

Helena Halkola

Dokumentaatioprosessin kehittäminen Metropolian tietohallinnon ohjelmointiryhmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma



Tekijä Otsikko Sivumäärä Aika	Helena Halkola Dokumentaatioprosessin kehittäminen Metropolian tietohallinnon ohjelmointityöryhmässä 40 sivua 26.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	
Ohjaaja	Lehtori Olli Hämäläinen
<p>Insinööriyössä oli tavoitteena kehittää Metropolian tietohallinnon ohjelmointityöryhmän dokumentaatioprosessia. Lisäksi tarkoitus oli myös laatia Metropolian käyttäjätietojen hallintajärjestelmästä Ammeesta dokumentaatiota, joka oli aiemmin puutteellista ja vanhentunutta. Työn tarkoituksena olikin kehittää ratkaisuja, joilla voitaisiin ehkäistä dokumentaation vanhentuminen ja huono laatu ja niistä aiheutuvat ongelmat ja riskit.</p> <p>Työssä käsiteltiin erilaisia dokumentaatio-suosituksia liittyen dokumenttien kirjoittamiseen, laatuun ja tarpeeseen. Suositusten lisäksi esiteltiin lyhyesti erilaisia dokumentointityökaluja, joita voidaan hyödyntää ohjelmistoprojektien eri vaiheissa. Näitä suosituksia ja dokumentointityökaluja käytettiin Amme-järjestelmän dokumentoimiseen.</p> <p>Työn aikana laaditun Amme-dokumentaation ansiosta uuden järjestelmän pääkehittäjän perehdyttäminen on helpompaa. Lisäksi järjestelmän jatkokehittäminen helpottuu, kun dokumentaatio on ajan tasalla. Laadittua dokumenttia voidaan jatkossa myös käyttää mallina sille, millaista dokumentaatiota eri projekteista tulisi tehdä.</p> <p>Työn tuloksena työryhmälle luotiin dokumentaatioprosessin kuvaus, joka ohjeistaa tarvittavan dokumentoinnin laatimista projektin alusta aina ylläpitovaiheeseen. Lisäksi määriteltiin, että jatkossa projekteista täytyy laatia vähintään tekninen ja toiminnallinen määrittely.</p>	
Avainsanat	dokumentointi, Wiki, ohjelmistotuotanto, Amme

Author Title Number of Pages Date	Helena Halkola Development of a documentation process for the software team of Metropolia's IT Management 40 pages 26 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	
Instructor	Olli Hämäläinen, Senior Lecturer
<p>The goal of this study was to improve the documentation process for the software team of Metropolia's IT Management. Furthermore, the goal was to also create documentation for Metropolia's Identity Management System called Amme, as the previous documentation was partly insufficient and outdated.</p> <p>Therefore, the purpose for this study was also to develop solutions that would prevent the documentation from becoming outdated and being of poor quality and essentially preventing the problems and risks caused by such matters.</p> <p>The thesis handles various different documentation recommendations that are affiliated with writing a documentation, its quality and the means it has to fulfill. In addition to these, various different documentation tools are briefly introduced in the thesis, which can be used at different stages in a project's development. These recommendations and tools were used to create the documentation for the Amme system.</p> <p>The documentation that was created during this study makes the orientation process easier for the new head developer for the Amme system. In addition, it eases the further development of Amme. The documentation can also be used as guideline in terms of what documentation should be like in a project.</p> <p>As a result, a documentation process description was created for the software team. The process instructs the developers to create and maintain documentation through the life cycle of a project. It was also concluded that in the future at least technical and functional definition documents are required in every project.</p>	
Keywords	documentation, software development, Wiki, Amme

Sisällys

Lyhenteet ja määritelmät

1	Johdanto	1
2	Ohjelmistotuotanto	2
2.1	Ohjelmistotuotannon osa-alueet	2
2.2	Ohjelmistoprojektin elinkaaren vaiheet	3
2.3	Vaihejakomallit	4
3	Dokumentointi	8
3.1	Dokumentointi yleisesti	8
3.2	Dokumentaation puutteen riskit	9
3.3	Dokumentointi ohjelmistoprojekteissa	10
4	Dokumentaatioesityksiä	14
4.1	Dokumentin kirjoittaminen ja laatu	15
4.2	Dokumentaation tarve	16
4.3	Dokumentoinnin ajankohta	17
5	Dokumentointityökaluja	18
5.1	CASE-työkalut	18
5.2	Tehtävienhallintaohjelmistot	19
5.3	Wiki-sivut	20
5.4	Pilvipalvelut ja verkkolevyt	20
5.5	Microsoft Office -tuotteet	21
5.6	Ohjelmakoodin dokumentointityökalut	22
6	Kehityskohteet ohjelmointiryhmän dokumentaatioprosessissa	24
7	Amme-järjestelmän dokumentaation kehittäminen	27
7.1	Tausta	27
7.2	Ammeen toiminta ja rakenne lyhyesti	28
7.3	Ammeen tietokanta	31

7.4	Järjestelmän tutkiminen	31
7.5	Dokumentaation laatiminen	33
7.6	Dokumentin tarkastaminen ja korjailu	34
7.7	Haasteet ja kokemukset	34
7.8	Projektin lopputulos ja opetukset	35
8	Dokumentaatioprosessin kehittäminen	37
8.1	Dokumentaation tarpeen määrittely	37
8.2	Dokumentaation laatimisprosessi	38
8.3	Dokumenttien sijoitus ja ylläpitäminen	39
9	Yhteenveto	40
	Lähteet	41

Lyhenteet ja määritelmät

CASE	<i>Computer-Aided Software Engineering</i> -työkaluja käytetään ohjelmistotuotannon prosesseissa organisoimiseen ja visualisoimiseen.
HTML	<i>Hypertext Markup Language</i> on merkintäkieli, joilla voidaan kuvata www-sivujen sisällön rakennetta.
IEEE	<i>Institute of Electrical and Electronics Engineers</i> on kansainvälinen tekniikan alan järjestö, joka määrittelee erilaisia standardeja.
ISO	<i>International Organization for Standardization</i> on kansainvälinen standardisointijärjestö.
LDAP	<i>Lightweight Directory Access Protocol</i> on verkkoprotokolla, jota käytetään hakemistopalveluissa.
MySQL	MySQL on relaatiotietokantaohjelmisto, joka on nykyisin Oraclen omistuksessa.
RTF	<i>Rich Text Format</i> on yleinen muotoillun tekstin tallennusmuoto.
SQL	<i>Structured Query Language</i> on alun perin IBM:n kehittämä standardoitu kyselykieli relaatiotietokantojen käsittelyä varten.
UML	<i>Unified Modeling Language</i> on standardoitu mallinnuskieli kaavioiden kuvaamiseen esimerkiksi ohjelmistokehitystä varten.
XML	<i>Extensible Markup Language</i> on merkintäkieli, jota käytetään dokumenttien tallentamiseen ja tiedonvälityksessä järjestelmien välillä.

1 Johdanto

Tämän työn aiheena on kehittää Metropolia Ammattikorkeakoulun tietohallinnon ohjelmointityöryhmän dokumentaatioprosessia. Alun perin tarkoitus oli tehdä dokumentaatiota Metropolian käyttäjätietojen hallintajärjestelmästä Ammeesta. Aikaisempi dokumentaatio järjestelmästä oli joko vanhaa tai osittain puutteellista. Koska Amme on tärkeä järjestelmä Metropoliaassa, on tärkeää, että sen dokumentaatio on ajan tasalla.

Puutteellinen ja vanha dokumentaatio Ammeesta ilmensi myös ohjelmointityöryhmän yleistä dokumentointikulttuuria. Amme-järjestelmä ei ollut ainoa, jonka kohdalla dokumentaatio puuttui tai oli vanhentunut. Suurin osa ohjelmoijien omista pienemmistä ohjelmistoprojekteista kärsi dokumentaation puutteesta.

Puutteellinen dokumentaatio saattaa esimerkiksi vaatia kallista työaikaa muilta työtehtäviltä. Tästä syystä tämä työ laajeni käsittelemään dokumentaatioprosessin kehittämistä kyseisessä ohjelmointityöryhmässä. Lukija voi soveltaa työssä esitettyjä ratkaisuehdotuksia dokumentaation ja dokumentaatioprosessin parantamiseen.

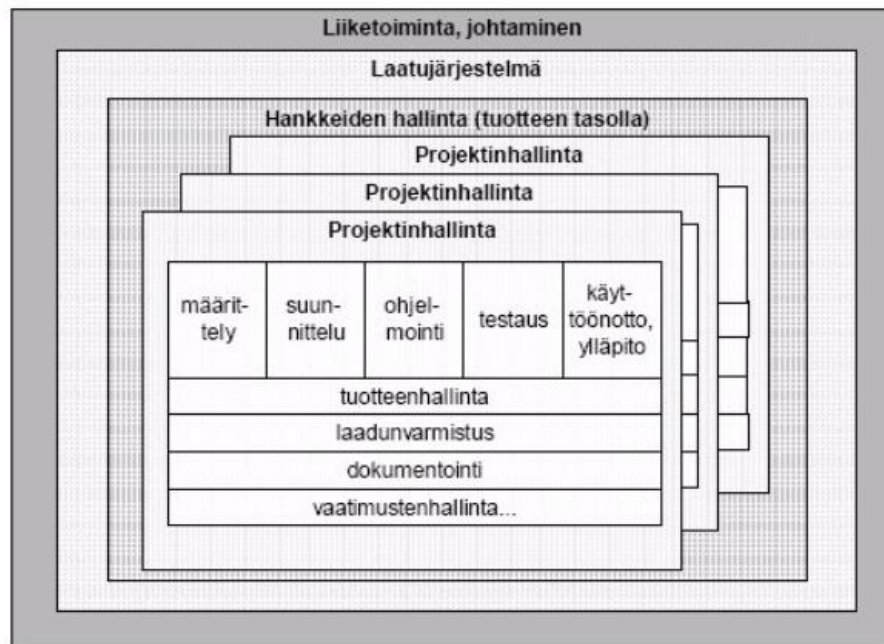
Työssä esitellään ensiksi ohjelmistotuotantoa ja dokumentointia yleisellä tasolla. Tämän jälkeen tarkastellaan dokumentaatiosuosituksia ja -työkaluja. Seuraavaksi esitellään Metropolian tietohallinnon ohjelmointiryhmässä havaittuja ongelmia, minkä jälkeen käsitellään Amme-järjestelmän dokumentaation laatimisprosessia. Lopuksi työssä keskitytään dokumentaatioprosessin kehitysehdotuksiin, joita on valittu suositusten ja kokemusten pohjalta.

Työ rajattiin käsittelemään dokumentaatioprosessin kehittämistä itsenäisissä ja pienissä projekteissa, joille ei ole määritelty varsinaista projektipäällikköä.

2 Ohjelmistotuotanto

2.1 Ohjelmistotuotannon osa-alueet

Ohjelmistotuotanto voidaan jakaa eri osa-alueisiin, jotka näkyvät kuvassa 1. Tässä työssä keskitytään kuvassa näkyvään projektinhallinnan osa-alueeseen. Projektinhallintaa voidaan toteuttaa esimerkiksi ohjelmiston kehitysprosessissa, jossa voidaan erottaa seuraavat vaiheet: määrittely, suunnittelu, ohjelmointi, testaus, käyttöönotto ja ylläpito. Projektiin liittyy koko ohjelman elinkaaren ajan lisäksi tukitoimintoja, joita ovat esimerkiksi dokumentointi, laadunvarmistus ja tuotteenhallinta. [1, s. 35.] Tässä työssä keskitytään tarkastelemaan dokumentointia.



Copyright by Haikala

Kuva 1. Ohjelmistotuotannon osa-alueet

Kuvassa voidaan nähdä myös laatujärjestelmä yhtenä ohjelmistotuotannon osa-alueena. Suurissa ohjelmistoyrityksissä laatujärjestelmä on tärkeä osa, joka ohjaa tuotantoa määrittelemällä yrityksen toimintatavat eli toimintaprosessit. [1, s. 35.]

Laatujärjestelmiä voidaan suunnitella esimerkiksi ISO 9000 -standardin avulla. Standardeja noudattamalla pyritään siihen, että organisaatio toimii laatujärjestelmänsä mukaisesti. Organisaatiolle voidaan myöntää ISO-laatujärjestelmäsertifikaatti, jos organisaation laadunhallinta vastaa standardivaatimuksia. Pienemmissä yrityksissä tällaiset laatujärjestelmät eivät ole välttämättä tarkoituksenmukaisia.

2.2 Ohjelmistoprojektin elinkaaren vaiheet

Ohjelmiston elinkaari (life cycle) käsittää koko sen välisen ajan, kun ohjelmiston kehittäminen alkaa ja kun ohjelmisto poistetaan käytöstä. Vaihejakomallit perustuvat pitkälti elinkaaren vaiheisiin, mutta jokaisessa mallissa vaiheiden välillä eteneminen eroaa toisistaan. [1, s. 36.]

Määrittelyvaihe aloittaa projektin. Esitutkimus voidaan ajatella osaksi määrittelyvaihetta, tai se voidaan ajatella olevan osa tukitoimintaa, jolla hallitaan vaatimuksia. Esitutkimus sisältää joka tapauksessa asiakkaan antamat vaatimukset järjestelmälle. Niiden pohjalta tehdään toiminnalliset vaatimukset, jotka määrittelevät toteutettavan järjestelmän. Muita käytettäviä synonyymejä toiminnallisille vaatimuksille on järjestelmävaatimukset, ominaisuudet tai ohjelmistovaatimukset. Määrittelyvaiheen tuloksena luodaan **toiminnallinen määrittelydokumentti**. [1, s. 39.] Toiminnalliseen määrittelydokumenttiin palataan tarkemmin seuraavassa luvussa.

