Millaisia kysymyksiä ohjelmoijat esittävät ja miten he hakevat niihin vastauksia

Jarmo Isotalo

Referaatti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 16. 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 [4], [5], [1]. Kysymykset liittyvät niin vieraiden ohjelmistorajapintojenen (api) käyttöön [3] 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 [5]

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 [5] 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. 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.

2.2 Riippuvuuksien hahmottaminen

Sopivan aloituskohdan ollessa selvillä, usein [5] 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 [5, 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 What and why they ask?

Tutkimuksessaan he selvittivät kysymysten lisäksi, mite erilaiset työkalut auttoivat vastaamaan kysymyksiin. He havainnoivat, että useasti kysymys oli liian avoin, jotta työkalut olisivat osanneet vastata niihin. Sillito, Murphy ja De Volder [5]

4 Eliciting Design Requirements for Maintenance-Oriented IDEs: A Detailed Study of Corrective and Perfective Maintenance Tasks

[2] Vaikka työkalujen määrä ohjelmistokehityksessä on kasvanut, suuri osa työkaluista on tehty kehitys-, ei ylläpitovaiheeseen. Tapaustutkimuksessa he keskittyvät viiden eri ylläpitotehtävän tekemiseen, tekiöinä kokeineita java ohjelmistokehittäjiä. [2]

5 Tyypillisiä ongelmia

Tiivistä ylhäältä APIt on vaikeita Jungloid mining

6 Työkaluja avun hakuun

6.1 Grep

6.2 IDE

6.3 Stackoverflow

6.4 Javadoc

6.5 OSS - Github

6.6 Blogit

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] 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`.

[4] 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.

[5] 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`.