Pekka Tammi

Dokumentaatio ketterässä kehityksessä ja siihen liittyvät käytänteet ja työkalut

Tieto- ja ohjelmistotekniikan kandidaatintutkielma

29. huhtikuuta 2024

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pekka Tammi

Yhteystiedot: ptjtammi@student.jyu.fi

Työn nimi: Dokumentaatio ketterässä kehityksessä ja siihen liittyvät käytänteet ja työkalut

Title in English: Documentation in Agile Development and Related Practices and Tools

Työ: Kandidaatintutkielma

Sivumäärä: 20+0

Tiivistelmä: Hyvä dokumentaatio selkeyttää ohjelmiston tavoitteita, auttaa uusien kehittäjien perehdyttämisessä ja parantaa viestintää ketterissä kehitystiimeissä. Vaikka yksi ketterän ohjelmistokehityksen julistuksen pääperiaatteista on asettaa toimiva ohjelmisto kattavan dokumentaation edelle, tehokkaalla dokumentaatiolla on silti tärkeä rooli.

Tutkielma keskittyy siihen, mitä nykyaikaisesta ohjelmistosta dokumentoidaan. Tyypillisesti dokumentoitavat osat, kuten käyttäjätarinat, toiminnalliset ja ei-toiminnalliset vaatimukset. Nämä auttavat kehittäjiä ymmärtämään ja hallitsemaan projektin laajuutta ja vaatimuksia. Lisäksi tässä tutkimuksessa tarkastellaan työkaluja, kuten wikejä, jotka tukevat kehittäjiä dokumentaation ylläpidossa ja kirjoittamisessa. Nämä työkalut mahdollistavat kehittäjille dokumentaation yhteisöllisen ja iteratiivisen kirjoittamisen, säästäen aikaa ja parantaen yhteistyötä ketterissä tiimeissä.

Tutkimus nostaa esiin kysymyksen siitä, miksi dokumentaation laatiminen on usein välteltävä tehtävä ketterien ohjelmistokehittäjien keskuudessa sekä näennäistä ristiriitaa, jossa kehittäjät tunnustavat silti tehokkaan dokumentaation tärkeyden työssään.

Avainsanat: dokumentaatio, ketterä kehitys, dokumentaatiotyökalut

Abstract: Documentation in agile software development is a frequently discussed topic. Although one of the main principles of agile software development is to prioritize working software over comprehensive documentation, effective documentation still plays an important role. Effective documentation clarifies the software's goals, helps in the onboarding of new developers, and improves communication within agile development teams.

The thesis focuses on various aspects of documentation in modern software. It consists of multiple different elements. These elements, such as user stories, functional and non-functional requirements, assist developers in understanding and managing the project's scope and specifications. Additionally, this study examines tools, e.g. wikis, that support developers in maintaining and writing documentation. These tools enable developers to collaboratively and iteratively write documentation, saving time and enhancing synergy within agile teams.

This study raises the question of why documentation is often viewed as a dreaded task among agile software developers. It also explores the apparent contradiction where developers still recognize the importance of effective documentation in their work.

Keywords: documentation, agile software development, documentation tools

Kuviot

Sisällys

1 Johdanto

Dokumentaatiolla tarkoitetaan ohjelmistoon liittyvää relevanttia kirjoitettua tietoa, jonka avulla välitetään informaatiota ohjelmistosta. Dokumentaatio on siis ennen kaikkea kommunikaation väline ohjelmistokehittäjien sekä ohjelmistokehittäjien ja loppukäyttäjien välillä. Pätevän ja kattavan dokumentaation päivittäminen ja ylläpitäminen nopeasti muuttuvassa ohjelmistokehitysympäristössä ei aina ole mahdollista tai tarkoituksenmukaista (Forward ja Lethbridge 2002). Dokumentaatio tehostaa kuitenkin pitkässä juoksussa ohjelmistokehittäjien työtä (Voigt ym. 2016).

Dokumentaation ylläpitämisen tueksi on kehitetty erilaisia työkaluja. Näiden avulla voidaan helpottaa dokumentaation tekemistä ja ylläpitämistä, jotta sen tekemiseen kuluisi vähemmän aikaa ja vaivaa. Ylläpidetty ja tarkoituksenmukainen dokumentaatio helpottaa ohjelmistokehittäjien työtä, kommunikaatiota ja uusien kehittäjien perehdyttämistä (Islam, Hasan ja Eisty 2023).

Voigt ym. (2016) tekemän tutkimuksen mukaan hyvin pieni osa ohjelmistokehittäjien työajasta kuluu lähdekoodin kirjoittamiseen ja vielä pienempi osa dokumentaation tekemiseen. Tiedon etsimiseen taas kuluu hyvin suuri osa päivästä. Tiedon hankkimiseen käytettyä aika voi tällöin vähentyä, jos dokumentaatio on ajantasaista.

Dokumentaation kirjoittaminen on pitkään ja laajasti keskusteltu aihe ohjelmistokehittäjien keskuudessa, mikä on johtanut useisiin tutkimuksiin tämän aihealueen syvällisemmästä ymmärtämisestä. Tämä korostaa tarvetta syvemmälle ymmärrykselle dokumentaation merkityksestä, sen vaikutuksista ohjelmistokehityksessä ja parhaiden käytäntöjen tunnistamisesta.