Suunnitteluvaiheessa suunnitellaan järjestelmän toimintoja, joita määriteltiin toiminnallisessa määrittelydokumentaatioissa. Suunnitteluvaiheessa vastataan kysymykseen, ”miten” järjestelmä suorittaa sille määritellyt toiminnot. [1, s. 40.] Tämän vaiheen jälkeen saadaan tulokseksi **tekninen määrittelydokumentaatio**, johon palataan myöhemmin seuraavassa luvussa.

Ohjelmointivaihe (toteutus) koostuu ohjelmakoodin tekemisestä. Tämä vaihe kestää niin kauan, kunnes koodi kääntyy ensimmäistä kertaa virheettömästi. Tämän jälkeen ollaan valmiita etenemään tärkeään testausvaiheeseen. [1, s. 40.]

Testauksessa yritetään etsiä virheitä kehitetystä ohjelmistosta. Testausta toteutetaan usein monella tasolla, jolloin testaaminen voidaan jakaa kolmeen osaan: moduulitestaukseen, integrointitestaukseen ja järjestelmätestaukseen. Moduulitestauksessa testataan yksittäisten moduulien toimintaa. Integrointitestauksessa testataan moduulien yhteistoimintaa. Järjestelmätestauksessa keskitytään testaamaan koko järjestelmää kokonaisuudessaan. [1, s. 40.]

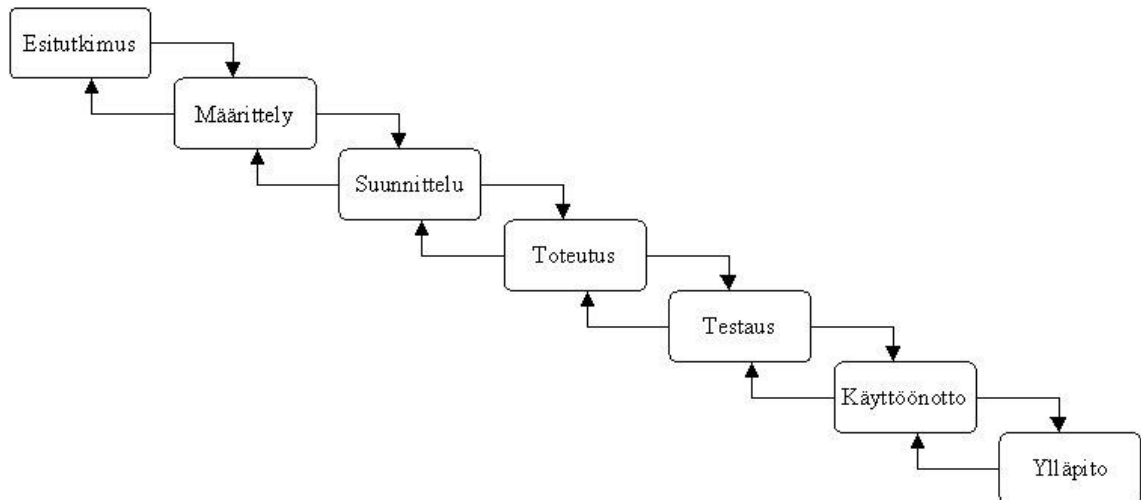
Testausvaiheen jälkeen järjestelmä otetaan käyttöön, ja järjestelmä siirtyy **ylläpitovaiheeseen**. Tässä vaiheessa korjataan ilmenneitä virheitä ja muutetaan järjestelmää uusien vaatimusten ilmaantuessa. [1, s. 40.]

2.3 Vaihejakomallit

Ohjelmistoprojektin kehitystyössä edetään tyypillisesti vaiheittain. Varsinkin laajat ja monimutkaiset projektit tarvitsevat ositusta. Ohjelmistoprojekteja voidaan toteuttaa eri vaihejakomalleilla, jotka tarjoavat erilaisia ratkaisuja projektien hallintaan.

Vesiputousmalli

Vesiputousmalli on perinteisimpiä ja vanhimpia prosesseja ohjelmistotuotannossa. Tässä mallissa suunnittelu- ja toteutusprosessi etenee vaiheesta seuraavaan järjestyksessä. Kuvassa 2 nähdään vesiputousmallin vaiheet. Tarkoitus on, että edelliseen vaiheeseen ei palata missään vaiheessa. Tämä onkin yksi vesiputousmallin haaste, sillä usein tuotantoprosessi on käytännössä iteratiivista, eli vaiheita voidaan toistaa monta kertaa.



Kuva 2. Vesiputousmallin vaiheet

Dokumentit ovat tärkeitä tuotoksia vesiputousmallissa. Jokaisesta vaiheesta on tarkoitus tuottaa dokumentti tai dokumentteja, jotka pohjustavat seuraavaa vaihetta. Esimerkiksi vaatimusanalyysi toimii pohjana toiminnalliselle määrittelylle, sillä määrittely tehdään vaatimusanalyysin perusteella. On myöskin tarkoituksenmukaista, että edellisen vaiheen dokumenttia ei muokata enää, kun se on jo tehty. Dokumentti siis ikään kuin jäädytetään. Tästä syystä on tärkeää, että vaiheet saatetaan loppuun huolellisesti, ennen kuin siirrytään seuraavaan. [2]

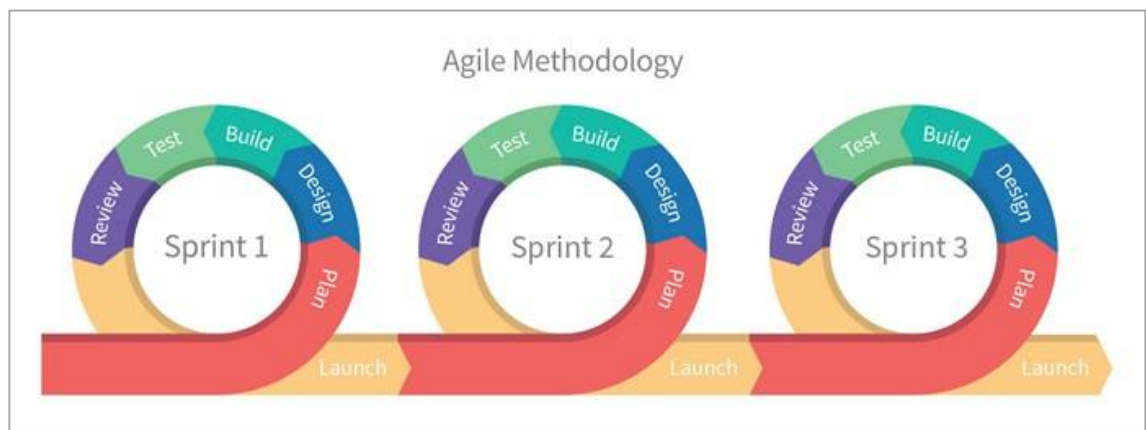
Huolellisuus on yksi vesiputousmallin eduista. Jos ohjelmisto suunnitellaan aluksi tarkasti, se voi johtaa merkittäviin säästöihin projektin myöhemmissä vaiheissa. Tutkimusten mukaan varhaisessa vaiheessa löydetty virhe on halvempi korjata aikaisemmin kuin myöhemmin. [3] Vesiputousmallin vankka dokumenttikeskeisyys on myös tärkeä etu. Uuden työntekijän on helppo liittyä projektiin, jos kaikki aiemmat dokumentit ovat huolellisesti tehtyjä. Vastaavasti avainhenkilön menettäminen ei aiheuta välttämättä liian suurta riskiä, jos kaikki tieto on dokumentoitu hyvin.

Vesiputousmallissa on myös ongelmia. Koska malli on ehdoton vaiheiden etenemisen suhteen, se ei sovellu välttämättä projekteihin, joissa iterointi tuntuu luonnolliselta toimintatavalta. Asiakasprojekteissa asiakas ei aina välttämättä osaa kerralla määrittellä vaatimuksia projektille. Tällöin halutaan usein kokeilla jonkinlaista prototyyppiä. On myös tavallista, että vaatimukset voivat muuttua kesken projektin asiakkaan tahdosta. Vesiputousmallin kannalta aiemmin tehty vaatimusmäärittely täytyisi tehdä uudelleen.

Vesiputousmalli soveltuneekin parhaiten pienemmille projekteille, joissa vaatimusten määrittely on selkeää ja helppoa, eikä niihin kohdistu todennäköisesti muutoksia. Lisäksi vesiputousmallista on tehty muokattuja versioita, jotka jättävät sijaa myös maltilliselle iteroimiselle.

Ketterät menetelmät

Ketteriä menetelmiä ovat esimerkiksi Scrum ja Extreme Programming (XP). Näille ohjelmistokehityksen menetelmille yhteistä on toimivaan ohjelmistoon panostaminen viestimällä suoraan ja nopealla reagoinnilla muutoksiin. Ohjelmistokehitys jaetaan lyhyisiin 1-4 viikon kestäviin iteraatioihin, jotka ovat kuin pieniä ohjelmistoprojekteja sisältäen suunnittelun, määrittelyn, toteutuksen, testauksen ja dokumentoinnin. Kuvassa 3 nähdään esimerkki Agile-menetelmien vaihejakomallista.



Kuva 3. Agile-menetelmien vaihejakomalli. (<http://www.screenmedia.co.uk/blog/2014/08/what-is-agile-development-a-brief-introduction/>)

Kuva 3 havainnollistaa, että ketterissä menetelmissä käydään samoja vaiheita läpi kuin esimerkiksi vesiputousmallissa. Siinä missä vesiputousmallissa käydään jokainen vaihe vain kerran läpi, ketterissä menetelmissä vaiheet toistetaan, sillä kaikki vaiheet käydään läpi jokaisessa projektin ”pyrähdyksessä” (sprint).

Ketterät menetelmät pohjautuvat Agile Manifesto -julistukseen, joka määrittelee ketterien menetelmien perusajatuksen. [6]

Manifestin sisältö on seuraava:

- Yksilöt ja vuorovaikutus ovat tärkeämpiä kuin prosessit ja työkalut.
- Toimiva sovellus on tärkeämpi kuin kokonaisvaltainen dokumentaatio.
- Asiakasyhteistyö on tärkeämpää kuin sopimusneuvottelut.
- Muutokseen reagoiminen on tärkeämpää kuin suunnitelman noudattaminen.

Ketterät menetelmät suosivat päivittäisiä lyhyitä tilannepalavereja, johon kaikki tiimin jäsenet osallistuvat. Palaverissa käydään läpi tyypillisesti kysymyksiä kuten: **Mitä teit edellisenä päivänä? Mitä aiot tehdä tänään? Mitkä seikat haittaavat työskentelyäsi?** [5]

Prototyyppimalli

Prototyyppimallissa tehdään heti aluksi yksinkertainen malli ohjelmasta asiakkaalle nähtäväksi, jotta asiakas hahmottaisi helpommin, mitä lisävaatimuksia ohjelmalle voidaan asettaa. Asiakkaalta tulleeseen palautteeseen voidaan näin reagoida nopeasti.

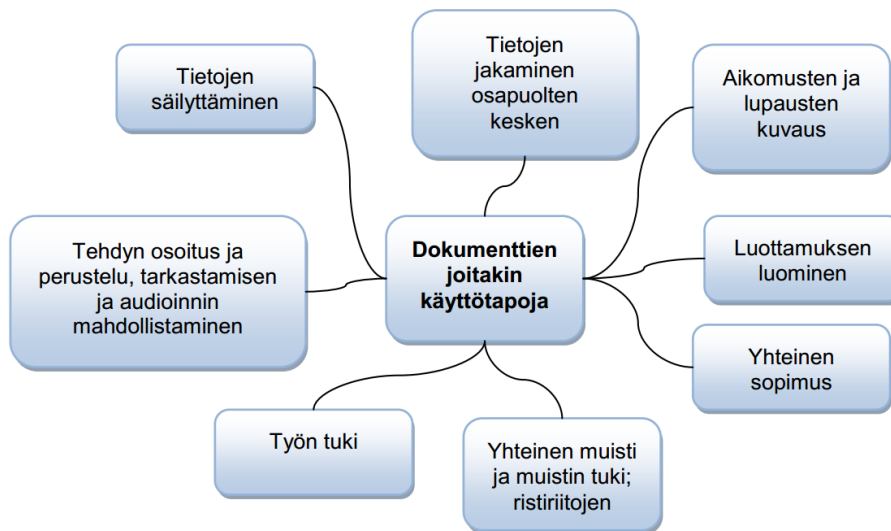
Prototyyppimalleja on kahta tyyppiä: *hylättävät* ja *kehittyvät* prototyypit. Hylättävää prototyyppiä ei ole tarkoitus viedä tuotantoon, vaan sitä käytetään vain eräänlaisena apuvälineenä, kun varsinaista sovellusta aletaan kehittää. Hylättävää prototyyppiä ei saisi koskaan käyttää lopullisen sovelluksen ytimenä, sillä se voi olla huonosti suunniteltu. Kehittyvää prototyyppiä sen sijaan jalostetaan tuotantoversioksi asti. Prototyyppiä kehitetään asiakkaan palautteiden pohjalta niin kauan, kunnes se vastaa asiakkaan tarpeita. [6, s. 434.]

3 Dokumentointi

3.1 Dokumentointi yleisesti

Tässä luvussa tarkastellaan, mitä merkitystä ja tarkoituksia dokumentoinnilla on.

Dokumentointi tarkoittaa jonkin asian kuvaamista useimmiten esimerkiksi kirjoitetussa muodossa. Dokumentteja tehdään eri tarkoituksiin. Kuvassa 5 on esitetty joitakin dokumenttien käyttötarkoituksia.



