Kako se (ne)izgubiti tokom zadatka za izmenu softvera

(eksperimentalna analiza procesa razumevanja programa)

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 12, DECEMBER 2004 889

How Effective Developers Investigate Source Code: An Exploratory Study

Martin P. Robillard, Wesley Coelho, and Gail C. Murphy, *Member, IEEE Computer Society*

- proučavanje kako programeri sprovode zadatak izmene funkcije autosave programa jEdit
- Cilj je bio razumeti faktore uspešnog i efikasnog razumevanja programa, koji vode uspešno realizovanom zadatku izmene

Postavke eksperimenta

- JEdit editor napisan u Javi
  - 65000 linija koda, 301 klasa u 20 paketa
- Zadatak
  - Omogućiti korisniku da eksplicitno onemogući autosave funkcionalnost
  - Teškoću predstavlja činjenica da nema projektne dokumentacije (osim javadoc) i funkcionalnost se može razumeti samo inspekcijom programskog koda
  - Očekivana izmena: 60tak linija of modifikovanih, izbrisanih, ili dodatih u program, u 5 fajlova
- 5 programera je rešavalo zadatak

Tok eksperimenta

Eksperiment je podeljen u tri faze:

1. Obuka za rad u Eclipse-u (30 minuta)
2. Proučavanje jEdit programa (1 sat)
   - Dato: pisani material sa opisom zadatka, test primeri za osnovne zahteve, početna tačka za pregled koda
   - Dozvoljen pregled i izvršavanje koda jEdit-a
   - Dozvoljeno beleženje pronađenih informacija
   - Restrikcije
     - Bez upotrebe dibagera
     - Bez menjanja koda

Tok eksperimenta

3. Izmena programa (2 sata)

- Napraviti planirane izmene u programu
- Uspešnost obavljenog zadatka ocenjivana je sa 8 test primera (koliko testova će uspešno proći)

(na sajtu se može naći jEdit_izmena_resenje.zip)

Generalna strategija rešavanja zadatka

1. Pregledati izvorni kod sistema
2. Locirati i razumeti kod koji je od značaja za zadatak
   - Programski kod jEdita je obiman
   - Relevantan kod je razbacan u raznim paketima
   - Programski kod od značaja za zadatak je deo nekog koda koji nije od značaja
3. Primeniti izmene
4. Pospremanje
5. Validacija promena, takođe u odnosu na ostale nepromenjene funkcije
   - Možda ne znamo kako su te ostale funkcije radile pre promena

Šta čini zadatak teškim?

- Veliki programski kod
  - Odakle početi?
- Nedeskriptivna imena u programu
- Teškoće u razumevanju zahteva i njegovih podrazumevanih pretpostavki
  - Oskudno znanje o domenu (npr. dizajn tekst editora, koriršćeni java APIji)
- Dokumentacija neažurna, nepostojeća ili nedovoljna
  - Neki detalji nisu dokumentovani, jer je originalni programer pretpostavljao da su svima očigledni

Šta čini zadatak teškim?

- Potreba da se razumeju mnoge druge klase da bi se razumeo mali fragment koda
- Mogućnost dezorijentacije i zaboravljanja glavnog cilja istraživanja
  - Odakle sam došao na ovo?
  - Zašto sam otvorio ovu klasu?
- Statički pogled u odnosu na izvršavanje programa
  - Koji objekti se zaista kreiraju?
  - Koji metodi se zaista izvršavaju?

Robillarov Eksperiment

- Prikupljeni su sledeći podaci:
  - Izmenjeni artefakti
  - Video snimci ekrana svakog programera dok su rešavali zadatak
- Uspeh rešavanja zadatka je ocenjivan
  - Potrošenim vremenom
  - Kvalitetom izmena
    - Rešenje je savršeno ako implementira korektno zahteve i uklapa se u postojeći dizajn jEdita
      - Ne sme se npr. staviti konstanta tamo gde vrednost može da se pročita iz nekog property objekta

Rezultati eksperimenta

Time Taken to Complete the Change Phase of the Study

Subject	1	2	3	4	5
Time (minutes)	125	62	72	114	120

Programming Experience of Subjects

Subject	1	2	3	4	5
Experience (years)	1	3	5	5	1

Solution Quality for Each Subject

Sub-task/Subject	1	2	3	4	5
1–Check box	**Success**	**Success**	**Success**	Inelegant	**Success**
2–State reset	Not attempted	Buggy	**Success**	Buggy	Not attempted
3–Disabling	Unworkable	**Success**	**Success**	**Success**	Unworkable
4–Deletion	Unworkable	**Success**	**Success**	Buggy	Unworkable
5–Recovery	Unworkable	**Success**	**Success**	**Success**	Not attempted

