Musa Jallow

Lähdekoodin kommentointi

Tietotekniikan kandidaatintutkielma

8. syyskuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Musa Jallow

Yhteystiedot: musa.j.jallow@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Lähdekoodin kommentointi

Title in English: Source code commenting

Työ: Kandidaatintutkielma

Sivumäärä: 30+0

Tiivistelmä: Tässä tutkielmassa selvitetään, miten alan kirjallisuudessa lähestytään lähdekoodin kommentointia. Tutkielman tavoite on tuoda esille hyviksi todettuja tapoja kommentoida lähdekoodia. Lisäksi tarkastellaan erilaisia normeja ja yleisiä käytänteitä liittyen lähdekoodin kommentointiin. Lopputulemana lukijalle on selvää, kenen vastuulla on kommentoida koodia, miten koodia tulisi kommentoida ja milloin. Lisäksi lukijalle tulee selväksi minkälaisia työkaluja systemaattiseen ja laadukkaaseen kommentointiin on tarjolla.

Avainsanat: Lähdekoodi, dokumentointi, kommentointi, koodaaminen ja ohjelmointi

Abstract: This study takes a look at how source code commenting is seen in the literature in the field of IT. The aim of this study is to bring up the best practices doing source code commenting. In addition, the study takes a look at different kinds of norms and common practices related to source code commenting. The end result is to have an in-depth view who is in charge of code commenting, how and when you should comment, and which kind of tools you can use to get systematic and quality source code comments.

Keywords: Source code, documentation, commenting and programming

Kuviot

Sisältö

1 Johdanto

Ohjelmistot ovat kehittyneet alati monimutkaisemmiksi ja suuremmiksi kokonaisuuksiksi. Nykypäivän ohjelmistokehityksessä yhä suuremmat tiimit vastaavat ohjelmiston kehitysprosessista ja tästä syystä kokonaisuuden hallinta vaikeutuu. Dokumentoinnista on tullut tärkeä osa ohjelmistotuotantoa. Kattava ja kirjallinen suunnitelma ohjelmistonkehityksestä ja -kehitysprosessista on välttämätöntä nykypäivän ohjelmistokehityksessä. (Wu ja Simmons 2000; ISO/IEC/IEEE 2015, s. 14, 19, 21–22; Wiegers ja Beatty 2013, s. 181, 61–64) Tähän on kehitelty erilaisia malleja ja toimintatapoja. Yksi hyväksi todettu menetelmä on lähdekoodin kommentointi. Erityisesti monimutkaiset funktiot ja laajat kokonaisuudet tarvitsevat lähdekoodissa esiintyviä kommentteja. (Ahuja, Dhaiya ja Sadana 2013; Steidl, Hummel ja Juergens 2013)

Tutkielmassa tarkastellaan kommentointia osana dokumentaation viitekehystä. Tutkielmassani keskityn lähdekoodin kommentointiin. Tarkastelen kuitenkin myös dokumentointia ylemmän tason käsitteenä. Dokumentoinnin kokonaisuuteen kuuluvat kaikki dokumentoinnin erilaiset muodot, mukaan luettuna lähdekoodin kommentointi. Kandidaatin tutkielmassani keskityn ainoastaan järjestelmään liittyvään dokumentointiin.

Ohjelmistotuotanto ympäristöissä on suuri tarve laadukkaalle lähdekoodin kommentoinnille. On tärkeää tuoda hyviksi todettuja käytänteitä esille. (Wang, Wang ja Wang 2019). Tutkielmassa tuodaan esille alan kirjallisuudessa esiin nousevia käytänteitä, sekä kuinka lähdekoodin kommentointiin suhtaudutaan. Tutkielmassa nostetaan myös esille, mitä dokumentaatiolla tarkoitetaan, mitä lähdekoodin kommentoinnilla tarkoitetaan ja minkälaisia lähdekoodin kommentoinnin toimintatapoja alan kirjallisuudessa nousee esille. Lisäksi esille tuodaan lähdekoodin kommentoinnin vaikutuksia ohjelmistokehityksessä ja erityisesti ohjelmiston ylläpidon näkökulmasta.

Tutkielman tutkimusmenetelmänä on kirjallisuuskartoitus. Vertailen esiin nousaavia tapoja lähestyä lähdekoodin kommentointia. Hyödynnän tieteellisissä artikkeleissa esiin nousevia käsitteitä, mielipiteitä, kokemuksia, sekä tutkimustuloksia aiheeseen liittyen. Pyrin löytämään yhtäläisyyksiä ja eroavaisuuksia näkökulmissa sekä erilaisia termejä, joiden kautta

selitän laajempia käsitteitä. Lisäksi pyrin löytämään tutkimusten kautta hyviksi todettuja toimintatapoja ja käytänteitä. Tutkielmassani vastaan kysymyksiin:

1. Kuka lähdekoodia kommentoi?
2. Miten ja minne lähdekoodia tulisi kommentoida?
3. Miten lähdekoodin kommentointi kuvataan tieteellisessä kirjallisuudessa?

Tutkimusaineistona käytetään IT-alan kirjallisuutta sekä kommentointiin ja dokumentaatioon liittyviä tutkimuksia. Teokset on valittu laaja-alaisesti, jotta saadaan tarjottua mahdollisimman kattava kirjallisuuskatsaus. Tutkielmassa yhdeksän artikkelia toimivat tutkielman varsinaisena aineistona analyysille. Artikkelit on julkaistu vuoden 2003 ja vuoden 2019 välillä. Kirjallisuuskatsausta varten lähdeaineistoa on etsitty Jyväskylän Yliopiston kirjaston JYKDOK hakupalvelusta sekä IEEE Xplore -digitaalikirjastosta. Näiden lisäksi on etsitty tutkimuksen kannalta relevanttia aineistoa myös muista Jyväskylän Informaatioteknologian tiedekunnan tieteellisesti hyväksytyistä tietokannoista. Lisäksi teoksien validiteetti on tarkastettu käyttäen Julkaisufoorumi-sivustoa.

2 Dokumentointi

Luvussa esittelen dokumentointia ja mitä sillä tarkoitetaan. Tarkastelen myös esiin nousevaa termistöä. Pohjaan havaintoni erityisesti Wiegers ja Beatty (2013) kirjaan; Software requirements.

2.1 Dokumentaatiosta yleisesti

Wiegers ja Beatty (2013, s. 181) kuvailevat dokumentaation olevan sopimus sidosryhmien välillä rakennettavasta tuotteesta. Dokumentaatio on mitä tahansa kirjoitettua tai kuvitettua informaatiota, joka pyrkii kuvailemaan, määrittelemään tai raportoimaan toiminnallisuudesta, vaatimuksista, prosesseista tai tuloksista (Wiegers ja Beatty 2013, s. 8; ISO/IEC/IEEE 2015, s. 3; Lethbridge ym. 2003).

Dokumentointi on tärkeä osa jokaista projektia. Se tarjoaa informaatiota niin kehittäjille, sidosryhmille kuin myös asiakkaille. Projektin alkuvaiheessa luodut dokumentit luovat perustan sille, miten projektin elinkaari viedään alusta loppuun (Wiegers ja Beatty 2013, s. 14).

ISO/IEC/IEEE (2015, s. 9–11) standardin mukaan ohjelmiston kehittäjän tulisi dokumentoida ohjelman toiminnan sekä ohjelman kehityksen kannalta olennaista informaatiota. Dokumentaation tulisi antaa informaatiota niin suunnittelu- ja toteutusprosesseista aikatauluineen kuin ohjelmiston toiminnallisuudesta, käytöstä ja ylläpidosta. Lisäksi dokumentaation tulisi antaa informaatiota muusta ohjelmistokehittäjän näkökulmasta tärkeästä informaatiosta. (De Souza, Anquetil ja De Oliveira 2006, s. 31–33). De Souza, Anquetil ja De Oliveira (2006, s. 32) kuvailevat ohjelmisto dokumentaation tuovan esille informaatiota ohjelmistosta ja tämän informaation olevan tarkoitettu ihmisille luettavaksi.

