

Ongelmat Ohjelmointiprosessissa ja niiden ratkaiseminen

Jarmo Isotalo

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 28. joulukuuta 2015

Tiedekunta — Fakultet — Faculty		Laitos — Institution — Department	
Matemaattis-luonnontieteellinen		Tietojenkäsittelytieteen laitos	
Tekijä — Författare — Author			
Jarmo Isotalo			
Työn nimi — Arbetets titel — Title			
Ongelmat Ohjelmointiprosessissa ja niiden ratkaiseminen			
Oppiaine — Läroämne — Subject			
Tietojenkäsittelytiede			
Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages	
Kandidaatintutkielma	28. joulukuuta 2015	21	
Tiivistelmä — Referat — Abstract			
<p>Ohjelmistoihin joudutaan virhetilanteiden ja ominaisuuksien lisäysten takia tekemään muutoksia. Usein muutoksen tai korjauksen tekee ohjelmoija, joka ei alunperin ole ollut tuottamassa kyseistä ohjelmakoodia. Ohjelmoija pyrkii selvittämään järjestelmän toimintaa, jotta hän pääsisi nopeasti tekemään korjauksen. Usein ohjelmoija esittää itselleen kysymyksiä, joiden avulla hän pyrkii selvittämään ohjelmiston toimintaa.</p> <p>Ohjelmoijan tueksi on kehitetty useita erilaisia työkaluja ja ohjelmistoja. Moni ei kuitenkaan osaa käyttää näitä työkaluja täysimääräisesti taikka tehokkaasti osana omaa ohjelmointiprosessiaan.</p> <p>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.</p> <p>ACM Computing Classification System (CCS):</p> <p>D.2.1 [Software Engineering]: Tools</p> <p>D.2.2 [Software Engineering]: Programmer workbench</p>			
Avainsanat — Nyckelord — Keywords			
ohjelmistotuotanto, ohjelmointiprosessin kysymykset, virheen etsintä, virheen korjaus			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Sisältö

1 Johdanto	1
1.1 Tutkimusmenetelmä	3
1.2 Tutkimuskysymykset	4
2 Ongelmatilanteita	4
2.1 Ohjelmointitehtävän aloituskodan löytäminen	5
2.2 Riippuvuuksien hahmottaminen	6
2.3 Koodin konseptien selvittäminen	7
2.4 Ohjelmointirajapinnan käyttäminen	7
3 Työkaluja ohjelmointiprosessin tueksi	9
3.1 Ohjelmointiympäristö	9
3.2 API- ja projektien dokumentaatio	10
3.3 S6-projekti	12
3.4 Jungloid-louhinta	12
3.5 Whyline	13
3.6 GitHub	13
4 Sosiaalisen median hyödyntäminen	14
4.1 Kysymys- ja vastauspalstat	14
4.2 ExampleOverflow	15
4.3 Wikit	15
4.4 Blogit	15
5 Yhteenveto	16
Lähteet	18

1 Johdanto

Ohjelmistot ovat keskeisessä asemassa ihmisen elämässä ja liiketoiminnassa. Ihmisten arki on muuttunut älypuhelisten ja tablettien yleistyttyä ja niissä toimivia sovelluksia käytetään yhä enemmän. Harva liiketoiminta on mahdollista ilman tietojärjestelmiä ja esim. sähkökatkon sattuessa/pitkittyessä usean yrityksen toiminta vaiheutuu.

Myös tietojärjestelmissä olevat virheet haittaavat liiketoimintaa ja ihmisten elämää. Järjestelmiin päätyvien virheiden syntymistä pyritään estämään erilaisin menetelmin eri vaiheissa ohjelmiston kehitysprosessia. Dokumenttien (mm. määrittely) katselmointi on hyväksi havaittu keino estämään tilanetta, jossa ohjelmoija ohjelmoi virheettömän toiminnon, mutta toiminto ei vastaa loppukäyttäjän tarvetta. Ohjelmakoodiin päätyviä virheitä pyritään välttämään mm. pariohjelmoinnilla ja kattavalla ohjelmiston testauksella.