Rezultati eksperimenta

- 2 uspešno rešenje, 1 nekvalitetno, 2 neuspešna
- Uspešni programeri primenili su ***sistematski pristup*** tokom analiziranja programa i pre primene izmena:
  - Razumevanjem opšte strukture programa kroz usmerene pretrage
  - Potpunim planiranjem svih izmena unapred
- Neuspešni programeri sledili su ***nesistematski pristup*** površnim pregledom koda i pogađanjem
  - "sistematiski" ne znači pregled liniju po liniju

Zapažanje 1 (Robillard)

- Neuspešni programeri su sve svoje modifikacije koda izvršili na jednom mestu, čak i kad je trebalorasporediti na više mesta u skladu sa postojećim dizajnom.
  - Jedan programer realizovao sve promene u jednom jedinom metodu
- Zaključak: Nedovoljno proučavanje dizajna pre obavljanja promene dovelo je do toga da se menja jedino ono mesto u kodu koje je bolje proučeno od strane programera.

Zapažanje 2

- **(Nepažnja tokom proučavanja programa)**. Programski segmenti koji su bili jasno relevantni za zadatak promene nisu uočeni kao takvi kada su prikazani slučajno.
  - Uspešni programeri više su koristili unakrsno referenciranje i pretragu po ključnim rečima
  - Neuspešni programeri više su koristili obično pregledanje i skrolovanje teksta
- Zaključak: *Informacija relevantna za promenu se otkriva samo ako se traži eksplicitno.*

Zapažanje 3

- Uspešni programeri napravili su detaljan i kompletan plan pre sprovođenja izmene, a neuspešni to nisu uradili.
- Zaključak: Pravljenje detaljnog plana izmena omogućava programeru
  1. da razmisli o obimu analiziranog koda i proceni da li je ta analiza bila dovoljna
  2. da sprovede fokusiranu programsku pretragu da razradi rešenje

Zapažanje 4

- Uspešni programeri nisu se vraćali na proučavanje istih metoda toliko često koliko neuspešni.
- Zaključci:

  - Programeri koji su u stanju da bolje procene važnost proučavanog metoda su efikasniji.

  - Programeri koji su u stanju da bolje razumeju i zapamte metode koje su istraživali su efikasniji

  - Programeri koji imaju teškoće da otkriju nove relevantne metode su manje efikasni

Zapažanje 5

- Uspešni programeri uglavnom su obavljali strukturno vođene pretrage, a ne pretrage na osnovu intuicije ili usklađene sa dekompozicijom sistema po fajlovima
- Zaključci: Bez detaljnog poznavanja implementacije sistema, pogađanje koje metode treba proučiti (na osnovu imena/lokacije), nije tako efikasno kao obavljanje fokusirane pretrage.
  - Programeri treba da se odupru iskušenju da pokušaju da pogode koji je kod relevantan za promenu na osnovu nestrukturnih ključeva

Koje informacije su programeru potrebne kada vrši izmene koda?

- Rad autora Sillito et al. [2008]
- Sprovedene dve studije da se sazna koja pitanja programeri formulišu tokom zadatka izmena u kodu

Informacije potrebne programeru

- Nalaženje inicijalnih tačaka fokusa u kodu od značaja za zadatak
  - Npr, nalaženje klasa koje reprezentuju domenske koncepte
  - Nalaženje klasa koje odgovaraju UI elementima
  - Nalaženje klasa od značaja za posmatranu funkciju programa
  - Da bi našli informacije, progamer koristi
    - Tekstualne pretrage
    - Debugger
    - Pregled (čitanje) koda

Informacije potrebne programeru

- Dalje sticanje znanja na bazi inicijalnih tačaka
  - Uključuje višestruke informacije o istom entitetu (koje smo ranije pomenuli)
  - Relacije nasleđivanja
  - Upotreba u drugim klasama
  - Pozivaoci/Pozvani određenog metoda (posebno u slučaju polimorfizma; dinamičko ponašanje)
  - Radi dolaska do ovih informacija, programmer koristi:
    - Call stack viewer u debugger-u
    - Pretragu korišćenja imena

Informacije potrebne programeru

- Izgradnja modela povezanih informacije o više entiteta

- Moraju se razumeti višestruke relacije među entitetima

- Moraju se razumeti opštiji koncepti (bit picture)

