleen. Esim. jos sertifikaatti on vanhentunut voi tämän johdosta yhteys jäädä saamatta, vaikka palvelin olisikin ok. SSL kuitenkin pakko olla, jotta saadaan lupa käyttää TIHAn palveluita. (Trade-off).
Analyysi	Tilanne pitää havaita ja pystyä vaihtamaan varapalvelimeen. Jos ei voida varmentaa henkilöllisyyttä ei tietoja saa näyttää. Jos molemmat palvelimet nurin, pitää yrittää uudestaan niin kauan että onnistuu. Mikäli on muita autentikointitapoja, niin niitä voitaisiin käyttää, koska IDLE tukee useita autentikointitapoja (esim. lintulan LDAP). IDLE:ssä havainnointiin aikakatkaisu, jonka jälkeen koitetaan toiseen palvelimeen. Palvelimet kovakoodattu eli ei kovin helposti konfiguroitavissa.
Arvioitu	kyllä
Priorisointipisteet	0
Lähde	Laatupuu

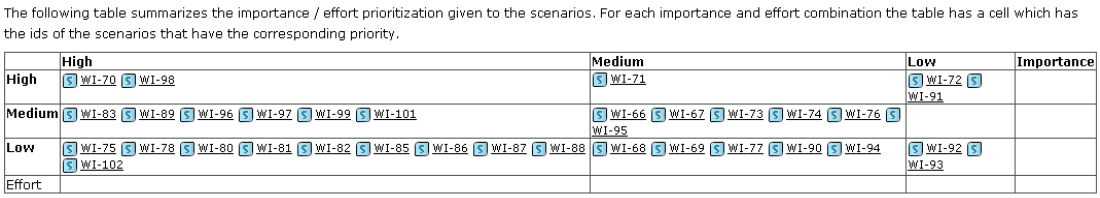
Taulukko 7.4: Autentikointipalvelin on nurin -skenaariotyöalkion sisältö.



Kuva 7.3: ATAM:issa löydetyt riskit jaoteltuna laatuominaisuuksien mukaan.

tulee huomioida. Muualta raportista voidaan nähdä, että arvioiduista seitsemästä skenaariosta neljä liittyy tietoturvallisuuteen ja loput kolme jokainen eri laatuomi-

naisuuteen, joten verrattain suuri tietoturvallisuuteen liittyvien riskien määrä ei ole niin huolestuttavaa. Riskien lista siis kertoo siitä, että skenaarioiden priorisoitiin perustuen tietoturvallisuus on katsottu tärkeimmäksi IDLE:n laatuominaisuudeksi.



The following table summarizes the importance / effort prioritization given to the scenarios. For each importance and effort combination the table has a cell which has the ids of the scenarios that have the corresponding priority.

	High	Medium	Low	Importance
High	WI-70 WI-98	WI-71	WI-72 WI-91	
Medium	WI-83 WI-89 WI-96 WI-97 WI-99 WI-101	WI-66 WI-67 WI-73 WI-74 WI-76 WI-95		
Low	WI-75 WI-78 WI-80 WI-81 WI-82 WI-85 WI-86 WI-87 WI-88 WI-102	WI-68 WI-69 WI-77 WI-90 WI-94	WI-92 WI-93	
Effort				

Kuva 7.4: IDLE:lle suoritetun ATAM-arvioinnin priorisointiyhteenveto.

Kuvassa 7.4 on ATAM-raportin kohdan 4.3 priorisointiyhteenvetotaulukko, joka tiivistää skenaarioiden tärkeyteen ja työmäärään pohjautuvan priorisoinnin. Taulukosta nähdään, että korkean tärkeyden skenaarioita on paljon enemmän kuin korkean työmäärän skenaarioita. Korkean tärkeyden skenaarioita on 18 ja työmäärän viisi. Korkean tärkeyden ja matalan työmäärän skenaarioita on myös paljon. Tällaisia skenaarioita on kymmenen, kun seuraavaksi yleisimmällä prioriteetilla olevia skenaarioita on kuusi. Korkean työmäärän skenaarioita ei siis ehkä löytynyt tarpeeksi. Korkean tärkeyden ja matalan työmäärän skenaarioiden suuri määrä johtuu ainakin osittain siitä, että osa näistä skenaarioista käsitteli IDLE:n perusvaatimuksia, jotka useamman vuoden ajan kehityksessä ja käytössä ollut IDLE toteuttaa.