Jotta tuotantoon päätyisi mahdollisimman vähän virheitä, niitä pyritään havaitsemaan ja korjaamaan ohjelmistotuotannon aikana erilaisilla koodianalyysillä ja testaamisella (mm. yksikkötestit, integraatiotestit, end-to-end-testit, suorituskykytestit ja rasiustestit). Kaikista toimista huolimatta sovelluksia ei kuitenkaan saada täysin virheettömiksi. Erityisesti monimutkaisissa ja laajoissa ohjelmistokokonaisuuksissa on mahdotonta tunnistaa kaikki virhetilanteet. Lisäksi kattavaan virheiden etsintään ei ole aina varaa.

Kaikki virheet eivät ole samanarvoisia. Lönnroth ja Törmälä luokittelevat virheet neljään luokkaan [PL15]:

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

Oleellista on määritellä virheen korjauksen kiireellisyys. Korjauksen kiireellisyyteen vaikuttaa virheen laajuus, moneenko käyttäjään virhe vaikuttaa, ja virheen vaikutukseen, onko virhe helposti kierrettävissä vai estääkö se kokonaan kyseisen ominaisuuden käytön.

Virheidenkorjausten lisäksi ohjelmistoja muutetaan tekemällä niihin mm. pieniä käyttäjän työtä helpottavia toiminnallisuusparannuksia ja sovelluksen

suorituskykyä tai ylläpidettävyyttä parantavia korjauksia. Usein järjestelmiä myös täydennetään lisäämällä niihin uusia toiminnallisuuksia.

Paitsi järjestelmissä olevat ohjelmointivirheet myös järjestelmien ylläpitoon liittyvät 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.

Testauksella pyritään ehkäisemään muutoksen yhteydessä syntyviä sivuvaikutuksia, mutta testit harvoin tunnistavat kaikkia mahdollisia virhetilanteita. Muiden käyttöön tarkoitettujen kirjastojen kohdalla mahdollisten sivuvaikutusten riski kasvaa, mikäli muutos muuttaa ohjelman toimintaa tai korjaa pitkäaikaisen bugin. Laajassa käytössä olevat kirjastot noudattavat usein semanttista versiointia [PW15]. Tämä sallii pienissä päivityksissä vain pienet korjaukset, jotka eivät riko taikka muuta kirjaston toimintaa ja vaatii selkeän versionumeron päivityksen muutoksiin, jotka muuttavat kirjaston toimintaa siten, että kirjaston käyttäjien tulisi varautua muutoksiin kirjastoa päivitettäessä.

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. Tehokas prosessi mahdollistaa myös sen, että ohjelmoija pääsee tekemään aitoa lisäarvoa tuottavia uusia ominaisuuksia sinänsä tarpeettomien virhekorjausten sijaan.

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

1.1 Tutkimusmenetelmä

Tässä 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 tietolähde, joka johdattaa tutkijan toisen tietolähteen 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 Digital Librarystä.

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 käännoksillä. 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 määrätietoisesti 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.

1.2 Tutkimuskysymykset

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

2 Ongelmatilanteita

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 päättyy tuottamaan uutta koodia olemassaolevan koodin uusiokäytön sijaan. Ohjelmoijat muistuttavat pienimuotoisesti taiteilijoita [Gra03] siinä, että he tuntuvat ha-

luavan 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 olemassaolevan 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.

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 ongelmia tyypillisesti ilmenee. Kuvaan myös, millaisia kysymyksiä kussakin tilanteessa ongelmaa ratkaistaessa esitetään sekä miten näihin kysymyksiin haetaan vastauksia erilaisia apuohjelmia käyttäen. Sen jälkeen tarkastellaan muutamia vastaamiseen käytettyjä työkaluja ja ohjelmistoja. Tarkastelussa mukailen artikkeleissa Eliciting Design Requirements for Maintenance-oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks [KAM05] ja Questions Programmers Ask During Software Evolution Tasks [SMDV06] käytettyä hyväksi kokemaani rakennetta.

2.1 Ohjelmointitehtävän aloituskodan 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 ajatteleman 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 ohjelmoijan 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.