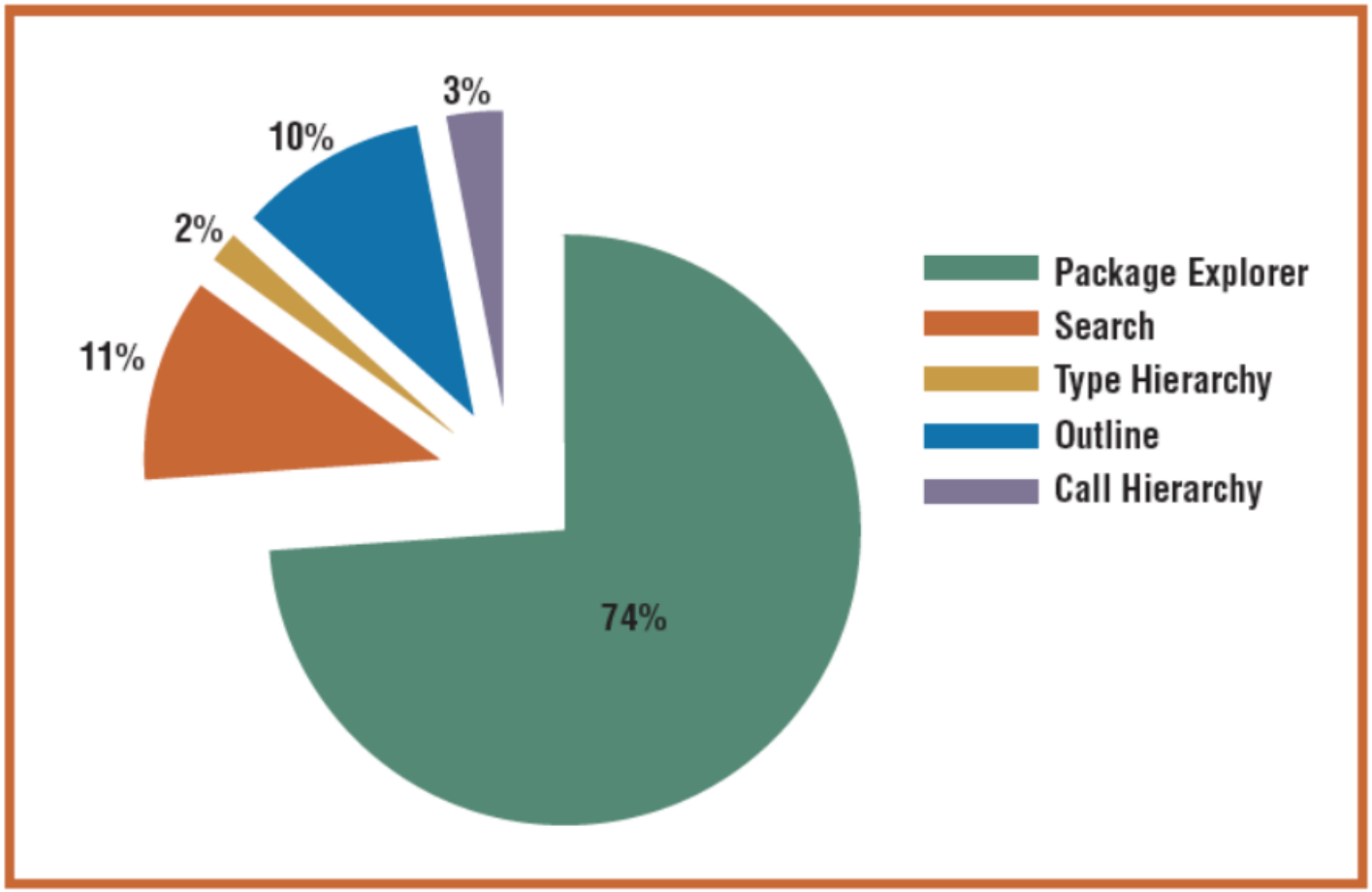
- Tok kontrole I podataka prilikom poziva metoda

- Problem kod pravljenja ovog modela je zaboravljanje ranije ostvarenih saznanja

Zapažanja o upotrebi alata

- Raspoloživi alati uglavnom daju odgovore na konkretna pitanja o jednom entitetu
- Nema puno podrške za pitanja o više entiteta niti za opštu sliku
- Među rezultatima upita koje daju alati ima mnogo irelevantnih

Statistika korišćenja navigacionih pogleda u Eclipse-u



Literatura

- M. Robillard, W. Coelho, and G. Murphy. How effective developers investigate source code: An exploratory study. IEEE Transactions on Software Engineering, 30 (12), 2004.
- J. Sillito, G. Murphy and K. De Volder. Questions programmers ask during software evolution tasks. IEEE Transactions on Software Engineering, 34(4), 2008.
- G. Murphy, M. Kersten, and L. Findlater. How are Java Software Developers using the Eclipse IDE?, IEEE Software, 2006.