8. ARVIOINTI

8.1 ATAM-Työkalu

ATAM-työkalun toiminnan ja käytettävyyden arvioinniksi sitä käytettiin IDLE-oppimisympäristölle (katso kohta 7.1) suoritetussa ATAM-arvioinnissa. Ajan ja sopivien henkilöiden puutteen vuoksi ATAM:ista suoritettiin vain ensimmäinen vaihe. Vaikka toisen vaiheen toiminnallisuus, jota ei oltu vielä edes tässä vaiheessa toteutettu, jäikin testaamatta, antoi tämä tilaisuus hyvän kuvan työkalun toimivuudesta, sillä suurinta osaa työkalun toiminnoista käytettiin.

Työkalun havaittiin toimivan hyvin eikä suurempia ongelmia tullut. Työkalun käyttäjän eli tietojen syöttäjän mielestä työkalun käyttö oli sujuvaa. Työkalun käyttäjä on kokenut ATAM-tilaisuuksien vetäjä ja hänellä oli aiempia kokemuksia ATAM-työkalusta, joten työkalun sopivuutta uudelle käyttäjälle tai ATAM:ia huonosti tuntevalle henkilölle ei voi tämän perusteella arvioida. Muiden osallistujien mielestä prosessin seuraaminen ja siihen osallistuminen kuten skenaarioon liittyvien päätösten nimeäminen onnistui hyvin, kun käytettiin dataprojektoria tietokoneen näytön sisällön näyttämiseen kaikille. Esimerkki esiin tulleesta hyvästä käyttöliittymäominaisuudesta on laatupuu kohteen korostuminen eli tekstin taustan värin vaihtuminen hiirellä osoitettaessa, jolloin läsnäolioille on helppoa osoittaa, mistä puhutaan.

Työkalusta löydettiin myös korjattavaa ja parannettavia asioita. Näitä olivat esimerkiksi seuraavat kolme asiaa, jotka korjattiin työkalun viimeisimpään versioon.

- Mahdollisuus tehdä välitallennus skenaarioita priorisoitaessa tärkeyden ja työmäärän mukaan. Käytetyssä versiossa tämä ei ollut mahdollista, koska työkalu vaati, että jokaisen skenaarion prioriteetti kentissä on tallennettaessa kelvollinen arvo eli kenttä ei saanut olla tyhjä.

- Skenaariota arvioitaessa, kun mennään lisäämään päätöksiä, näytetään listassa myös jo valitut päätökset, joita voi myös poistaa. Aiemmin näytettiin vain ne päätökset, jotka eivät liity skenaarioon, jolloin jo valittuja päätöksiä ei voinut tarkastaa samasta näkymästä.

- Kun laatupuuhun lisätään kohta, palautetaan sivu lisäyksestä seuraavan sivun uudelleen latauksen jälkeen lähemmäksi juuri lisättyä kohtaa. Aiemmassa versiossa palattiin sivun ylälaitaan, jolloin sivua joutui vierittämään oikean

kohdan löytämiseksi.

Yleisesti työkalun web-pohjaisuus tarjoaa etuja. Vaikka testitilaisuudessa työkalua käytti vain yksihenkilö, ja se on ensisijaisesti suunniteltu siten, että yksi henkilö syöttää tietoja, voisi päätöksiä kerättäessä useampi henkilö kirjata päätöksiä työkaluun yhtäaikaa. Lopuksi voitaisiin sitten katsoa luotujen päätösten listaa yhteisesti ja poistaa mahdolliset päällekkäiset päätökset. Tällainen toimintatapa voisi tehostaa päätösten keräämistä. ATAM-tilaisuuden läsnäolijat voivat myös selata ja tutkia syötettyä tietoa tilaisuuden aikana omilla tietokoneillaan, jolloin heidän ei tarvitse rajoittua siihen, mitä dataprojektorilla näkyy. Varjopuolena web-pohjaisuudessa on puolestaan riski verkkoyhteyden katkeamisesta, hidastelusta tai palvelimen hidastelusta suuren kuorman takia, jotka voivat suurestikin haitata ATAM-tilaisuuden sujumista. Lisäksi Polarion voi kirjata käyttäjän ulos, jos sitä ei käytetä vähään aikaan. Näin on ATAM-työkalua käytettäessä mahdollista menettää tietoa, jos sitä ei ole tallennettu ennen esimerkiksi kahvitaukoa. Tämä ongelma on kuitenkin mahdollista kiertää, kun polarioniin kirjauduttaessa valitsee asetuksen, että järjestelmä pitää käyttäjän sisään kirjautuneena.

