Millaisia kysymyksiä ohjelmoijat esittävät ja miten he hakevat niihin vastauksia

Jarmo Isotalo

Referaatti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 17. lokakuuta 2015

Sisältö

1 Johdanto

Ihmiset ovat aina hakeneet apua oppiakseen ja suoriutuakseen paremmin erilaisista tehtävistä. Ohjelmistoprojektien eri vaiheiden aikana tulee esille erilaisia kysymyksiä ohjelmakoodista [8], [9], [1]. Kysymykset liittyvät niin vieraiden ohjelmistorajapintojenen (api) käyttöön [5] kuin yleisemmin.

2 Tilanteita missä tulee ongelmia - miksi kyselee ja mitä

Ohjelmoitaessa tulee usein esiin erilaisia kysymyksiä, joihin vastaamiseen noin puolet ohjelmointiajasta kuluu [2]. Näiden ohjelmointitehtävään tutustuessa ja ohjelmointitehtävän aikana esiin tulleiden kysymysten avulla ohjelmoja pyrkii sisäistämään koodin toimintaa tarvittavissa määrin saadakseen tarvittavan tehtävän tehtyä. Tällaisia tehtäviä on esimerkiksi uuden ominaisuuden luominen olemassa olevaan ohjelmaan sekä ohjelmassa olevan ohjelman ohjelmointivirheen korjaaminen.

Usein kysymykseen vastausta haettaessa tulee esiin useita kysymykseen liittyviä ja sitä tarkentavia kysymyksiä, joihin vastauksia hakemalla vastaus alkuperäiseen kysymykseen tarkkenee [9]

Alla tarkastellaan erilaisia tilanteita ohjelmointityön aikana, missä tyypillisesti tulee kysymyksiä esille, sekä millaisilla kysymyksillä näitä on ratkottu. Myöhemmässä kappaleessa on esitelty erilaisten työkalujen soveltuvuutta ohjelmointiprosessin aikana tulleisiin kysymyksiin vastaamisesta.

2.1 Oikean aloituskohdan löytäminen

Ohjelmointitehtävän alussa tehtävään liittyvien kohtien paikantaminen on haastavaa. Sillito, Murphy sekä De Volder havaitsivat artikkelissaan [9] havaitsivat ohjelmoijien käyttävän runsaasti aikaa sopivan aloituskohdan löytämiseen. Osan he havaitsivat aloittavan oleellisten koodin kohtien etsimisen tekstipohjaisella haulla, kun taas toiset käyttivät ohjelmointiympäristön kehittyneempää tyyppi -pohjaista hakua. Tekstipohjasta hakua käyttäessä muodostuu haasteeksi sopivan avainsanan keksiminen, sekä suuri määrä epäoleellisia tuloksia [7]. Myös luokka ja paketti selaimia käytetään sopivan aloituskohdan löytämiseen. Tämä olettaa kuitenkin luokkien ja pakettien nimien olevan kuvaavia. Löydettässä mahdollisesti oleellisia kohtia, tulee ohjelmoijan voida tunnistaa, liittyyko löydetty kohta oleellisesti työnalla olevaan tehtävään. Eräs tapa tunnistaa koodin oleellisuus on käyttää debuggeria, asettaen mahdollisesti oleelliseen kohtaan pysähdyskohta (breakpoint) ja suorittamalla ohjelmaa debuggerin kautta voi huomata pysähtyykö ohjelma odotetusti pysähdyskohtiin. Myös selvittämällä mahdollisesti oleellisen kohdan riippuvuuksia muihin tyyppeihin, saa paremman vaikutelman siitä, onko kyseinen kohta oleellinen työn aloitukseen.

Avainsanahaut tuottavat usein paljon turhia tuloksia avainsanat, rakenne, metodisignaturet ja loopit yms (ast) tosin toi on turhaa infoa, tee testcaseja ja arvo keywordejä

2.2 Riippuvuuksien hahmottaminen

Sopivan aloituskohdan ollessa selvillä, usein [9] seuraava askel riippuvuuksien hahmoittaminen. Tässä ohjelmoija tarkastelee itse oleellisia luokkia, mitä ne perivät, mitä rajapintoja ne toteuttavat sekä mitä metodeja luokka toteuttaa. Tarkastelu etenee hitaasti laajemalle alueelle, mitä tyyppejä oleellisessa kodassa käytetään ja miten ne liittyvät tähän luokkaan. Näin he mallintavat selkeiden riippuvuuksien ja oleellisten kohtien välisiä suhteita hahmoittaakseen paremmin, kuinka ohjelman oleellinen osa toimii. Näihin kysymyksiin ohjelmoijat voivat helposti vastata käyttäen ohjelmointiympäristön perustyökaluja.