Kuva 4. Dokumentoinnin käyttötarkoituksia (Matti Vuori: 125 pointtia dokumentoinnista)

Dokumentteja käytetään esimerkiksi tietojen säilyttämiseen, jotta voidaan esimerkiksi tarkistaa myöhemmin, miten jokin asia on tehty. Näin ne toimivat myös ihmisen muistin jatkeena ja toimivat hyvin seurannan välineenä. Voidaan seurata, mitä on tehty ja mitä on vielä tekemättä. [7]

Dokumentin luomisessa on tärkeää, että tiedot tallennetaan luotettavasti. Näin niitä voidaan jakaa eri osapuolille sekä sidosryhmille. Kun tieto kirjoitetaan ylös, ei jätetä varaa henkilöiden eriäville muistikuville, ja niistä syntyneille väärinkäsityksille. Tuotetusta dokumentista voi myös aina varmistaa asioita, mikäli epäselvyyksiä ilmenee tai jos muistiin ei voi luottaa. Kaikkien henkilöiden muisti ei kuitenkaan toimi samalla tavalla, joten on hyvä, että olemassa on objektiivinen asiakirja. [7]

On melko harvoja tilanteita, jolloin dokumentaatiota ei tarvita. Tällaisissa tilanteissa kaikki osapuolet ymmärtävät asian samalla tavalla. Jos asia on ajankohtainen vain hetken eikä toistuvuutta tule olemaan jatkossa, ei ehkä ole välttämätöntä luoda dokumentaatiota. [7]

Itsestään selvät ja yksinkertaiset asiat eivät välttämättä siis vaadi dokumentaatiota, mutta tämän dokumentaation tarpeettomuuden määrittely ei ole välttämättä helppoa. Uusien osapuolien ilmaantuessa on kuitenkin aina mahdollisuus, että aiemmin kaikille itsestäänselvyys ei olekaan sitä uudelle henkilölle. Asian opettaminen kyseiselle henkilölle voi olla nopeaa ja helppoa, mutta siitä voi olla hyvä varmuuden vuoksi tehdä pieni dokumentaatio, sillä vastaavanlaisia tilanteita tulee jatkossakin. Lisäksi yksinkertaisen asian dokumentoiminen ei oletettavasti vaadi paljon aikaa.

3.2 Dokumentaation puutteen riskit

Dokumentaation puute on yleensä haitallista. Siitä voi koitua suoria ja välillisiä haittoja. Yrityksessä saattaa olla esimerkiksi järjestelmä, jonka tuntee vain yksi henkilö. Hän on siis avainhenkilö tässä suhteessa. Henkilö saattaa vaihtaa työnantajaa, joutua pitkälle sairauslomalle tai vaikka menehtyä äkillisesti. On siis riskialtista jättää tieto vain yhden ihmisen varaan. Varsinkin, jos tietoa ei ole dokumentoitu millään tasolla. On myös tavallista, että ohjelmistokehittäjä unohtaa vuosien saatossa, kuinka hänen ohjelmakoodinsa toimiikaan.

Aikaa on saatettu säästää aiemmin, kun dokumentaatiota ei ole tehty. Dokumentaation suorittaminen jälkikäteen vie luultavasti enemmän aikaa, sillä järjestelmän toiminnan tutkiminen ja asioiden palauttaminen mieleen on raskaampaa, kuin jos dokumentaation olisi laatinut asioiden ollessa vielä tuoreessa muistissa. Pahimmissa tapauksissa dokumentoinnin tekee joku muu kuin se henkilö, joka siitä tietää eniten. Tämän seurauksena työaikaa menee oletettavasti vielä enemmän.

Dokumenttien puute vaikuttaa myös vanhoihin ja uusiin työntekijöihin. Esimerkiksi jos uusi työntekijä aloittaa ohjelmointitehtävien parissa, hänen täytyy ehkä selvittää itse perinpohjaisesti nykyisen järjestelmän ja ohjelmakoodin toiminta. Tämä vaikuttaa työsken-

telytehokkuuteen todennäköisesti vähentävästi. Vanhat työntekijät saattavat kärsiä dokumentaation puutteesta, kun vanhaan järjestelmään halutaan tehdä muutoksia. Muutoksia on hankala tehdä, jos järjestelmästä ei tiedetä mitään entuudestaan. Järjestelmän tutkimiseen kuuluu arvokasta työaikaa.

3.3 Dokumentointi ohjelmistoprojekteissa

Dokumenttien tuottaminen on osa ohjelmointityötä, mutta käytännössä laadukas dokumentointi on useimmiten heikoin lenkki ohjelmistokehityksessä. Ohjelmistoprojekteissa aikataulut kiristyvät herkästi, jolloin dokumentointi saatetaan tehdä vasta jälkikäteen. Tällöin kaikkia suunnitteluvaiheessa tehtyjä määrittelyjä ja muita asioita ei välttämättä kirjata tarpeeksi tarkasti. [1, s. 70.]

Suosittelujen dokumenttien määrä vaihtelee projektin suuruudesta riippuen. Pienissä projekteissa voidaan selvittää vähäiselläkin dokumentaatiolla. Pienistäkin projekteista olisi hyvä olla esimerkiksi määrittely- ja suunnitteludokumentteja, sillä niiden täydellinen puute voi johtaa koko ohjelman uudelleenkodeeraamiseen tulevaisuudessa. Minimidokumentaatioon tulisi kuulua projektisuunnitelma, määrittelydokumentti (toiminnallinen määrittely), suunnitteludokumentti (tekninen määrittely) ja testaussuunnitelma. [1, s. 51.]

Seuraavaksi esitetään kuitenkin tarkemmin vain toiminnallinen ja tekninen määrittelydokumentaatio, jotka ovat omasta mielestäni keskeisemmät dokumentaatiot varsinkin pienemmissä ohjelmistokehitysprojekteissa. Niiden pohjalta voidaan tuottaa ylläpitodokumentaatiota.

Ylläpitodokumentaatio tehdään projektin päätteeksi. Projektin aikana tuotetut tuotedokumentit muutetaan osaksi tuotekohtaista tuotedokumentaatiota eli ylläpitodokumentaatiota. Lopputuloksena voi olla neljä erilaista dokumenttityyppiä: käyttöohje, asennus- ja operointiohje, koulutusmateriaali sekä tekninen dokumentaatio. Tekniseen dokumentaatioon kootaan *toiminnallisen* ja *teknisen määrittelyn* lisäksi testaukseen ja tuotteenhallintaan liittyvät dokumentit. [1, s. 75.]

Toiminnallinen määrittely

Toiminnallisella määrittelydokumentilla on tarkoitus selvittää ohjelmistolle asetettavat vaatimukset ja toiminnalliset tavoitteet. Toiminnallinen määrittely syntyy usein asiakasvaatimusten, eli esitutkimuksen, pohjalta. Määrittelyllä voidaan tarkoittaa koko järjestelmän (laitteisto, ohjelmisto, järjestelmäsuunnittelu) määrittelyä, ohjelmiston määrittelyä tai ohjelmiston osan määrittelyä. Määrittelyvaiheen jälkeen saadaan tulodokumentiksi toiminnallinen määrittelydokumentti. Lisäksi määrittelyvaihe toimii pohjana alustavalle käyttöohjeelle. [1, s. 79.]

<ul style="list-style-type: none"> ▲ 1 JOHDANTO <ul style="list-style-type: none"> 1.1 Tarkoitus ja kattavuus 1.2 Tuote 1.3 Määritelmät, termit ja lyhenteet 1.4 Viitteet ja muut liittyvät dokumentit 1.5 Yleiskatsaus dokumenttiin ▲ 2 YLEISKUVAUS <ul style="list-style-type: none"> 2.1 Ympäristö 2.2 Toiminta 2.3 Käyttäjät 2.4 Yleiset rajoitteet 2.5 Oletukset ja riippuvuudet ▲ 3 TIEDOT JA TIETOKANNAT <ul style="list-style-type: none"> 3.1 Tietokannan yleiskuvaus 3.2 Taulun 1 kuvaus 3.3 Taulun 2 kuvaus ▲ 4 TOIMINNOT <ul style="list-style-type: none"> 4.1 Yleistä 4.2 Järjestelmän toiminnot 	<ul style="list-style-type: none"> ▲ 5 ULKOISET LIITTYMÄT <ul style="list-style-type: none"> 5.1 Laitteistoliittymät 5.2 Ohjelmistoliittymät 5.3 Tietoliikenneliittymät ▲ 6 MUUT OMINAISUUDET <ul style="list-style-type: none"> 6.1 Suorituskyky ja vasteajat 6.2 Käytettävyys, toipuminen, turvallisuus, suojaukset 6.3 Ylläpidettävyys 6.4 Siirrettävyys ja yhteensopivuus 6.5 Käyttäjän ylläpitotoimet ▲ 7 SUUNNITTELURAJOITTEET <ul style="list-style-type: none"> 7.1 Standardit ja suositukset 7.2 Laitteistorajoitteet 7.3 Ohjelmistorajoitteet 7.4 Muut rajoitteet 8 HYLÄTYT RATKAISUVAIHTOEHDOT 9 JATKOKEHITYSAJATUKSIA 10 VIELÄ AVOIMET ASIAT
---	---

Kuva 5. Toiminnallisen määrittelydokumentin runko.

Kuvassa 6 esitetään esimerkki toiminnallisen määrittelyn sisältörungosta. Ensimmäisen luvun on tarkoitus antaa lukijalle nopea ymmärrys dokumentista ja mitä hänen kannattaa siitä lukea. Ensimmäiseksi kerrotaan tuotteen tarkoituksesta ja käytettävistä lyhenteistä. Lyhenteiden ja termien määrittely on tärkeää, jotta lukija ymmärtää mitä esimerkiksi tietyllä termillä tarkoitetaan kyseisessä dokumentissa. Lisäksi luvussa kerrotaan, mitä

muita asiaan liittyviä dokumentteja on olemassa. Luvun lopussa selitetään dokumentin rakennetta. [1, s. 80.]

Toisessa luvussa selitetään järjestelmän toiminnan yleiskuva. Luvussa kuvataan järjestelmän liittymät ja ympäristö. Myös toiminta kuvataan yleisellä tasolla. Kohdassa 2.3 kerrotaan, millaisia käyttäjiä järjestelmällä on. Yleisten rajoitteiden kuvaamisella tarkoitetaan esimerkiksi lainsäädäntöön tai toteutustyökaluihin liittyviä rajoitteita. Viimeisessä kohdassa esitetään oletukset, jotka vaikuttavat oleellisesti järjestelmän toimintaan. Oletukset voivat koskea esimerkiksi käytettävissä olevan laitteiston tehoa. [1, s. 80.]

Luvussa 3 kuvataan järjestelmän tietokanta, sen tietosisältö sekä järjestelmän käsittelemät tiedot. Luku 4 keskittyy kuvaamaan järjestelmän toimintaa toiminto kerrallaan. Jokaisesta toiminnosta esitetään sen tarkoitus, tarvittut syötteen, miten käsittely tapahtuu ja mitä saadaan tulosteeksi. [1, s. 80.]

Ulkoiset liittymät esitetään luvussa 5. Esimerkiksi käyttöliittymä voidaan kuvata tässä joko tarkasti tai yleisemmällä tasolla. Tarkka kuvaus voidaan tehdä myös suunnitteluvaiheessa eli teknisessä määrittelyssä. Luvussa 6 kuvataan järjestelmän ominaisuudet, jotka eivät ole toiminnallisia ominaisuuksia. Luku 7 keskittyy rajoitteiden kuvaamiseen. Viimeiset luvut käsittelevät hylättyjä ratkaisuvaihtoehtoja, jatkokehitysajatuksia ja avoimeksi jääneitä asioita. [1, s. 81.]

Tekninen määrittely

Teknisen määrittelydokumentin tarkoituksena on esittää järjestelmän tekninen toteutus. Tekninen määrittelydokumentti on suunnittelutyön tulosta. Suunnittelu voidaan jakaa kahteen tasoon: *arkkitehtuurisuunnitteluun* ja *moduulisuunnitteluun*. Arkkitehtuurisuunnittelussa järjestelmä jaetaan moduuleihin ja niiden rajapinnat määritellään. Moduulisuunnittelussa suunnitellaan puolestaan moduulien sisäinen rakenne. [1, s. 81.]

Arkkitehtuurisuunnittelussa on kyse eri osien välisestä työnjaon suunnittelusta. Tarkoitus on pyrkiä suunnittelemaan riippumattomia moduuleja. Tällöin niitä on mahdollista käyttää uudelleen muualla. Lisäksi muutosten tekeminen helpottuu, kun riippuvuuksia ei ole. [1, s. 82.]

- ▲ 1. JOHDANTO
 - 1.1 Tarkoitus ja kattavuus
 - 1.2 Tuote ja ympäristö
 - 1.3 Määritelmät, merkintätavat ja lyhenteet
 - 1.4 Viitteet
 - 1.5 Yleiskatsaus dokumenttiin
- ▲ 2. JÄRJESTELMÄN YLEISKUVAUS
 - 2.1 Sovellusalueen kuvaus
 - 2.2 Järjestelmän liittyminen ympäristöönsä
 - 2.3 Laitteistoympäristö
 - 2.4 Ohjelmistoympäristö
 - 2.5 Toteutuksen keskeiset reunaehdot
 - 2.6 Sopimukset ja standardit
- ▲ 3. ARKKITEHTUURIN KUVAUS
 - 3.1 Suunnitteluperiaatteet
 - 3.2 Ohjelmistoarkkitehtuuri, moduulit ja prosessit
 - 3.3 Tietokanta-arkkitehtuuri
 - 3.4 Virhe- ja poikkeusmenettelyt
- ▲ 4. MODUULI /LUOKKA/PROSESSI-KUVAUKSET
 - ▲ 4.1 Moduuli X (kustakin moduulista oma alakohtansa)
 - 4.1.1 Yleiskuvaus
 - 4.1.2 Rajapinta yleisesti
 - 4.1.3 Rajapintafunktiot
 - 4.1.4 Moduulin toteutus
 - 4.1.5 Virhekäsittely
- 5. VALMISOSAT JA ERITYISET TEKNISET RATKAISUT
- 6. HYLÄTYT RATKAISUVAIHTOEHDOT
- 7. JATKOKEHITYSAJATUKSIA
- 8. VIELÄ AVOIMET ASIAT

Kuva 6. Teknisen määrittelydokumentin runko.