8.2 Skenaariopohjainen arkkitehtuuridokumentti

Näkymän arviointia varten IDLE:n (katso kohta 7.1) arkkitehtuuritietämyksestä generoitiin dokumentti laajimmilla mahdollisilla parametreilla, jolloin dokumenttiin saatiin mahdollisimman paljon päätöksiä ja arkkitehtuurikuvauksia. Dokumentista luotu PDF-versio lähetettiin kahdelle IDLE:ä hyvin tuntevalle henkilölle: asiakkaana olevalle TTY:n ohjelmistotekniikan laitoksen työntekijälle ja pääarkkitehdille. He tutustuivat dokumenttiin, jonka jälkeen pidettiin palautetilaisuus, jossa dokumenttia käytiin läpi. Tarkoituksena oli kerätä yleistä palautetta dokumentista kuten sen rakenteesta sekä selvittää, miltä dokumentti näyttää tiettyä järjestelmää tuntevien henkilöiden mielestä eli antaako se hyvän kuvan järjestelmästä.

Yleisesti dokumenttiin oltiin melko tyytyväisiä. Generoitu dokumentti vastasi asiakkaan mielestä melko hyvin hänen käsitystään arkkitehtuuridokumentista rakenteensa ja sisältönsä puolesta. Pääarkkitehti uskoo, että uusi IDLE:n kehittäjä saisi dokumentista irti aika paljon hyödyllistä tietoa.

Suurimmat ongelmat dokumentissa liittyivät sen rakenteeseen ja tiedon esittämisen järjestykseen. Osa ongelmista korjattiin näkymän uusimpaan versioon. Arvioidusssa versiossa kaikki korkeimman tason arkkitehtuurikuvausten lapsikuvaukset päätöksineen olivat päätökset nimisen luvun alla. Lisäksi korkean tason arkkitehtuurikuvauksiin liittyvät päätökset olivat tämän luvun ensimmäisenä alakohtana ja siis erillään arkkitehtuurikuvauksistaan. Tätä päätökset-lukua pidettiin liian pitkänä muuhun dokumenttiin nähden, joten se jaettiin osiin nostamalla jokainen toisen

tason arkkitehtuurikuvaus omaksi luvukseen. Tietosisällön esittäminen arkkitehtuurikuvauksena päätökset luvussa koettiin ongelmalliseksi. Korkean tason päätösten erottaminen niihin liittyvistä arkkitehtuurikuvauksista omaan kohtaansa koettiin myös hieman häiritseväksi, joten nämä päätökset siirrettiin muiden päätösten tapaan päätökseen liittyvän arkkitehtuurikuvauksen alle. Dokumentin johdantoon haluttiin myös järjestelmän yleiskuvauksen lisäksi kuvaus dokumentin rakenteesta.

Tärkeänä huomiona todettiin myös, että dokumentin automaattinen generointi voi laittaa päätöksiä ja arkkitehtuurikuvauksia lukijan näkökulmasta melko satunnaiseen järjestykseen. Jos dokumentti tehtäisiin käsin, joutuisi dokumentin kirjoittaja miettimään asioiden esitysjärjestystä, jolloin siitä voisi tulla lukijan kannalta parempi kokonaisuus. Osittain tähän liittyen tuli esiin jatkokehitysidea ohjatusta dokumentin generoinnista, jossa käyttäjä pääsisi enemmän vaikuttamaan dokumentin rakenteeseen ja korostamaan tärkeitä asioita omiksi luvuikseen. Arviointiyhteenvedosta todettiin, että se ei ehkä sellaisenaan tarjoa paljoa hyödyllistä lisätietoa, vaan lähinnä osoittaa lukijalle, mistä asioista kuten riskeistä kannattaa etsiä lisätietoa muualta tietämyskannasta.

Tämä arviointi ei ollut kaikilta osin erityisen kattava. Tässä ei tutkittu dokumentin eri parametreilla luotuja versioita, joten niiden hyödyllisyyttä ei ole arvioitu. Dokumenttia on myöhemmin tarkoitus arvioida siten, että IDLE:ä tuntemattomat henkilöt saavat IDLE:en liittyviä tehtäviä, jotka heidän pitää ratkaista dokumentin avulla. Näin selviää onko dokumentti hyödyllinen esimerkiksi ylläpidon tukena. Arvioinnissa tuli myös esiin se, että IDLE on suhteellisen pieni järjestelmä, joten dokumentti ei ehkä toimi yhtä hyvin isolle järjestelmälle. Suoritetun arvioinnin pohjalta voidaan kuitenkin sanoa, että dokumentin idea on toimiva.