Tässä tutkielmassa käsitellään, mitä dokumentaatio on, sen yhteyttä ketterään ohjelmistokehitykseen ja mitä nykyaikaisista ohjelmistoista dokumentoidaan. Tämän lisäksi tarkastellaan työkaluja, joita on kehitetty dokumentoinnin tekemisen ja ylläpidon tueksi.

2 Dokumentaatio ketterässä kehityksessä

Ketterän ohjelmistokehityksen ydintarkoituksena on karsia mahdollisimman paljon perinteiseen ohjelmistokehitykseen tyypillisesti liitettyä kankeutta. Karsimalla kankeutta voidaan edistää nopeaa reagointia muuttuviin olosuhteisiin kuten käyttäjien vaatimusten muutoksiin sekä aikataulujen muutoksiin. Perinteiset, vakiintuneet menetelmät, kuten vesiputousmalli, ovat liian jäykkiä ja hitaita, jotta ne voisivat vastata alati muuttuviin olosuhteisiin. Esimerkiksi, ketterässä kehityksessä tiimi voi päivittää sovelluksen toiminnallisuutta viikoittain asiakaspalautteen perusteella, kun taas vesiputousmallissa tällaiset muutokset odottaisivat seuraavaa projektivaihetta, mikä voisi kestää kuukausia. On olemassa erilaisia tyylejä tai tapoja harjoittaa ketterää ohjelmistokehitystä, kuten eXtreme Programming, Scrum, Lean, ominaisuuslähtöinen kehitys (DFD) ja monia muita. Vaikka edellä mainitut menetelmät eroavat toisistaan on niiden kaikkien ideana välttää kankeutta ja turhaa työtä (Erickson, Lyytinen ja Siau 2005; Dingsøyr ym. 2012).

Dokumentaatio on olennainen osa mitä tahansa kehitettävää ohjelmistoa (Cioch, Palazzolo ja Lohrer 1996). Dokumentaatio määritellään ihmisille luettavaksi tarkoitetuksi materiaaliksi, jolla välitetään informaatiota jostakin tietystä ohjelmistosta (Forward ja Lethbridge 2002). Dokumentaation tulisi myös tarjota perusteluita sille, miksi ohjelmisto on rakennettu juuri sillä tavalla (Selic 2009). Esimerkiksi, ohjelmistoprojektissa, jossa kehitetään asiakashallintajärjestelmää, dokumentaatio voisi selittää, miksi järjestelmä on suunniteltu tukemaan eri käyttäjärooleja ja miten tämä suunnittelupäätös auttaa eri käyttäjätyyppejä hallitsemaan asiakastietoja tehokkaasti. Tämä antaa kehittäjille ja sidosryhmille ymmärryksen siitä, miten järjestelmän ominaisuudet tukevat liiketoimintaprosesseja.

Ketterän kehityksen julistuksessa (Beck, Grenning ja Martin 2001) arvostetaan toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota. Tästä syystä jotkut ohjelmistokehittäjät pitävät dokumentaation kirjoittamista turhana tehtävänä (Prause ja Durdik 2012). Tämän kohdan kirjaimellinen tulkinta on kuitenkin harhaanjohtavaa sillä julistuksen historia- osassa julistuksen kirjoittavat painottavat, että dokumentaatio on tärkeä osa ohjelmistoa. Sen sijaan satojen sivujen mittaiseen ja harvoin käytettyyn dokumentaatioon ei kannusteta (Beck, Grenning ja Martin 2001).

Lähes kaikki ohjelmistokehittäjät kokevat hyvän dokumentaation tärkeäksi (Stettina ja Heijstek 2011). Voigt ym. (2016) tutkimuksessa todetaan, että myös ketterässä kehityksessä on oltava pätevää dokumentaatiota, jotta ohjelmistokehittäjät pystyvät tekemään työtään tehokkaasti. Ohjelmistokehittäjät kuitenkin saattavat pitää esimerkiksi lähdekoodin dokumentaatiota tärkeämmässä asemassa kuin ohjelmiston arkkitehtuurin dokumentaatiota (De Souza, Anquetil ja De Oliveira 2005).

Ohjelmistokehittäjien työajasta suurin osa ajasta menee johonkin muuhun kuin ohjelmakoodin kirjoittamiseen. Esimerkiksi Voigt ym. (2016) tekemän tutkimuksen mukaan vain 12 % ohjelmistokehittäjien ajasta käytetään ohjelmakoodin kirjoittamiseen. Samassa tutkimuksessa todetaan, että tiedon etsimiseen, digitaaliseen kommunikointiin ja keskusteluihin kuluu 33 % työajasta. Hyvä dokumentaatio siis voi vähentää tiedon etsimiseen käytettyä aikaa ja täten tehostaa ohjelmistokehittäjien työtä.

Kerran kirjoitettu dokumentaatio ei ole riittävä vaan sitä tulisi iteratiivisesti päivittää samalla kun ohjelmistoa kehitetään. Dokumentaation iteratiivinen päivittäminen voi myös parantaa lähdekoodin laatua (Stettina, Heijstek ja Fægri 2012). On kuitenkin esitetty, että vanhentunutkin dokumentaatio on hyödyllisempää kuin ei dokumentaatiota lainkaan (Lethbridge, Singer ja Forward 2003).