Luku 1 sisältää samanlaisen kuvailun kuin toiminnallisessa määrittelyssä. Järjestelmän ympäristö käsitellään luvussa 2. Luvussa 3 esitetään valitut arkkitehtuuriratkaisut koskien tietokantaa ja ohjelmistoa. Neljännessä luvussa kuvataan kunkin moduulin tarkemat tiedot. Viidennessä luvussa selostetaan mahdollisia käytettyjä ulkoisia komponentteja, ja esitetään erityisiä tai poikkeavia teknisiä ratkaisuja. [1, s. 83.]

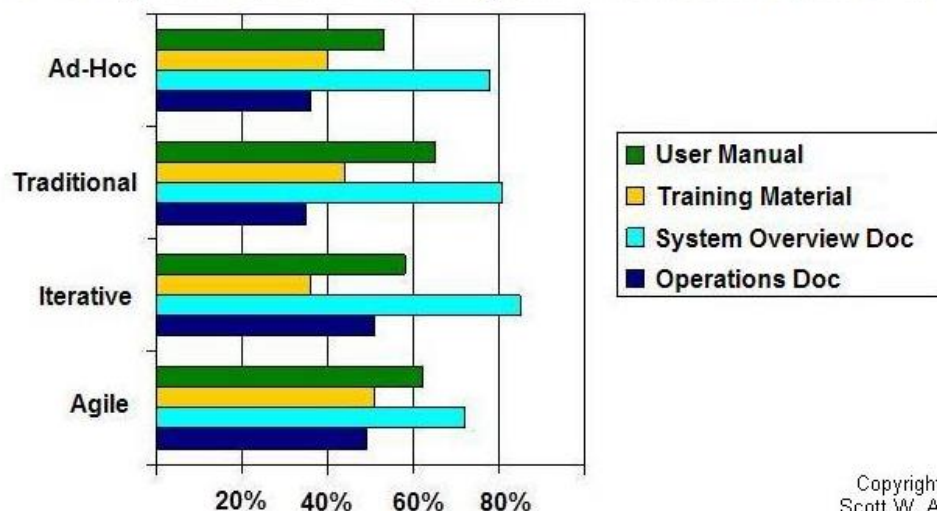
4 Dokumentaatio-suosituksia

Tässä luvussa perehdytään dokumentoimisen parhaisiin käytäntöihin. Dokumenttikeskeinen vesiputousmalli määrittelee pitkälti, mitä dokumentteja tulee milloinkin luoda. Tässä luvussa käydään läpi muun muassa Scott Amblerin [8] esittämiä parhaita käytäntöjä ketterässä dokumentoinnissa. Usein projektinhallinnassa ei käytetä mustavalkoisesti vain yhtä vaihejakomallia, vaan se voi olla hybridi kahdesta mallista. Siksi ketterän dokumentoinnin suosituksia voidaan soveltaa dokumentoimiseen muissakin projektimalleissa.

Vastoin yleisiä uskomuksia, ketteriä menetelmiä noudattavat ohjelmoijaryhmät eivät tuota juurikaan yhtään vähemmän dokumentteja kuin perinteisiä menetelmiä käyttävät ryhmät (kuva 8). Vaikka ketterät menetelmät pitävät toimivaa ohjelmistoa tärkeämpänä kuin dokumentteja, nämä eivät kuitenkaan sulje toisiaan pois. Toimivan ohjelmiston rinnalla voidaan myös toteuttaa laadukkaita dokumentteja. [8]

Ketterissä menetelmissä jää kuitenkin sijaan tulkintaan dokumenttien tärkeydestä. Tällöin on mahdollista, että henkilöt, jotka eivät erityisesti välitä dokumenttien tekemisestä, perustelevat dokumentoimattomuutta ketterien menetelmien periaatteilla.

Percentage of Teams Creating Deliverable Documentation



Kuva 7. Eri menetelmiä käyttävien ryhmien tuottamat dokumenttityypit ja niiden määrät

4.1 Dokumentin kirjoittaminen ja laatu

Amblerin mukaan testiraporttien käyttäminen dokumentaatioissa on kannattavampaa kuin, että asioita vain kirjoittaisi dokumenttiin staattisesti. Testiraportit tuovat kahdenlaista etua dokumenttiin: niillä voidaan määritellä vaatimuksia, ja ne vahvistavat tehdyt toteutukset. [8]

Ketterissä menetelmissä dokumenttien tekeminen sijoitetaan usein projektin loppupäähän, jolloin voidaan keskittyä tehtyjen ja pysyvien päätöksiä dokumentoimiseen. Tämä ajatusmalli ei kuitenkaan tarkoita sitä, että dokumentaatiota ei voisi tehdä koko projektin ajan. On hyvä kirjoittaa projektin aikana tehtyjä päätöksiä ylös lopullisia dokumentteja varten.

Kaikkea dokumentaatiota ei tarvitse tehdä käsin. On olemassa työkaluja, joilla voidaan tehdä takaisinmallinnuksia (*reverse-engineering*) esimerkiksi kirjoitetusta koodista tai tietokannasta. Takaisinmallinnus voi tuottaa esimerkiksi graafisen esityksen tietokannan taulujen suhteista. Tällaisten dokumenttien tuottaminen automaattisesti säästää aikaa ja siten myös rahaa. [8]

Myös järjestelmän lähdekoodin dokumentoiminen on tärkeää. Esimerkiksi tilanteessa, jossa ajantasaista dokumentaatiota ei ole saatavilla, lähdekoodi on ainut luotettava lähde, josta toiminnallisuutta voidaan lähteä selvittämään. Jos koodi on vaikeaselkoista, eikä sitä ole dokumentoitu, voi järjestelmän ymmärtäminen viedä paljon aikaa. Tarkoitus ei ole kuitenkaan täyttää koodia kommentteilla, sillä kommentitkin saattavat vanhentua koodin mukana.

lhanteellisesti koodi olisi kirjoitettu itseselittävään muotoon, jolloin luokilla, muuttujilla ja metodeilla on kuvaavat nimet. Kommentteja voidaan lisätä kohtiin, jotka ovat vaikeaselkoisia. Kommenttien tarkoitus olisi kertoa, miksi jokin asia tehdään eikä miten asia tehdään. Koska koodi voi muuttua, täytyy kommenttiakin muuttaa. Jos kommentti on miksi-muodossa, sitä ei tarvitse muuttaa yhtä herkästi. [8]

Vaikka dokumentaatio onkin tärkeä osa ohjelmistotuotantoa, sen ei tulisi viedä liikaa aikaa muusta ohjelmointityöstä. Sopiva dokumentaatioon käytetty aika on noin 10 prosenttia projektiin käytettävästä työajasta. [9]

Dokumentaation puute on todettu haitalliseksi. Toisaalta liiallinen dokumentointi on myös huono asia. Ensinnäkin dokumentaatioon on tällöin kulunut ehkä paljon työaikaa, mutta myös dokumenttien ylläpito hankaloituu, jos dokumenttimateriaalia on paljon. On tarkoituksenmukaista tehdä dokumentteja, jotka sisältävät kaiken tarpeellisen tiedon, eikä mitään enempää tai vähempää. [8] Esimerkiksi kuvien käyttö dokumenteissa on tehokasta ja parantaa dokumentin laatua. Kuviin voidaan pakata runsaasti informaatiota.

Yksi suositus on myös, että suuremmat dokumentit koostetaan pienemmistä dokumenteista. Dokumentaatio koostuu tällöin ikään kuin pienistä toisistaan riippumattomista moduuleista, jolloin dokumentit eivät sisällä toistoa. Wikiä voidaan käyttää tällaiseen dokumenttien osittamiseen. [8] Wikiin palataan seuraavassa pääluvussa.

4.2 Dokumentaation tarve

On tärkeää määritellä, mitä dokumentoidaan. Jokainen projekti on erilainen, jolloin yksi malli ei välttämättä käy kaikille projekteille. Asia kannattaa dokumentoida, jos siitä on hyötyä projektille. Hyöty voi näkyä lyhyellä tai pitkällä aikavälillä, ja hyöty voi myös vaikuttaa suoraan tai epäsuoraan. Lisäksi jonkun osapuolen tulisi päättää, kuinka paljon dokumentaatioon sijoitetaan. Tärkeää on, että dokumentista saatavan hyödyn tulisi olla suurempi kuin siihen käytetty aika, joka menee sen laatimiseen ja ylläpitämiseen. Tarpeen tulisi olla ensisijaisesti tärkein syy dokumentaation laatimiseen, eikä esimerkiksi halun. [8]

Dokumentointitarpeiden määrittely kannattaa aloittaa miettimällä kohdeyleisöä, jolle dokumentti on ensisijaisesti tarkoitettu. Esimerkiksi asiakasprojekteissa on hyvä kysyä asiakkaan tarpeista ja minimivaatimuksista koskien dokumentaatiota. [8]

Dokumentin tekeminen tulisi olla samanlainen vaatimus kuin esimerkiksi jonkin ohjelmistokomponentin toteuttaminen. Sitä tulisi käsitellä kuten muitakin työtehtäviä, eli sille voidaan asettaa prioriteetti ja lisätä työjonoon muiden tehtävien lisäksi.

4.3 Dokumentoinnin ajankohta

Ihanteellisesti dokumentaatiota tehdään koko projektin aikana, mutta on tavallista, että dokumentointia tehdään projektin loppupuolella. Huolimatta dokumentoimisen ajankohdasta olisi hyvä dokumentoida iteroiden. Eli käytännössä kannattaa kirjoittaa vähän kerrallaan, pyytää palautetta ja parantaa dokumenttia annetun palautteen perusteella. Dokumentointia ei siis kannata hoitaa yhdellä kertaa.

Vaikka dokumentointi on myös kommunikointia, kaikkea kommunikointia ei kannata jättää pelkästään dokumentaation varaan. Esimerkiksi kasvotusten käyty keskustelu on tehokasta kommunikointia, mutta sitä kannattaa tukea dokumentoimisella. Dokumentti toimii käydyn keskustelun luotettavana tallennusvälineenä, sillä ihmisen muistiin ei voida aina luottaa. Lisäksi ihmiset muistavat asioita eri tavalla. [5]

Ambler kehottaa, että dokumentteja tulisi päivittää vain, kun sille on erityinen tarve. Ei ole välttämättä aina haitallista, vaikka dokumentti olisikin hieman vanhentunut, jos dokumentista on kuitenkin vielä hyötyä. Liian usein päivittäminen voi viedä arvokasta aikaa, ja sen sijaan saman ajan voi käyttää esimerkiksi koodin kehittämiseen. [8]

5 Dokumentointityökaluja

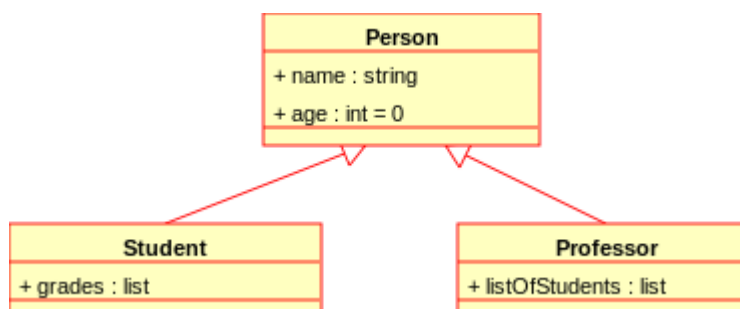
Tässä luvussa esitellään joitakin tavallisimpia työkaluja, joita voidaan käyttää dokumentoinnin apuna. Työkaluja esitetään yleisellä tasolla kuvaten niiden tärkeimmät toiminnot ja ominaisuudet.

5.1 CASE-työkalut

Ohjelmistojen kehityksessä käytetään CASE-työkaluja. Työkaluja tukevat jotain ohjelmistokehityksen työvaihetta. Työkaluihin voidaan lukea kaikki ohjelmistot, joita käytetään ohjelmointityössä. Esimerkiksi projektinhallintaohjelmistot, kääntäjät, editorit, piirtotyökalut ja sovelluskehittimet voidaan ajatella olevan CASE-välineitä. [1, s. 83.]

CASE-välineet voidaan jaotella kahteen ryhmään: edustavälineisiin (upper-CASE) ja taustavälineisiin (lower-CASE). Edustavälineitä käytetään määrittely- ja suunnitteluvaiheissa (esimerkiksi piirtotyökalut) ja taustavälineitä käytetään toteutustyön aikana (esimerkiksi kääntäjät). [1, s. 83.]