8.3 ATAM-raportti

IDLE:lle tehdyn ATAM-arvioinnin lopuksi (katso 8.1) katsottiin myös nopeasti ATAM-raporttia ja sen tarjoamia näkymiä arvioinnissa syntyneisiin tietoihin. Tärkeänä huomiona todettiin, että raportin tarjoamien näkymien osoittamia asioita ei tule hyväksyä sellaisenaan, vaan niitä tulee pohtia ja käyttää päätelmien tekemisen tukena. Riskit-kohdan päätöksiin liittyvien riskien määrän näyttävästä taulukosta nähtiin, että kaksi päätöstä olivat kahdesssa skenaariossa riskejä, jolloin voitaisiin olettaa näiden päätösten olevan tärkeimmät riskit. Järjestelmää tuntevan henkilön mukaan kuitenkin päätös, johon liittyi vain yksi riski, on merkittävämpi riskitekijä. Tätä voidaan ehkä pitää koko ATAM-menetelmän heikkoutena, kun riskien merkittävyyttä ei arvioida. Yleisesti ATAM-raportin osittaisen automaattisen luonnin ja erityisesti valmiiden yhteenvetoa tarjoavien taulukoiden ja näkymien seurauksena voi käydä niin, että ATAM:in tuloksia ei pohdita ja analysoida niin tarkkaan kuin, jos joku joutuisi kirjoittamaan ja koostamaan koko raportin kokonaan esimerkiksi

muistiinpanoista tai muusta ATAM-tilaisuudessa tietojen tallentamiseen käytetystä lähteestä. Toisaalta suurilta osin automaattisesti generoidun ATAM-raportin ehdottomana hyötynä on ajan säästyminen manuaaliselta raportin kokoamiselta. Näkymän ansiosta kaikki ATAM-raportit ovat myös rakenteeltaan yhtenäisiä, jolloin lukija tietää aina, mitä raportista löytyy ja mistä se löytyy.

8.4 Komponentin teknologianäkymä

Komponentin teknologianäkymän arvioinniksi IDLE:n (katso kohta 7.1) autentikoinnista vastaavasta CAuth-luokasta luotiin näkymä. Tästä näkymästä kerättiin palautetta samassa tilaisuudessa, jossa skenaariopohjaista arkkitehtuuridokumenttia arvioitiin. Tarkoituksena oli selvittää olisiko autentikointiluokan teknologianäkymästä IDLE:ä tuntevien henkilöiden mielestä hyötyä komponentin kanssa tekemisissä olevalle henkilölle kuten ylläpitäjälle eli sisältääkö näkymä tarvittavan tiedon ja onko se järkevästi organisoitua.

IDLE:ä tuntevien henkilöiden mielestä näkymässä vaikutti olevan hyvin koottuna komponenttiin liittyvää tietoa. Arvioinnissa näkymän rakennetta pidettiin hieman ongelmallisena ja sitä muutettiinkin lopulliseen versioon siten, että komponentin sisäinen rakenne siirrettiin viimeiseksi, kun aikaisemmin se oli vaatimusten jälkeen. Tärkeiksi komponentista esiteltäviksi asioiksi nostettiin esiin komponentin rajapinta ja vastuut, jolle ei näkymässä ole varsinaisesti omaa paikkaansa, vaan ne tulee sisällön tuottajan kuvata joko komponentin kuvauksessa, tarkennetuissa vaatimuksissa tai päätöksissä. Ongelmallista voi myös olla se, että arkkitehtuuritietämyksen kuvat ja tekstit ovat staattisia ja samoja eri näkymissä eikä niistä voi korostaa tai muuten tuoda esiin juuri tietyn näkymän kannalta oleellista tietoa. Jos komponentti on esimerkiksi osa jotain suurta arkkitehtuurikuvausta, tulee komponentin teknologianäkymään tästä kuvauksesta kaikki sisältö, josta suurin osa ei koske komponenttia ollenkaan.