3 Dokumentaation ulottuvuudet

Voigt ym. (2016) mukaan ohjelmistokehittäjien tulisi keskittyä dokumentoimaan niitä asioita, mitkä ovat ohjelmiston toiminnan kannalta olemmaisimpa asioita. Dokumentoimalla kattavasti ohjelmiston eri osia pystytään kuitenkin helpottamaan tulevien ohjelmistokehittäjien työtä, kun ohjelmistoa ylläpidetään ja jatkokehitetään (Islam, Hasan ja Eisty 2023). Seuraavaksi käsitellään tarkemmin, mitä ulottuvuuksia nykyaikaisista ohjelmistoista dokumentoidaan sekä, millä perusteilla dokumentaation ulottuvuuksia on jaoteltu.

3.1 Dokumentaation jaottelu

Modernit ohjelmistot ovat isoja ja kompleksisia kokonaisuuksia, joten yhtenäisen ja kattavan dokumentaation kirjoittaminen voi olla hyvin haasteellista. Ei ole olemassa universaalia määrittelyä siitä, mitä ohjelmistoista tulisi dokumentoida. Kirjallisuudessa dokumentaatiosta on kuitenkin tunnistettu esimerkiksi seuraavia ulottuvuuksia: käyttäjätarinat, toiminnalliset ja ei-toiminnalliset vaatimukset, lähdekoodi, ohjelmiston arkkitehtuuri, suunnittelupäätökset, virhelokit, ohjelmiston arviointikokoukset ja teknisen velan dokumentointi (Islam, Hasan ja Eisty 2023; Behutiye, Rodriguez ja Oivo 2022; Antoniol ym. 2002).

Dokumentoitavat ulottuvuudet riippuvat siitä kenelle tietoa on tarkoitus välittää. Ohjelmistokehittäjille tarkoittu dokumentaatio kuvaa ohjelmiston suunnittelua, lähdekoodia, arkkitehtuuria, rajapintoja ja ohjelmiston toimintaa ohjelmistokehittäjän näkökulmasta (kts. Kuvio 1). Ohjelmistojen vaatimusmäärittelyssä dokumentoidaan toiminnallisia ja ei-toiminnallisia vaatimuksia. Toiminnalliset ja ei-toiminnalliset vaatimukset ovat siis olennainen osa, kun ohjelmistoa tai uusia toimintoja suunnitellaan (Behutiye, Rodriguez ja Oivo 2022). Käyttöoppaat sen sijaan selittävät ohjelmiston toimintaa loppukäyttäjille eli miten ohjelmistoa tulisi käyttää (Aghajani ym. 2019).

3.2 Käyttäjätarinat

Käyttäjätarinat on laajalti käytetty tapa dokumentoida ketterässä kehityksessä (Lucassen ym. 2016). Käyttäjätarina on lyhyt teksti, jossa kuvataan jokin vaatimus, jota ohjelmistolta odotetaan. Lucassen ym. (2016) mukaan käyttäjätarinan tulisi muodostaa kokonainen lause, joka sisältää käyttäjän roolin ja keskittyy yhteen asiaan tai ongelmaan. Lisäksi käyttäjätarinaan voi sisältyä syy sille, miksi toiminto on tärkeä. Esimerkiksi "Projektipäällikkönä haluan asettaa hälytysrajan mittalaitteelle"tai "Päivystävänä insinöörinä haluan saada tekstiviestin antamaani numeroon, jos mittalaitteen arvo ylittää ennalta määritetyn rajan".

Koska käyttäjätarinat ovat lyhyitä ja ytimekkäitä ne toimivat enemmän oikotienä kattavammalle dokumentaatiolle. Ne edustavat tärkeää käyttäjälähtöisiä vaatimuksia, jotka ovat tärkeitä ohjelmistokehittäjille (Islam, Hasan ja Eisty 2023). Käyttäjätarinat siis auttavat ohjelmistokehittäjiä keskittymään niihin ominaisuuksiin, jotka ovat loppukäyttäjän kannalta olennaisimpia.

3.3 Toiminnalliset ja ei-toiminnalliset vaatimukset

Toiminnallisilla vaatimuksilla tarkoitetaan loppukäyttäjien määrittämiä vaatimuksia, joita loppukäyttäjät olettavat pääsevänsä käyttämään. Toiminnalliset vaatimukset näyttävät päällisin puolin hyvin samalta kuin käyttäjätarinat. Näiden kahden erona on se, että käyttäjätarinat ovat hyvin lyhyitä, kun taas toiminnalliset vaatimukset syventyvät enemmän ohjelmiston teknisiin vaatimuksiin (Pasuksmit, Thongtanunam ja Karunasekera 2021). Esimerkki toiminnallisesta vaatimuksesta voisi olla "Käyttäjänä haluan pystyä lataamaan raportit suoraan järjestelmästä Excel-muodossa, jotta voin tehdä nopeita datan analysointeja ilman erillistä tietojen siirtoa."Tämä vaatimus tarjoaa yksityiskohtaisemman ja teknisemmän kuvauksen siitä, mitä ohjelmistolta vaaditaan.

