Ohjelmointiprosessissa kohdatut ongelmat ja niiden ratkaiseminen työkalujen ja ohjelmistojen avulla

Jarmo Isotalo

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 20. joulukuuta 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Jarmo Isotalo

Työn nimi — Arbetets titel — Title

Ohjelmointiprosessissa kohdatut ongelmat ja niiden ratkaiseminen työkalujen ja ohjelmistojen avulla

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	20. joulukuuta 2015	21

Tiivistelmä — Referat — Abstract

Tuotantokäytössä oleviin ohjelmistoihin joudutaan erilaisten virhetilanteiden ja uusien ominaisuuksien lisäysten takia tekemään muutoksia. Usein muutoksen tai korjauksen tekee ohjelmoija, joka ei alunperin ole ollut tuottamassa kyseistä ohjelmakoodia. Ohjelmistokehittäjä pyrkii erilaisin keinoin selvittämään järjestelmän toimintaa, jotta hän pääsisi tehokkaasti tekemään korjauksen. Usein koodari esittää itselleen kysymyksiä, joiden avulla hän pyrkii selvittämään ohjelmiston toimintaa ja löytääkseen koodista sen kohdan, johon muutos tai korjaus tehdään.

Ohjelmoijan tueksi ja kysymyksiin vastaamista varten on kehitetty useita erilaisia työkaluja ja ohjelmistoja, joista osa on kaupallisia, mutta suuri on avoimesti kaikkien saatavilla ja käytettävissä. Käyttämällä erilaisia työkaluja tehokkaasti ohjelmoija löytää vastauksia esille tulleisiin kysymyksiinsä helpommin ja nopeammin. Haasteena on se, etteivät ohjelmistokehittäjät useinkaan tunne riittäästi ongelmanratkaisuun soveltuvia työkaluja tai he eivät osaa käyttää työkaluja täysimääräisesti ja tehokkaasti osana omaa ohjelmointiprosessiaan.

Esittelen tässä tutkielmassa yleisiä ongelmatilanteita, joissa ohjelmoija selvittää ohjelman rakennetta ja toimintaa itselleen esittämiensä kysymysten avulla ja esittelen erilaisia ratkaisukeinoja. Lisäksi esittelen joitakin vastaamiseen tarkoitettuja työkaluja sekä työkaluja, joita voi käyttää vastausten etsimiseen, vaikkei niitä alunperin ole tarkoitettu siihen käyttöön.

ACM Computing Classification System (CCS):
D.2.1 [Software Engineering]: Tools
D.2.2 [Software Engineering]: Programmer workbench

Avainsanat — Nyckelord — Keywords
ohjelmistotuotanto, ohjelmointiprosessin kysymykset, virheen etsintä, virheen korjaus

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Ohjelmistot ovat näinä päivinä keskeisessä asemassa ihmisen elämässä ja kaikessa liiketoiminnassa. Ihmisten arki on muuttunut älypuhelinten ja tablettien yleistyttyä ja niissä toimivia sovelluksia käytetään koko ajan enemmän ja enemmän. Tuskin mikään liiketoiminta on enää mahdollista ilman tietojärjestelmiä eikä mikään bisnes pyörisi pitkään esim. sähkökatkon sattuessa/pitkittyessä.

Haastavia tilanteita tulee myös tietojärjestelmissä olevista virheistä. Järjestelmiin päätyvien virheiden syntymistä pyritään estämään erilaisin menetelmin eri vaiheissa järjestelmän kehitysprosessia. Dokumenttien (mm. määrittely) katselmointi on hyväksi havaittu keino estämään tilannetta, jossa ohjelmoija ohjelmoi virheettömän toiminnon, mutta toiminto ei vastaa loppukäyttäjän liiketoimintatarvetta. Ohjelmakoodiin päätyviä virheitä pyritään välttämään mm. pariohjelmoinnilla.

Jotta tuotantoon päätyisi mahdollisimman vähän virheitä, niitä pyritään havaitsemaan ja korjaamaa ohjelmistotuotannon aikana erilaisilla koodianalyyseillä ja testaamisella (mm. yksikkötestit, integraatiotestit, systeemitestit, integraatiotestit, end-to-end-testit, suorituskykytestit ja rasitustestit). Kaikista toimista huolimatta sovelluksia ei kuitenkaan saada täysin virheettömiksi. Erityisesti monimutkaisissa ja laajoissa ohjelmistokokonaisuuksissa on mahdotonta tunnistaa kaikki virhetilanteet, sillä erilaisia if-else-haaroja on kertakaikkiaan liian paljon. Lisäksi täydellisen kattavaan virheiden etsintään ei ole varaa, sillä täydellinen testaus on liian kallista. Niinpä tuotantoon päätyy sovelluksia, joissa on virheitä/puutteita.

Virheet luokitellaan usein neljään luokkaan:

1. Kriittinen virhe, joka estää järjestelmän käytön [PL15]
2. Virhe, joka vaikeuttaa järjestelmän käyttöä [PL15]
3. Virhe, joka voidaan ohittaa tekemällä toiminto tilapäisesti eri tavalla [PL15].
4. Kosmeettinen haitta, joka ei estä järjestelmän käyttöä[PL15].

Oleellista on myös tunnistettujen virheiden kiireellisyyden määrittely. Osa virheistä voi huoleti odottaa seuraavaa korjauspakettia, mutta jotkut virheet on korjattava välittömästi esim. hotfix-prosessia noudattaen.

Esimerkiksi yrityksen liiketoiminta voi olla kokonaan halvaantunut vakavan järjestelmävirheen takia, jolloin korjauksella on kiire!

Välitön
: Vaatii välittömän korjauksen. Ohittaa kaikki muut jonossa olevat työt.

Kiireellinen
: Korjaus on tehtävä viikon kuluessa.

Normaali
: Korjaus voi odottaa seuraavaa versiota.

Alhainen
: Korjauksella ei ole kiire.

Virheenkorjausten lisäksi ohjelmistoja muutetaan tekemällä niihin mm. pieniä käyttäjän työtä helpottavia toiminnallisuusparannuksia ja sovelluksen suorituskykyä tai ylläpidettävyyttä parantavia teknisiä korjauksia. Usein järjestelmiä myös täydennetään lisäämällä niihin kokonaan uusia toiminnallisuuksia tai toimintokokonaisuuksia.

Paitsi järjestelmissä olevat ohjelmointivirheet, myös järjestelmien TODO infrassa tai tietoliikenteessä tapahtuvat virheet/ongelmatilanteet saavat helposti liiketoiminnan ja ihmisen arjen sekaisin. Hiljattain Facebook oli jonkun aikaa pois käytöstä ja ihmiset soittivat hädissään hätäkeskukseen [Pel15]!