Ohjelmistojen kehityksessä dokumentaatio alkaa jo projektin suunnitteluvaiheessa. Wu ja Simmons (2000) mukaan ennen projektin aloittamista projekti tulee suunnitella hyvin. Tämä on yksi kriittisimmistä vaiheista, sillä nykyaikaisten monimutkaisen ohjelmistojen hahmottaminen etukäteen voi olla vaikeaa. Lisäksi dokumentoitu suunnitelma tarjoaa työkalut

projektin hallintaan. (Wu ja Simmons 2000).

Ahuja, Dhaiya ja Sadana (2013) mukaan dokumentaatio on ulkoista ja sisäistä dokumentointia, kommentteja sekä readme -tiedostoja (suom. lue minut). Tutkielmassani käytän readme -tiedostosta sen englanninkielistä ilmaisua.

Ulkoinen dokumentaatio kuvailee tapaa, jolla ohjelmisto on kehitetty. Sisäinen dokumentaatio taas kertoo, miten koodia tulisi toteuttaa. Näiden lisäksi on kommentit, jotka kertovat koodin toteutuksesta. Readme -tiedostosta ilmenee ohjelman tarkoitus, funktionaalisuus sekä tiedot ulkoisista riippuvuuksista. (Ahuja, Dhaiya ja Sadana 2013). Sisäinen dokumentaatio kattaa tiedon siitä, millä tavalla ohjelmistokehittäjien tulisi lähdekoodia tehdä (Ahuja, Dhaiya ja Sadana 2013). Tähän liittyy vahvasti tieto siitä, miten lähdekoodia kommentoidaan. Seuraavassa luvussa tarkastellaan sisäistä dokumentaatiota ohjelmistokehityksessä. Erityisesti tarkastellaan sitä, miten ohjelmiston toteutusta ja lähdekoodin kommentointia ohjataan erilaisten dokumenttien kautta.

2.2 Dokumentointi ohjelmistokehityksessä

Ohjelmistokehityksessä tarvitaan erilaisia dokumentteja, joiden avulla ohjelmiston toteutusta lähdetään suunnittelemaan. Tärkeä seikka ohjelmistokehityksessä on luoda dokumentti, jossa kuvaillaan yrityksen tai alan normeja (Wiegers ja Beatty 2013, s. 128, 291–292). Dokumentti on osa yrityksen sisäistä dokumentaatiota, ja tarjoaa informaatiota ohjelmistokehittäjille, kuinka heidän tulisi rakentaa ohjelmisto. Tähän asiakirjaan tulisi myös sisällyttää yrityksen standardit lähdekoodin kommentointiin liittyen (ISO/IEC/IEEE 2015, s. 14, 19, 21–22).

ANSI/IEEE (1984) mukaan standardit miten lähdekoodia kommentoidaan tulisi olla osa ohjelmiston laadunvarmistussuunnitelmaa. Laadunvarmistussuunnitelmassa tulisi määritellä vähintään dokumentoinnin-, loogisten rakenteiden, ohjelmoinnin ja kommentoinnin standardit. (ANSI/IEEE 1984). Tämän kaltaiset asiakirjat ja standardit kertovat ohjelmistokehittäjille, millä tavalla heidän tulisi kehittää toivottua ohjelmistoa (Wiegers ja Beatty 2013, s. 261–265).

Ilman riittävää resurssointia ja johtamista jäävät hyvät suunnitelmat, vain hyviksi suunnitelmiksi. Wiegers ja Beatty (2013, s. 366) viittaavat Hooks ja Farryn (2011) tekemään tutkimukseen, jonka mukaan parhaiten menestyneissä projekteissa käytettiin 28 % projektin resursseista vaatimusmäärittelyyn, mallintamiseen, validointiin ja todentamiseen. On siis tärkeää, että ohjelmistokehityksessä suunnittelu on resurssoitu hyvin. Ohjelmiston kehitykseen ja toiminnallisuuteen tulee luoda ohjenuorat, miten asioita halutaan tehdä. Kuten Hooks ja Farry (2011) tuovat esille, hyvin resurssoitu suunnitteluprosessi ja kattavat dokumentit projektin toteutuksen apuna, ovat avain laadukkaan tuotteen tekemiseen (Wiegers ja Beatty 2013, s. 366). Ohjelmistokehittäjät tulee ottaa jo heti varhaisessa vaiheessa mukaan dokumentaation laatimiseen. Eri toimijoiden sisällyttäminen jo varhaiseen suunnitteluun tukee projektin onnistumista sekä ehkäisee suunnitteluvirheitä. (Wiegers ja Beatty 2013, s. 61–64).

Asianmukaisten dokumenttien, joiden avulla pystytään ohjaamaan ohjelmistotuotantoa ja projektin läpiviemistä tulisi olla osa jokaista ohjelmistotuontatoympäristöä (Wu ja Simmons 2000; Wiegers ja Beatty 2013, s. 14, 261–265). Näiden dokumenttien avulla pystytään luomaan yrityksen sisäiset standardit (Wiegers ja Beatty 2013, s. 261–265; ANSI/IEEE 1984). Yksi osa näitä standardeja tulisi olla lähdekoodin kommentointi (ANSI/IEEE 1984). Jotta ohjelmistotuontatoympäristössä onnistutaan luomaan laadukkaat ja kattavat dokumentit lähdekoodin kommentoinnin tueksi, on ymmärrettävä yleiset käytänteet ja hyvät tavat. Tulevissa luvuissa paneudutaan siihen, miten lähdekoodin kommentointia tulisi tehdä.

3 Mitä lähdekoodin kommentointi on?

Lähdekoodin kommentoinnille on käytössä erilaisia käsitteitä. Jotkut puhuvat esimerkiksi dokumentoinnista, kun taas toiset kommentoinnista. Luvussa tarkastellaan alan kirjallisuudessa esiin nousevia huomioita lähdekoodin kommentoinnista. Mitä lähdekoodin kommentoinnilla tarkoitetaan?

3.1 Lähdekoodin kommentoinnin määrittely

Alan kirjallisuudessa ei eroteta dokumentaatiota ja kommentointia toisistaan selkeästi. Niillä tarkoitetaan välillä samaa asiaa, ja välillä dokumentaatiolla tarkoitetaan laajemmin ohjelmistossa esiintyvää informaatiota. Lisäksi alan kirjallisuudessa toistuu myös erilaiset termit dokumentaation osalta. Todettakoon, että yhteistä termistöä ja määritelmiä ei siis ole.

Tutkielmassa käytetään Ahuja, Dhaiya ja Sadana (2013) erottelua lähdekoodin dokumentaation ja lähdekoodin kommentoinnin välillä. Lähdekoodin dokumentaatio on ohjelmiston mukana tuleva teksti, joka selittää ohjelmiston toimintaa ja käyttöä (Ahuja, Dhaiya ja Sadana 2013). Lähdekoodin kommenttien Ahuja, Dhaiya ja Sadana (2013) kuvailevat sen sijaan olevan lähdekoodiin lisättyä tekstimuotoista informaatiota, joka selittää yksityiskohtaisesti koodin toteutusta.

Tilley ja Parveen (2013) kuvailevat ohjelmiston kommenttien olevan elintärkeä apu ymmärtää ohjelman funktionaalista toimintaa ja toteutusta. Erilaiset kommentointitavat ja -tyylit voivat kuitenkin tehdä kommenttien ymmärtämisestä vaikeaa (Tilley ja Parveen 2013). Tästä syystä onkin äärimmäisen tärkeää luoda yhteisiä toimintatapoja ja tyylejä kommentoinnille.

Termillä lähdekoodin kommentointi tarkoitetaan lähdekoodiin kirjoitettuja tekstimuotoisia kommentteja, jotka tarjoavat informaatiota ohjelman tai sen osan toiminnallisuudesta sekä tekevät lähdekoodista helpommin luettavaa (Wang, Wang ja Wang 2019). Toisin sanottuna lähdekoodissa ilmenevien kommenttien tehtävänä on kertoa, miten koodi toimii ja miten sitä tulisi käyttä (Ahuja, Dhaiya ja Sadana 2013). Kommentit tarjoavat luonnollisen kielen (eng. natural language) muodossa selityksen toiminnallisuudesta ja nostavat esille tietoa ohjelman

osista, kuten lauseista (eng. statements), ohjelmasegmenteistä ja toiminnoista eli funktioista. (Wang, Wang ja Wang 2019). Lähdekoodin kommentit tarjoavat siis informaatiota ohjelmiston logiikasta ja funktionaalisuudesta (Huang ym. 2019, s. 88). Lisäksi kommentit edustavat tärkeintä informaation lähdettä järjestelmän dokumentaatioon osalta. (Steidl, Hummel ja Juergens 2013). Tästä syystä ne ovat avainasemassa lähdekoodin ymmärtämisen, kehittämisen ja ylläpidon suhteen. (Steidl, Hummel ja Juergens 2013). Automaattiset API -työkalut kuten javadoc ja doxygen sekä niiden luomat kommentit nähdään alan standarina (Spinellis 2010). Automaattisesti generoituihin kommentteihin paneudumme tarkemmin luvussa neljä.