Ei-toiminnalliset vaatimukset ovat tärkeä osa ohjelmistokehityksessä ja ohjelmiston dokumentaatiossa. Laatuvaatimuksilla tarkoitetaan ohjelmistolta toivottuja ominaisuuksia kuten esimerkiksi luotettavuus, turvallisuus ja ylläpidettävyys (Aljallabi ja Mansour 2016). Ohjelmistokehittäjät pitävät laatuvaatimusten dokumentaatiota tärkeänä osana ohjelmistokehitystä. Niiden dokumentoinnin avulla voidaan selventää laatuvaatimuksia, parantaa päätöksen-

tekoa ja helpottaa laadunvaatimusten valvontaa (Behutiye ym. 2020). Behutiye ym. (2017) tekemän tutkimuksen mukaan ei-toiminnallisia vaatimuksia ei dokumentoida yhtä tarkasti ja usein kuin toiminnallisia vaatimuksia, mikä vaikeuttaa niiden jäljitettävyyttä. Esimerkki laatuvaatimuksesta voisi olla, että ohjelmiston on kyettävä palautumaan virhetilanteista minuutin sisällä 99 % ajasta.

3.4 Tekninen dokumentaatio

Tekninen dokumentaatio on tarkoitettu kehittäjille ohjelmiston ylläpitämisen ja jatkokehityksen tueksi. Tekniseen dokumentaatioon voi laskea esimerkiksi lähdekoodin, käyttöliittymän rakenteen, järjestelmän arkkitehtuurin ja API-viiteoppaiden dokumentaation (Islam, Hasan ja Eisty 2023). Allison, Turner ja Kwasny (2021) tekemän tutkimuksen mukaan kokemattomampien ohjelmistokehittäjien oli vaikeampaa oppia uusia teknologioita puutteellisen dokumentaation takia. Erityisesti dokumentaation puute haittasi silloin, kun ohjelmistokehittäjien piti saada useat eri komponentit toimimaan keskenään. Tämä painottaa puolestaan hyvän teknisen dokumentaation tarvetta.

Lähdekoodia dokumentoidaan suoraan lähdekoodin kommentteihin, mikä auttaa kehittäjiä ymmärtämään kirjoitettua koodia paremmin (Steidl, Hummel ja Juergens 2013). Lisäksi lähdekoodin dokumentointi on tärkeää ohjelmistokehityksessä, koska se antaa mahdollisuuden löytää lähdekoodista se kohta, joka ei toimi halutulla tavalla (Antonino ym. 2014). De Souza, Anquetil ja De Oliveira (2005) tekemän kyselytutkimuksen mukaan lähdekoodin ja sen kommenttien ymmärtäminen on tärkein asia, kun ylläpidetään ohjelmistoa. Lähdekoodin dokumentointi on erityisen tärkeää kun ohjelmistoa kehitetään useissa paikoissa ja useiden organisaatioiden välillä. Hyvä dokumentaatio helpottaa kommunikaatiota erityisesti silloin, kun projektissa työskentelevät asuvat eri aikavyöhykkeillä (Phalnikar, Deshpande ja Joshi 2009).

Samalla tavalla kuin lähdekoodin dokumentointi, myös ohjelmiston arkkitehtuurin dokumentointi on tärkeää ohjelmiston ylläpidettävyyden kannalta. Ohjelmiston arkkitehtuuri on korkean tason kuvaus, jonka tarkoituksena on määritellä ohjelmiston rakenteelliset komponentit ja niiden välistä vuorovaikutusta. Ohjelmiston arkkitehtuurin dokumentaatio on usein suuri ja kompleksinen ja koostuu joskus jopa useista sadoista sivuista (Hadar ym. 2013).

Ohjelmiston arkkitehtuuria voidaan kuvata erilaisilla kaavioilla, kuten esimerkiksi UML-kaavioilla (Unified Modeling Language). Kaavioiden lisäksi on olemassa arkkitehtuurin kuvauskieliä (ADL), joita käytetään ohjelmistoarkkitehtuurin kuvaamiseen, analysointiin ja dokumentointiin (Abbasi ym. 2017). Ohjelmiston arkkitehtuurin dokumentaatio voi esimerkiksi sisältää tietoa siitä, miten data virtaa järjestelmän eri osien välillä. Siinä voidaan kuvata prosesseja ja rajapintoja, jotka mahdollistavat datan liikkumisen komponenttien välillä sekä, miten järjestelmän turvallisuutta varmistetaan ohjelmiston eri tasoilla.

Ohjelmiston kompleksisuuden takia sen arkkitehtuurin kuvaaminen on usein vaikein asia dokumentoida moderneissa ohjelmistoissa (Hadar ym. 2013), (Islam, Hasan ja Eisty 2023). De Souza, Anquetil ja De Oliveira (2005) tutkimuksen mukaan ohjelmiston arkkitehtuurin dokumentaatiota tarvitaan harvemmin kuin lähdekoodin dokumentaatiota. Tätä perustellaan sillä, että arkkitehtuurin dokumentaatiota käytetään kerran, jotta saadaan hyvä ymmärrys ohjelmiston laajasta toiminnasta. Tämän jälkeen dokumentaatioon ei tarvitse enää palata. Tutkimuksessa kuitenkin painotetaan sitä, että arkkitehtuurin dokumentointi on silti tärkeää vaikka se ei olekaan usein käytetty.