IT-alalla kaikille on tuttu sanonta: "Jos se toimii, älä koske siihen", sillä jos ohjelmistokokonaisuutta ja/tai käytettyä logiikkaa ei tunne, saattaa ohjelmakoodin saada sekaisin tekemällä pieneltä tuntuvan muutoksen: muutos itsessään on virheetön, mutta se aiheuttaa uuden virheen johonkin toiseen, ennalta-arvaamattomaan paikkaan. Automaattisella testaamisella pyritään tunnistamaan muutoksen yhteydessä syntyvät sivuvaikutukset, mutta testit harvoin tunnistavat kuitenkaan kaikkia mahdollisia virhetilanteita. Muiden käyttöön tarkoitettujen kirjastojen kohdalla mahdollisten sivuvaikutusten riski kasvaa, mikäli muutos muuttaa ohjelman toimintaa tai korjaa pitkäaikaisen bugin. Yleisesti käytössä olevat kirjastot noudattavat usein laajassa käytössä olevaa semanttista versiointia [PW15]. Tämä sallii pienissä päivityksissä vain pienet korjaukset, jotka eivät riko taikka muuta kirjaston toimintaa, vaan vasta isompien versionumeroiden päivitysten on hyväksyttyä muuttaa kirjaston toimintaa siten, että kirjaston käyttäjien tulisi varautua muutoksiin kirjastoa päivitettäessä.

Erilaisista, yllämainituista syistä johtuen tuotantokäytössä oleviin järjestelmiin joudutaan usein tekemään korjauksia tai täydennyksiä ja tuon työn tekee ohjelmoija.

Jos alkuperäisen koodin tehnyt ohjelmoija on edelleen saatavilla, hän kykenee varsin nopeasti aloittamaan haluttujen muutosten tekemisen, koska koodi, käytetty logiikka ja ohjelman rakenne ovat hänelle entuudestaan tuttuja. Usein kuitenkin käy niin, että muutoksen joutuu tekemään henkilö, jolle koko ohjelmisto on täysin tuntematon. Korjausten ja täydennysten tekemisen tulee olla mahdollisimman tehokasta, jotteivat korjauskulut kasva kohtuuttomiksi ja jotta häiriön vaikutus loppukäyttäjille olisi mahdollisimman lyhyt. Aika on siis rahaa, tässäkin. Tehokas prosessi mahdollistaa myös sen, että ohjelmoija pääsee tekemään aitoa lisäarvoa tuottavia uusia ominaisuuksia sinänsä tarpeettomien virhekorjausten sijaan.

Miten ohjelmoija pääsee mahdollisimman nopeasti kiinni työhön? Ohjelmoijan käytössä on (toivottavasti) kattava tekninen dokumentaatio, joka auttaa ohjelmistoon tutustumisessa. Dokumentin lukemisen lisäksi ohjelmoija ryhtyy yleensä selvittämään tehtävää/ongelmaa esittämällä itselleen erilaisia kysymyksiä ohjelmakoodin toiminnasta ja toimimattomuudesta. Kysymyksillä pyritään luomaan kuvaa siitä, miten ohjelma toimii ja mistä osista se rakentuu [SSE15, SMDV06, DER12].

Ennalta tuntemattomassa ohjelmistossa on niin tehtävän laajuutta kuin sopivan aloituskohdan löytymistä vaikea selvittää. Erityisesti sovelluksen koon kasvaessa sopivan kohdan löytäminen vaikeutuu. Myös uusien ohjelmointirajapintojen (API, Application Programming Interface) kohdalla on useasti vaikea hahmottaa ohjelmointirajapinnan rakenne ja sen tarjoamien luokkien suhteet toisiin luokkiin [MXBK05]. Kuten normaaleissa projekteissa niin myös rajapintojen käytössä suuri koko vaikeuttaa oikean kohdan ja tarvittavien luokkien löytämistä.

Erilaisten, yllä mainittujen ongelmien ratkaisemiseen on luotu useita erilaisia työkaluja, joita ohjelmoija voi käyttää. Ohjelmoijien käytössä on myös erilaisia sosiaalisia medioita, joista löytyy ratkaisuja ongelmiin.

Tutkielman rakenne muodostuu seuraavasti: alussa tarkastellaan tyypillisiä ongelmatilanteita ja niissä syntyneitä kysymyksiä, joita ohjelmoijat ongelmia ratkoessaan esittävät. Sen jälkeen tarkastellaan muutamia juuri näihin kysymyksiin vastaamiseen suunnattuja työkaluja ja ohjelmistoja sekä lo-

puksi kerron, kuinka ohjelmoija voi löytää sosiaalisesta mediasta vastauksia kysymyksiinsä.

2 Tutkimusmenetelmä

Tässä laadullisessa tutkielmassani käytin aineiston valinnassa

1. harkinnanvaraista otantaa ja
2. lumipallotekniikkaa.

Harkinnanvarainen otanta ja aineistolähtöisyys ovat keskeisiä tekijöitä laadullisessa tutkimuksessa [Tuo09, s. 16-20]. Harkinnanvaraisella otannalla tarkoitetaan sitä, että aineisto valitaan tutkijan asettamien kriteereiden perusteella [SK06].

Tuomi ja Sarajärvi kuvaavat aineistonhankintamenetelmänä lumipallotekniikkaa, jossa aineiston hankinnan alkuvaiheessa tiedetään tiedonantaja, joka johdattaa tutkijan toisen informantin pariin. Aineiston hankinta etenee siten, että tutkija etenee tiedonantajasta toiseen sitä mukaa, kun hän löytää uusia informantteja [Esk96, s. 88]. Termi lumipallo tulee siitä, että otos kertyy kuin pyörivä lumipallo, kasvaen kierros kierrokselta.

Tutkimuskysymykset vaikuttavat luonnollisesti tutkittavan aineiston määrään ja luonteeseen. Lähdeaineistoja etsin Google Scholarista sekä ACM:stä.

Oli haastavaa löytää oikeat avainsanat etsiessäni lähtöartikkeleita tutkielmaani. Tein lukuisia hakuja käyttämällä erityyppisiä yhdistelmiä sanoista ohjelmisto, virheenkorjaus, virheentunnistus, virhe, järjestelmäkehitys, ohjelmistokehitys, työkalut, virheenetsintä jne. niiden englanninkielisillä käännöksillä. Tutustuin kunkin haun tuloksena löytyneen artikkelin otsikkoon, tiivistelmään, viitteisiin (reference) sekä viittauksiin (cited by). Valitsin työni lähtöartikkeleiksi harkinnanvaraisella/teoreettisella otoksella arvioni perusteella tutkimuskysymyksiini parhaiten sopivat artikkelit:

1. How Developers Search for Code: A Case Study [SSE15]
2. Specifying What to Search for [Rei09]

: Valittuani lähtöartikkelit etenin aineiston valinnassa lumipallotekniikalla käyden läpi lähdeartikkelien viitteet ja viittaukset sekä näiden artikkelien

viitteet ja viittaukset arvioiden artikkelin oleellisuutta tutkielmassani otsikon ja tiivistelmän perusteella. Hakuavaruuden kasvettua noin kahteensataan artikkeliin lopetin määrätietoisen hakuavaruuden laajentamisen ja keskityin pienentämään hakuavaruutta karsomalla mahdollisesti oleellisten artikkelien joukkoa artikkelien otsikon, tiivistelmän ja ensimmäisten kappaleiden perusteella. Näin sain valittua hakuavaruudesta noin 15 varmasti tutkielmaan sopivaa artikkelia. Kirjallisuuskatsauksen edetessä lähdeaineistoon nousi mukaan muutama aiemmassa karsinnassa hylätty, mutta valituissa artikkeleissa kiinnostavasti viitattu artikkeli.