Tämän arvioinnin laajuus ei ollut kovin suuri. Lisäksi osittain ehkä puutteellinen tietämyskannan sisältö ei antanut näkymästä ja komponentista täyttä kuvaa. Näistä ongelmista huolimatta voidaan todeta, että näkymän idea on toimiva, mutta se vaatii arkkitehtuuritietämyksen tuottajilta sopivaa sisältöä.

9. YHTEENVETO

Tässä työssä toteutettiin internet-selaimella käytettävä arkkitehtuuritietämyskanta käyttämällä toteutusalustana Polarion ALM -ohjelmiston elinkaaren hallintajärjestelmää. Polarion saatiin toimimaan tietämyskantana konfiguroimalla sinne talletettavan tiedon rakenne tietämyskantaa varten kehitetyn arkkitehtuuritietämyksen metamallin mukaiseksi. Metamallin tarkoituksena oli yhdistää perinteinen arkkitehtuuri arkkitehtuuriarvioinneissa syntyvään tietoon ja yleiseen arkkitehtuuritietämykseen.

Polarioniin kuuluvan wikin avulla tietämyskantaan voitiin lisätä omia toimintoja tiedon syöttämiseen ja sen tarkasteluun, joilla täydennettiin Polarionin valmiita ominaisuuksia. Näin toteutettiin yksi tiedonsyöttötyökalu ja kolme näkymää arkkitehtuuritietämykseen. Tiedon syöttötyökaluna toteutettiin ATAM-työkalu, jonka avulla ATAM-arkkitehtuurinarviointimenetelmää sovellettaessa syntyvä tieto voidaaan tallentaa suoraan tietämyskantaan sitä mukaa, kun tietoa syntyy ATAM-prosessin aikana. Kehitettyjä näkymiä olivat puolestaan ATAM-työkaluun liittyvät ATAM-raportti -näkymä arvioinnin tulosten tarkasteluun, komponentin teknologianäkymä tietyn komponentin kanssa tekemisissä olevan henkilön tarpeisiin sekä skenaariopohjainen arkkitehtuuridokumentti -näkymä, joka on räätälöitävä koko arkkitehtuurin kattava dokumentti.

Arkkitehtuuritietämyskannan ja siihen toteutettujen ominaisuuksien toimivuutta testattiin TTY:n ohjelmistotekniikan laitoksella kehitetyn web-pohjaisen IDLE-oppimisympäristön arkkitehtuuritietämyksellä. IDLE:lle suoritettiin arkkitehtuurin arviointi ATAM-työkalua käyttäen, ja arkkitehtuuritietämystä kerättiin lisäksi IDLE:n dokumentaatiosta. Mukana olleilta IDLE:n kehitysprojektin osanottajilta saatiin palautetta työkalusta ja näkymistä. Tietämyskanta todettiin periaatteessa toimivaksi järjestelmäksi, mutta ongelmiakin löydettiin, joista osa on jo korjattu. Löydetyt ongelmat liittyivät lähinnä näkymien rakenteeseen.

Arviointi ei ollut kaikilta osin kovin kattavaa, joten sen perusteella arkkitehtuuritietämyskantaa ei voi kutsua valmiiksi. Sen toiminnallisuus on myös vielä hieman rajoittunutta. Uusia tiedonsyöttötyökaluja ja näkymiä pitäisi siis vielä toteuttaa. Tietämyskantaa tulisi myös testata kokonaisella IDLE:ä suuremmalla järjestelmällä, sillä melko pienikokoisesta IDLE:stä ei syötetty tietämyskantaan kaikkea mahdollista tietoa. Tietämyskannan toimivuus osana ohjelmistoprojektia vaatii myös

lisätutkimusta siitä, miten se sopii osaksi työnkulkua ja millaista ohjeistusta sen tehokas käyttö vaatii. Tietämyskannan web-pohjaisuuden ja toteutuksen luonteen vuoksi tietämyskantaa ei ehkä missään vaiheessa voi sanoa täysin valmiiksi, sillä uusia tiedon syöttötyökaluja ja näkymiä voidaan tehdä helposti tarpeen ilmetessä, ja vanhoja näkymiä ja työkaluja voidaan muokata tai niitä voidaan käyttää pohjana hieman erilaisiin ominaisuuksiin. Koska kuka tahansa riittävät oikeudet omaava henkilö voi periaatteessa muokata näkymiä ja työkaluja, voivat eri käyttäjät myös muokata niitä itselleen sopivimmiksi. Tässä työssä kuvattu jo suoritettu tietämyskannan kehitys antaa kuitenkin näille tulevaisuuden mahdollisuuksille sekä jatkokehitys- ja tutkimussuunnnitelmille hyvät lähtökohdat.