Ohjelmointirajapinta (engl. application programming interface, lyh. API) on tärkeä osa ohjelmistojen rakentamisesta. Ohjelmistokehittäjät käyttävät API:ja työssään päivittäin (Ajam, Rodriguez ja Benatallah 2021). On tärkeää, että API-dokumentaatio on pätevä ja ajantasainen ohjelmistokehittäjien työn helpottamiseksi. Tämän lisäksi se helpottaa uusien ohjelmistokehittäjien perehdyttämistä (Islam, Hasan ja Eisty 2023). API-dokumentaatiossa voidaan esimerkiksi antaa yleiskatsaus API:n käytöstä, kuvata pyyntöjen formaattia ja luetella mahdollisia virhekoodeja.

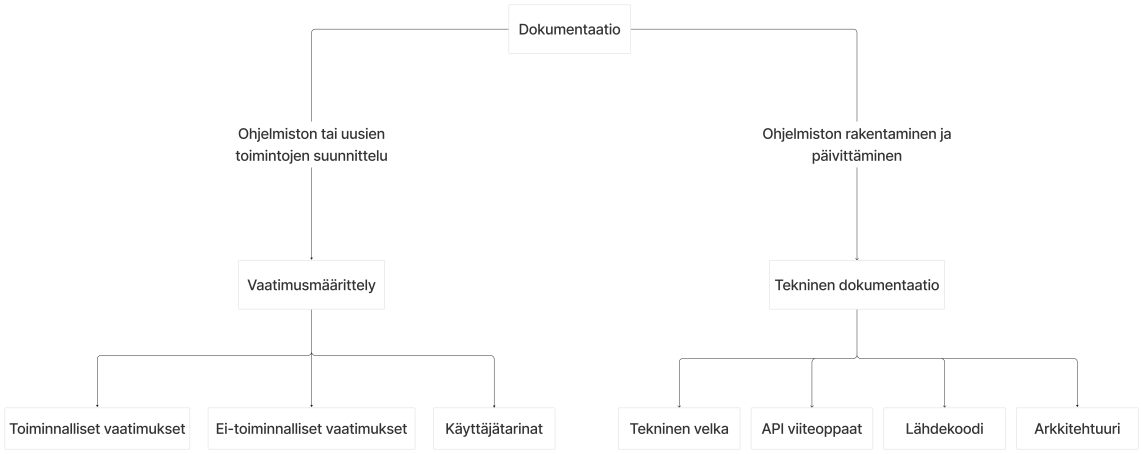
3.5 Teknisen velan dokumentaatio

Teknisellä velalla tarkoitetaan ohjelmistokehityksen toimien seurauksia, jotka tahallisesti tai tahattomasti asettavat etusijalle asiakasarvon tai projektin rajoitteet, kuten aikataulut, teknisen toteutuksen sijaan. Tämä sisältää esimerkiksi testikattavuuden saavuttamisen ja ylläpitämisen sekä koodin laajennettavuuden. Tekninen velka on siis analogia taloudelliselle velalle, johon liittyy käsitteitä velan tasoista, velan kertymisestä ajan myötä, velan todennäköiset

seuraukset ja paine velan maksamisesta tulevaisuudessa (Holvitie ym. 2018). Esimerkiksi, jos ohjelmistokehitystiimi päättää käyttää nopeaa, mutta vähemmän skaalautuvaa ratkaisua tietokantakyselyihin, jotta ohjelmisto saadaan nopeammin markkinoille, he keryttävät teknistä velkaa. Tulevaisuudessa, kun käyttäjämäärät nousevat, tämä voi aihettaa ongelmia suorityskyvyn kanssa.

Teknisen velan dokumentaatio on tärkeää, koska tällöin ohjelmistokehittäjät, jotka ylläpitävät ja jatkokehittävät kyseistä ohjelmistoa, ovat tietoisia teknisen velan aiheuttamista mahdollisista rajoitteista tai ongelmista(Islam, Hasan ja Eisty 2023). Holvitie ym. (2018) mukaan teknistä velkaa tulisi dokumentoida hyvin aktiivisesti, mutta sen aiheuttamien konkreettisen haittojen dokumentointi on haasteellista.

Nykyaikaisten ohjelmistojen laajuuden ja kompleksisuuden takia ne tarvitsevat kattavan ja erityisesti tarkoituksenmukaisen dokumentaation. Tämä voi olla ristiriidassa ketterän kehityksen julistuksen kanssa, jossa arvostetaan toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota (Beck, Grenning ja Martin 2001).



Kuvio 1. Kaaviokuva ohjelmistokehittäjille suunnatun dokumentaation eri ulottuvuuksista (mukaillen Islam, Hasan ja Eisty (2023))

4 Työkalut dokumentaatiossa

Osa ohjelmistokehittäjistä ei pidä dokumentaation kirjoittamista tarpeellisena tehtävänä (Stettina, Heijstek ja Fægri 2012). Sen lisäksi ulkopuolisten tekijöiden, kuten aikapaineiden, takia ohjelmistokehittäjät joutuvat jättämään dokumentaation alemmaksi prioriteettilistalla (Prause ja Durdik 2012; Lethbridge, Singer ja Forward 2003). Dokumentaation tekemisen helpottamiseksi on olemassa useita erilaisia työkaluja ja käytänteitä, joilla pyritään vähentämään dokumentaation tekemiseen kuluvaa aikaa.