Tutkimuksen edetessä löytyi muutamia kiinnostavia ja tutkimusta tukevia lähdeaineistoja, jotka eivät olleet löytyneet alkuperäisessä haussa.

Lähdeaineistona on lisäksi laadullista tutkimusta valottavia teoksia.

2.1 Tutkimuskysymykset

1. Millaisia ongelmia ohjelmistokehittäjät kohtaavat ohjelmoidessaan.
2. Mitä työkaluja näiden ongelmien ratkaisemiseen on.

3 Tilanteita, joissa ongelmia esiintyy

Tutkiessaan jo olemassaolevaa koodia ohjelmoija pyrkii erilaisilla loogisilla kysymyksillä sisäistämään koodin toimintaa. Näihin kysymyksiin vastaaminen ei ole aina helppoa saati nopeaa: kysymysten vastausten löytymiseen saattaa kulua jopa puolet ohjelmointiin käytetystä ajasta [KAM05]. Erilaisia kysymyksiä tulee mieleen erityisesti silloin, kun työskentelee uuden ominaisuuden tai muutostyön parissa tai kun työstettävä ohjelmakoodi on ohjelmoijalle tuntematonta. Ohjelmoija ei voi löytää kunnollista vastausta ongelmiinsa perehtymättä riittävässä määrin ohjelmakoodin toimintaan.

Riskinä on, että ohjelmoija ei löydä ongelmakohtaa vaan päätyy tuottamaan uutta koodia olemassaolevan koodin uusiokäytön sijaan. Ohjelmoijat muistuttavat pienimuotoisesti taiteilijoita [Gra03] siinä, että he tuntuvat haluavan mieluummin tuottaa kaiken itse sen sijaan, että käyttäisivät aikaa tutustuakseen muiden tuottamaan koodiin ja jatkojalostaakseen sitä.

Ohjelmoitaessa on kuitenkin aina tärkeää uusiokäyttää mahdollisimman paljon olemassaolevaa koodia ja kirjoittaa uutta koodia vain mikäli valmista toteutusta ei ole olemassa. Koodia uusiokäyttämällä säästyy jo olemassaole-

van koodin uudelleen kirjoittamiselta ja samalla riski uusien bugien tuottamisesta pienenee. Ohjelmakoodin uusiokäyttämisessä on kuitenkin myös riskinsä; ohjelmakoodissa voi olla tuntemattomia ohjelmointivirheitä, tietoturvariskejä tai käytössä oleva algoritmi saattaa olla aivan liian tehoton/hidas.

Usein kysymykseen vastausta haettaessa tulee esiin useita uusia kysymykseen liittyviä, alkuperäistä kysymystä tarkentavia kysymyksiä, joihin vastaaminen lopulta edesauttaa alkuperäiseen kysymykseen vastaamista [SMDV06].

Alla tarkastellaan sellaisia tilanteita ohjelmointiprosessissa, jossa kysymyksiä tyypillisesti ilmenee. Kuvaan myös, millaisia kysymyksiä kussakin tilanteessa esitetään sekä miten tällaisiin kysymyksiin haetaan vastauksia mahdollisesti erilaisia apuohjelmia käyttäen. Sen jälkeen tarkastellaan muutamia vastaamiseen käytettyjä työkaluja ja ohjelmistoja.

3.1 Tehtävän oleellisten aloituskohtien löytäminen

Uuden ohjelmointitehtävän aloittaminen on usein haastavaa, vaikka tiedossa olisi suurinpiirtein se toiminnallisuus tai korjaus, joka on tarkoitus tehdä. Tilanne on sama, oli sitten kyseessä olemassaolevan ohjelmiston toimintavirheen korjaaminen tai uuden ominaisuuden lisääminen ohjelmaan. Erityisen haastavaksi aloittamisen tekee se, jos ohjelman koodi ei ole ennalta tuttua. Tällöin ennen tehtävän aloittamista tulee selvittää itselleen tarkemmin ohjelman rakennetta ja toimintaa sekä selvittää, mitkä ovat oleellisia kohtia tehtävän muutoksen kannalta.

Eräs tapa aloituskohdan löytämiseen on arvata sopivia aiheeseen liittyviä avainsanoja, joiden perusteella voi hakea kohtia ohjelmakoodista. Tässä metodissa on haasteena niin oikeiden avainsanojen keksiminen [Rei09] kuin epäoleellisten tulosten suuri määrä, mikä hidastaa prosessia entisestään. Haku saattaa johdatella kokonaan väärään suuntaan hidastaen entisestään oleellisen ohjelmakoodin kohdan paikantamista.

Toinen tapa sopivan aloituskohdan löytämiseen on ohjelmistoympäristöjen tarjoamat luokka- ja pakkauskaaviotyökalut. Nämä saattavat nopeuttaa hakua, mutta myös näiden käytön haasteena on sopivien avainsanojen löytäminen [Rei09]. Kaikki ohjelmoijat eivät myöskään osaa tehokkaasti käyttää tarjolla olevia luokka- ja pakkauskaaviotyökaluja.

Myös virheenjäljitintä (debugger) voi käyttää sopivan aloituskohdan paikantamiseen, merkitsemällä ohjelmakoodiin pysähdyskohtia (break point)

kohtiin, joiden arvioi olevan sopivia. Seuraamalla ohjelman suoritusta virheenjäljittimellä näkee helposti, pysähtyykö ohjelman suoritus merkittyihin pysähdyskohtiin eli eteneekö koodin logiikka ohjelmoijan ajattelemalla tavalla. Tässäkin, kuten aiemmassakin aloituskohtien paikantamiseen käytetyissä tavoissa, on haasteena sopivien pysähdyskohtien löytäminen.

Virheenjäljitintä voidaan käyttää myös toisella tapaa: varmistetaan oleellinen kohta [KAM05] käyttämällä muilla tavoilla saatuja arvauksia aloituskohdasta pysähdyspisteiden kohtana, jolloin virheenjäljittimellä tarkistetaan, suoritetaanko ohjelmakoodia oikeasti näistä kohdista.

