Kuinka ohjelmoijat etsivät koodia

Jarmo Isotalo

Referaatti
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 14. 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 foobar

Mitä on koodin hakeminen miksi koodia haetaan

S llito, Murphy ja De Volder esittelivät Kysymyksiä, mitä kehittäjät kysyvät artikkelissaan [5].

- mitä ne koittaa selvittää - lista syistä

miten koodia haetaan -työkalut, grep, kysyminen

Mitä koodihakuja on - pari työkalua Foorumit -IRC -Stackoverflow - Github -blogs

3 What and why they ask?

Sillito, Murphy ja De Volder tutkivat artikkelissaan [5] millaisia kysymyksiä ohjelmistokehittäjät kysyvät työskennellessään ohjelmistokoodimuutosten parissa. He järjestivät kaksi tapaustutkimusta. Tutkimuksen, jossa ohjelmistokehittät työskentelivät pareina toteuttaenpyydetyn muutoksen heille aiemmin tuntemattomaan ohjelmakoodiin. Tähän heillä oli aikaa 45 minuuttia; tehtävässä tarkoituksena ei ollut saada pyydettyä tehtävää valmiiksi, vaan tehtävän odotettiin jäävän kesken. Pareista aina kokemattomampi oli ohjeistettu olemaan näppäimistöllä, ja kokeneempi tarkkailemaan vierestä. Näin he pyrkivät saamaan luonnollisempaa keskustelua aikaan.

Toisessa tapaustutkimuksessaan ohjelmistokehittäjät työskentelivät itselleen tutun ohjelmistokoodin parissa. Tässä heitä pyydettiin tekemään heille tyypillinen ohjelmointitehtävä.

Näistä tapaustutkimusista kertätystä datasta he tunnistivat neljä yleistä katogoriaa, mihin kaikki tutkimuksen aikana kysytyt kysymykset jakautuivat. Ensimmäisten tehtävänantoon liittyvien kohtien löytäminen ohjelmakoodista, missä tiettyä tyyppiä/metodia käytetään ja missä ne sijaitsevat koodissa, koodiin liittyvien konseptien ymmärtäminen, kokonaiskuvan saavuttaminen ohjelman suorituksesta (omaan tehtävään liittyvästä osasta ainakin). He havaitsivat, että useasti vastatakseen korkeamman tason kysymykseen, kysyjä kysyi useampia tukikysymyksiä saadakseen vastauksen korkean tason kysymykseen. Useasti tukikysymykset johtivat väärään vastaukseen, ja tutkittavien tuli palata kysymyksissään taaksepäin ja tarkentaa kysymyksiään eri suuntaan.

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.

4 asking and answering questions about unfamiliar apis

AB ja C selvittivät artikkelissaan [1] millaisia kysymyksiä ohjelmistokehittäjät kysyvät kohdatessaan uuden ohjelmistorajapinnan (APIn). Tapaustutkimuksessaan he tunnistivat kaikenkaikkiaan kaksikymmentä kysymystä, joista he tunnistivat viisi vaikeinta

Aluksi isoin kysymys uuden ohjelmistorajapinnan käytössä on, miten eri tyypit liittyvät toisiinsa. He havaitsivat, että ohjelmistokehittäjillä oli toive, miten ohjelmistorajapinta toimisi, ja he ärsyyntyivät, kun ohjelmistorajapinta ei toiminutkaan odotetusti. Usein kysymyksiin ei ollut helppoja vastauksia. He havainnoivat, että erityisiä haasteita tutkittaville tuotti kahden tai useamman tyypin suhde. Mitkä avainsanat kuvaavat ohjelmistorajapinnan toimintoja; erityisesti haettaessa lisätietoa ohjelmistorajapinnan käytöstä. Useilla oli vaikeuksia arvata aluksi avainsanoja oikein. Miten luoda esiintymä tyypistä, ilman että sille on tarjolla julkista konstruktoria, miten selvittää metodikutsun tulos; metodin paluuarvon lisäksi heittääkö metodi poikkeuksia, ilmaisten että metodin suoritus päättyi virhetilanteeseen. Tässä he havaitsivat ohjelmistokehittäjien oletuksen metodin toiminnasta versus siitä, miten se oli toteutettu vaikuttavan paljon. Esimerkkinä he nostivat esille xml tiedoston validaatiometodin. Ohjelmistokehittäjät odottivat validaatio metodin palauttavan arvon joka kuvaa validaation onnistumista, kun taas todellisuudessa metodi ei palauta mitään kun xml tiedosto on validi, ja heittää poikkeuksen xml tiedoston ollessa epävalidi. Myös onnistuminen ja virhe oli ohjelmistokehittäjien ja ohjelmistorajapinnan luojien mielestä erilainen. Kävi myös ilmi, että tutkittavat kuvittelivat poikkeusten liittyvän paljon vakavempiin ongelmiin, kuin tiedoston epävalidiuuteen, eivätkä he siten olleet sisäistäneet poikkeuksien käyttötapoja. He korostavat, että poikkeusten käyttäminen kaikkien virhetilanteiden esittämiseen vaikeuttaa ohjelmistorajapinnan sisäistämisen. 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. Useat ohjelmointiympäristöjen työkalut koodin täydennyksessä yms, eivät tarjoa kunnollista tukea työhön liittyvien tyyppien tunnistamiseen, vaan olettavat, että ohjelmistokehittäjä tuntee jo tarvittavat tyypit. Myös tyyppien ja objektien relaatioita