3.2 Lähdekoodin kommentointi ohjelmistokehityksessä

Lähdekoodin kommentoinnin nähdään olevan olennainen osa monimutkaisia ja laajoja ohjelmistoja. Alati vakiintuva normi on, että ohjelmistot ja niiden osat tulisi kommentoida laadukkaasti. Menestyneet ohjelmistoalustat kommentoivat koodia kokonaisvaltaisesti ja johdonmukaisesti. Kokonaisvaltaisuus ja johdonmukaisuus ovat tärkeimmät tekijät laadukkaan kommentoinnin saavuttamiseksi (Spinellis 2010). Lisäksi yrityksellä tulisi olla kehittämisohjeet, jonka avulla ohjataan kommenttityylejä (Ahuja, Dhaiya ja Sadana 2013).

Ohjelmistokehityksessä lähdekoodin kommenttien on lähes yksiselitteisesti nähty olevan olennainen osa ohjelmaa. Kommentointi koetaan varsin tärkeäksi ohjelmistokehityksen eri osa-alueilla, kuten ohjelmiston toiminnan oppimisessa, testaamisessa, uuden ohjelmiston kanssa työskentelyssä ja ongelmanratkaisussa (Lethbridge ym. 2003). Kommentointi tarjoaa ohjelmoijille informaatiota ohjelman toiminnallisuudesta ja tästä syystä nopeuttaa ohjelmiston opettelua. Mitä paremmin ja systemaattisemmin ohjelmisto on kommentoitu, sitä nopeammin ja pienemmällä vaivalla ohjelmoijat ymmärtävät ohjelmistoa ja sen toimintaa (Tilley ja Parveen 2013). Aineiston perusteella voisikin siis todeta lähdekoodin kommenttien tarjoavan eräänlaisen manuaalin ohjelmiston toiminnallisuudesta. Lisäksi se avaa erilaisia ajatuksia ja metodeja, joita ohjelmoija on koodissa käyttänyt. Ohjelmoijat saavat kommenttien perusteella parhaimmillaan yksityiskohtaisen ymmärryksen koodin toiminnasta ymmärtämättä itse koodin logiikkaa. De Souza, Anquetil ja De Oliveira (2006, s. 38, 40) tekemän tutkimuksen mukaan lähdekoodi kommentteineen on tärkein esine (eng. artifact) ohjelmiston ymmärtämisen näkökulmasta.

Toinen huomio, joka nousi esille Lethbridge ym. (2003, s. 35–38) havainnoista on, että vaikka ohjelmiston tekijä lähtisi yrityksestä, on uusien ohjelmoijien mahdollista ymmärtää ohjelman toiminnallisuutta kommenttien avulla. Pee ym. (2014, s. 1) kertovat IT-alan yrityksissä olevan tyypillisesti korkea työntekijöiden vaihtuvuus. Pee ym. (2014, s. 1–6) jakavat työntekijöiden vaihtuvuuden kahteen, ulkoiseen ja sisäiseen. Sisäisessä vaihtuvuudessa työntekijä siirtyy yrityksen sisällä uusiin tehtäviin, jolloin tietotaito säilyy yrityksessä. Ulkoisessa vaihtuvuudessa taas työntekijä vaihtaa täysin yritystä. (Pee ym. 2014, s. 1–6). Tästä syystä yrityksessä tai projektissa tulee miettiä, kuinka uusi tekijä saadaan nopeasti perehdytettyä tehtävään. Tähän yhtenä ratkaisuna on lähdekoodin kommentit. Lähdekoodin kommenttien avulla uusi ohjelmoija voi saada nopeasti ymmärryksen ohjelman toiminnallisuudesta ja tästä syystä jatkaa työskentelyä ohjelman parissa (Lethbridge ym. 2003).

3.3 Lähdekoodin kommentoinnin tyylit ja formaatit

Kommentointityylejä on kolme, rivi- (eng. inline), lohkokommentit (eng. block) ja dokumenttikommentit (eng. document comment) (Wang, Wang ja Wang 2019). Rivi- ja lohkokommentit ovat yleisesti käytettyjä tapoja, sillä ne on määritelty kullekin ohjelmointikielelle erikseen. Dokumenttikommentit taas ovat spesifimpi kommentointi tyyli, sillä se esiintyi vain Wang, Wang ja Wang (2019) julkaisussa. Esittelen seuraavaksi tarkemmin nämä tavat.

3.3.1 Rivikommentit

Rivikommentit ovat kommentteja, jotka käyttävät erotinmerkkiä tai sekvenssissä olevia merkkejä kommentin aloitukseen ja rivinvaihtoa lopetukseen (Ahuja, Dhaiya ja Sadana 2013). Wang, Wang ja Wang (2019) kuvailevat rivikommentin määritteleväksi piirteeksi sen, että se on yhteydessä samalla rivillä olevaan lähdekoodiin. Ohjelmointikielissä on syntaksiin määritelty, millä merkillä tai merkkijonolla rivikommentti aloitetaan. C++, C#, D, Go, Java, JavaScript, Object Pascal (Delphi), Objective-C ja PHP ohjelmointikielet käyttävät // eli kahta kenoviivamerkkiä kommentin aloitukseen. Bash, Cobra, Perl, Python, Ruby, Windows PowerShell, PHP ja Maple ohjelmointikielet käyttävät risuaitaa eli # kommentin aloitukseen. TeX, Prolog, MATLAB,Erlang, S-Lang ja Visual Prolog käyttävät prosenttimerkkiä eli % määrittelemään kommentin aloitusta. (Ahuja, Dhaiya ja Sadana 2013).

```
// Esimerkki rivikommentti
```

Kuvio 1. Esimerkki rivikommentista

3.3.2 Lohkokommentit

Lohkokommentit ovat kommentteja, jotka käyttävät erotinmerkkiä kommentin aloitukseen ja toista erotinmerkkiä kommentin lopetukseen. Lohkokommenteissa ei välilyöntiä ja rivinvaihtoa lasketa kommentin alkua tai loppua määritteleväksi erotinmerkiksi. (Ahuja, Dhaiya ja Sadana 2013). C, C++, C#, D, Go, Java, JavaScript, Objective-C, PHP, Visual Prolog ja CSS ohjelmointikielissä lohkokommentti alkaa kenoviivan ja tähden yhdistelmällä /* ja päättyy samaan yhdistelmään, mutta toisin päin */. Object Pascal (Delphi) ja Pascal ohjelmointikielissä lohkokommentti tulee aaltosulkeiden väliin. Aaltosulku oikealle aloittaa lohkokommentin ja aaltosulku vasemmalle lopettaa sen. (Ahuja, Dhaiya ja Sadana 2013). Wang, Wang ja Wang (2019) kuvailevat lohkokommentin erityispiirteeksi, että se kirjoitetaan ennen koodiriviä tai rivejä joihin se linkittyy.

```
/* Lohkokommentti alkaa
jatkuu ...
ja loppuu */
```

Kuvio 2. Esimerkki lohkokommentista

3.3.3 Dokumenttikommentit

Dokumenttikommentit ovat kommentti tyyppi, joka nousi esille vain Wang, Wang ja Wang (2019) julkaisussa. Wang, Wang ja Wang (2019) nostavat dokumenttikommentit esille ohjelmointikieli Pythonin yhteydessä. Dokumenttikommentti eli docstring, on kommenttien tapainen ja sen tulisi pitää sisällään informaatiota moduuleista, funktioista, luokista sekä metodeista (Wang, Wang ja Wang 2019). Wang, Wang ja Wang (2019) tekemässä julkaisussa dokumenttikommentteihin ei paneuduta tämän tarkemmin.