Erilaisten kaavioiden piirtäminen CASE-välineillä on yleistä ohjelmistokehityksessä, sillä ne ovat täsmällisiä ja niihin voidaan upottaa tiiviisti informaatiota, jota olisi muuten vaikea selittää sanoin. Kaavioiden laatimisessa noudatetaan usein standardoitua UML-mallinnuskieletä, joka sisältää kokoelman erilaisia notaatioita, joilla voidaan luoda eri tyyppisiä kaavioita. Eri kaavioilla voidaan kuvata esimerkiksi ohjelmiston rakennetta, käyttäytymistä ja vuorovaikutusta. UML-työkaluja on esimerkiksi Microsoft Visio ja Astah. Niiden lisäksi tarjolla on paljon muita maksullisia sekä avoimen lähdekoodin työkaluja.



Kuva 8. UML-mallinnuskielellä tehty luokkakaavio

Kuvassa 9 esitetään UML-mallinnuskielellä toteutettu luokkakaavio, jossa kuvataan kahden luokan periytyminen yhdestä luokasta.

Lisäksi CASE-työkaluilla voidaan generoida koodia. Esimerkiksi Astah-ohjelmalla voidaan generoida java-koodia laadittujen kaavioiden pohjalta. Koodin generoiminen säästää aikaa ja vähentää virheiden mahdollisuutta.

5.2 Tehtävienhallintaohjelmistot

Tehtävienhallintaohjelmistot mahdollistavat resurssien suunnittelun, organisoinnin ja hallinnan projektikohtaisesti.

Esimerkiksi JIRA on Atlassianin kehittämä kevyt, helppokäyttöinen ja joustava tehtävienhallintaohjelmisto, jota käytetään projektimuotoisessa työskentelyssä. Projektia voidaan harvoin tehdä kerralla valmiiksi, jolloin tehtävienhallintaohjelmistolla voidaan pilkkoa tehtäviä osatehtäviksi tai kirjata irrallisiakin työtehtäviä järjestelmään. [10]

JIRA-ohjelmisto on selainpohjainen ja se voidaan ottaa käyttöön asentamalla se omalle palvelimelle tai tilaamalla se pilvipalveluna. Yrityksessä JIRAn käyttäjiä voivat olla esimerkiksi projektipäälliköt, projektin työntekijät, asiakkaat, asiakastukihenkilöt ja esimiehet. [10]

Projektit, tehtävät ja työnkulut ovat JIRAn perusta. **Projekti** voi olla esimerkiksi ohjelmistoprojekti tai olla vain paikka, johon kerätään tiettyjä tehtäviä. **Tehtävä** voi olla esimerkiksi projektiin kuuluva työtehtävä, virheilmoitus tai asiakastukipyyntö. Tehtävään voidaan liittää erilaisia tietoja, kuten kuvaus, vastuuhenkilö ja tehtävän suorittamiseen annettava aikaraja tai -arvio. Kun tehtävä on luotu, se on jossakin työvaiheessa. Työvaiheiden ketjusta muodostuu tehtävän **työnkulku**. Työn edistymistä, eli työvaiheita on helppo seurata JIRAn avulla. [10]

JIRA mahdollistaa myös raporttien tuottamiseen. Raportteja voidaan tehdä esimerkiksi tietyn ajanjakson aikana tehdyistä tehtävien määrästä, ratkaistujen tai ratkaisemattomien tehtävien määrästä tai tehtävien ratkaisuun suunnitellusta tai käytetystä ajasta. [10]

5.3 Wiki-sivut

Wikillä tarkoitetaan verkkosivustoja, jonka sisältöä käyttäjät voivat muokata vapaasti käyttämällä web-selainta. Nykyään useilla organisaatiolla on käytössään sisäiset wiki-sivut, jotka on luotu wikiohjelmistojen, kuten Atlassian Confluencen, avulla. Wikejä käytetään esimerkiksi sisäiseen viestintään ja projektinhallintaan.

Wikiin voidaan luoda organisaation eri ryhmille omat ryhmäkohtaiset tilat, jonne pääsy määritellään oikeuksien avulla. Wiki-sivuille voidaan luoda esimerkiksi ohjeita ja dokumentaatiota. Esimerkiksi ohjelmointitiimi voi luoda jokaiselle ohjelmistoprojektille omat sivut, jotka sisältävät kaikenlaista tietoa projektista. Sivulle voidaan liittää esimerkiksi projektia varten tehdyt määrittely- ja suunnitteludokumentit. Dokumentit voidaan myös ikään kuin kirjoittaa auki hierarkkisesti luvuittain eri wiki-sivuiksi, joilla voi olla lapsisivuja (child pages).

Wikin käytön etuna on niiden helppo muokattavuus. Koska tieto on kaikkien saavutettavissa, voi olla pienempi kynnyksensä päivittää tiettyä sivua. Wiki tarjoaa myös mahdollisuuden tarkastella, mitä muutoksia sisältöön on tehty ja kuka niitä on tehnyt. Tämä on tärkeää jäljitettävyyden ja versioinnin kannalta. [4]

Wikissä on myös haittapuolia. Jos wikiin tallennetaan paljon tietoa, sitä voi olla hankala hallita. Vanhaa tietoa ei myöskään välttämättä poisteta, kun tieto ei ole enää relevanttia. Wikistä on joskus hankala tuoda tietoa esimerkiksi Word-dokumenttiin ilman, että dokumenttia joutuu ”siistiä” muodollisesti. Lisäksi käyttäjät saattavat arastella toisten käyttäjien tekemien sivujen muokkaamista. [4]

5.4 Pilvipalvelut ja verkkolevyt

Erilaiset pilvipalvelut toimivat dokumentoinnissa monella tasolla. Pilvipalveluita voidaan käyttää esimerkiksi **tallennusalustoina**, jollaisia on esimerkiksi Google Drive, Microsoft Sharepoint ja OneDrive.

Jaetut **verkkolevyt** ovat tarkoitettu dokumenttien varastointiin. On tyypillistä, että

vuosien saatossa verkkolevyille kertyy paljon kansioita ja dokumentteja. Kansioita ei välttämättä ole helppo siivota ja järjestellä uudelleen, sillä niiden sisällöstä ei ole ehkä varmaa tietoa. Lopulta voi olla hankalaa etsiä tarvittavaa dokumentaatiota, ja kynnys verkkolevyjen käyttämiseen nousee. Lisäksi verkkolevyjen hakutoiminto ei ole ihanteellisin, jos ei ole varma siitä, mitä etsii.

Tallennusalustojen lisäksi pilvipalvelut käsittävät myös **pilvipalvelusovellukset**. Esimerkiksi Office 365-pilvipalvelu tarjoaa toimistotyökaluja kuten Exceliä ja Wordia käytettäväksi pilvessä.

Google Docs on tekstinkäsittelypilvipalvelu, jolla käyttäjät voivat luoda, muokata ja jakaa dokumentteja muiden henkilöiden kanssa. Yksi tärkeimmistä eduista palvelussa on kuitenkin se, että dokumentteja voidaan muokata reaaliaikaisesti muiden valittujen henkilöiden kanssa. Google Docs on lisäksi yhteensopiva Word-dokumenttien kanssa, joten niitä voidaan tuoda helposti Google Docsiin käsiteltäväksi.

5.5 Microsoft Office -tuotteet

Office-tuoteperheeseen kuuluu tavallisesti Microsoft Word, Microsoft Excel, Microsoft PowerPoint ja Microsoft Access. Lisäksi on vielä Microsoft Visio, joka ei virallisesti ole kuulunut Office-tuoteperheeseen, vaan se on erikseen hankittava ohjelmisto. Microsoftin toimistotyökaluja voidaan käyttää joko paikallisesti tai niitä voidaan hankkia pilvipalveluina.

Suurin osa dokumenteista tuotetaan tavanomaisesti erilaisilla tekstinkäsittelyohjelmilla. [10]. Microsoft Word on suosittu tekstinkäsittelyohjelma, jossa on paljon erilaisia toimintoja. Uusin Word 2016 -versio sisältää uusia toimintoja vanhempiin versioihin (Word 2013) verrattuna. Word 2016 kuuluu sekä Office 2016 -sovellusperheeseen että Office 365 -pilvipalvelutuoteperheeseen.

Uusi versio mahdollistaa reaaliaikaisen yhteistyön, kun asiakirja tallennetaan OneDriveen tai SharePointiin. Asiakirjaa työstävillä henkilöillä täytyy olla myös käytössä uusin versio. Lisäksi versiohistoriaa on parannettu tässä versiossa. Tiedoston muutoshistoriaa pääsee tarkastelemaan ja myös aiemmat versiot ovat käytettävissä. Dokumentteja on

myös yksinkertaisempi jakaa muille henkilöille esimerkiksi SharePointin, OneDriven tai OneDrive for Businessin kautta. Word 2016 tarjoaa myös mahdollisuuden lähettää PDF-tiedosto sähköpostiin suoraan Wordista. [18]

Microsoft Visio -ohjelma avustaa visuaalisessa työskentelyssä. Sen avulla voidaan piirtää esimerkiksi erilaisia vuo- ja prosessikaavioita. Visioin käyttö on helppoa, sillä ohjelmassa voidaan käyttää valmiita aloituskaavioita ja tarvittaessa etsiä lisää hakusanoilla. Käytettävät mallit ja muodot ovat lisäksi esimerkiksi erilaisten standardien (kuten UML ja IEEE) mukaisia. Visio tukee myös esimerkiksi Excel-tiedostojen upottamista Visio-dokumenttiin. Visio-dokumentteja voidaan myös jakaa ja niitä voidaan työstää yhtä aikaa työryhmän jäsenten kesken, jos käytössä on SharePoint tai Office 365.

5.6 Ohjelmakoodin dokumentointityökalut

Toistaiseksi työssä on keskitytty järjestelmän ulkoiseen dokumentoimiseen. Myös lähdekoodin dokumentointi on tärkeää, sillä sovelluksen toiminnan selvittäminen jälkikäteen on vaikeaa ilman dokumentointia. [16] Dokumentointia voidaan tehdä joko tavallisilla koodin joukkoon lisätyillä kommentteilla, tai sitten voidaan käyttää erityisiä kommentointityökaluja.

Lähdekoodia voidaan dokumentoida esimerkiksi Javadoc-työkalulla, jota tarkastellaan tässä aliluvussa. Muita samankaltaisia työkaluja on esimerkiksi PHPDoc ja Doxygen. Näillä työkaluilla voidaan tuottaa automaattisia dokumentteja esimerkiksi sovelluksen toiminnasta ja rajapinnoista.

Javadoc tuottaa oletuksena HTML-muotoisia dokumentteja eli valmiiksi muotoiltuja dokumentteja. Laajennusten avulla on mahdollista tuottaa myös esimerkiksi XML- tai RTF-dokumentteja. [19]

Javadoc-kommenteilla voidaan dokumentoida luokkia, metodeja ja attribuutteja. Kommentti aloitetaan `/**`-merkinnällä, ja se sijoitetaan ennen kutakin dokumentoitavaa kohdetta. Kommentti päätetään `*/`-merkinnällä. Seuraavana on esimerkki Javadoc-kommentoinnista. [17]

```
/** Class Description of MyClass */
public class MyClass
{
    /** Field Description of myIntField */
    public int myIntField;
    /** Constructor Description of MyClass() */
    public MyClass()
    {
        // Do something ...
    }
}
```

Esimerkkikoodi 1. Esimerkki Javadoc-kommentin rakenteesta.

Javadoc-kommenttiin voidaan myös lisätä tunnisteita (tags) eli niin kutsuttuja ”tägejä”. Näillä tunnisteilla määritellään, millaista tietoa kommentti koskee. Yleisesti käytettyjä tunnisteita ovat esimerkiksi:

- `@author [tekijän nimi]` - Luokan tai rajapinnan tekijän nimi(merkki)
- `@version [versio]` - Versiotieto luokasta tai rajapinnasta
- `@param [parametrin nimi] [parametrin kuvaus]` - Kuvaa metodin parametreja
- `@return [kuvaus palautettavasta arvosta]` - Kuvaa metodin palauttamaa tietoa
- `@throws [heitettävä poikkeus] [selitys poikkeuksesta]` - Kuvaa metodin heittämän poikkeuksen.

6 Kehityskohteet ohjelmointiryhmän dokumentaatioprosessissa

Tässä luvussa esitellään syitä dokumentoimattomuuteen ja niiden seurauksia. Havainnot pohjautuvat Metropolian tietohallinnon ohjelmointiryhmästä tehtyihin havaintoihin. Ryhmä koostuu usein noin neljästä ohjelmoijasta, joista yksi tai muutama on usein tuoreempia aloittelevia ohjelmoijia. Työntekijävaihtuvuus on melko suurta, sillä nuoremmat työntekijät ovat usein opiskelijoita, jotka valmistuttuaan saattavat vaihtaa työpaikkaa. Tästä syystä on myös tärkeää, että järjestelmästä on olemassa dokumentaatiota myös seuraaville uusille ohjelmoijille.

Työskennellessäni harjoittelijana Metropolian tietohallinnon ohjelmointiryhmässä havaittiin kolme yksinkertaista ongelmaa:

- Dokumentaatiota ei ole.
- Dokumentaatio on puutteellista.
- Dokumentaatio on vanhaa.

Voidaan huomata, että ongelmat ovat yleisesti tuttuja ohjelmistotuotannon alalla. [7; 13.] Internetissä on paljon keskustelupalstoja, joissa ohjelmoijat kertovat ajatuksiaan siitä, miksi ohjelmoijat eivät pidä dokumentoimisesta. [13] Seuraavaksi pohditaan syitä yllä mainittuihin ongelmiin.

Syyt

Työryhmässä ei ole koskaan ollut selviä toimintatapoja dokumentoinnin suhteen, eli mitään yhtenäistä käytäntöä ei ole sovittu. Moni tämän luvun ongelmista on yhteydessä **kulttuuriin**. Pinttyneitä toimintatapoja ja asenteita voi olla hankala muuttaa, jos työryhmässä on tehty dokumentaatiota kukin omalla tyylillään jo vuosia.