Ohjelmistojen laajuuden takia niistä pitäisi dokumentoida huomattavan paljon asioita. Tämän takia ei ole olemassa vain yhtä työkalua, jolla voisi ratkaista tämän ongelman. Työkalut, joita käytetään dokumentoinnin tukena, ovat esimerkiksi ohjelmat, jotka tuottavat dokumentaatiota lähdekoodin perusteella, erilaiset wikit sekä kaaviot ja diagrammit ohjelmiston arkkitehtuurin kuvaamisen avuksi (Islam, Hasan ja Eisty 2023).

Lähdekoodin kommentteja pidetään yhtenä tärkeimmistä asioista ohjelmistojen dokumentaatiossa (De Souza, Anquetil ja De Oliveira 2005). Lähdekoodien kommenteista luotaviin dokumentaatioihin on kehitetty ohjelmia esimerkiksi JavaDoc ja Doxygen. Nämä ohjelmat luovat minimaalista dokumentaatiota suoraan kehittäjien lähdekoodiin kirjoittamista dokumentaatiokommenteista (Islam, Hasan ja Eisty 2023). Näiden ohjelmien tuottama minimaalinen dokumentaatio sopii hyvin ketterään ohjelmistokehitykseen. Tällöin ohjelmistokehittäjien työaikaa ei kulu niin paljon dokumentaation kirjoittamiseen.

Wikeillä tarkoitetaan yhteistyöllisiä verkkosivuja, joita useat eri käyttäjät voivat muokata. Erilaisten wikien käyttäminen ohjelmiston dokumentointiin on suosittua ohjelmistokehittäjien keskuudessa niiden yksinkertaisuuden takia (Voigt, Huttemann ja Gohr 2016). Wikien helppokäyttöisyydellä pyritään vähentämään eroavaisuutta lähdekoodin ja dokumentaation välillä. Wikien hyvänä puolena on myös se, että joitain niistä, esimerkiksi sprintDoc ja XSDoc, voi yhdistää ohjelmistokehitysympäristöihin, kuten Visual Studio Codeen (Islam, Hasan ja Eisty 2023). Tällöin ohjelmistokehittäjän ei tarvitse poistua kehitysympäristöstään kirjoittaakseen dokumentaatiota tai kommentteja juuri kirjoittamastaan koodista.

Ohjelmistojen arkkitehtuuri on hyvin kompleksinen kokonaisuus ja se on usein myös laajasti

dokumentoitu osa ohjelmistoa (Hadar ym. 2013). Ohjelmiston arkkitehtuuria voidaan kuvata esimerkiksi UML-kaavioilla tai arkkitehtuurin kuvauskielillä, esimerkiksi Unicon, Adage ja Drawin. Arkkitehtuurin kuvauskielet tarjoavat selkeän syntaksin ja tukea ohjelmiston arkkitehtuurin jatkokehittämiselle. Näitä kieli käytetään eniten turvallisuuden kannalta kriittisissä ohjelmistoissa kuten prosessien hallinnassa, lääketieteessä ja avaruuslennoissa. Nämä kielet eivät siis ole laajasti käytössä ohjelmistokehityksessä. Ohjelmiston arkkitehtuuria dokumentoidessa kallistutaan usein käyttämään UML-kaavioita niiden selkeyden ja kustannustehokkuuden takia (Abbasi ym. 2017).

Arkkitehtuurin dokumentoinnin tueksi on kehitetty myös ohjelmia, jotka luovat dokumentaatiota perustuen lähdekoodiin, esimerkiksi Rubin ja Rubin (2011) esittämä ratkaisu, jossa muutokset lähdekoodissa muuttavat dokumentaatiota ja toisin päin. Tällaiset työkalut helpottavat ohjelmistojen arkkitehtuurin dokumentointia.

5 Johtopäätökset

Ollaan todettu, että dokumentaatio on laajalti keskusteltu ja tutkittu aihe ketterässä ohjelmistokehityksessä. Moderneissa ohjelmistoissa on paljon eri elementtejä, joita voidaan dokumentoida. Dokumentointityyli riippuu siitä, kenelle dokumentaatio on tarkoitettu. Esimerkiksi ohjelmistokehittäjille tarkoitettu dokumentaatio eroaa huomattavasti loppukäyttäjille tarkoitetusta dokumentaatiosta. Dokumentaation tueksi on kehitetty useita eri ohjelmia, jotka pyrkivät helpottamaan tiettyä osa-aluetta dokumentoinnissa. Ohjelmistojen laajuuden takia ei kuitenkaan ole yhtä työkalua, jolla voisi tehdä koko ohjelmiston dokumentaation.

On myös todettu, että ohjelmistokehittäjät eivät pidä dokumentaation kirjoittamista korkeana prioriteettina. Tämä yhdistettynä tiukkoihin aikarajoihin johtaa vanhentuneeseen ja epäpätevään dokumentaatioon. Puutteellinen dokumentaatio puolestaan lisää tiedonhakuun käytettävää aikaa tulevaisuudessa. Tästä voi muodostua ikävä kierre, joka on vaikeaa ratkaista rajallisilla resursseilla.