4 Miten ja minne lähdekoodia kommentoidaan?

Alan kirjallisuudessa nousee esille monia erilaisia tapoja kommentoida lähdekoodia. Tässä luvussa katsomme tarkemmin miten lähdekoodia tulisi kommentoida, ja minne lähdekoodin kommentit tulisi sijoittaa.

4.1 Erilaiset tavat kommentoida lähdekoodia

Lähdekoodia voi kommentoida monin eri tavoin. Ahuja, Dhaiya ja Sadana (2013) nostavat esille esipuheen (eng. prefacing), versiohistorian sisällyttämisen ja loogisten lohkojen kommentoinnin. Tässä tutkielmassa käytetään esipuheesta nimitystä otsikko (eng. header), sillä se toistui aineistossa esipuhetta useammin. Loogisilla lohkoilla Ahuja, Dhaiya ja Sadana (2013) tarkoittavat aliohjelmia, funktioita tai muita ohjelman osia.

4.1.1 Otsikko

Otsikko (eng. header) on tapa, jossa ohjelman alkuun kirjoitetaan kommentti, joka tuo esille lyhyesti ohjelman toiminnallisuutta (Ahuja, Dhaiya ja Sadana 2013; Steidl, Hummel ja Juergens 2013). Otsikko auttaa nopeasti ymmärtämään mistä ohjelmassa on kyse sekä tarjoaa informaatiota. Esimerkiksi yleinen funktionaalisuus, luokan kirjoittaja ja versionumero ovat tietoja, jotka tulisi käydä ilmi otsikosta (Steidl, Hummel ja Juergens 2013).

4.1.2 Versiohistoria

Sisältää tiedon siitä, milloin ja miksi muutoksia on tehty. Lisäksi versiohistoria voi pitää sisällään tekijätiedot esimerkiksi kuka ohjelman on tehnyt tai minkä yrityksen omaisuutta se on. (Ahuja, Dhaiya ja Sadana 2013). Versiohistorian voi sisällyttää otsikkoon. Kuviossa 3 versiohistoria on sisällytetty otsikkoon. Todettakoon, että nykyään versiohistorian säilyttämiseksi ja tallentamiseksi on olemassa hajautettuja versiohallinnan työkaluja, jotka hoitavat asian paremmin.

```
/**
* Ohjelman tarkoitus on demonstroida kuinka
* luokka tulisi kommentoida
* @author Esimerkki
* @version 24.08.2020
*
*/
public class Esimerkki {

    //some code here

}
```

Kuvio 3. Esimerkki otsikosta Javassa

4.1.3 Loogisten lohkojen kommentit

Loogisten lohkojen eli aliohjelmien ja funktioiden kommentit takaavat, ettei ohjelmoijien tarvitse lukea koodia kokonaisuudessaan (Ahuja, Dhaiya ja Sadana 2013). Lukemalla loogisen lohkon kommentin ohjelmoija saa välittömästi käsityksen, mitä ohjelman eri osat tekevät. Tästä syystä tietyn ohjelman osan tai sen sisällä olevan komponentin etsiminen helpottuu ja nopeutuu.

```
//Metodi ottaa kaksi parametria ja palauttaa niiden summan
public static int  esimerkkiMetodi(param1, param2) {
    return param1+param2;
}
```

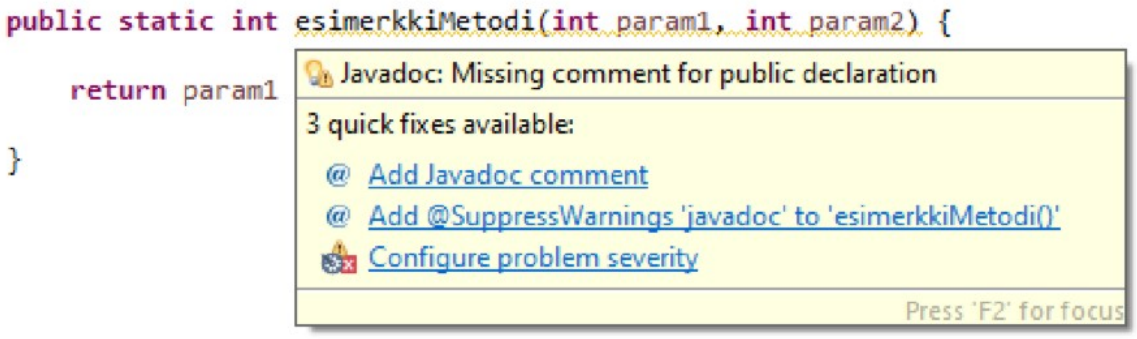
Kuvio 4. Esimerkki metodin kommentoinnista Javassa

4.2 Työkalut lähdekoodin kommentointiin

Ohjelmistokehittäjät käyttävät monenlaisia työkaluja lähdekoodin kommentointiin. Monessa ohjelmointikielessä ohjelmointirajapinta (eng. API) tarjoaa dokumentointigeneraattorin. Dokumentointigeneraattori on työkalu, joka luo automaattisesti dokumentaatiota ohjelmistoon. Tämän kaltaiset työkalut takaavat, että dokumentointia tehdään säännöllisesti. Yksi tämän kaltainen työkalu on Java ohjelmointikielessä oleva javadoc. Javadoc on automaattinen työkalu, jonka tehtävä on luoda kommentteja. Javadoc luo ohjelmointirajapintadokumentaation HTML-muodossa lähdekoodiin sekä vastaa kommenttien lisäämisestä. (Ahuja, Dhaiya ja Sadana 2013). Kun ohjelmoija luo esimerkiksi uuden metodin Javassa, vastaa javadoc kommenttien lisäämisestä. Javadoc-kommentit sijaitsevat luokkien ja metodien edessä (Huang ym. 2019, s. 94).

Ohjelmoijan tehtäväksi jää täyttää kommentit halutulla informaatiolla. Se, mikä tekee javadoc:n kaltaisista kommentteja generoivista työkaluista nykypäivän standardin ohjelmistokehityksessä on se, että ne pitävät huolen systemaattisesta kommentoinnista. ohjelmistotuontatoprosessin ollessa käynnissä käyttöliittymä ilmoittaa ohjelmoijalle, jos javadoc kommentteja ei ole kommentoitu rajapinnan määrittelemällä tavalla. Kuviossa 5 näkyy, että käyttöliittymä on alleviivannut metodin ja sen parametrit keltaisella kertoakseen ohjelmoijalle kommentin uupumisesta. Kun ohjelmoija laittaa hiiren kursorin alleviivatun osan päälle, avautuu ikkuna, josta ohjelmoija voi klikkaamalla lisätä javadoc-kommentin. Tämä toimii oivana muistutuksena ohjelmoijalle lisätä kommentit välittömästi metodia tehdessä. Javadoc-kommentit eroavat muista kommenteista yksilöllisen syntaksinsa puolesta. Javadoc-kommentit käyttävät /** -merkkiyhdistelmää määrittelemään kommentin alun, * -merkkiä määrittelemään kunkin kommenttirivin, ja */ -merkkiyhdistelmää kertomaan kommentin päättymisestä. Kuviossa 6 näkyy javadoc:n automaattisesti luoma kommentti syntakseineen.

Monet muutkin ohjelmointikielet sisältävät API-työkaluja lähdekoodin dokumentointiin, mutta Java ja javodc on näistä tunnetuin (Spinellis 2010). Ilman API-työkalua on kommentin lisääminen luokkien ja metodien eteen varsin helppoa käyttäen lohko- tai rivikommentteja.