Lisäksi jotkut ohjelmoijat saattavat valitettavasti ajatella, että ”oikeat ohjelmoijat eivät kirjoita dokumentaatiota”. Ajatus on mahdollisesti lähtöisin Ed Postin vuonna 1982 huumorimielessä tehtyyn kirjoitukseen, jossa luonnehditaan ”oikeiden ohjelmoijien” menettelytapoja ohjelmoinnin suhteen. [15] Vaikka kirjoitus on tehty huumorilla, osa ohjelmoijista saattaa ajatella, että dokumentointi ei ole tarpeellista tai se ei ole heidän työtään.

Tärkein ja tavanomaisin syy lienee **ajanpuute**. Ohjelmistoprojekteissa on usein erilaisia määräaikoja, jolloin on ensisijaisesti tärkeää, että ohjelma toimii. Dokumentaatio jää silloin toissijaiseksi tehtäväksi. Usein saattaa käydä niin, että määräaikojen mentyä dokumentointia ei muisteta enää tehdä jälkikäteen, kun vastaan tulee taas uusia työvaiheita.

Pienemmissä ohjelmointiryhmissä työ voi olla hyvin **itsenäistä**. Ohjelmoijat saattavat olla itseohjautuvia, ja heillä on omia projekteja. Usein myös saatetaan olettaa, että ohjelmoija huolehtii projektistaan koko sen elinkaaren ajan. Tämä vähentää motivaatiota dokumentoimiselle, sillä dokumentointi tehdään ikään kuin vain itselle tai mahdolliselle tulevaisuuden seuraajalle. Ajatus saattaa tuntua kaukaiselta, jolloin dokumentointi lykääntyy ja unohtuu helposti. Oletus projektin pysymisestä yhden henkilön varassa on ongelmallinen, jos projekti on pitkäikäinen. Henkilö, joka tietää ohjelmistosta eniten, saattaa esimerkiksi jäädä sairauslomalle, vanhempainvapaalle tai vaihtaa työpaikkaa.

Koska ohjelmoijat toimivat itsenäisesti projektiansa parissa, muut eivät myöskään kysele dokumentaatioiden perään tai niitä ei anneta muille luettavaksi. Työryhmässä ei ole ketään, joka vastaisi dokumentaation laadusta tai siitä, että sitä tehtäisi. Tällöin jokainen on ollut vastuussa omien projektien dokumentoinnista. Projektista huolehtinut työntekijä voi esimerkiksi vaihtaa työpaikkaa, ja hän jättää jälkeensä dokumentaation, jonka muut työntekijät saattavat nähdä vasta ensimmäistä kertaa. Lopputulos voi olla se, että dokumentaatiosta puuttuu hyvin oleellisia tietoja. Dokumenttien laatua tai tekemistä ei siis **valvota**.

Työryhmässä voi olla myös harhaluulo siitä, että tehtyjä **dokumentteja ei lueta**. Tämä käsitys on kuitenkin osoittautunut osittain virheelliseksi seuraamassani ohjelmointiryhmässä. Tehtyjä dokumentteja on haluttu mielellään lukea, mutta oletettavasti silloin, kun dokumentaatioiden puute on tiedostettu.

Jos dokumentteja ei kuitenkaan lueta, ne saattavat helposti vanhentua. Dokumenttien säännöllinen käyttäminen ja lukeminen helpottavat dokumentin ylläpitämistä, sillä esimerkiksi virheet huomataan helpommin, kun joku lukee ja käyttää dokumenttia.

Ohjelmoijat eivät aina tiedä, millaisia dokumentteja heidän täytyisi tuottaa. He voivat myös kokea, että he eivät osaa dokumentoida. Kynnys tehdä dokumentaatiota voi olla suuri, jos sitä ei ole aikaisemmin tehnyt.

Muut ongelmat

Edellä mainituilla syillä ja seurauksilla on seuraavia haitallisia seurauksia.

Usein **koodin kommentoiminen** ja dokumentointi on myös puutteellista. Yhdessä muun dokumentaation puutteen kanssa voi uuden ohjelmoijan olla hankalaa ymmärtää järjestelmän tarkkaa toimintaa.

Sen lisäksi, että dokumentoiminen tärkeää, on myös tärkeää ylläpitää tehtyjä dokumentteja. Sisällöltään **vanhentunut dokumentti** on lähes yhtä haitallista, ellei haitallisempaa kuin kokonaan puuttuva dokumentaatio. Vanhentunut tieto voi hämätä ja aiheuttaa väärinkäsityksiä. Kerran jo väärin luetun ja ymmärretyn asian oikominen voi olla hankalampaa kuin se, että ymmärrys olisi rakentunut tarkistamalla järjestelmän toiminnan konkreettisesti tutkimalla esimerkiksi ohjelmakoodia.

Dokumentit saattavat olla myös **hajautettuna** ympäriinsä. Niitä voi löytyä jaetulta verkkolevyltä, omalta verkkolevyltä tai kovalevyltä, Wiki-sivuilta tai Google Driveltä. Dokumentit tai muut tiedot voivat myös jäädä vain joidenkin henkilöiden sähköpostiviesteihin, joita ei välttämättä koskaan kirjata kyseistä järjestelmää koskevaan dokumenttiin. Uusi työntekijä ei näin ollen välttämättä saa tietää koskaan sähköpostiviesteissä käydyistä keskusteluista, jotka saattavat sisältää uusia määrittelyjä ja toimintoihin tehtyjä muutoksia.

Yksi ongelma on se, että dokumenttien olemassa olosta ei aina edes tiedetä. Dokumentit saattavat lymyillä esimerkiksi verkkolevyn syövereissä, mutta niiden tarkkaa sijaintia ei ole tiedossa henkilöillä, jotka saattavat tarvita dokumentteja.

7 Amme-järjestelmän dokumentaation kehittäminen

7.1 Tausta

Amme-järjestelmän dokumentaatio oli puutteellinen ennen tämän työn aloittamista. Puutteellinen dokumentaatio esimerkiksi tietokannasta vaikutti uusien kehittäjien tehokkuuteen kehittää järjestelmää. Aluksi tarkoitus oli selvittää Ammeen tietokannan toimintaa kehittäjiä varten, sillä järjestelmän pääasiallisesti luonut henkilö ei enää toiminut järjestelmän pääkäyttäjänä. Pian huomattiin, että pelkän tietokannan eri taulujen tarkoitusten selvittäminen ei riitä. Jotta ymmärtäisi taulujen tarkoituksia, tulisi ymmärtää myös järjestelmän toimintaa. Tästä syystä dokumentaation aihe laajeni käsittelemään tietokannan lisäksi myös järjestelmän toimintaa.

Luotu dokumentaatio mahdollistaa myös sen, että muihin työtehtäviin siirtynyttä asiantuntijaa ei tarvitse enää häiritä yhtä paljon erilaisilla kysymyksillä koskien järjestelmää. Dokumenttia voidaan käyttää uuden kehittäjän perehdytykseen, jotta hän saa kokonaiskuvan järjestelmän toiminnasta. Tämän jälkeen on helpompi esittää tarvittaessa tarkempia kysymyksiä, kun kokonaiskuva on hahmottunut.

Lisäksi tarkoitus oli, että luotua dokumenttia voitaisiin käyttää mallina myös muissa pienempien projektien dokumentoimisessa. Amme-järjestelmä eroaa kuitenkin pienemmistä projekteista sen laajuuden ja omaleimaisuuden vuoksi, mistä syystä siitä tehtyä dokumenttia ei välttämättä kannata suoraan käyttää mallina muille projekteille.

Amme-järjestelmä on myös hyvin laaja ja keskeinen järjestelmä, joka on vastuussa käyttäjätunnusten luomisesta sekä tietojen jakamisesta ja päivittämisessä kohdejärjestelmille. Tämän perusteella näin tärkeän järjestelmän dokumentoiminen on kannattavaa.

Järjestelmän dokumentaatiossa käytettiin seuraavia työkaluja:

- MySQL Workbench 6.3 -tietokantojen hallintaohjelma
- Microsoft Word 2013 -tekstinkäsittelyohjelma
- Eclipse Jee Mars -ohjelmistokehitysympäristö

- Microsoft Visio 2013 -piirustusohjelma
- Microsoft Access 2013 -tietokantojen käsittelyohjelma.

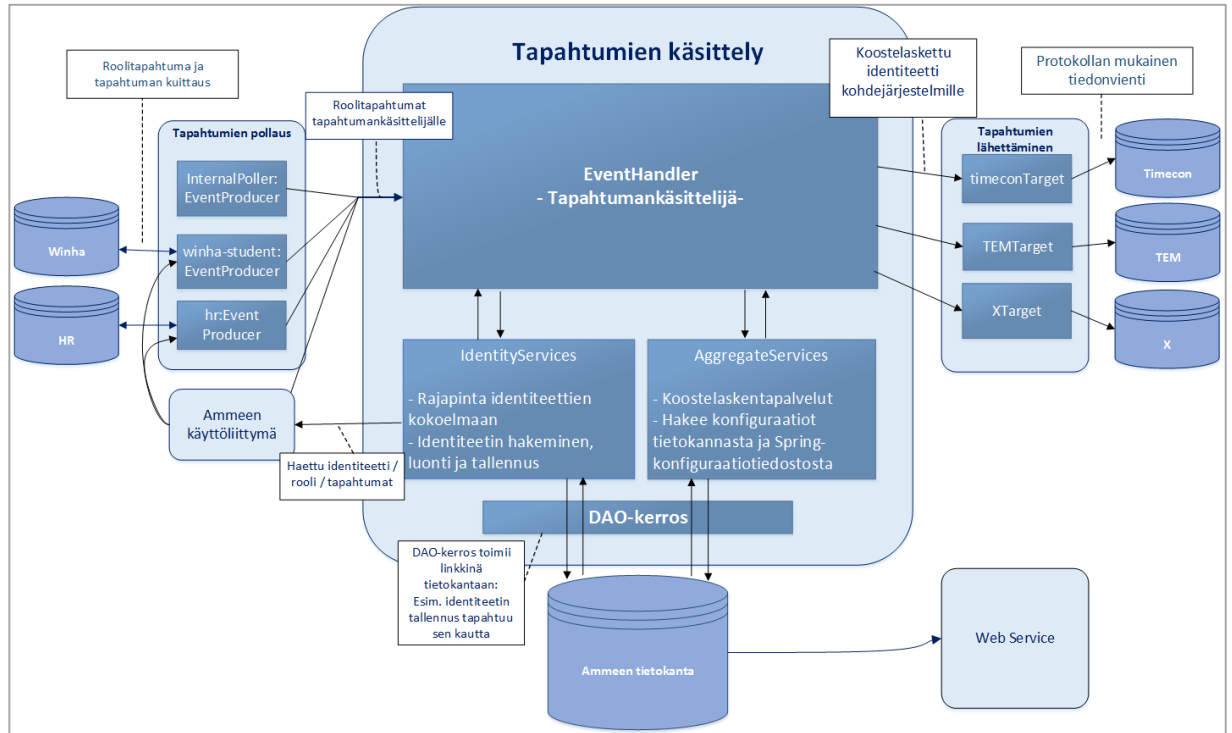
7.2 Ammeen toiminta ja rakenne lyhyesti

Metropolia käyttää identiteetinhallinnassa Amme-järjestelmää ja toteuttaa pääsynhallintaa LDAP-hakemistolla, joka on samalla myös Ammeen kohdejärjestelmä. Kokonaisuudessaan identiteetin- ja pääsynhallintaa toteutetaan keskitetyllä mallilla, jossa Amme toimii koko järjestelmän keskuksena, johon on liitetty erilaisia lähde- ja kohdejärjestelmiä.

Amme on Java-ohjelmointikielellä toteutettu tapahtumapohjainen järjestelmä. Tapahtumapohjaisuus tarkoittaa sitä, että lähdejärjestelmässä tapahtuva muutos viedään Ammeeseen käsiteltäväksi saman tien. Amme on siis reaaliaikainen järjestelmä, eikä esimerkiksi eräajopohjainen järjestelmä, jossa muutokset päivitetäisiin esimerkiksi kerran päivässä.

Tapahtumapohjainen ratkaisu mahdollistaa yhdessä keskitetyn hallinnan kanssa muun muassa seuraavia asioita:

- Uudet käyttäjät saavat nopeasti tarvittavat oikeudet kaikkiin kohdejärjestelmiin.
- Roolin muuttuessa myös käyttöoikeudet kohdejärjestelmiin muuttuvat.
- Poistuneiden käyttäjien käyttöoikeudet eivät jää voimaan.
- Voidaan helposti seurata käytössä olevia käyttöoikeuksia.



Kuva 9. Ammeen arkkitehtuuri

Ammeen arkkitehtuuri esitetään kuvassa 10. Vasemmalla voidaan nähdä lähdejärjestelmät ja oikealla kohdejärjestelmät. Keskellä toimii Ammeen tapahtumankäsittelijä, joka on koko toiminnan perusta. Tapahtumankäsittelijä käyttää erilaisia palveluja (*identityServices* ja *AggregateServices*) toteuttaakseen identiteetin luomisen ja laskemisen kohdejärjestelmiä varten. Tapahtumien käsittely on yhteydessä Ammeen tietokantaan, josta luetaan ja jonne kirjoitetaan tietoja.

Identiteetti

Identiteetti on abstraktio tosielämän henkilöstä, jota voidaan kuvailla tarvittavilla attribuuteilla esimerkiksi tietokannassa. Esimerkiksi tietokantataulussa oleva tietue, joka sisältää henkilön tietoja, voi kuvastaa henkilön identiteettiä. Amme-järjestelmä koostaa identiteetin roolien tiedoista.