Kuitenkaan mikään edellä mainituista tavoista ei ole erityisen tehokas löytämään oleellista aloituskohtaa. Haasteeksi kaikissa tavoissa muodostuu suhteellisen suuri määrä täysin epäolennaisia tuloksia [KAM05], jotka vievät ohjelmistokehittäjän aikaa ja saattavat pahimmillaan johdatella ohjelmoijan pitkäksi aikaa tarkastelemaan väärää aluetta ohjelmakoodista, ennen kuin kohdan epäolennaisuus selviää hänelle.

Tärkeää on kuitenkin löytää oikea aloituskohta, olipa keino mikä tahansa.

3.2 Riippuvuuksien hahmottaminen

Kun sopiva aloituskohta on tiedossa, voi ohjelmoija viimein keskittyä tarkastelemaan kyseistä ohjelmakoodin kohtaa, sen suhdetta ja riippuvuuksia muuhun ohjelmakoodiin [SMDV06].

Tarkastelemalla aloituskohdan ohjelmakoodissa olevia luokkia ja niiden suhdetta muuhun ohjelmakoodiin, kuitenkin aluksi keskittyen läheisiin luokkiin, niiden rakenteisiin ja metodeihin, on tarkoitus pyrkiä parantamaan ja sisäistämään omaa käsitystä oleellisesta ohjelmakoodista. Tässä tyypillisesti tarkastellaan, mitä tietorakenteita ja luokkia oleellisessa ohjelmakoodin kohdassa käytetään sekä mitä metodeja on luokassa, jossa oleellinen kohta sijaitsee. Tyypillisesti tarkastellaan myös luokan perimää sekä sen toteuttamia rajapintoja keskittyen tarkastelussa vain lähimpien luokkien tarkasteluun, jolloin muodostetaan pieneltä alueelta tarkka yleiskuva [SMDV06]. Useat ohjelmointiympäristöt tukevat yllämainittua hyvin tarjoamalla helpon navigoinnin luokkien ja niiden riippuvuuksien välillä. Myös ohjelmointiympäristöjen tarjoamat luokka- ja pakettitason tarkasteluun tarkoitetut työkalut auttavat ohjelmoijaa visaisissa kysymyksissä.

3.3 Koodin konseptien selvittäminen

Kun oleellisen aloituskohdan läheiset riippuvuudet ja rakenne ovat selvillä, on seuraava askel tarkastella koodin korkean tason konsepteja. Samalla muodostetaan kokonaiskuva ohjelmiston toiminnasta. Samalla tarkastellaan sitä, missä oleellisen aloituskohdan sisältävä esiintymä luokasta luodaan ja mitä parametrejä sille annetaan. Tietorakenteiden kohdalla myös tietorakenteen oikeaoppisen käytön selvittäminen auttaa [KAM05]. Lisäksi tarkastellaan sitä, missä oleellisen kohdan luokkaa ja sen esiintymiä käytetään sekä mitkä kapseloivat kyseisiä esiintymiä ja mihin niitä käytetään [SMDV06].

Ohjelmointiympäristöistä on tässä apua. Useat ohjelmointiympäristöt tarjoavat mahdollisuuden hakea paikkoja, mistä metodia kutsutaan sekä paikkoja, missä tiettyyn tyyppiin (Type) viitataan. Nämä auttavat selvittämään kyseisen kohdan käytön laajuutta ja paikkoja ohjelmakoodissa. Aloittelijoille tavanomainen, mutta ei kovin hyvä, tapa vastaavaan selvitykseen on esimerkiksi muuttaa metodin määrittelyä (signature) siten, että metodia ei enää löydy. Tämän jälkeen kääntämällä ohjelmakoodin uudestaan voi nähdä kaikki kyseistä metodia käyttävät kohdat, sillä kääntäjä raportoi käännösvirheestä niistä kohdista, jotka käyttivät (= yrittävät käyttää) kyseistä metodia. Vastaavasti luokan nimeä muuttamalla ja sen jälkeen ohjelmankoodia kääntämällä ja virheitä tarkastelemalla saa selville, missä luokkaa on käytetty. Tämä ei ole suositeltu tapa, mutta on kuitenkin aloittelijoiden keskuudessa tyypillinen.

3.4 Ohjelmointirajapinnan käyttäminen

Niin ohjelman toimintaa tarkastellessa kuin uutta ominaisuutta luodessa tulee useasti vastaan ennalta tuntemattomia ohjelmointirajapintoja (API). Joskus ohjelmointirajapinnat ovat helppokäyttöisiä, mutta useasti, varsinkin isompien kirjastojen (Library) kohdalla ohjelmointirajapinnat ovat monipuolisia ja siten vaikeammin sisäistettävissä.

Uuteen ohjelmointirajapintaan tutustuminen aloitetaan usein varmistamalla ensin, että kyseinen ohjelmointirajapinta varmasti tarjoaa tarkoitukseen sopivan toiminnallisuuden. Tämän jälkeen voi aloittaa tarkemman tutustumisen ohjelmointirajapintaan.

Vastaavasti, kuten uuteen ohjelmistoon tutustuttaessa, pyritään aluksi paikantamaan ne luokat ja metodit, jotka ovat tarpeen. Niiden löydyttyä

ryhdytään selvittämään löydettyjen luokkien ja metodien suhteita muihin ohjelmointirajapinnan käyttämiin tyyppeihin (type). Myös dokumentaation lukeminen auttaa, mutta se harvoin tarjoaa yleiskuvaa siitä, mihin luokka yleiskuvassa asettuu tai minkä muiden luokkien kanssa sitä tyypillisesti käytetään [DER12].

Ohjelmointiympäristöt tarjoavat vain heikosti apua uusien ohjelmointirajapintojen käytössä [MXBK05]. Usein ohjelmistoympäristöt tarjoavat automaattista täydennystä vasta silloin, kun ohjelmoija tietää, mitä luokkia ja rajapintoja tarvitaan. Niinpä useimmiten ohjelmointirajapinnan käytön sisäistämiseen käytetään vain ohjelmointirajapinnan tarjoamaa sisäistä dokumentaatiota, mikä harvoin auttaa yleiskuvan luomisessa. Myös tästä dokumentaatiosta oleellisien kohtien löytäminen on vaikeaa, sillä sopivan avainsanan keksiminen on useasti haastavaa [Rei09]. Toinen tapa tutustua ohjelmointirajapinnan käyttöön on etsiä omaan tarkoitukseen sopivia esimerkkejä kyseisen ohjelmointirajapinnan käytöstä. Tällöin haasteeksi muodostuu esimerkkien suuri määrä sekä esimerkkien vaihteleva laatu [ZBY12]. Lisäksi käytössä olevasta ohjelmointirajapinnasta riippuen sopivien esimerkkien löytäminen on joskus hyvinkin haastavaa, ja ohjelmoijat, jotka hakevat esimerkkejä, päätyvät usein palaamaan ohjelmointirajapinnan dokumentaatioon [DER12]. Tämä kuitenkin riippuu pitkälti käytössä olevasta ohjelmointirajapinnasta sekä siitä, miten tyypillistä asiaa sillä on tekemässä.