LÄHTEET

[1] ANSI/IEEE 1471-2000. *Recommended Practice for Architecture Description of Software-Intensive Systems.* Institute of Electrical and Electronics Engineers, 2000.

[2] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry. Architectural knowledge and rationale: issues, trends, challenges. *ACM SIGSOFT Software Engineering Notes*, 32(4):41–46, 2007.

[3] J. Bosch. Software architecture: The next step. In *Software Architecture, First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004*, pages 194–199. Springer, 2004.

[4] R. de Boer, R. Farenhorst, P. Lago, H. Van Vliet, V. Clerc, and A. Jansen. Architectural knowledge: Getting to the core. In *Proceedings of the Third International Conference on the Quality of Software Architectures, Medford, Massachusetts, USA, July 12 - July 13, 2007*, pages 197–214. Springer, 2007.

[5] R. Farenhorst and R.C. Boer. Knowledge Management in Software Architecture: State of the Art. In M. Ali Babar, T. Dingsøyr, P. Lago, and H. van Vliet, editors, *Software Architecture Knowledge Management Theory and Practice*, pages 21–38. Springer-Verlag, Berlin Heidelberg, 2009.

[6] Apache Software Foundation. Apache lucene - overview. [Viitattu 01.09.2011]. Saatavissa: http://lucene.apache.org/java/docs/index.html.

[7] Apache Software Foundation. Velocity käyttäjän opas. [Viitattu 19.08.2011]. Saatavissa: http://velocity.apache.org/engine/devel/translations/user-guide_fi.html.

[8] C. Hofmeister, P. Kruchten, R.L. Nord, H. Obbink, A. Ran, and P. America. Generalizing a model of software architecture design from five industrial approaches. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, Pennsylvania, USA, November 6-10, 2005*, pages 77–88, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[9] A. Jansen, P. Avgeriou, and J.S. Van Der Ven. Enriching software architecture documentation. *Journal of Systems and Software*, 82(8):1232–1248, 2009.

[10] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, Pennsylvania, USA, November 6-10, 2005*, pages 109–120, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[11] A. Jansen, J. Bosch, and P. Avgeriou. Documenting after the fact: Recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536–557, 2008.

[12] R. Kazman, M. Klein, and P. Clements. Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, USA, 2000.

[13] P. Kruchten, P. Lago, and H. van Vliet. Building up and reasoning about architectural knowledge. In *Proceedings of the Second International Conference on the Quality of Software Architectures, Västerås, Sweden, June 27-29, 2006*, pages 43–58. Springer, 2006.

[14] Rami Laine. Idle-oppimisympäristön suunnittelu ja toteuttaminen. Diplomityö, Tampereen teknillinen yliopisto, 2008. Saatavissa: http://www.cs.tut.fi/~idle/idle_dipl.pdf.

[15] D.E. Perry and A.L. Wolf. Foundations for the study of software architecture. *Software Engineering Notes*, 17(4):40–52, 1992.

[16] Polarion Software. Polarion alm : Integrated application lifecycle management software. [Viitattu 18.08.2011]. Saatavissa: http://www.polarion.com/products/alm/index.php.

[17] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. Ali Babar. a comparative study of architecture knowledge management tools. *the Journal of Systems and Software*, 83(3):352–370, 2010.

[18] J.S. Van Der Ven, A. Jansen, P. Avgeriou, and D.K. Hammer. Using architectural decisions. In *the Second International Conference on the Quality of Software Architecture (Qosa 2006), Västerås, Sweden, June 27-29, 2006*.

A. VELOCITY-ESIMERKKI: AVOIMET VAAATIMUKSET

Alla on kohdassa 3.3 esitellyn avoimet vaatimukset wiki-sivun lähdekoodi.

```
1 Avoimet vaatimukset

#set( $projektiId = $page.getSpace().getProjectId() )
#set( $vaatimukset = $trackerService.queryWorkItems( "type:requirement AND
                     project.id:$projektiId", "id" ) )

#foreach( $vaatimus in $vaatimukset )
  #if( $vaatimus.getStatus().getId() == "open" )
    * {workitem:$vaatimus.getId()|display=long}
  #end
#end
```