Mielenkiintoiseksi huomioksi voidaan nostaa myös se, että suurin osa ohjelmistokehittäjistä pitää minimaalisen, mutta pätevän dokumentaation ylläpitämistä ja kirjoittamista tärkeänä. Dokumentaation tekemistä kuitenkin vältellään tai se annetaan vähemmän pätevien kehittäjien vastuulle (Stettina, Heijstek ja Fægri 2012). Dokumentaation laiminlyömiselle ei myöskään anneta tarkempia perusteluita lukuun ottamatta aikarajoitteita ja budjettia.

Ohjelmistokehittäjät vaikuttavat siis suhtautuvan dokumentaatioon hyvin ristiriitaisesti. Tämä nostaa kysymyksen siitä, miten ohjelmistokehittäjät saadaan kirjoittamaan tarpeellista dokumentaatiota. Onko ratkaisu asennemuutos vai uuden työkalun kehittäminen, joka tekee hyvän dokumentaation ilman, että ohjelmistokehittäjän tarvitse varata enempää aikaa sen kirjoittamiseen.

Lähteet

Abbasi, M.A., D.-E.-B. Batool, R. Butt ja T.M. Anjum. 2017. “Comparative Analysis of Software Architecture Documentation and Architecture Languages” [kielellä English], 199–204. ISBN: 978-1-5090-5753-5. https://doi.org/10.1109/APWC-on-CSE.2016.041.

Aghajani, E., C. Nagy, O.L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota ja M. Lanza. 2019. “Software Documentation Issues Unveiled” [kielellä English], nide 2019-May, 1199–1210. ISSN: 0270-5257. ISBN: 978-1-72810-869-8. https://doi.org/10.1109/ICSE.2019.00122.

Ajam, George, Carlos Rodriguez ja Boualem Benatallah. 2021. “Scout-bot: Leveraging API Community Knowledge for Exploration and Discovery of API Learning Resources” [kielellä en]. Number: 2, *CLEI Electronic Journal* 24, numero 2 (elokuu): 5:1–5:24. ISSN: 0717-5000, viitattu 12. maaliskuuta 2024. https://doi.org/10.19153/cleiej.24.2.5. https://www.clei.org/cleiej/index.php/cleiej/article/view/498.

Aljallabi, B.M. ja A. Mansour. 2016. “Enhancement approach for non-functional requirements analysis in Agile environment” [kielellä English], 428–433. ISBN: 978-1-4673-7869-7. https://doi.org/10.1109/ICCNEEE.2015.7381407.

Allison, M., S. Turner ja M. Kwasny. 2021. “Assessing Cognitive Load for Junior Software Engineers: A Mixed-Method Study” [kielellä English], 883–888. ISBN: 978-1-66545-841-2. https://doi.org/10.1109/CSCI54926.2021.00206.

Antonino, Pablo Oliveira, Thorsten Keuler, Nicolas Germann ja Brian Cronauer. 2014. “A Non-invasive Approach to Trace Architecture Design, Requirements Specification and Agile Artifacts”. Teoksessa *2014 23rd Australian Software Engineering Conference,* 220–229. ISSN: 2377-5408. Huhtikuu. Viitattu 11. maaliskuuta 2024. https://doi.org/10.1109/ASWEC.2014.30. https://ieeexplore.ieee.org/document/6824127.

Antoniol, G., G. Canfora, G. Casazza, A. De Lucia ja E. Merlo. 2002. “Recovering traceability links between code and documentation” [kielellä English]. *IEEE Transactions on Software Engineering* 28 (10): 970–983. ISSN: 0098-5589. https://doi.org/10.1109/TSE.2002.1041053.

Beck, Kent, James Grenning ja Robert C. Martin. 2001. *Manifesto for Agile Software Development.* Viitattu 6. helmikuuta 2024. https://agilemanifesto.org/.

Behutiye, W., P. Karhapää, D. Costal, M. Oivo ja X. Franch. 2017. “Non-functional requirements documentation in agile software development: Challenges and solution proposal” [kielellä English]. ISBN: 9783319699257, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10611 LNCS:515–522. ISSN: 0302-9743. https://doi.org/10.1007/978-3-319-69926-4_41.

Behutiye, W., P. Rodriguez ja M. Oivo. 2022. “Quality Requirement Documentation Guidelines for Agile Software Development” [kielellä English]. *IEEE Access* 10:70154–70173. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2022.3187106.

Behutiye, W., P. Rodriguez, M. Oivo, S. Aaramaa, J. Partanen ja A. Abherve. 2020. “How agile software development practitioners perceive the need for documenting quality requirements: A multiple case study” [kielellä English], 93–100. ISBN: 978-1-72819-532-2. https://doi.org/10.1109/SEAA51224.2020.00025.

Cioch, Frank A., Michael Palazzolo ja Scott Lohrer. 1996. “Documentation suite for maintenance programmers” [kielellä English], 286–295.