4 Työkaluja ohjelmointiprosessin tueksi

Ohjelmistokehittäjä ei ole aivan yksin “tyhmän” tekstieditorin kanssa etsiessään vastauksia ohjelmointiprosessin aikana esiintulleisiin kysymyksiinsä, sillä ohjelmistokehittäjiä varten on luotu erilaisia ohjelmistoja ja apuvälineitä työn tueksi. Yksi haasteista on sopivan työkalun valinta kulloinkin kyseessä olevaan tehtävään/kysymykseen: kaikki ohjelmoijan avuksi tehdyt työkalut kun eivät osaa vastata kaikkiin kysymyksiin tai kenties lainkaan auttaa juuri akuuttiin kysymykseen. Toinen haaste on se, että vaikka ohjelmistokehittäjä valitsee sopivan työkalun, käyttää hän todennäköisesti vain osaa tarkoitukseen luoduista ominaisuuksista [KM04].

Shepherd ja Murphy ehdottavat ratkaisuksi kysymyksiin vastaamiseen työkalua, joka tarkkailee ohjelmoijan toimintaa automaattisesti samalla tun-

nistaen kohdat, missä ohjelmoijalla on tyypillisesti vaikeuksia. Sillä on myös valmiiksi määritelty informaatio auttamaan alkuun pääsyssä, joka auttaa löytämään sopivan työkalun vastaamaan yleisiin kysymyksiin. Ajanmittaan työkalu myös auttaa ohjelmoijaa oppimaan yhä tehokkaammin käyttämään tarjolla olevia työkaluja ja ohjaa aktiivisesti niiden käyttöön [SM08].

Alla esittelen tyypillisiä ohjelmistokehittäjän elämän helpottamiseksi luotuja työkaluja. Osa työkaluista on tarkoitettu auttamaan yleisellä tasolla, kun taas toiset auttavat vain pienessä, mutta sitäkin haastavammassa osassa.

4.1 Ohjelmointiympäristö (IDE, Interactive Development Environment)

Ohjelmointiympäristö on ohjelmisto, joka pyrkii tukemaan ohjelmistokehittäjän arkea tarjoamalla kaikki olennaisimmat työkalut ohjelmiston kehittämiseen [MKF06]. Ohjelmointiympäristö koostuu tyypillisesti tekstieditorista, joka tukee ohjelmakoodin syntaksiväritystä, ja tukee ohjelmakoodin kirjoittamista tarjoamalla automaattista koodin täydennystä kielen standardikirjaston ja jo käytettyjen kirjastojen perusteella. Ohjelmointiympäristö ei kuitenkaan osaa ehdottaa uusien aiemmin projektissa käyttämättömien kirjastojen käyttöönottoa ilman, että ohjelmoija lisää itse viitteen kyseiseen kirjastoon [MXBK05]. Ohjelmointiympäristö tarjoaa myös joko hyvän integraation ulkoiseen kääntäjään tai sisäänrakennetun kääntäjän, jonka avulla editori kertoo ohjelmoijalle ohjelmakoodin syntaksivirheistä samanaikaisesti, kun ohjelmakoodia kirjoitetaan. Näin ohjelmoija tunnistaa ja pystyy korjaamaan syntaksivirheet mahdollisimman nopeasti. Useissa ohjelmointiympäristöissä on myös sisäänrakennettu virheenetsin, joka avustaa ohjelmakoodin suorituksen tarkastelun ohjelmakoodille rivi riviltä. Ohjelmointiympäristöt tarjoavat myös tiedostoselaimen projektin tiedostoille sekä erilaisia luokka- ja pakettitason visualisointi- sekä selaustyökaluja [MKF06].

4.2 API-doc ja projektien doc

Varsinkin isompien ohjelmistoprojektien ja muiden käyttöön tarkoitettujen ohjelmistokirjastojen kohdalla dokumentaation merkitys kasvaa, sillä koodin katselmointipohjainen koodin omaksuminen ei onnistu enää nopeasti.

Ohjelmakoodin yhteydessä olevan dokumentaation tarkoituksena on selventää kyseisen luokan käyttötarkoitusta ja toimintaa. Vastaavasti luokan

metodien yhteydessä oleva dokumentaatio avaa kyseisen metodin toiminnallisuutta [Kra99]. Dokumentaation tavoitteena on siis saada koodin jatkokehittäjä taikka uudelleenkäyttäjä ymmärtämään koodin toiminta siinä määrin, että hänen ei tarvitse lukea ja tutustua kaikkeen ohjelmakoodiin (mikä on hidasta) vaan hän onnistuu lukemalla kompaktin dokumentaation sisäistämään nopeasti ohjelmakoodin toiminnan ja käyttötarkoituksen.

Projektien dokumentaatiota on ohjelmakoodin seassa olevan dokumentaation lisäksi erillisissä tiedostoissa. Tällaiset dokumentit ovat useasti projektissa ohjelmoineiden ohjelmoijien tekemiä sekä projektin tuotoksen käyttäjien luomia [Lou06].

Dokumentaation käytettävyyden ja saatavuuden parantamiseksi on myös luotu useita projekteja, kuten: `https://readthedocs.org/`Read the Docs. Read the docs pyrkii tekemään ohjelmoijille helpoksi julkaista ohjelman dokumentaatio ja loppukäyttäjien näkökulmasta se on muodostumassa luotettavaksi paikaksi hakea dokumentaatiota eri avoimen lähdekoodin ohjelmistoihin.

Ohjelmistojen dokumentaatio muodostuu sekä ohjelmakoodin yhteyteen kirjoitetusta ohjelmakoodin dokumentaatiosta että ohjelmiston/kirjaston wikiin [Lou06] tai muualle koodista irralleen kirjotetuista ohjeista ja neuvoista ohjelmiston käyttöön. Tällainen ohjelmakoodista irrallaan oleva dokumentaatio kattaa useasti ohjeita ohjelmiston käyttöönottamisessa, tarjoaa ohjeita sen konfiguroimiseen, korjaa yleisiä väärinymmärryksiä ohjelmistosta sekä muita projektiin sopivia ohjeita.

Useat kirjastot tarjoavat dokumentaation lisäksi esimerkkejä kirjaston käytöstä, mutta tyypillisesti nämä esimerkit ovat vain hyvin yksinkertaisista kirjaston käyttötarkoituksista.

4.3 S^6-projekti

Vaikka avointa lähdekoodia ja erilaisia kirjastoja on tarjolla paljon, ei niiden uudelleenkäyttäminen ole kuitenkaan aina helppoa. Erityisen vaikeaa on käyttötarkoitukseen sopivan kirjaston löytäminen.