Roolit

Roolit koostavat identiteetin. Roolit jaetaan viiteen eri roolityyppiin, jotka eroavat toisistaan sen perusteella, millainen asema henkilöllä on organisaatiossa (taulukko 1). Henkilö voi olla esimerkiksi opiskelija, työntekijä tai ostopalveluhenkilökunnan työntekijä. Roolityypit siis mukailevat erilaisia sopimuksia, joita on esimerkiksi työsopimus, opiskelu-oikeus tai jokin yhteistyösopimus. Roolitiedot ovat aina lähtöisin jostain lähdejärjestelmästä. Amme käsittelee roolitietoja ja luo niiden pohjalta identiteetin.

Taulukko 1. Roolien luokittelu.

Roolityyppi	Rooli(t)	Selitys	Lähde
staff	metropolia.staff	Metropolian henkilökunta	HR
student	metropolia.student	Metropolian opiskelijat	Winha
external	partner.staff konsis.staff konsis.student popjazz.staff popjazz.student heltech.staff heltech.student metka.staff	Ostopalveluhenkilökunta ja erilaiset kumppaniorganisaatiot	Web-käyttöliittymä
unidentified	external service	Ulkopuoliset Hallinnolliset tunnukset	Web-käyttöliittymä
guest	guest	Lyhytaikaiset vierailijat	Web-käyttöliittymä

Koostelaskenta

Koostelaskennalla on kaksi erityyppistä tehtävää: 1) tietojen rikastaminen ja 2) tietojen koostaminen. Rikastaminen tarkoittaa uusien attribuuttien laskemista rooleille ja identiteetille. Esimerkiksi käyttäjätunnuksen luominen on rikastamista. Koostamista on puolestaan esimerkiksi vahvimman arvon valitseminen roolista identiteettiin. Vahvin rooli määräytyy muun muassa roolityypin ja läsnäolokoodin perusteella, missä esimerkiksi henkilökuntarooli on vahvempi kuin opiskelijarooli. Niin kutsutut aggregaattorit (koostefunktiot) suorittavat koostelaskennan.

7.3 Ammeen tietokanta

Ammeen tietokanta on toteutettu MySQL-relaatiotietokantana. Tietokanta sisältää noin kolmekymmentä tietokantataulua, joista suurin osa on aktiivisesti käytössä. Muutama tauluista ei ole käytössä, mutta ovat valmiina tulevaisuuden toimintojen kehittämistä varten. Lisäksi tietokannassa on vanhentuneita ja tyhjiä tauluja.

Tietokantataulujen välillä on relaatioita eli yhteyksiä, mutta niitä ei ole asetettu MySQL-tietokantakielellä. Eli esimerkiksi vierasavaimia ei ole asetettu taulujen välille. Eheyttä ylläpidetään kuitenkin ohjelmallisesti esimerkiksi transaktioiden avulla. Jos tietojen käsittelyn aikana tapahtuu virhe, tapahtuma keskeytetään, eikä tietokantaan tallenneta mahdollisesti puutteellisia tietoja.

Vierasavainten määrittely ei ole välttämättä kovin yksinkertaista Ammeen omaleimaisen toiminnan takia, missä valitaan attribuutin arvot identiteettiin roolien vahvuuden mukaan. Esimerkiksi jossain roolissa tieto voi olla vanhaa, eikä tietoa haluta päivittää vanhalle roolille. Vierasavaimen määrittely huolehtisi tällöin ”liikaa” eheydestä.

7.4 Järjestelmän tutkiminen

Dokumentointiprosessi alkoi järjestelmän tutkimisella. Tutkiminen aloitettiin vanhoista suunnitteludokumenteista, joita oli onneksi melko paljon. Suunnitteludokumenttien lisäksi tutkittiin Ammeen tietokantaa MySQL Workbench -työkalulla.

Tietokantataulujen yhteyksiä selvitettiin tekemällä MySQL-kyselyitä. Jos halutaan esimerkiksi tietää, löytyykö jokin tietokenttä myös tietokannan toisesta taulusta, voidaan käyttää alla olevaa kyselyä:

```
SELECT DISTINCT COLUMN_NAME, TABLE_NAME  
  
FROM INFORMATION_SCHEMA.COLUMNS  
  
WHERE column_name like 'haettavaKenttä'  
  
AND TABLE_SCHEMA='database';
```

Esimerkkikoodi 2. MySQL-kysely, jolla etsitään tietokenttää tietokannan tauluista.

Kysely palauttaa tietokentän ja taulun nimen, josta haettava tietokenttä löytyi. Tämä kysely helpotti taulujen välisien yhteyksien tutkimista. Taulujen väliset yhteydet olisi helposti voinut tarkastaa MySQL Workbench -ohjelman toiminnolla, joka automaattisesti luo graafisen näkymän yhteyksistä, jos ne on määritelty MySQL-tietokantakielellä. Koska taulujen välille ei ollut määritelty MySQL-tietokantakielellä yhteyksiä, kyseistä toimintoa ei voinut käyttää, vaan yhteydet piti tarkastaa manuaalisesti.

Tutkimisen aikana kirjattiin kysymyksiä valmiiksi tapaamisiin, joita pidettiin järjestelmän luoneen asiantuntijan kanssa. Kysymykset ja keskustelu olivat tärkeä apu järjestelmän ymmärtämisessä, sillä vanhojen suunnitteludokumenttien mukaisia suunnitelmia ei tietysti aina ollut toteutettu kuten oli suunniteltu. Vanhimmat suunnitteludokumentit olivat lähes kahdeksan vuotta vanhoja, kun taas tuoreimmat dokumentit olivat muutaman vuoden vanhoja.

Myöhemmin työn aiheen laajentuessa aloitin myös ohjelmakoodin tutkimisen. Erityisesti koostelaskennan toteuttavien aggregaattorien toiminta piti tarkastaa erikseen ohjelmakoodista, sillä tietokantatauluista ei saanut kokonaisvaltaista käsitystä niiden yksityiskohteisesta toiminnasta. Välillä jonkin tietokentän toiminta oli epäselvää, jolloin piti tarkistaa, löytyykö ohjelmakoodista viitteitä kyseiseen attribuuttiin. Usein tietokentän tarkoitus selveni, mutta toisinaan huomattiin, että tietokentällä ei ollut jostain syystä mitään käyttöä.

7.5 Dokumentaation laatiminen

Dokumentaation laatiminen aloitettiin tietokannan dokumentoimisella. Jokaisesta tietokantataulusta tehtiin taulukko, jossa käy ilmi tietokentän nimi, selite ja esimerkkiarvo(t). Muutamisiin taulukoihin lisättiin myös sarakkeen, josta käy ilmi, onko tietokentän lähde jokin lähdejärjestelmä vai tuottaako Amme itse tietokentän.

Lisäksi käytettiin MySQL Workbench -työkalua, jolla pystyi luomaan tietokantataulusta graafisen esityksen, josta käyvät ilmi seuraavat asiat tietokenttäkohtaisesti:

- Toimiiko tietokenttä avaintietokenttänä?
- Mikä on tietokentän tietotyyppi?
- Voiko tietokenttä olla tyhjä? (NULLABLE/NOT NULL)

Taulukko 2. Tietokantataulun tietokenttien dokumentointiesimerkki

Tietokenttä	Lähde	Selite	Esimerkkiarvo(t)
givenName (etunimi)	A (Amme)	Henkilön etunimi	Essi
ownEntryRole (roolityyppi)	A (Amme)	Roolityypin nimi	metropolia.student
ownStudentID (opiskelija- tunniste)	W (Winha)	Winhan sisäinen opiskelijanumero (OPISK) Huom. Arvon on tarkoitus olla uniikki. Opiskelijanumero liitetään aina yhteen henkilöön.	12345

Tietokantataulujen dokumentaation jälkeen tutkittiin taulujen välisiä yhteyksiä, ja toisiinsa liittyvät taulut ryhmiteltiin yhteen. Taulujen väliset yhteydet merkittiin manuaalisesti, sillä MySQL Workbench ei voinut esittää yhteyksiä automaattisesti vierasavainmäärittelyiden puutteen vuoksi.

Kuvista puuttuu toistaiseksi kuitenkin taulujen väliset tarkemmat suhdemerkinnot, joilla kuvataan taulujen ja tietokenttien välisiä ”äiti-lapsi”-suhteita. Esimerkiksi identiteetillä voi olla monta roolia, mutta roolilla voi olla vain yksi identiteetti.

7.6 Dokumentin tarkastaminen ja korjailu

Dokumentin tarkastaminen järjestelmän asiantuntijan kanssa aloitettiin vasta melko myöhään. Työtä tehtiin tutkimisen ja keskusteluiden pohjalta. Ensimmäisen tarkastamisen jälkeen sovittiin, että työn sisällöllistä rakennetta muutetaan hieman. Tämän jälkeen työtä tarkastettiin aina luku kerrallaan. Tarkastamisen yhteydessä korjaukset kirjattiin muistiin ja ne toteutettiin myöhemmin.

7.7 Haasteet ja kokemukset

Kohtasin työn aikana melko paljon haasteita. Ensinnäkin järjestelmä oli entuudestaan lähes tuntematon. Tämän lisäksi en onnistunut löytämään referenssikirjallisuutta samankaltaisista identiteetinhallintajärjestelmistä, joita vasten toiminnallisuutta olisi voinut verrata. Näiden seikkojen takia oli haastavaa ymmärtää itselle tuntematonta aihealuetta. Asioiden ymmärtäminen on kuitenkin tärkeä edellytys dokumentoimiselle. Lisäksi referenssikirjallisuuden puute vaikutti siihen, että termistöä ja käsitteitä täytyi keksiä jonkin verran itse. Sopivan kuvailevan käsitteen keksiminen oli välillä haastavaa.

Haasteellisuutta lisäsi myös se, että dokumentaation laatiminen oli hyvin riippuvaista haastateltavan asiantuntijan aikatauluista. Hänellä oli paljon omia työtehtäviä ja kiireitä, minkä takia tapaamisia ei voinut pitää milloin tahansa.

Koin myös, että lähdin tekemään työtä ikään kuin nurinkurisesti, sillä aloitin tutkimisen tietokannasta. Lähdin liikkeelle liian triviaaleista asioista, joten kokonaiskuvan hahmot-

taminen tapahtui paljon myöhemmin. Kokonaiskuvan ymmärtäminen helpottaa triviaalien tietojen ymmärtämisessä, sillä silloin asiat ovat helpommin yhdisteltävissä jo opittuihin kokonaisuuksiin.

7.8 Projektin lopputulos ja opetukset

Amme-järjestelmän tutkiminen ja dokumentoiminen oli opettavainen kokemus. Pääsin seuraamaan melko läheltä ohjelmointiryhmän toimintatapoja ja kulttuuria, mitä käsiteltiin edellisessä luvussa.

Yksi selkeimmistä opeista oli, että järjestelmän dokumentoiminen vie paljon aikaa. Vielä enemmän aikaa vie se, jos kirjoittamisen tekee toinen henkilö, jolle järjestelmä on entuudestaan tuntematon. Ulkopuolisen kirjoittajan täytyy joka tapauksessa olla tiiviissä yhteistyössä järjestelmän tehneen henkilön kanssa, jos se on vain mahdollista.

Jälkikäteen dokumentointi saattaa tällöin viedä enemmän aikaa, kuin jos sen olisi tehnyt järjestelmän toteuttamisen aikana. Toisaalta dokumenttien tekemättömyys ja päivittämättömyys toteutuksen aikana on mahdollistanut ajankäytön muihin kehitystehtäviin. Tämä aiemmin säästetty aika saattaa kuitenkin kumoutua jälkikäteen tehtyyn dokumenttiin käytettyyn aikaan.

Opin myös, että on tärkeää määritellä heti aluksi, mitä aiotaan dokumentoida ja miten. Työssäni oli aluksi hieman epäselvyyksiä, että mitä dokumentoidaan. Tästä syystä dokumentin rakenne meni uusiksi työn puolivälissä. Dokumentin rakenteen muokkailu jälkikäteen on raskasta ja vie aikaa.

Ketterän dokumentoinnin käyttö olisi ollut kannattavaa tämän dokumentaation tekemisessä. Huomasin, että tein dokumenttia liian paljon itsenäni, ja annoin sen tarkasteltavaksi asiantuntijalle liian myöhäisessä vaiheessa. Jos olisimme keskustelleet heti aluksi yhdessä dokumentin rakenteesta ja sisällöstä, dokumentti olisi mahdollisesti syntynyt nopeammin ja tehokkaammin. Käytännössä ketterä dokumentointi olisi kuitenkin saattanut vaatia enemmän aikaa asiantuntijan kanssa, mikä ei olisi ollut välttämättä mahdollista.