2.2 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 omaa käsitystään 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.

2.3 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. Tarkastellen 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 kohtia ohjelmakoodista, mistä metodia kutsutaan sekä kohtia, missä tiettyä tyyppiä käytetään. 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 kyseistä metodia. Vastaavasti luokan nimeä muuttamalla ja sen jälkeen ohjelmakoodia kääntämällä ja virheitä tarkastelemalla saa selville, missä luokkaa on käytetty. Tämä ei ole suositeltu tapa, mutta on kuitenkin aloittelijoiden keskuudessa tyypillinen.

2.4 Ohjelmointirajapinnan käyttäminen

Niin ohjelman toimintaa tarkastellessa kuin uutta ominaisuutta luodessa tulee useasti vastaan ennalta tuntemattomia ohjelmointirajapintoja (API). Jos-

kus 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 heikosti apua uusien ohjelmointirajapintojen käyttöönotossa [MXBK05]. Ohjelmistoympäristöt tarjoavat automaattista täydennystä vasta silloin, kun ohjelmoija tietää, mitä luokkia ja rajapintoja tarvitaan ja on mahdollisesti käynyt itse lisäämässä käytetyt riippuvuudet projektiin. Niinpä useimmiten ohjelmointirajapinnan käytön sisäistämiseen käytetään vain ohjelmointirajapinnan tarjoamaa sisäistä dokumentaatiota, mikä harvoin auttaa kattavan yleiskuvan luomisessa ohjelmointirajapinnasta. Myös tästä dokumentaatiosta oleellisten kohtien löytäminen on vaikeaa, sillä sopivan avainsanan keksiminen on useasti haastavaa [Rei09]. Toinen tapa tutustua ohjelmointirajapinnan käyttöön on etsiä omaan käyttötarkoitukseen sopivia esimerkkejä kyseisen ohjelmointirajapinnan käytöstä. Tällöin haasteeksi muodostuu esimerkkien suuri määrä ja niiden vaihteleva laatu [ZBY12]. Lisäksi käytössä olevasta ohjelmointirajapinnasta riippuen hyvien esimerkkien löytäminen on hyvin haastavaa, ja ohjelmoijat, jotka haavevat esimerkkejä, päätyvät usein palaamaan ohjelmointirajapinnan dokumentaatioon [DER12]. Tämä kuitenkin riippuu pitkälti käytössä olevasta ohjelmointirajapinnasta sekä siitä, miten tavanomaista asiaa sillä on tekemässä. Tutustumme myöhemmässä kappaleessa *ExampleOverflow*hun [ZBY12], joka pyrkii tekemään esimerkkien etsimisestä helpompaa.

3 Työkaluja ohjelmointiprosessin tueksi

Ohjelmoija ei ole aivan yksin “tyhmän” tekstieditorin kanssa vaan usemmien ohjelmoijalla on käytössään ohjelmointiympäristö (IDE, Integrated Development Environment). Ohjelmoijien tueksi on myös luotu työkaluja ja apuvälineitä auttaman ohjelmointiprosessissa kohdattujen ongelmien ratkaisussa. Yksi haasteista on sopivan työkalun valinta kulloinkin kyseessä olevaan tehtävään: kaikki ohjelmoijan avuksi tehdyt työkalut kun eivät osaa auttaa kaikissa ongelmissa kunnolla tai kenties lainkaan. Toinen haaste on se, että vaikka ohjelmoija valitsee sopivan työkalun, käyttää hän todennäköisesti vain osaa tarkoitukseen luoduista ominaisuuksista [KM04].

Shepherd ja Murphy ehdottavat ratkaisuksi ongelmien ratkaisuun ja ongelman ratkaisuprosessin aikana keksityjen vastaukseen johdattelevien kysymysten vastaamiseen työkalua, joka tarkkailee ohjelmoijan toimintaa automaattisesti tunnistuen kohdat, missä ohjelmoijalla on tyypillisesti vaikeuksia. Tunnistaessaan, että ohjelmoijalla on vaikeuksia edetä, se osaa ehdottaa mitä työkaluja käyttämällä ohjelmoija voisi edetä ja ratkaista kohtaamansa ongelman. Ajanmittaan työkalu myös auttaa ohjelmoijaa oppimaan yhä tehokkaammin käyttämään tarjolla olevia työkaluja ohjaten ohjelmoijaa aktiivisesti näiden työkalujen käyttöön [SM08].