S^6-projekti pyrkii helpottamaan sopivan koodin hakemista ja tukee siten koodin uudelleenkäyttöä sekä erilaisten ohjelmistokirjastojen löytämistä. S^6-projekti mahdollistaa sopivien luokkien ja metodien haun siten, että ohjelmoija tarjoaa testitapauksia, joiden rakennetta ja semantiikkaa automatisoidusti tarkastelemalla S^6-projekti pystyy rajaamaan hakutuloksia. Projekti

pyrkii myös automatisoimaan tarvittavat muunnokset eri tyyppien (Type) välillä, jotta eriävät tyypit eivät rajoittaisi järjestelmän tarjoamia hakutuloksia liikaa [Rei09]. Haku ei kuitenkaan tarjoa ainoastaan juuri ohjelmistokehittäjän tarjoamaan esimerkkiin sopivaa vastausta, vaan näyttää myös suunnilleen siihen sopiva ratkaisuja. Tämä ominaisuus on kehitetty S^6-projektiin siksi, että ohjelmistokehittäjä tietää usein vain suuntaa-antavasti, mitä hän oikeasti tarvitsee.

4.4 Jungloid-louhinta

Suuri määrä mahdollisia ohjelmointirajapintoja tekee kaikkien tarpeellisten ohjelmointirajapintojen ulkoa opettelun mahdottomaksi sekä vaikeuttaa tarpeeseen soveltuvan ohjelmointirajapinnan löytämistä. Mahdollisesti sopivan ohjelmointirajapinnan löydettyään ohjelmoijan haasteeksi saattaa muodostua se, että ohjelmointirajapinta käyttää eri tyyppejä kuin mitä on tarjolla. Lähtötyypin muuntaminen ohjelmistorajapinnan tarpeeseen sopivaan tyyppiin ei ole aina helppoa eikä suoraviivaista. Joskus tyyppi tulee kierrättää usean muun tyypin kautta, jotta kohdetyyppiin päästään. Se tuottaa ohjelmoijalle paljon haasteita erityisesti silloin, jos tarjolla olevat tyypit eivät ole ohjelmoijalle entuudestaan tuttuja.

Jungloid-louhinta pyrkii auttamaan ohjelmoijaa tällaisessa tilanteessa. Jungloidit määritellään seuraavasti: $\lambda x.\, e \;:\; \tau_{in} \;\rightarrow\; \tau_{out}$ ja jungloid-haku määritellään parina: (τ_{in}, τ_{out}), missä τ_{in} ja τ_{out} ovat tyyppejä, joilla kuvataan, mistä tyypistä mihin tyyppiin muunnos halutaan tehdä. Jungloid-louhinnan taustalla on tietovarasto erilaisista tyypeistä ja niiden suhteista. Haku toimii siten, että kun tiedetään lähtötyyppi τ_{in} sekä kohdetyyppi τ_{out}, voidaan tyyppien välille hakea erilaisia reittejä tyyppien suhdeverkosta. Tämä haku onnistuu perusverkkoalgoritmeilla. Haun laadun parantamiseksi tyyppejä voidaan muunnella ja yleistää ennen haun suorittamista. Esimerkiksi perintää tai rajapintoja käyttävä luokka voidaan tulkita jonain yläluokan tyyppinä tai rajapinnan tyyppinä, kun jungloid-haku suoritetaan [MXBK05].

4.5 Whyline

Ohjelmointiprosessin aikana ohjelmistokehittäjien kysymyksiin vastaamiseksi on luotu myös Whyline-ohjelma [KM04], joka pyrkii auttamaan ohjelmistokehittäjiä vastaamaalla seuraavanlaisiin kysymyksiin:

- Miksi jokin toimii?
- Miksi jokin ei toimi odotetusti?
- Miksi jotain tapahtuu?
- Miksi jotain ei tapahdu?

Tämän tyyppisiin kysymyksiin harva työkalu tarjoaa vastauksia - suurin osa työkaluista tarjoaa korkeintaan mahdollisuuden tarkastella ohjelman toimintaa suorituksen aikana, mutta jättää kokonaan vastauksen selvittämisen ohjelmoijan tehtäväksi. Whyline pyrkii helpottamaan tämän tyyppisiin kysymyksiin vastaamista; se ei kerro suoraan oikeaa vastausta, vaan ohjaa ja opastaa ongelmanratkaisuprosessissa ja siinä käytettyjen työkalujen tehokkaassa käytössä - ja siten vähentää ohjelmointiongelman ratkaisuun käytettyä aikaa [KM04].

4.6 GitHub

GitHub tarjoaa niin yksittäisille ohjelmistokehittäjille kuin organisaatioille keinon hallita julkisia ja yksityisiä ohjelmistovarantoja (repository) Git-versionhallintajärjestelmällä. GitHubissa on yli 5 miljoonaa avointa lähdekoodivarastoa. Ison lähdekoodivarastomäärän hallintaa helpottamaan GitHubissa voi merkitä itseään mahdollisesti kiinnostavia lähdekoodivarastoja tähdellä. Tähdet on tarkoitettu nimenomaan kuvaamaan kiinnostusta ja auttamaan lähdekoodivarastojen löytymistä helposti myöhempää käyttöä varten - ei niinkään osoittamaan pitävänsä tai käyttävänsä kyseisen lähdekoodivaraston ohjelmaa [BBS13], kuten merkistä voisi olettaa. GitHubissa voi myös seurata muiden projekteja sekä GitHubiin rekisteröityneitä käyttäjiä.

GitHub tarjoaa kielipohjaisen lajittelun suosittujen lähdekoodivarastojen tarkasteluun. Lisäksi GitHubista voi hakea ohjelmakoodia kielen ja avainsanojen perusteella [BBS13]. Kuten muissakin avainsanapohjaisissa hauissa avainsanojen arvaaminen/löytäminen on haastavaa [Rei09], mutta sopivan ohjelmointirajapinnan käyttöesimerkin haku on taas muita työkaluja helpompaa, sillä sopivat avainsanat, luokat ja metodit ovat tyypillisesti silloin tiedossa. GitHubista esimerkin hakemisessa on myös se hyvä puoli, että koodia ei ole kirjoitettu vain esimerkiksi, vaan se on oikeasti jossain tuotantokäytössä.

5 Sosiaalisen median hyödyntäminen

Sosiaalinen media, kuten Facebook ja Imdb, tarjoavat paljon hyödyllisiä suosituksia kaikille internetin käyttäjille, kuten mitä elokuvia kannattaisi katsoa tai mitä tuttavat ja lähipiiri tekevät parhaillaan. Ammattilaisten sivustot, kuten StackOverflow, ovat vastaavasti ohjelmistokehittäjille erityisesti suunnattuja sosiaalisia medioita.

Esimerkkien haku on oleellinen osa nykyaikaista ohjelmistokehitystä [ZBY12].