Kuvio 5. Esimerkki javadoc kommentin uupumisesta

```
/**
 * @param param1 ensimmäinen parametri
 * @param param2 toinen parametri
 * @return ensimmäisen ja toisen parametrin summa
 */
public static int esimerkkiMetodi(int param1, int param2) {

    return param1 + param2;

}
```

Kuvio 6. Esimerkki javadoc kommentista

4.3 Miten lähdekoodia tulisi kommentoida

Ohjelmistokehityksessä on monia vakiintuneita käytänteitä, kuinka lähdekoodia tulisi kommentoida. Tilley ja Parveen (2013) mukaan jo pelkästään tarkastelemalla menestyneiden ohjelmistoalustojen kommentointia voidaan nähdä, miltä laadukas kommentointi näyttää. Laadukkaasti kommentoiduissa ohjelmistoalustoissa korostuu kommentoinnin johdonmukaisuus ja kokonaisvaltaisuus (Tilley ja Parveen 2013). Jotta ohjelmistotuontato ympäristössä päästään tilanteeseen, jossa lähdekoodin kommentointi on laadukasta, pitää sopia yhdessä tavat ja menetelmät kommentoida. Lisäksi ohjelmistokehittäjien tulee muistaa kirjoittaa kommentteja säännöllisesti (Huang ym. 2019, s. 88–89). Monet ohjelmointikielet, kuten Java ja PHP, sisältävät sovellusliittymägeneraattorin (API). Javassa Javadoc ja PHP:ssä PHPdoc ovat hyviä esimerkkejä API:sta, jotka vaativat koodia kirjoittaessa otsikon ja selityksen funktioiden toiminnallisuudesta muuttujineen ja paluuarvoineen. (Ahuja, Dhaiya ja Sadana 2013; Spinellis 2010; Head ym. 2018, s. 652). Tämän kaltaiset työkalut muistuttavat ohjel-

mistokehittäjää kommentoimaan koodia sitä mukaa kun sitä kirjoitetaan.

Isona haasteena, miten lähdekoodia tulisi kommentoida, nousee esille ohjelmistokehittäjien päätöksenteko. Vaikka Javadoc:n kaltaiset generaattorit tukevat päätöksentekoa, eivät ne pysty korvaamaan ohjelmistokehittäjien vastuuta siitä, miten ja mitä tulisi kommentoida (Huang ym. 2019, s. 88–90, 100). Vaikean asiasta tekee erityisesti se, ettei ole olemassa standardia, joka ohjaisi ohjelmistokehittäjien päätöksentekoa kommentointiin liittyen. Kokemus ohjelmoinnista nähdään suurimpana tekijänä, miten hyvin ohjelmoija näitä päätöksiä tekee. (Wang, Wang ja Wang 2019). ACM:n tietotekniikan (eng. Computer Science) vuoden 2018 opetussuunnitelmassa pyrittiin parantamaan ohjelmoinnin opiskelijoiden taitoja ja asenteita dokumentoida koodia. Uudessa opetussuunnitelmassa luotiin uusi kurssi, jonka tavoite oli lisätä opiskelijoiden käyttämää aikaa valmiin koodin ylläpidon ja testauksen parissa. Tästä syystä opiskelijat saavat kokemusta siitä, miten he pystyvät välittämään informaatiota dokumentoinnin avulla. (Doyle ym. 2011). Doyle ym. (2011) esiin nostamat tavat tukevat vähäisellä kokemuksella varustettujen tulevaisuuden ohjelmistokehittäjien dokumentoinnin päätöksentekokykyä.

Tutkielmassa käytetään Steidl, Hummel ja Juergens (2013) esiin nostamaa määrittelyä miten lähdekoodia tulisi kommentoida.

- Johdonmukaisuus: Lähdekoodissa tulisi kommentoida johdonmukaisesti metodien ja koodialueiden funktionaalisuutta. Näiden kommenttien tulisi liittyä metodin nimeen sekä tarjota lisäinformaatiota. Kommentin vahva linkittyminen metodin nimeen takaa tiedon kommentin ajantasaisuudesta sekä tarjoaa selkeän tunnisteen metodille.
- Hyödyllisyys: Kommenttien tulee tukea ohjelmiston ymmärtämistä selkeyttämällä koodin tarkoitusta. Yleisellä tasolla koodin lukijan tulee kokea, että kommentti on hyödyllinen.
- Täydellisyys: Ohjelmistossa tulisi olla tekijänoikeudet sijoitettuna kuhunkin tiedostoon sekä otsikot kullekin luokalle. Lisäksi jokainen metodi ja koodialue tulisi kommentoida.
- Johdonmukaisuus: Kaiken kommentoinnin tulisi olla johdonmukaista läpi ohjelmiston. Kommentit tulisi kirjoittaa samalla kielellä, ja jokaisella tiedostolla tulisi olla samat tekijänoikeustiedot.

Lisäksi seuraavia asioita tulisi huomioida koodia kommentoidessa (Ahuja, Dhaiya ja Sadana 2013):

- Monimutkaisten asioiden kommentointi on erittäin tärkeää. Vaikka koodi olisi kirjoitus hetkellä selkeää ja ymmärrettävää, ei se välttämättä ole sitä myöhemmin.
- Kommentoi sitä mukaan, kun kirjoitat tai ylläpidät koodia.
- Kirjoita helposti luettavia, mahdollisimman lyhyitä ja pelkistettyjä kommentteja.

4.4 Kommenttien sijainti ja sisältö

Kuten yllä mainittiin, monissa kielissä on automaattisia työkaluja muistuttamaan ohjelmoijaa kommenttien lisäämisestä. Silti ohjelmoijan tulee itse tehdä päätös parhaasta paikasta lisätä kommentti. Hyvää kommenttipaikkaa etsiessä tulisi huomioida, että kommentti kattaa ohjelman ydinkoodipätkän. (Huang ym. 2019, s. 88). Ydinkoodipätkä voi olla esimerkiksi funktio tai aliohjelma, ja niitä on usein enemmän kuin yksi, jolloin ne kaikki tulisi kommentoida. Huang ym. (2019, s. 88) havaitsivat luettuaan monia teknisiä dokumentteja sekä tutkimusraportteja, että on vaikeaa löytää perusteelliset ohjeistukset ohjelmoijille kommenttipaikkojen päättämiseen. Huang ym. (2019, s. 88–89) määrittelevät omassa ohjeistuksessaan kolmen tyyppiset koodikontekstit (eng. code context information):

1. Rakenteelliset kontekstiominaisuudet
2. Syntaktiset kontekstiominaisuudet
3. Semanttiset kontekstiominaisuudet

Rakenteelliset kontekstiominaisuudet edustavat koodilausekkeiden välisiä suhteita suhteessa sijaintiin, etäisyyteen, kytkentäsuhteisiin, ja niin edelleen. Syntaktiset kontekstiominaisuudet ja semanttiset kontekstiominaisuudet edustavat asiayhteyskoodin (eng. context code) syntaksia ja semanttista jakaumaa. (Huang ym. 2019, s. 88). Näiden kolmen määritelmän avulla Huang ym. (2019, s. 88–90) luoma ”kommenttien ehdottaja” -ohjelmisto pystyy määrittelemään minne kommentti tulisi sijoittaa. Vaikka tämä määrittely onkin tarkoitettu Huang ym. (2019, s. 88–90) ”kommenttien ehdottaja” -ohjelmiston perustaksi, voidaan sen avulla tukea myös ohjelmoijien tekemiä päätöksiä liittyen kommenttien sijaintiin. Huang ym. (2019, s. 88–90) määritelmien kautta johdin oman kahden kohdan ohjeistuksen kom-

menttien sijainnille:

1. Lähdekoodin rakenteen avulla voidaan määritellä, minne kommentit tulisi sijoittaa. Rakenteeseen vaikuttaa paljon ohjelmistokehittäjän tekemät valinnat, millä tavalla hän lähdekoodia jäsentelee. Yksi tapa olisi kategorioida yksittäiset funktiot tai aliohjelmat isommaksi ryhmäksi (Steidl, Hummel ja Juergens 2013). Tällöin voitaisiin kommentoida korkeammalla tasolla, minkälaisia funktioita tai aliohjelmia tässä funktioryppäässä esiintyy.
2. Lähdekoodissa käytetyn kielen perusteella voidaan tehdä päätös siitä, mitä tarvitsee kommentoida. Tärkeää olisi kommentoida kaikki muuttujat, funktiot tai aliohjelmat, sekä muut ohjelman osat, jotka eivät ole yksiselitteisiä. Jos esimerkiksi funktiota tulkittaessa ei ole yksiselitteisesti selvää, miten funktio toimii ja miksi se on toteutettu tietyllä tavalla, tulisi tämä kommentoida. Lisäksi ohjelmistokehityksessä ja ohjelmoinnissa on monia yleisesti sovittuja toteutustapoja, kuten muuttujien määrittely ja nimeäminen. Jos ohjelmoija luo koodia yleisten normien vastaisesti, tulisi kommenttien avulla selittää toteutusta.