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

3.1 Ohjelmointiympäristö

Ohjelmointiympäristö on ohjelmisto, joka pyrkii tukemaan ohjelmoijan arkea tarjoamalla kaikki olennaisimmat työkalut ohjelmiston kehittämiseen [MKF06]. Ohjelmointiympäristö koostuu tyypillisesti tekstieditorista, joka tukee ohjelmakoodin syntaksiväritystä sekä ohjelmakoodin kirjoittamista tarjoamalla automaattista koodin täydennystä kielen standardikirjaston ja jo käytettyjen kirjastojen perusteella.

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 eikä erillistä aikaa vievää käännös

askelta enää tarvita, sillä ohjelmointiympäristö on kääntänyt ohjelmakoodin sitä tahtia kuin se on kirjoitettu. Tämä säästää myös ohjelmoijan aikaa.

Ohjelmointiympäristöt eivät kuitenkaan ratkaise kaikkia ohjelmoijien ongelmia. Vaikka ohjelmointiympäristöt ovat usein erikoistuneet vain muutama kieliin, tarjoavat ne usein jonkinasteisen tuen hyvin monelle eri kielelle. Riippuen käytetystä ohjelmointiympäristöstä ja kielestä ohjelmointiympäristön ohjelmoijalle tarjoavat ominaisuudet saattavat siis olla eri. Useissa ohjelmointiympäristöissä on myös sisäänrakennettu virheenetsin kielille, joiden kirjoittamiseen ohjelmointiympäristö on erikoistunut, joka avustaa ohjelmakoodin suorituksen tarkastelun ohjelmakoodille rivi riviltä. Ohjelmointiympäristöt tarjoavat myös tiedostoselaimen projektin tiedostoille sekä erilaisia luokka- ja pakettitason visualisointi- sekä selaustyökaluja [MKF06].

Ohjelmointiympäristöjä voi laajentaa kuitenkin lisäohjelmilla (plugin), joiden avulla ohjelmointiympäristöön voi tuoda erilaisia lisäominaisuuksia [MKF06], kuten tuen ei ohjelmointiympäristön valmiiksi tukemalle kielille tai uuden työkalun auttamaan ongelmanratkaisussa. Kuitenkaan kaikkia ohjelmoijan työtä helpottavia työkaluja ei ole sisäänrakennettu tai saatavissa lisäohjelmana ohjelmointiympäristöön. Siksi on tärkeää, että ohjelmoija osaa käyttää myös muita työkaluja. Esimerkiksi harva ohjelmointiympäristö osaa automaattisesti ottaa käyttöön ja ehdoittaa ohjelmoijan tarvitsemaa ulkoista kirjastoa ja sen luokkia [MXBK05].

3.2 API- ja projektien dokumentaatio

Isompien ohjelmistoprojektien ja muiden käyttöön tarkoitettujen ohjelmistokirjastojen kohdalla dokumentaation merkitys kasvaa, sillä koodin katselointipohjainen koodin omaksuminen ei onnistu enää nopeasti. Ulkoisten ohjelmistojen tarkoituksena on myös se, että niiden käyttö onnistuu tunte-matta tarkalleen kyseisen ohjelmakoodin toimintaa, vaan riittää, että ohjelmoija lukee tarvittavan dokumentaation ohjelmistokirjaston julkiseen käyttöön tarkoitetuista rajapinnoista.

Ohjelmakoodin yhteydessä olevan dokumentaation tarkoituksena on selventää kyseisen luokan käyttötarkoitusta ja toimintaa. Vastaavasti 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. Ohjelmakoodin yhteydessä oleva dokumentaatio on usein eritelty ohjelmiston käyttäjän ja jatkokehittäjän välille, jolloin ohjelmiston käyttäjä saa ohjelmiston nopeammin käyttöönsä.