Shepherd ja Murphy ehdottavat ratkaisuksi kysymyksiin vastaamiseen työkalua, joka tarkkailee ohjelmoijan toimintaa automaattisesti samalla tunnistaen kohdat, missä ohjelmoijalla on tyypillisesti vaikeuksia. Sillä on myös valmiiksi määritelty informaatio auttamaan alkuun pääsyssä, joka auttaa löytämään sopivan työkalun vastaamaan yleisiin kysymyksiin. Ajanmittaan työkalu myös auttaa ohjelmoijaa oppimaan yhä tehokkaammin käyttämään tarjolla olevia työkaluja ja ohjaa aktiivisesti niiden käyttöön [SM08].

5.1 Kysymys- ja vastauspalstat

Kysymys- ja vastauspalstat (Q&A) ovat yleinen tapa hakea vastauksia ongelmiin kaikilla elämän osa-alueilla. Sama pätee ohjelmointikehityksessä: kysymys- ja vastauspalstoja käytetään paljon etsittäessä sopivia ratkaisuja ongelmiin sekä haettaessa esimerkkejä tietyn ohjelmointirajapinnan käyttöön. StackExchangen StackOverflow on erityisen suosittu ohjelmistokehittäjien keskuudessa. Sivusto keskittyy tarkasti pysymään olennaisessa eli ohjelmistokehitysaiheisten kysymyksissä ja niiden vastauksissa. Sivuston toimintaperiaate on yksinkertainen: se kerää ison osan aiheen asiantuntijoita käyttäjikseen, jotka vastaavat toistensa kysymyksiin. StackOverflowlla on yli 16 miljoonaa eri käyttäjää kuukaudessa. Käyttäjät saavat pisteitä hyvistä vastauksista, käyttäjät äänestävät jokaiseen kysymykseen tulleet vastaukset ja ne näytetään paremmuusjärjestyksessä. Näin sivulta omaan kysymykseensä vastausta etsivä ohjelmoija löytää nopeasti aiheeseen sopivimman kunnollisen vastauksen [BBS13].

StackOverflown ja GitHubin käyttäjien aktiivisuutta tutkittaessa on havaittu, että GitHubissa aktiiviset ovat keskimääräistä aktiivisempia myös vastaamaan StackOverflowssa esitettyihin kysymyksiin. Vastaavasti

GitHubissa vähemmän aktiiviset olivat aktiivisempia kysymään apua StackOverflowssa [VFS13].

5.2 ExampleOverflow

StackOverflown ja muiden vastaavien sivustojen käyttäjien haasteeksi muodostuu valtava esimerkkien ja vastausten määrä. Zagalskyn, Barzilayn ja Yehudain sivusto ExampleOverflow [ZBY12] kokoaa StackOverflowsta parhaiksi vastauksiksi merkityt koodiesimerkit. Lisäksi se tarjoaa optimoidun haun niistä. ExampleOverflow tarjoaa koodihaussa aina viisi sopivinta vastausta ja vastauksen sopimattomaksi merkitessään käyttäjä saa tilalle aina seuraavaksi sopivimman vaihtoehdon. Sivusto tarjoaa suoraan koodiesimerkkejä, joiden alla on linkit alkuperäisiin kysymyksiin ja vastauksiin. Sivuston tavoitteena on tehdä esimerkkien löytäminen mahdollisimman vaivattomaksi. Kirjoittajien havaintojen mukaan ExampleOverflow löytää sopivia esimerkkejä useissa tapauksissa yhtä hyvin tai paremmin kuin StackOverflow [ZBY12].

5.3 Wikit

Wikit ovat yhteisöllinen tapa luoda kattavia tietopankkeja, joissa usean ihmisen tietotaito ja osaaminen yhdistyvät luoden asiasta kiinnostuneille laadukasta materiaalia. Wikejä käytetään myös avoimen lähdekoodin ohjelmien dokumentointiin sekä parantamaan ohjelmointirajapintojen dokumentaatiota. Wikit koetaan myös sähköpostilistoja ja muita perinteisempiä keskustelu- sekä dokumentointimuotoja helpommiksi ja selkeämmiksi käyttää juuri yhteisöllisen muokkauksen ja tiedon ajantasaisuuden ansiosta [Lou06]. Myös GitHub tarjoaa helppokäyttöisiä wikejä ohjelmistojen dokumentointiin [sta15].

5.4 Blogit

Sosiaalisen median kasvun myötä on blogien käyttö on kasvanut myös ohjelmistokehittäjien keskuudessa. Isoissa avoimen lähdekoodin projekteissa kuten Postgresql:ssä, Gnomessa ja Python:ssa julkaistaan keskimäärin noin kahdeksan tunnin välein uusi blogijulkaisu. Julkaisut ovat keskimäärin 150-273 sanan pituisia ja ne käsittelevät:

- ohjelmiston vaatimuksia

- ohjelmiston ympärille muodostunutta yhteisöä
- ohjelmistosta lisätiedon kertomista
- ohjelmiston käyttöönottoa ja jakelua
- suunnitteluratkaisuja
- ylläpitoa
- prosessin koordinointia ja hallintaa
- lisätietoja siitä, miten asia on toteutettu ohjelmistossa.

Useimmiten aihe liittyy sellaiseen kokonaisuuteen, jonka parissa kirjoittaja on äskettäin työskennellyt [PM11]. Myös henkilökohtaiset ohjelmistokehittäjien blogit toimivat paikkana löytää tietoa uusista ohjelmointirajapintojen ominaisuuksista sekä esimerkkejä niistä.

6 Yhteenveto

Ohjelmoitaessa on aina tärkeää uusiokäyttää mahdollisimman paljon olemassaolevaa koodia ja kirjoittaa uutta koodia vain mikäli valmista toteutusta ei ole olemassa. Koodia uusiokäyttämällä säästyy jo olemassaolevan koodin uudelleen kirjoittamiselta ja samalla riski uusien bugien tuottamisesta pienenee.

Oikean korjauspaikan löytäminen ei ole helppoa. Toisen tekemään koodiin tutustuminen (ja erityisesti tutustumisen aloittaminen!) on vaikeaa ja vie helposti turhaan aikaa, joten prosessin tehokkuuteen kannattaa panostaa. Ohjelmoijan ei kannata yrittää ratkoa ongelmiaan yksin, vaan hänen kannattaa hyödyntää avuksi ja työn tueksi tehtyjä työkaluja.

Ohjelmistotuotantoprosessin aikana ohjelmistokehittäjän kohtaamiin kysymyksiin vastaaminen ei ole aina yksinkertaista, mutta erilaiset työkalut tehostavat työtä. Työkaluja on tehty erilaiseen käyttötarkoitukseen. Yksi parhaista työkaluista on Jungloid-louhinta [MXBK05], jonka avulla on mahdollista ratkaista suuri osa ohjelmointirajapintojen käyttöön ja tyyppeihin liittyvistä kysymyksistä. Ohjelmistokehitysprosessin aikana tulevien kysymysten vastaamiseen liittyvät työkalut, kuten Whyline, on tarkoitettu ohjaamaan ohjelmistokehittäjä käyttämään tehtävään tarkoitettuja työkaluja ja opettaa niiden tehokkaampaa käyttöä [KM04].