Steidl, Hummel ja Juergens (2013) lisäävät, että ohjelmistossa pitäisi olla kommentoituna tekijänoikeudet, jotka sijoitetaan kunkin tiedoston alkuun. Lisäksi otsikko, jota Steidl, Hummel ja Juergens (2013) kutsuvat otsikko kommentiksi, tulisi löytyä jokaisesta luokasta. Javassa otsikko kirjoitetaan tuontien (eng. import) jälkeen, mutta kuitenkin ennen luokan määrittelyä (Steidl, Hummel ja Juergens 2013).

5 Käytänteet lähdekoodin kommentoinnissa

Luvussa tarkastellaan minkälaisia hyötyjä ja haittoja lähdekoodin kommentointiin liittyy. Tässä luvussa keskitytään erityisesti koodin luettavuuden parantamiseen lähdekoodin kommentteja hyödyntäen sekä, minkälaisia käytänteitä alan kirjallisuudessa ilmenee.

5.1 Lähdekoodin kommentoinnin hyödyt

Lähdekoodin kommentoinnilla on paljon hyötyjä sekä ohjelmistokehittäjälle että muille ohjelmiston parissa työskenteleville. Koodin lukemisen helpottaminen on suurimpia syitä, miksi lähdekoodia tulisi kommentoida ohjelmistoja kehittäessä. Huang ym. (2019, s. 88) kuvailevat koodissa olevien kommenttien olevan tärkeä osa ohjelmistoprojekteja, sillä ne tuovat esille ohjelman logiikkaa ja funktionaalista toteutustapaa sekä parantavat ohjelman lukemista ja ymmärtämistä. Erityisesti ylläpidon näkökulmasta lähdekoodin kommentit ovat elintärkeitä. De Souza, Anquetil ja De Oliveira (2006, s. 32) nostaa esille, että 40 % – 60 % ylläpitoon käytetystä ajasta menee ohjelmiston ymmärtämiseen. Ohjelmiston ajan tasalla pitäminen helpottuu ja nopeutuu huomattavasti merkityksellisen ja säännöllisen kommentoinnin myötä. Burki ja Vogt (2014, s. 4) väittävät hyvän dokumentoinnin laskevan 12 % ylläpidon kokonaiskustannuksia.

Lisäksi lähdekoodin kommentointi helpottaa ohjelmistokehittäjän työtä jäsennellä omia ajatuksiaan kehitystyössä. Kommenttien, erityisesti ohjelman alkuun tehtävän otsikon, uskotaan helpottavan ohjelmistokehittäjää miettimään ohjelmistoa kokonaisuudessaan ja siihen kuuluvia osia. Tästä syystä ohjelmistokehittäjän voi olla helpompi orientoitua yksittäisiin ohjelman osiin ja niiden tehtäviin osana laajempaa kokonaisuutta. (Ahuja, Dhaiya ja Sadana 2013).

Ahuja, Dhaiya ja Sadana (2013) mukaan Storey ym. tekemässä tutkimuksessa ilmeni, että ohjelmistokehittäjät käyttävät yleisesti tehtävä kommentteja. TODO, BUG, FIXME ovat yleisesti käytössä olevat tavat kirjoittaa tehtävä kommentteja (Wang, Wang ja Wang 2019). Myös Huang ym. (2019, s. 94–95) nostavat esille todo -kommentit ja niiden yleisyyden ohjelmistokehityksessä. Tehtäväkommentit ovat merkkinä ohjelmistokehittäjälle keskeneräisestä

tehtävästä, korjausta tarvitsevasta viasta (eng. bug) tai huomautus toteutuksen hakkeroinnista (Steidl, Hummel ja Juergens 2013).

Lisäksi lähdekoodin kommentoinnissa nousee esille tapa kommentoida koodia pois. Eräs käytetty tapa ohjelmistokehityksessä on kommentoida testit sekä pois käytöstä otettavat koodin osat pois. (Huang ym. 2019, s. 94–95). Tällöin koodia ei poisteta, vaan sen eteen lisätään kommentoinnille ominaiset merkit. Vaikka koodin osa ei ole enää käytössä, se voi tarjota informaatiota ohjelmistokehittäjille. Lisäksi se voi tulla tarpeen yhteensopivuuden näkökulmasta, jos projektin versiossa mennään taaksepäin. (Wang, Wang ja Wang 2019).

5.2 Haasteet lähdekoodin kommentoinnissa

Vaikka lähdekoodin kommentointi nähdään yleisesti positiivisena ja tärkeänä asiana, liittyy siihen myös haittoja. Lähdekoodin kommentointia ei nähdä yleisesti kovin hyödyllisenä osana lähdekoodia ja tästä syystä laadukkaan kommentoinnin laatimiseen tarvittavaa aikaa ei arvosteta (Tilley ja Parveen 2013). Tämä luo monia haasteita kommentoinnin tekemiseen sekä siihen, miten kommentointi yleisesti nähdään. Jos kommentointia ei tehdä oikein ja kunnolla, voivat huonot kokemukset luoda ohjelmistokehittäjälle negatiivisen kuvan kommentoinnista (Spinellis 2010).

Koodin tulisi olla itsessään helposti luettavaa, jolloin sitä ei tarvitse kommentoida (Ahuja, Dhaiya ja Sadana 2013; Huang ym. 2019, s. 100; Spinellis 2010). Tämä korostuu erityisesti muuttujien osalta, sillä nimeämällä muuttujan osuvammin ei kommenttia tarvita selittämään sen toiminnallisuutta. Spinellis (2010) mukaan huonoa koodia ei tulisi kommentoida, vaan se tulisi kirjoittaa paremmin. Tämä viittaa siihen, että looginen ja hyvin nimetty koodi on itsestään selvää. Jos koodista aletaan kommentoimaan kaikkea mahdollista, tulee siitä helposti liian runsasta. Runsas kommentointi haittaa koodin luettavuutta (De Souza, Anquetil ja De Oliveira 2006, s. 31–32). Lisäksi se lisää kommenttien ylläpitämiseen käytettävää aikaa huomattavasti (De Souza, Anquetil ja De Oliveira 2006, s. 31–32; Wang, Wang ja Wang 2019). Myös Huang ym. (2019, s. 101) katsovat liiallisen määrän kommentteja tarkoittavan ylimääräistä työtä, jonka ohjelmistokehittäjät joutuvat käyttämään kommentteihin. He uskovat tämän vaikuttavan ohjelmoinnin tehokkuuteen, sillä kommentteihin käytetty aika on

pois itse ohjelmointiin käytetystä ajasta. Spinellis (2010) taas nostaa esille, kuinka huonot kommentit vievät ohjelmistokehittäjien huomioita epäolennaiseen sekä vievät näytöltä tilaa.

Tilley ja Parveen (2013) esittävät, että lähdekoodin kommentit ovat usein puutteellisesti tehtyjä. Oli syy ajanpuute tai huonosti toteutettu kommentti, jättävät epätäydelliset kommentit paljon tulkinnan varaa. Tämä jo itsessään tekee lähdekoodin lukemisesta huomattavasti vaikeampaa ja aikaa vievää.

Suurimpana haittana nähdään kommenttien pitäminen ajan tasalla. Erityisesti ylläpidon näkökulmasta kommentit eivät yleensä ole ajan tasalla ja tästä syystä lähdekoodin ja sen kommenttien välillä voi olla loogisia eroja (De Souza, Anquetil ja De Oliveira 2006, s. 31–32; Spinellis 2010). Kääntäjät eivät tarkista kommentteja ja niiden oikeellisuutta, jolloin se jää ohjelmoijan tehtäväksi. Jos ohjelmoija ei muista, huomaa tai kerkeä muokkaamaan kommenttia koodin muokkaamisen yhteydessä, on lopputuloksena ristiriita koodin ja kommenttien välillä (Spinellis 2010). Tilley ja Parveen (2013) tukevat väitettä, jonka mukaan lähdekoodissa esiintyvät kommentit voivat usein olla ristiriidassa itse koodin kanssa.