Ohjelmistojen dokumentaatio muodostuu siis sekä ohjelmakoodin yhteyteen kirjoitetusta ohjelmakoodin dokumentaatiosta että ohjelmiston wikiin [Lou06] tai muualle koodista irralleen kirjoitetuista ohjeista ja neuvoista ohjelmiston käyttöön. Projektien dokumentaatiota on ohjelmakoodin seassa olevan dokumentaation lisäksi erillisissä tiedostoissa. Tällaiset dokumentit ovat projektissa ohjelmoineiden ohjelmoijien tekemiä sekä projektin tuotoksen käyttäjien luomia [Lou06]. Tällainen ohjelmakoodista irrallaan oleva dokumentaatio kattaa useasti ohjeita ohjelmiston käyttöönottamisessa, tarjoaa ohjeita sen konfiguroimiseen, korjaa yleisiä väärinymmärryksiä ohjelmistosta sekä muita projektiin sopivia ohjeita. Tämä dokumentaatio on tyypillisesti omassa kansiossaan ja readme-tiedostoissa tai ohjelmiston käytössä olevassa wiki palvelussa [Lou06]. Wikipohjainen dokumentaatio on useasti helpommin myös ohjelmiston käyttäjien muokattavissa, sillä muutokset tavanomaisesti tehdään suoraan internetiselaimesta eikä ohjelmiston lähdekoodia tarvitse erikseen ladata muokatakseen dokumentaatiota. Näin wikit sallivat ohjelmiston käyttäjien helpommin muokata ja kehittää ohjelmiston dokumentaatiota.

Dokumentaation saatavuus on silti usein heikkoa ja ajantasaisen version löytäminen vaikeaa. [https://readthedocs.org/Read the Docs](https://readthedocs.org/Read%20the%20Docs) pyrkii helpottamaan ohjelman dokumentaation julkaisua, jolloin yhä useampi projekti julkaisisi dokumentaationsa selkeästi kaikkien löydettäväksi ja luettavaksi. Se on myös loppukäyttäjien näkökulmasta muodostumassa luotettavaksi paikaksi hakea dokumentaatiota avoimen lähdekoodin ohjelmistoihin.

Dokumentaation lisäksi useat ohjelmistokirjastot tarjoavat esimerkkejä niiden käytöstä. Esimerkit ovat kuitenkin hyvin yksinkertaisia ja ne auttavat vain helpoissa asioissa alkuun pääsyssä. Monimutkaisemmissa tapauksissa niistä ei ole lainkaan iloa.

3.3 S⁶-projekti

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

S⁶-projekti pyrkii helpottamaan sopivan koodin hakemista ja tukee siten koodin uudelleenkäyttöä sekä erilaisten ohjelmistokirjastojen löytämistä. S⁶-projekti mahdollistaa sopivien luokkien ja metodien haun siten, että ohjelmoija tarjoaa testitapauksia, joiden rakennetta ja semantiikkaa automatisoidusti tarkastelemalla S⁶-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 ohjelmoijan tarjoamaan esimerkkiin sopivaa vastausta, vaan näyttää myös suunnilleen siihen sopiva ratkaisuja. Tämä ominaisuus on kehitetty S⁶-projektiin siksi, että ohjelmoija tietää usein vain suuntaa-antavasti, mitä hän oikeasti tarvitsee.

3.4 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 tyyppisiä kuin mitä on tarjolla. Lähtötyypin muuntaminen ohjelmistorajapinnan tarpeeseen sopivaan tyyppiin ei ole aina helppoa eikä suoraviivaista. Joskus tyyppi tulee kierrättää usean muun tyyppin 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 tyyppisiä, joilla kuvataan, mistä tyyppistä 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].

3.5 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].

3.6 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.

4 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].

4.1 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 ohjelmistokehityksiaiheisten 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].

4.2 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].

4.3 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].

4.4 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.

5 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 ohjelmoijan 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 ohjelmoija 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 Eclipse 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>.