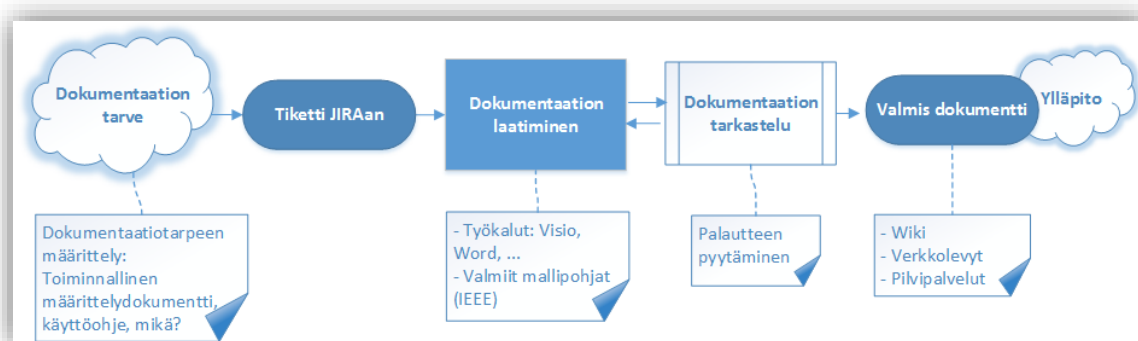
Lisäksi järjestelmän tutkiminen tarjosi mielenkiintoisen läpileikkauksen sen kehityksestä vuosien varrella. Oli mielenkiintoista huomata, kuinka historia vaikuttaa osaltaan ohjelmiston kehittämiseen. Järjestelmät ovat lisäksi harvoin täydellisiä, sillä ne ovat aina oman aikansa toimivia tuotoksia. Myöhemmin saatetaan huomata asioita ja toiminnallisuuksia, jotka olisi voinut toteuttaa toisin. Toisaalta saattaa ilmetä täysin uusia muun ympäristön asettamia vaatimuksia, joita ei olisi koskaan osattu ottaa huomioon järjestelmää suunniteltaessa. Lisäksi muiden järjestelmien ominaisuudet saattavat asettaa haasteita kehiteltävään järjestelmän suunnitteluun ja toteutukseen. Monesti siis hankalat lähtökohdat voivat aiheuttaa monimutkaisia ratkaisuja.

Lopuksi ymmärsin myös, että vaikka käytettävissä on paljon erilaisia standardeja, ohjeistuksia, malleja ja teorioita, niiden noudattaminen on oma lukunsa. Kokemukseni perusteella on tyypillistä, että kaikkea ei tehdä ohjeiden mukaan, ja harvoin mikään on täydellistä.

8 Dokumentaatioprosessin kehittäminen

Tässä luvussa keskitytään dokumentaatioprosessin kehittämiseen. Pääkohderyhmä kehitysehdotuksissa on Metropolian ohjelmointiryhmä, mutta esitetyt kehitysehdotukset ovat sovellettavissa yleisemminkin. Kaikkia ehdotuksia ei ole kuitenkaan välttämättä tarkoitus toteuttaa Metropolian ohjelmointiryhmässä.

Kuvassa 11 on esitetty dokumentaatioprosessin työnkulku. Prosessia voidaan ajatella eräänlaisena laatujärjestelmänä dokumentoinnin kannalta.



Kuva 10. Dokumentaatioprosessin työnkulku

8.1 Dokumentaation tarpeen määrittely

Prosessi alkaa dokumentointitarpeen määrittelyllä. Määrittelyn tuloksena päätetään, millainen dokumentti tulee tehdä. Tämän määrittelyn voi tehdä yksin tai keskustellen muiden kanssa. Määrittely voi myös tulla ylemmältä taholta, jos ohjelmistolta vaaditaan esimerkiksi käyttöohje asiakkaalle. Pienissä ja itsenäisissä projekteissa minimidokumentation tulisi koostua vähintään teknisestä ja toiminnallisesta määrittelystä, joista käy ilmi, miten ohjelmisto on toteutettu.

Kun dokumentin tarve on tiedostettu, siitä tehdään tiketti eli työtehtävä tehtävienhallintajärjestelmään. Kuvan esimerkissä viitataan JIRA-ohjelmistoon, mutta sen sijaan voidaan käyttää myös muuta samankaltaista järjestelmää. Tärkeää on, että dokumentaation laadimisesta tehdään konkreettinen työtehtävä, jotta se saataisiin mukaan työjonoon.

Ohjelmointiryhmällä on ollut JIRAn käyttömahdollisuus olemassa, mutta sen käyttö on jäänyt vähälle. Dokumentaatioprosessin kehityksen myötä JIRA-tehtävienhallintaohjelmistoa alettiin käyttää ahkerammin.

Aktiivisille projekteille luotiin omat projektiosiot. Vanhoille tai pian poistuville järjestelmille ei luotu omia osioita. Lisäksi aktiivisille projekteille luotiin tikettejä tehtävistä, jotka odottavat ratkomista. Tehtävät voivat olla erityyppisiä. Ne voivat käsitellä esimerkiksi parannusehdotuksia, bugeja tai uusia toimintoja. Tehtäväksi voidaan myös määritellä jonkin dokumentin luominen kyseisestä projektista.

JIRAn käyttö soveltuukin erinomaisesti myös dokumentaatioprosessin kehittämiseen, sillä dokumentaatiotarpeen voi kirjoittaa konkreettisesti työjonoon tehtäväksi. Dokumentointitarpeelle saadaan näin vastuuhenkilö ja lähtöpiste, josta voidaan edetä seuraavaan vaiheeseen, eli dokumentoinnin luomiseen. Tarvittava dokumentaatio on hyvä määritellä tehtävässä, jotta vastuuhenkilö tietää, mitä täytyy dokumentoida.

8.2 Dokumentaation laatimisprosessi

Seuraavaksi dokumenttia aletaan työstää. Työkaluina voidaan käyttää esimerkiksi Visiota ja Wordia. Mallipohjina kannattaa käyttää IEEE:n mukaisia dokumenttimalleja toiminnallisesta ja teknisestä määrittelystä, jotka esiteltiin luvussa 3.3.

Dokumenttimallit tarjoavat ikään kuin johdattelevia kysymyksiä, joihin vastaamalla syntyy dokumentti. Lisäksi, jos eri projektien dokumenttien tyylit yhtenäistyvät, luettavuus, ennustettavuus ja laatu paranevat.

Jos kyse on koodin dokumentoimisesta kommenttien avulla, voidaan käyttää esimerkiksi Javadoc-työkalua.

Dokumentaation laatimisen aikana on hyvä antaa dokumentti luettavaksi muille kommentoitavaksi. Jos dokumentin tekee alusta loppuun itse, omalle kirjoitukselle voi tulla sokeaksi. Palautteen perusteella voidaan korjata esimerkiksi kielioppivirheitä ja tehdä täsmennyksiä tai korjata rakennetta.

8.3 Dokumenttien sijoitus ja ylläpitäminen

Lopuksi valmis dokumentti täytyy siirtää sovittuun paikkaan, kuten verkkolevylle tiettyyn kansioon, Wikiin tai pilvipalveluun. On tärkeää sopia yhdessä, minne tehdyt dokumentit sijoitetaan ja miten niitä ylläpidetään. Kaikkien työryhmän jäsenten tulisi toimia samalla tavalla, jotta dokumentit eivät hajautuisi eri järjestelmiin ja jotta ne pysyisivät ajan tasalla. Dokumenttien sijoituspaikat voivat myös vaihdella projektikohtaisesti, sillä arkaluonteisia dokumentteja ei välttämättä kannata tai voi viedä pilvipalveluihin.

Vaikka dokumentti on valmis, sitä täytyy mahdollisesti ylläpitää. Käytännössä dokumentin ylläpitovastuu jää luultavasti henkilölle, joka on pääasiassa laatinut dokumentin. Dokumentin ja projektin luonne vaikuttavat ylläpitämisen tarpeeseen. Jos on todennäköistä, että järjestelmään tehdään myöhemmin uusia toimintoja, dokumentit täytyy pitää myös ajan tasalla. Uusien toimintojen ohella kannattaa tehdä myös tiketti dokumentin päivittämistarpeesta tehtävienhallintajärjestelmään.

9 Yhteenveto

Dokumentointi on tärkeä osa ohjelmistotuotantoa. Laadukkaalla ja ajantasaisella dokumentaatiolla tuetaan ohjelmistoprojektin hallintaa eri elinkaaren vaiheissa. Dokumentaation tärkeys korostuu varsinkin silloin, kun projektin pääkehittäjä vaihtuu uuteen. Jos järjestelmästä on tehty toiminnallinen ja tekninen määrittelydokumentaatio, uuden kehittäjän on helpompi ottaa järjestelmä haltuun.

Työn tarkoituksena oli kehittää ohjelmointiryhmän dokumentaatioprosessia. Prosessin kehittäminen ei kuitenkaan toteudu nopeasti. Tätä työtä kirjoittaessa kaikkia ongelmia ei ole ratkottu. Lähtökohdat ovat kuitenkin hyvät dokumentaatiokulttuurin luomiselle, sillä ohjelmoijat ovat tiedostaneet dokumentaation puutteen ongelmat. Ohjelmoijat ovat itse nimittäin joutuneet kokemaan puuttuvan dokumentaation seurauksia.

Selkeimpiä parannuksia on tehtävienhallintaohjelmiston käyttö dokumentaation tarpeen kirjaamiseen, ettei dokumentointitehtävä unohdu kiireen keskellä. Myös wikin käyttö esimerkiksi hiljaisen tiedon jakamiseen eri projekteista on lisääntynyt. Wikiin on tehty erilaisia ohjeartikkeleita eri ohjelmistoprojekteista, jotta kuka tahansa voisi esimerkiksi ratkaista tiettyjä ohjelmistossa ilmentyneitä ongelmia.

Muita parannuksia on esimerkiksi dokumenttien antaminen muille luettavaksi. Palautteen saaminen parantaa dokumentin laatua, eikä dokumentin sisällön laatiminen jää vain yhden ihmisen varaan.

Amme-järjestelmän dokumentointi osoitti, että ajantasainen dokumentaatio helpottaa järjestelmän jatkokehitystä, ja esimerkiksi uuden teknisen pääkehittäjän perehdyttämistä järjestelmän toimintaan ja ominaisuuksiin. Ammeesta tehty dokumentaatio toimii myös hyvänä esimerkkinä sille, minkälaista dokumentaation tulisi olla myös muissa ohjelmistoprojekteissa.

Tulevia haasteita lienee dokumentointikulttuurin kokonaisvaltainen jalkauttaminen, niin että uusista käytännöistä pidetään kiinni myös tulevaisuudessa. Nykyisiä kehitysehdotuksia on kokeiltu toistaiseksi nuorempien ja tuorempien ohjelmoijien keskuudessa.

Lähteet

1. Haikala Ilkka, Märijärvi Jukka. 2000. Ohjelmistotuotanto.
2. Taina Juha. 2009. Ohjelmistoprosessit ja ohjelmistojen laatu. Luentomateriaali. https://www.cs.helsinki.fi/u/taina/opol/k-2009/pdf/luku-6_2.pdf. Luettu 10.4.2016.
3. Ray Laura Henry. 2016. Creating a Culture of Documentation. <http://www.sciencedirect.com/science/article/pii/S009913331600015X>. Elsevier: The Journal of Academic Librarianship. Luettu 10.4.2016.
4. Rettich Kay. 2011. Using the Wiki to Deliver Paperless Software Documentation. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6087219>. Professional Communication Conference (IPCC), 2011 IEEE International. Luettu 10.4.2016.
5. Vuori, Matti. 2010. 125 pointtia dokumentoinnista. Verkkodokumentti. http://www.mattivuori.net/julkaisuluettelo/liitteet/satavartti_pointtia_dokumentoinnista.pdf. Luette 10.4.2016.
6. Ambler, Scott. 2012. Best Practices for Agile/Lean Documentation. Verkkodokumentti. <http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm>. Luettu 10.4.2016.
7. McAllister, Neil. 2012. How to get developers to document their code. Verkkodokumentti. <http://www.infoworld.com/article/2618550/application-development/how-to-get-developers-to-document-their-code.html>. Luettu 10.4.2016.
8. McConnell, S. 2002. Ohjelmistotuotannon hallinta. Edita Prima Oy. s. 434, 569-571.
9. Basaraner Cagdas. 2012. 10 Software Documentation Best Practices. Verkkodokumentti. <https://dzone.com/articles/10-software-documentation-best>. Luettu 10.4.2016.
10. Eduix Oy. JIRA. Verkkodokumentti. <http://www.eduix.fi/9-tuotteet-ja-palvelut/9-jira>. Luettu 10.4.2016.
11. Tasse, Gregory. 2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. <http://www.nist.gov/director/planning/upload/report02-3.pdf>. Luettu 10.4.2016.
12. Forward, Andrew & Lethbridge, Timothy C. xxxx. The Relevance of Software Documentation, Tools and Technologies: A Survey. <http://www.rose->

hulman.edu/Users/faculty/young/CS-Classes/csse575/Resources/DocumentationSurveyPaper.pdf. Luettu 10.4.2016.

13. Why Do Programmers Hate Documenting? 2003. Verkkodokumentti. <http://discuss.fogcreek.com/joelonsoftware/default.asp?cmd=show&ixPost=35336>. Luettu 10.4.2016.
14. Tilley, Scott. 1993. Documenting-in-the-large vs. Documenting-in-the-small. IBM Press: CASCON '93: Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing - Volume 2. Luettu 10.4.2016.
15. Post, Ed. 1982. Real Programmers Don't Use PASCAL. Verkkodokumentti. <http://www.ee.ryerson.ca/~elf/hack/realmen.html>. Luettu 10.4.2016.
16. Vesterholm, M. & Kyppö, J. 2006, s. 348.
17. Javadoc Tutorial. 2012. Verkkodokumentti. <https://students.cs.byu.edu/~cs240ta/fall2012/tutorials/javadoctutorial.html>. Luettu 10.4.2016.
18. Word 2016:n uudet ominaisuudet. 2016. <https://support.office.com/fi-fi/article/Word-2016-n-uudet-ominaisuudet-4219dfb5-23fc-4853-95aab13a674a6670>. Luettu 10.4.2016.
19. Javadoc 5.0 Tool. Verkkodokumentti. <http://docs.oracle.com/javase/1.5.0/docs/guide/javadoc/index.html>. Luettu 10.4.2016.