Tehtävään liittyvien koodipalojen etsinnässä moni valitsi epäolennaisia paloja, ja he käyttivät huomattavasti aikaa epäolennaisen koodin tarkasteluun. [2]

2.3 Koodin konseptien selvittämnen

Alkupaikan sekä läheisten riippuvuuksien selvittäminen on hyvä alku; kuitenkin usein on tarpeen saada selville tarkempi kokonaiskuva ohjelmiton oleellisesta osasta ja siten parantaa yleiskuvaa ohjelmiston toiminnasta. Tämän selvittämiseksi haetaan usein vastauksia kysymyksiin, missä tämä olio luodaan, mitä parametrejä sille annetaan, miten oikeaoppisesti käytetään tiettyä tietorakennetta sekä miten ohjelma käsittelee tietyt tapaukset. Ohjelmoijat tarkastelevat löydetyn oleellisen kohdan suhdetta ohjelmistoon, mistä ja mitkä luokat käyttävät tätä, ja mitä tehdessä, mitkä luokat kapseloivat tämän ja miksi ne tekevät niin [9, 2]

2.4 Miten tätä käytetään - apit

Niin ohjelman toimintaa tarkastellessa, kuin uutta ominaisuutta luodessa tulee useasti vastaan tuntemattomia ohjelmointirajapintoja (API). Joskus ohjelmointirajapinnat ovat helppokäyttöisiä, mutta useasti, varsinkin isompien kirjastojen (Library) ohjelmistorajapinnat ovat monipuolisia ja siten vaikeammin sisäistettävissä. Aluksi, uuden ohjelmointirajapinnan kohdatessaan ohjelmoija tyypillisesti aloittaa selvittämällä, kuinka eri tyypit liittyvät toisiinsa. Tämä tapahtuu osin samoin, kuten yllä mainitussa riippuvuuksien hahmoittaminen osiossa, mutta painottuu pitkälti dokumentaatioon ja luokka kaavioihin. Dokumentaatiosta sopivien luokkien löytämisen yksi suurimmista haasteista on oikeiden avainsanojen arvaaminen. Myös haasteita ohjelmistorajapintoihin tutustuttaessa tuottaa eri luokista esiintymien luominen. Erityisesti silloin, kun luokka ei tarjoa julkista konstruktoria, vaan käytössä

on rakentaja (builder) tai tehdas (factory) ohjelmointityylit. Myös luokkien dokumentoimattomat oletukset sekä ohjelmoijan mielestä epätyypillinen toteutus tuottaa haasteita [1].

2.5 hakemisen haasteet

He havaitsivat, että ohjelmistokehittäjillä oli toive, miten ohjelmistorajapinta toimisi, ja he ärsyyntyivät, kun ohjelmistorajapinta ei toiminutkaan odotetusti [1] Eri hakutapojen vaikutus, he havaitsivat noin puolen tutkittavista hakeneen esimerkkejä ja dokumentaatiota webistä, kun taas loput käyttivät vain paikallista dokumentaatiota. Kävi ilmi, että tällä hakutapaerolla ei ollut kummepaa vaikutusta tehtävästä selviämisessä. He havaitsivat, että tutkittavat aliarvioivat koodiesimerkkien haun vievän ajan. Tosin useat tutkittavat eivät onnistuneet löytämään sopivia esimerkkejä ja lopulta tyytyivät paikalliseen dokumentaatioon. [1]

Useat ohjelmointiympäristöjen työkalut koodin täydennyksessä yms, eivät tarjoa kunnollista tukea työhön liittyvien tyyppien tunnistamiseen, vaan olettavat, että ohjelmistokehittäjä tuntee jo tarvittavat tyypit. [1]

3 Tyypillisiä ongelmia

Tiivistä ylhäältä APIt on vaikeita Jungloid mining

4 Työkaluja avun hakuun

4.1 Grep

4.2 IDE

Eclipsen tarjoama tuki koodin täydennykseen olettaa, että ohjelmoija tietää tarpeelliset luokat, eikä ohjelmointiympäristö tarjoa mahdollisesti sopivia luokkia ohjelmoijalle [5].

4.3 S^6 projekti

S^6 projekti pyrkii helpottamaan sopivan koodin hakemista ja siten koodin uudelleenkäyttöä. Projekti mahdollistaa sopivien luokkien ja metodien haun siten, että ohjelmoija tarjoaa testitapauksia, joiden rakennetta ja semantiikkaa automatisoidusti tarkastelemalla projekti pystyy tarkentamaan hakutuloksia. Projekti pyrkii myös automatisoimaan tarvittavat muunnokset, jotta valmiit, eri tyyppejä käyttävät metodit, toimisivat oikein ahjelmoijan tarpeisiin. He korostavat, että hausta ei tule tehdä liian tarkaa, sillä harva toteutusratkaisu on lopullinen, vaan toteutus ja sen rakenne voi tarvittaessa mukautua sopimaan tarjolla olevaan koodiin. Siis vain korkean tason tieto siitä, mikä haluttu lopputulos on tiedossa. [7]