5.3 Miten lähdekoodia ei tulisi kommentoida

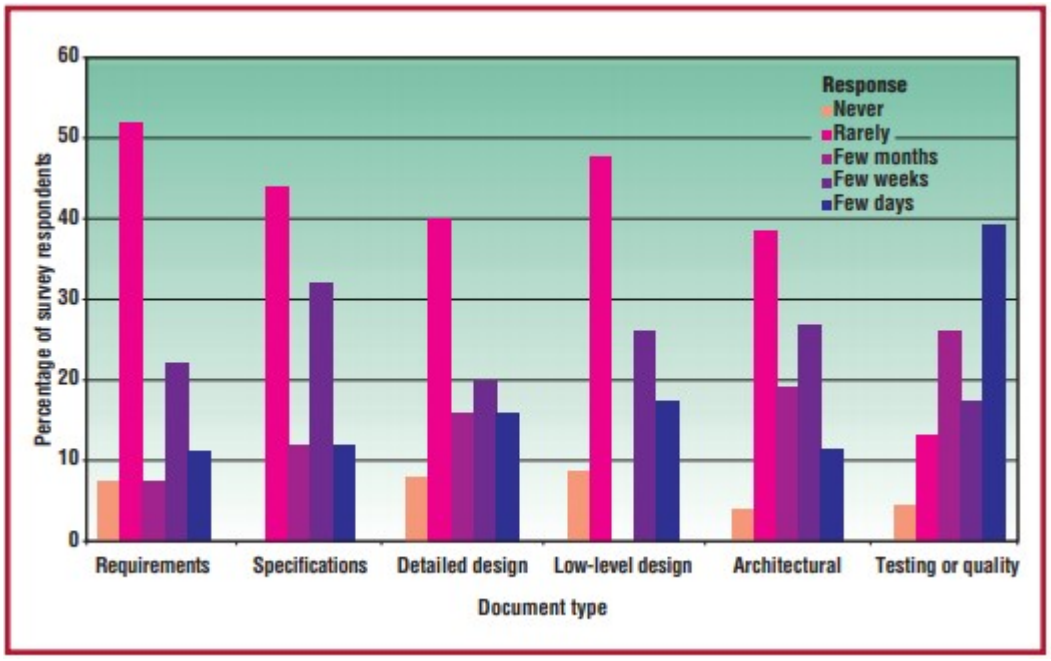
Alan kirjallisuudessa nousee esille, että lähdekoodin kommentointiin liittyy paljon asioita, joita ei tulisi tehdä. Erityisesti tulisi välttää liian runsasta, monitulkintaista sekä puutteellista tai vanhentunutta kommentointia. Ahuja, Dhaiya ja Sadana (2013) tuovat esille kahdeksan kohdan määrittelyn, mitä lähdekoodin kommentoinnissa ei tulisi tehdä:

1. Älä kirjoita kommentteja jokaiselle riville ja jokaiseen muuttujaan.
2. Käytä rivikommentteja ja vältä lohkokommentteja.
3. Kirjoita kommentteja sinne, minne niitä tarvitaan, mutta hyvä luettava koodi pitää sisällään hyvin vähän kommentteja. Jos muuttujat ja metodien nimet ovat merkityksellisiä, tekee se itsessään koodista helposti luettavaa. Jos koodi on helposti luettavaa, älä kirjoita kommentteja.
4. Muokkaa kommentteja sitä mukaan, kun muokkaat koodia.
5. Vähemmän kommenttirivejä tekee koodista elegantimman. Toisaalta epäselvä koodi, joka sisältää vähän kommentteja, on huonoin mahdollinen.

6. Jos kirjoitat monimutkaista koodia tai outoa logiikkaa, dokumentoi se hyvin selkeillä kommenteilla.
7. Jos alustat numeerisen muuttujan erikoisnumerolla, joka on muu kuin 0, -1 tai vastaava, dokumentoi valinnan syy.
8. Älä kirjoita koodia kommentteihin, ellei se ole tarpeellista

5.4 Kuinka kommentteja ylläpidetään

Lähdekoodin kommenttien ylläpito jää usein varsin vähälle huomiolle. Lethbridge ym. (2003) tekemien havaintojen mukaan kommenttien ylläpitoa ei nähdä tärkeänä. Sama havainto toistuu alan kirjallisuudessa. Ohjelmiston elinkaaren aikana tehdään paljon muutoksia ja päivityksiä ohjelmakoodiin. Kuitenkin kommentteja ei päivitetä yhtä säännöllisesti kuin itse ohjelmakoodia.



Kuvio 7. Aika, järjestelmä muutosten ja dokumentaatio päivitysten välillä erilaisille dokumentointi tyypeille (Lethbridge ym. 2003)

Lethbridge ym. (2003) tekemässä tutkimuksessa huomattiin, että ohjelmistokehittäjät harvoin muokkaavat kommentteja. Jos ohjelmistokehittäjät muokkaavat kommentteja, tehdään se useita viikkoja ohjelmakoodin muokkaamisen jälkeen. Kuvio 7 (Lethbridge ym. 2003)

osoittaa, että Lethbridge ym. (2003) tutkimukseen vastanneista (48 ohjelmistokehittäjää) hyvin harva muokkasi kommentteja välittömästi (muutaman päivän sisällä) ohjelmakoodin muokkaamisen jälkeen. Ohjelmistokehittäjät tekivät muokkauksia ainoastaan testeihin ja laadukkaaseen dokumenttiin (eng. quality document) nopeasti muutosten jälkeen (Lethbridge ym. 2003).

Yksi yleinen syy miksi kommenttien ylläpito jää usein varsin vähälle huomiolle on aikataulu-haasteet (Lethbridge ym. 2003). Ohjelmistokehittäjät kokevat kommenttien päivittämisen aikaa vieväksi erityisesti tilanteissa, joissa aikapaineet ohjelmistokoodin muokkaamiselle ovat etusijalla. Erityisesti monimutkaiset, pitkät ja paljon muokkaamista vaativat kommentit jäävät muokkaamatta. Väitettä tukee Lethbridge ym. (2003) havainto, että ohjelmistokehittäjät tekevät ja käyttävät lyhyitä ja ytimekkäitä kommentteja. Tämän vuoksi ohjelmistokehittäjät jättävät monimutkaiset ja aikaa vievät kommentit huomiotta.Head ym. (2018, s. 650–651) tutkivat, miksi ylläpitäjät eivät päivitä kommentteja ja havaintojensa perustaalla loivat kolme teemaa otsikolla: "Factors Impacting Whether Maintainers Will Update Comments":

1. Kommenttien pitäminen riittävän pieninä: Joskus informaatiota voi olla liikaa ja Head ym. (2018, s. 650–651) havaintojen mukaan jotkut ylläpitäjät pelkäävät, että kommenttien tarjoama informaatio tulkitaan väärin.
2. Kommenttia sekä muuta koodia ei tulla käyttämään tai lukemaan: Head ym. (2018, s. 650–651) mukaan osa kommenteista ja koodista on tullut siihen pisteeseen, ettei kukaan tule todennäköisesti sitä lukemaan tai muokkaamaan. Tämän vuoksi ylläpitäjät eivät koe tarpeelliseksi päivittää kommentteja. Lisäksi vanhentuneiden kommenttien päivittäminen voisi luoda muita ongelmia, kuten ristiriitoja muun vanhentuneen koodin ja sen kommenttien kanssa. (Head ym. 2018, s. 650–651).
3. Käytänteet kommentoinnissa Head ym. (2018, s. 650–651) huomasivat, että ylläpitäjät pyrkivät pitämään kiinni ohjelmiston kommentoinnissa käytetystä tyylistä. Ylläpitäjät eivät näe tarvetta kommentoida asioita koodista tai semmoisilla tavoilla, jotka eivät sovellu käytettyyn tyyliin. (Head ym. 2018, s. 650–651).

Vaikka kommenttien ylläpitäminen on haastavaa ja aikaa vievää, sen tekeminen on tärkeää laadukkaan ja helposti ymmärrettävän ohjelmakoodin ja dokumentoinnin kannalta. De Souza, Anquetil ja De Oliveira (2006, s. 32) kuvailevat kommenttien uupumisen pakottavan

ylläpitäjät tekemään töitä entistä enemmän löytääkseen informaatiota lähdekoodista. Tämän vuoksi Lethbridge ym. (2003) esittämä havainto, että ohjelmistokehittäjät tunnistavat arvon, jonka yksinkertainen kommentti tarjoaa, pitää paikkaansa. Kommenttien uupuminen ja se, etteivät ne ole ajan tasalla, on suuri haaste. Lähdekoodin kommenttien ylläpidolle tulisi resursoida aikaa ja se tulisi nähdä muuhun ohjelmistokehittämiseen kuuluvana osana. Oli sitten kyse bugin korjaamisesta tai monimutkaisemman funktion muokkaamisesta, voi ajan tasalla oleva kommentti nopeuttaa ohjelmiston parissa työskentelyä tulevaisuuden kehitystyössä. Tämän vuoksi tulisi laittaa hyvä kiertämään ja muokata kommentit tulevien ohjelmoijien työtä helpottamiseksi.