Työkaluja on moneen käyttötarkoitukseen ja ne kehittyvät jatkuvasti. Ohjelmoijan on tarpeen tuntea eri työvälineet ja pyrkiä löytämään niistä itselleen sopivin/sopivimmat ja opetella niiden käyttöä, jotta ohjelmointityö olisi mahdollisimman tehokasta.

Lähteet

[BBS13] Begel, Andrew, Bosch, Jan ja Storey, Margaret Anne: *Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder*. IEEE Softw., 30(1):52–66, tammikuu 2013, ISSN 0740-7459. http://dx.doi.org/10.1109/MS.2013.13.

[DER12] Duala-Ekoko, Ekwa ja Robillard, Martin P.: *Asking and Answering Questions About Unfamiliar APIs: An Exploratory Study*. Teoksessa *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, sivut 266–276, Piscataway, NJ, USA, 2012. IEEE Press, ISBN 978-1-4673-1067-3. http://dl.acm.org/citation.cfm?id=2337223.2337255.

[Esk96] Eskola, Jari.: *Johdatus laadulliseen tutkimukseen*. Lapin yliopisto, Rovaniemi, 1996, ISBN 951-634-468-2.

[Gra03] Graham, Paul: *Hackers and Painters*, 2003. http://www.paulgraham.com/hp.html, vierailtu 2015-12-20 .

[KAM05] Ko, Andrew J., Aung, Htet ja Myers, Brad A.: *Eliciting Design Requirements for Maintenance-oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks*. Teoksessa *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, sivut 126–135, New York, NY, USA, 2005. ACM, ISBN 1-58113-963-2. http://doi.acm.org/10.1145/1062455.1062492.

[KM04] Ko, Andrew J. ja Myers, Brad A.: *Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior*. Teoksessa *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, sivut 151–158, New York, NY, USA, 2004. ACM, ISBN 1-58113-702-8. http://doi.acm.org/10.1145/985692.985712.

[Kra99] Kramer, Douglas: *API Documentation from Source Code Comments: A Case Study of Javadoc*. Teoksessa *Proceedings of the 17th Annual International Conference on Computer Documentation*, SIGDOC '99, sivut 147–153, New York, NY, USA,

1999. ACM, ISBN 1-58113-072-4. http://doi.acm.org/10.1145/318372.318577.

[Lou06] Louridas, Panagiotis: *Using Wikis in Software Development.* IEEE Softw., 23(2):88–91, maaliskuu 2006, ISSN 0740-7459. http://dx.doi.org/10.1109/MS.2006.62.

[MKF06] Murphy, G.C., Kersten, M. ja Findlater, L.: *How are Java software developers using the Elipse IDE?* Software, IEEE, 23(4):76–83, July 2006, ISSN 0740-7459.

[MXBK05] Mandelin, David, Xu, Lin, Bodík, Rastislav ja Kimelman, Doug: *Jungloid Mining: Helping to Navigate the API Jungle.* SIGPLAN Not., 40(6):48–61, kesäkuu 2005, ISSN 0362-1340. http://doi.acm.org/10.1145/1064978.1065018.

[Pel15] Pelegrin, Williams: *Facebook went down for an hour and people acctually called 911 to report it*, 2015. https://help.github.com/articles/about-github-wikis/, vierailtu 2015-10-19 .

[PL15] Pauli Lönnroth, Arja Törmälä: *Hyväksymistestaussuunnitelma - hakeutujan palvelut ja todennetun osaamisen rekisteri*, 2015. https://confluence.csc.fi/download/attachments/27591656/Hyvaksymistestaussuunnitelma_-_Hakeutujan_palvelut_ja_TOR.pdf?version=8&modificationDate=1371458410538, vierailtu 2015-10-20 .

[PM11] Pagano, Dennis ja Maalej, Walid: *How Do Developers Blog?: An Exploratory Study.* Teoksessa *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, sivut 123–132, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0574-7. http://doi.acm.org/10.1145/1985441.1985461.

[PW15] Preston-Werner, Tom: *Semantic Versioning*, 2015. http://semver.org/, vierailtu 2015-12-20 .

[Rei09] Reiss, Steven P.: *Specifying What to Search for.* Teoksessa *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, SUITE '09, sivut 41–44, Washington, DC, USA, 2009. IEEE Computer Society,

ISBN 978-1-4244-3740-5. http://dx.doi.org/10.1109/SUITE.2009.5070020.

[SK06] Saaranen-Kauppinen: *Aineiston määrä ja tutkittavat*, 2006. http://www.fsd.uta.fi/menetelmaopetus/kvali/L6_2.html, vierailtu 2015-12-20 .

[SM08] Shepherd, David C. ja Murphy, Gail C.: *A Sketch of the Programmer's Coach: Making Programmers More Effective*. Teoksessa *Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '08, sivut 97–100, New York, NY, USA, 2008. ACM, ISBN 978-1-60558-039-5. http://doi.acm.org/10.1145/1370114.1370139.

[SMDV06] Sillito, Jonathan, Murphy, Gail C. ja De Volder, Kris: *Questions Programmers Ask During Software Evolution Tasks*. Teoksessa *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, sivut 23–34, New York, NY, USA, 2006. ACM, ISBN 1-59593-468-5. http://doi.acm.org/10.1145/1181775.1181779.

[SSE15] Sadowski, Caitlin, Stolee, Kathryn T. ja Elbaum, Sebastian: *How Developers Search for Code: A Case Study*. Teoksessa *Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 1600 Amphitheatre Parkway, 2015.

[sta15] staff, GitHub: *About GitHub Wikis*, 2015. http://www.digitaltrends.com/mobile/users-call-911-during-facebook-outage/, vierailtu 2015-12-20 .

[Tuo09] Tuomi, Jouni.: *Laadullinen tutkimus ja sisällönanalyysi*. Tammi, 2009, ISBN 978-951-31-4865-2.

[VFS13] Vasilescu, Bogdan, Filkov, Vladimir ja Serebrenik, Alexander: *StackOverflow and GitHub: Associations Between Software De-*

velopment and Crowdsourced Knowledge. Teoksessa *Proceedings of the 2013 International Conference on Social Computing*, SOCIALCOM '13, sivut 188–195, Washington, DC, USA, 2013. IEEE Computer Society, ISBN 978-0-7695-5137-1. http://dx.doi.org/10.1109/SocialCom.2013.35.

[ZBY12] Zagalsky, Alexey, Barzilay, Ohad ja Yehudai, Amiram: *Example Overflow: Using Social Media for Code Recommendation*. Teoksessa *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, RSSE '12, sivut 38–42, Piscataway, NJ, USA, 2012. IEEE Press, ISBN 978-1-4673-1759-7. http://dl.acm.org/citation.cfm?id=2666719.2666728.