4.4 Jungloid louhinta

Suuri määrä mahdollisia ohjelmointirajapintoja tekee kaikkien tarpeellisten ohjelmointirajapintojen ulkoa opettelun mahdottomaksi sekä vaikeuttaa tarpeeseen sopivan ohjelmointirajapinnan löytämistä. Mahdollisesti sopivan ohjelmointirajapinnan löydettyään, haasteeksi saattaa muodostua se, että se tarvitsee eri tyypit, kuin mitä on tarjolla. Lähtötyypin muuntaminen sovivaan tyyppiin ei ole aina helppoa eikä suoraviivaista. Joskus tyyppi tulee kierrättää usean muun tyypin kautta, jotta kohdetyyppiin päästään. Se tuottaa ohjelmoijalle joskus paljon haasteita, eityisesti silloin jos tarjolla olevat tyypit eivät ole tuttuja. Tässä avuksi tulee Jungloid louhinta. Jungloidit määritellään seuraavasti: $\lambda x.\, e \;:\; \tau_{in} \rightarrow \tau_{out}$. Jungloidhaku määritellään parina: (τ_{in}, τ_{out}) missä τ_{in} ja τ_{out} ovat tyyppejä, siis mistä tyypistä, mihin tyyppiin. Jungloid louhinnan taustalla on tietovarasto erilaisista tyypeistä ja niiden suhteista. Kun haussa tiedetään lähtö tyyppi τ_{in} sekä kohde tyyppi τ_{out} voidaan tyyppien suhde verkosta hakea mahdollisia reittejä näiden kahden tyypin väliltä, perus verkko algoritmeillä [5]

4.5 Whyline

Ohjelmoijat kysyvät usein miksi jotain tapahtuu ja miksi jotain ei tapahtunut osana ongeman ratkaisu prosessia. Kuitenkin suuri osa tarjolla olevista ohjelmakoodin debuggaus työkaluista ei tarjoa helppoa mahdollisuutta näihin kysymyksiin vastaamiseen, vaan tarjoavat vain tapoja tarkastella ohjelmaa ja sen tilaa tietyssä kohtaa ohjelman suoritusta, jättäen suuren osan ohjelman suorituksen tarkastelusta ja ongelmien tunnistamisesta ohjelmoijan vastuulle. Whyline luotiin ohjaamaan tarkemmin ongelmanratkaisu prosessia, ohjaamaan oikeisiin kysymyksiin vastaamista ja siten vähentää debuggaamiseen käytettyä aikaa [3].

4.6 Stackoverflow

4.7 Javadoc

4.8 OSS - Github

4.9 Wikit

Wikit ovat yhteisöllinen tapa luoda kattavia tietopankkeja, missä usean ihmisen tietotaito ja osaaminen yhdistyvät luoden asiasta kiinnostuneille laadukasta materiaalia. Wikejä käytetään myös avoimen lähdekoodin ohjelmien dokumentaatioon sekä parantamaan ohjelmistorajapintojen dokumentaatiota. Wikit koetaan myös sähköpostilistoja ja muita perinteisempiä keskustelu sekä dokumentaatiomuotoja helpommiksi ja selkeämmiksi käyttää [4].

4.10 Blogit

Sosiaalisen median kasvun myötä on blogien käyttö kasvanut myös ohjelmistokehittäjien keskuudessa. Isoissa avoimen lähdekoodin projekteissa, kuten Postgresql, Gnome ja Python julkaistaan keskimäärin noin kahdeksan tunnin välein uusi blogi julkaisu. Julkaisut ovat keskimäärin 150-273 sanaa ja ne käsittelevät pääosin joko ohjelmiston vaatimuksia, ohjelmiston ympärille muodostunutta yhteisöä, ohjelmistosta lisätiedon kertomista, ohjelmiston käyttöönottoa ja jakelua, prosessin koordinointia ja hallintaa, lisätietoja siitä, miten asia on toteutettu ohjelmistossa, suunnitteluratkaisuista sekä ylläpidosta. Useimmiten aihe liittyy asiaan, jonka parissa kirjoittaja on äskettäin työskennellyt. [6].

[6]

4.11 specifying what to search for

Avainsanahaut tuottavat usein paljon turhia tuloksia avainsanat, rakenne, metodisignaturet ja loopit yms (ast) tosin toi on turhaa infoa, tee testcaseja ja arvo keywordejä But even if the programmer could be precise here, it would not be enough. As programmers become more precise as to what they want, the odds of identifying code that exactly matches their specifications approaches zero.

Lähteet

[1] 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`.

[2] 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`.

[3] 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`.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.