5.5 Otsikointi ja koodin luettavuuden parantaminen

Aineistossa tuotiin esille näkökulma, että kommentteja tulisi olla mahdollisimman vähän (Spinellis 2010; Ahuja, Dhaiya ja Sadana 2013; Head ym. 2018, s. 643–644; Wang, Wang ja Wang 2019; Steidl, Hummel ja Juergens 2013). Koodia voidaan jakaa erilaisiin osiin kommenttien avulla helpommin luettavan lähdekoodin ja mahdollisimman vähäisen kommentoinnin saavuttamiseksi. Yksi tapa saavuttaa tämä voisi olla jakaa toiminnallisuudeltaan samantyyliset metodit omaksi ryhmäkseen ja erottaa tämä ryhmä kommentoimalla sen alku. Steidl, Hummel ja Juergens (2013) nostavat esille saman funktionaalisen näkökulman omaavien metodien ja koodialueiden koostamisen yhteen laajemmaksi kokoonaisuudeksi. Tämän vuoksi nämä koodialueet voitaisiin kommentoida jakso kommentein (eng. section comment). Kuviossa 8 Steidl, Hummel ja Juergens (2013) tuovat esille esimerkin siitä, kuinka tämänkaltainen koodialue voitaisiin kommentoida.

```
// ——— Getter and Setter Methods ———
```

Kuvio 8. Esimerkki jakso kommentista (eng. section comment) Javassa (Steidl, Hummel ja Juergens 2013)

6 Yhteenveto

Tutkielma antaa yleiskuvan lähdekoodin kommentoinnista. Tutkielmassa esitellään, miten lähdekoodia tulisi kommentoida. Minne kommentit tulisi sijoittaa? Kenen vastuulla on kommentoida lähdekoodia?

Dokumentointi on olennainen osa jokaista ohjelmistotuontatoprojektia. IT-alalla kehitetään yhä parempia tapoja dokumentoida suunnitteluprosesseja sekä välittää informaatiota ohjelmistosta ja sen toiminnallisuudesta (Wiegers ja Beatty 2013, s. 128, 291–292; ISO/IEC/IEEE 2015, s. 9–11). IT-alan yritysten tulee tulevaisuudessa kehittää dokumentointia ja linjauksia siitä, miten lähdekoodin kommentointia tulisi tehdä. Aineiston perusteella kommentit tulisi sijoittaa siten, että jokainen metodi ja luokka olisi kommentoitu. Näissä kommenteissa tulisi käytä ilmi ohjelman tai sen osan toiminnallisuus, käyttötavat sekä tietoa toteutuksesta. Lähdekoodin kommenttien tulee tarjota lukijalle sellaista informaatiota, mikä ei lähdekoodista ilmene. Liian runsas kommentointi on haaste nykypäivän ohjelmistoissa. Nimeämällä metodit ja muuttujat mahdollisimman osuvasti pystytään kommenttien määrää vähentämään huomattavasti. Metodien ja muuttujien nimeämiskäytänteistä on syytä sopia yrityksessä, ja tämä tieto tulisi löytyä yrityksen lähdekoodin kommentointiin liittyvästä ohjeistuksesta (ISO/IEC/IEEE 2015, s. 14, 19, 21–22). Näin toimimalla päästään ohjelmoijasta riippumatta johdonmukaisempaan ja yhtenäisempään kommentointiin projekteissa. Lähdekoodin kommentointi on ohjelmistokehittäjien sekä ylläpitäjien vastuulla, ja siksi yritysten tulee tarjota heille riittävästi resursseja kommentointiin.

Kommentointiin liittyvä päätöksenteko on yksi suurimmista haasteista erityisesti nuorille ja aloitteleville ohjelmoijille (Huang ym. 2019, s. 88). Jo varhaisessa vaiheessa tulisi antaa tukea ja opastusta, kuinka lähdekoodin kommentointia tulisi harjoittaa. Doyle ym. (2011) nostaa esille opiskelijoille suunnatun kurssin, joka tukee opiskelijoiden päätöksenteon kehittymistä. Tämänkaltaisia kursseja tulisi järjestää entistä laajemmin myös kokeneemmille ohjelmoijille. Lisäksi erilaisten kommentteja automaattisesti generoivien API:en tulisi olla entistä laajemmin käytössä. Näitä API-työkaluja olisi syytä jatkokehittää sekä sisällyttää sellainen ohjelmointikieliin, joissa vastaava ei vielä ole.

Tutkielman heikkoutena on, että lähdekoodin kommentointia on tutkittu varsin vähän. Jatkotutkimuksena voitaisiin tutkia Huang ym. (2019, s. 88–90) esiin nostamaa kommenttiehdottajaa sekä muita vastaavan kaltaisia automaattisia lähdekoodin kommentointiin luotuja työkaluja. Näiden pohjalta voitaisiin löytää uusia määrittelyjä lähdekoodin kommentoinnille.

Lähteet

Ahuja, Manjot Singh, Surender Dhaiya ja Neha Sadana. 2013. “An empirical survey of usefulness of comments in software programming languages”. *International Journal of Computing and Corporate Research* 3.

ANSI/IEEE. 1984. “IEEE Standard for Software Quality Assurance Plans”. *IEEE Standrad.*

Burki, Coen J., ja Harald H. Vogt. 2014. “How to save on software maintenance costs”. *Omnext – Software analysis as service.* `http://asq.org/public/wqm/how-to-save-on-software-maintenance-costs.pdf`.

De Souza, S. C. B., N. Anquetil ja K. M. De Oliveira. 2006. “Which documentation for software maintenance?” *Journal of the Brazilian Computer Society.* `https://www.researchgate.net/publication/317601060_Which_documentation_for_software_maintenance`.

Doyle, Maureen, Brooke Buckley, Wei Hao ja James Walden. 2011. “Work in progress — Does maintenance first improve student’s understanding and appreciation of clean code and documentation”. *2011 Frontiers in Education Conference (FIE).*

Head, Andrew, Caitlin Sadowski, Emerson Murphy-Hill ja Andrea Knight. 2018. “When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects”. *2018 International Conference on Software Engineering (ICSE).*

Huang, Yuan, Xinyu Hu, Nan Jia, Xiangping Chen, Yingfei Xiong ja Zibin Zheng. 2019. “Learning Code Context Information to Predict Comment Locations”. *IEEE Transactions on Reliability* 69.

ISO/IEC/IEEE. 2015. “ISO/IEC/IEEE International Standard for Systems and software engineering – Content management for product life-cycle, user, and service management documentation”. *IEEE: International standard.*

Lethbridge, T. C., J. Singer, A. Forward ja D. Consulting. 2003. “How Software Engineers Use Documentation: The State of the Practice”. *IEEE Software* 20 (6).

Pee, Lg, Atreyi Kankaanhalli, Gek Woo Tan ja Zhi-Choong Tham. 2014. “Mitigating the Impact of Member Turnover in Information Systems Development Projects”. *IEEE Transactions on Engineering Management.*

Spinellis, Diomidis. 2010. “Code Documentation”. *IEEE Software.*

Steidl, Daniela, Benjamin Hummel ja Elmar Juergens. 2013. “Quality analysis of source code comments”. *ICPC’13.*

Tilley, Scott, ja Tauhida Parveen. 2013. “On the similarities and differences between program documentation and test documentation”. *2012 IEEE International Professional Communication Conference.*

Wang, Renmin, Tao Wang ja Huaimin Wang. 2019. “Study of a Code Comment Decision Method Based on Structural Features”. *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS).*

Wiegers, Karl, ja Joy Beatty. 2013. *Software requirements: Third Edition.*

Wu, Ching-Seh, ja D.B. Simmons. 2000. “Software Project Planning Associate (SPPA): a knowledge-based approach for dynamic software project planning and tracking”. *2000 Computer Software and Applications Conference (COMPSAC), IEEE Annual International.*