De Souza, S.C.B., N. Anquetil ja K.M. De Oliveira. 2005. “A study of the documentation essential to software maintenance” [kielellä English], 68–75. ISBN: 978-1-59593-175-7. https://doi.org/10.1145/1085313.1085331.

Dingsøyr, T., S. Nerur, V. Balijepally ja N.B. Moe. 2012. “A decade of agile methodologies: Towards explaining agile software development” [kielellä English]. *Journal of Systems and Software* 85 (6): 1213–1221. ISSN: 0164-1212. https://doi.org/10.1016/j.jss.2012.02.033.

Erickson, John, Kalle Lyytinen ja Keng Siau. 2005. “Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research”. *J. Database Manag.* 16 (lokakuu): 88–99. https://doi.org/10.4018/jdm.2005100105.

Forward, A. ja T.C. Lethbridge. 2002. “The Relevance of Software Documentation, Tools and Technologies: A Survey” [kielellä English], 26–33. ISBN: 978-1-58113-594-7.

Hadar, Irit, Sofia Sherman, Ethan Hadar ja John J. Harrison. 2013. “Less is more: Architecture documentation for agile development”. Teoksessa *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE),* 121–124. Toukokuu. Viitattu 12. maaliskuuta 2024. https://doi.org/10.1109/CHASE.2013.6614746. https://ieeexplore.ieee.org/abstract/document/6614746.

Holvitie, J., S.A. Licorish, R.O. Spínola, S. Hyrynsalmi, S.G. MacDonell, T.S. Mendes, J. Buchan ja V. Leppänen. 2018. “Technical debt and agile software development practices and processes: An industry practitioner survey” [kielellä English]. *Information and Software Technology* 96:141–160. ISSN: 0950-5849. https://doi.org/10.1016/j.infsof.2017.11.015.

Islam, M.A., R. Hasan ja N.U. Eisty. 2023. “Documentation Practices in Agile Software Development: A Systematic Literature Review” [kielellä English], 266–273. ISBN: 9798350345889. https://doi.org/10.1109/SERA57763.2023.10197828.

Lethbridge, T.C., J. Singer ja A. Forward. 2003. “How Software Engineers Use Documentation: The State of the Practice” [kielellä English]. *IEEE Software* 20 (6): 35–39. ISSN: 0740-7459. https://doi.org/10.1109/MS.2003.1241364.

Lucassen, G., F. Dalpiaz, J.M.E.M. van der Werf ja S. Brinkkemper. 2016. “Improving agile requirements: the Quality User Story framework and tool” [kielellä English]. *Requirements Engineering* 21 (3): 383–403. ISSN: 0947-3602. https://doi.org/10.1007/s00766-016-0250-x.

Pasuksmit, J., P. Thongtanunam ja S. Karunasekera. 2021. “Towards Just-Enough Documentation for Agile Effort Estimation: What Information Should Be Documented?” [Kielellä English], 114–125. ISBN: 978-1-66542-882-8. https://doi.org/10.1109/ICSME52107.2021.00017.

Phalnikar, R., V.S. Deshpande ja S.D. Joshi. 2009. “Applying agile principles for distributed software development” [kielellä English], 535–539. ISBN: 978-0-7695-3516-6. https://doi.org/10.1109/ICACC.2009.93.

Prause, C.R. ja Z. Durdik. 2012. “Architectural design and documentation: Waste in agile development?” [Kielellä English], 130–134. ISBN: 978-1-4673-2352-9. https://doi.org/10.1109/ICSSP.2012.6225956.

Rubin, E. ja H. Rubin. 2011. “Supporting agile software development through active documentation” [kielellä English]. *Requirements Engineering* 16 (2): 117–132. ISSN: 1432-010X. https://doi.org/10.1007/s00766-010-0113-9.

Selic, B. 2009. “Agile documentation, anyone?” [Kielellä English]. *IEEE Software* 26 (6): 11–12. ISSN: 0740-7459. https://doi.org/10.1109/MS.2009.167.

Steidl, D., B. Hummel ja E. Juergens. 2013. “Quality analysis of source code comments” [kielellä English], 83–92. ISBN: 978-1-4673-3092-3. https://doi.org/10.1109/ICPC.2013.6613836.

Stettina, C.J. ja W. Heijstek. 2011. “Necessary and neglected? An empirical study of internal documentation in agile software development teams” [kielellä English], 159–166. ISBN: 978-1-4503-0936-3. https://doi.org/10.1145/2038476.2038509.

Stettina, C.J., W. Heijstek ja T.E. Fægri. 2012. “Documentation work in agile teams: The role of documentation formalism in achieving a sustainable practice” [kielellä English], 31–40. ISBN: 978-0-7695-4804-3. https://doi.org/10.1109/Agile.2012.7.

Voigt, S., D. Huttemann ja A. Gohr. 2016. “SprintDoc: Concept for an agile documentation tool” [kielellä English], nide 2016-July. ISSN: 2166-0727. ISBN: 978-989-98434-6-2. https://doi.org/10.1109/CISTI.2016.7521550.

Voigt, S., J. Von Garrel, J. Müller ja D. Wirth. 2016. “A Study of Documentation in Agile Software Projects” [kielellä English], nide 08-09-September-2016. ISSN: 1949-3770. ISBN: 978-1-4503-4427-2. https://doi.org/10.1145/2961111.2962616.