tulisi tarkentaa dokumentaatiossa, erityisesti niissä tapauksissa, missä tietty tyyppinen olio voidaankin luoda vain kokoaan toisessa luokassa, eikä sillä ole julkista konstruktoria. Monet kokivat sen hämääväksi.

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

Artikkelissa [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. He jakavat ylläpito-operaation kolmeen vaiheeseen, relevanttien tiedostojen ja koodipalojen löytäminen, näiden välisten suhteiden mallintaminen ja toteutettavat muutokset. Tutkimuksessa tutkittaville annettiin 7 käyttäjien virheraporttia, joista mahdollisimman moni kuvattu ongelma heidän tuli korjata tutkimuksen aikana. Tutkimus kesti 70 minuuttia. Tutkimusessa käytetty koodivarasto koostui yhdeksästä java-tiedostusta. Muutostyön alussa he havaitsivat tutkittavan usein kysyvän, kuinka X toimii ja miksi X ei tapautunut. Näiden pohjalta tutkittavat alkoivat tyypillisesti rajaamaan relevantteja koodipaloja. Löydettyään tarpeellisen koodin he selvittivät sen suhteita toisiin luokkiin. He kysyivät tyypillisesti, mikä vaikuttaa muuttujan arvoon, mikä tämä muuttuja on. Tehtävään liittyvien koodipalojen etsinnässä moni valitsi epäolennaisia paloja, ja he käyttivät huomattavasti aikaa epäolennaisen koodin tarkasteluun.

Aktiviteettejä tarkkailemalla, heille selvisi, että keskimäärin yhtä suuri osa ajasta menee koodin lukemiseen kuin koodin muokkaamiseen. kun taas koodin navigointiin ja esimerkkien hakuun käytettiin hieman vähemmän aikaa. Vähiten aikaa meni ohjelman testaukseen. He tunnistivat seuraavat tarpeet muutos töihin tarkoitettuihin työkaluigin. Workingset:in ylläpito, tehtävään oleellisten koodipalojen listaus, sekä näiden riippuvuuksien hallinta. Kopiointiin liittyvien virheiden korjaamiseksi he ehdoittivat kopionnin säilyvän koodin kontekstissa ja siten ohjelmointiympäristön korostavan, että kopioidussa koodissa voi olla vielä korjaamattomia virheitä. He ehdottivay myös koodipaloista koostuvaa editoria, jossa tiedostosta on esillä vain tarpeellinen osa. Näin saisi helpommin kaikki oleelliset osat koodista esille. Ympäristön tulisi visualisoida ohjelmakoodin riippuvuudet. Sen sijaan että ohjelmoija navigoi ja katsoo jokaista riippuvuutta erikseen, voisi ohjelmointiympäristö tarjota selkeämmän tavan nopeammin tarkastella riippuvuuksia. Vastatessaan alun kysymyksiinsä, miksi ja miten ohjelmistokehittäjät joutuvat tutkimuksen mukaan usein arvaamaan ja tekevät merkittäviä johtopäätöksiä ohjelmiston toiminnasta. Ohjelmistoympäristön pitäisi osata vastata kysymyksiin miks i ja miten.

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`.