Refaktorisanje

Poboljšanje dizajna postojećeg koda

Prevod drugog izdanja

Martin Fowler
Podrška Kent Beck

Addison-Wesley
Računarski fakultet
CET

Refaktorisanje: Poboljšanje dizajna postojećeg koda

ISBN 978-86-7991-431-6

Autorizovan prevod sa engleskog jezika drugog izdanja knjige
Refactoring: Improving the Design of Existing Code

Fotografija na korici: Waldo-Hancock Bridge & Penobscot Narrows Bridge by Martin Fowler Lightbulb graphic: Irina Adamovich/Shutterstock

Prevod	Željko Brković
Recenzija	Milena Vujošević Janičić
Lektura	Nataša Ninković
Gl. i odg. urednik	Jovana Ristić
Tehnički urednik	Vesna Petrinović
Slog i prelom	Predrag Bujić
Izdavači	**CET Computer Equipment and Trade** Beograd, Skadarska 45 tel/fax: 011 3243-043, 3235-139, 3237-246 http://www.cet.rs **Računarski fakultet** Beograd, Knez Mihailova 6/V tel: 011 2627-613, 2633-321 www.raf.edu.rs
Za izdavača	Dragan Stojanović, direktor
Tiraž	1000
Štampa	„Pekograf", Beograd

Nastavno-naučno veće Računarskog fakulteta na 142. elektronskoj sednici održanoj 27. 4. 2020. godine donelo je odluku da prevod knjige „Refaktorisanje" autora Martin Fowler-a bude štampan kao univerzitetski udžbenik.

Za Sidni
– Martin

Sadržaj

Predgovor

Reč „refaktorisanje" je skovana u krugovima Smalltalk-a, ali nije potrajalo njeno prihvatanje i u drugim programerskim krugovima. Pošto je refaktorisanje integralni deo razvoja okruženja, termin se brzo javi kada oni koji rade sa okruženjem govore o svom zanatu. Do njega dolazi kada poboljšavaju svoje hijerarhije klasa i kada zadivljeno pričaju o tome koliko redova koda su uspeli da obrišu. Projektanti okruženja znaju da okruženje neće biti kako valja čim ga naprave – mora se razvijati kako se stiče iskustvo. Znaju i da će kôd biti čitan i modifikovan češće nego što će biti pisan. Ključna stvar u održavanju koda lakim za čitanje i modifikovanje jeste refaktorisanje – posebno u slučaju razvoja okruženja, ali i za softver uopšte.

Dakle, u čemu je problem? Jednostavno u sledećem: refaktorisanje je rizično. Zahteva izmene ispravnog koda koje mogu dovesti do teško primetnih grešaka. Ako se ne izvrši kako treba, refaktorisanje vam može oduzeti dane, čak i nedelje. A refaktorisanje je još opasnije ako ga primenjujete neformalno i sporadično. Na primer, počnete da kopate po kodu. Uskoro uočite nove mogućnosti i kopate još dublje. Što više prekopavate, više stvari izlazi na videlo i pravite sve više izmena. Na kraju iskopate rupu u kojoj se nađete i iz koje ne možete izaći. Da biste izbegli kopanje sopstvenog groba, refaktorisanje se mora obavljati sistematski. Kada sam sa koautorima pisao knjigu *Design Patterns*, pomenuli smo da uzorci za projektovanje nude ciljeve za refaktorisanje. Ipak, utvrđivanje ciljeva je samo deo problema; transformisanje koda da biste ih ostvarili je sasvim drugi izazov.

Martin Fowler i autori koji su doprineli njegovoj knjizi dali su neprocenjiv doprinos razvoju objektno orijentisanog softvera bacivši svetlo na proces refaktorisanja. Ova knjiga objašnjava principe refaktorisanja i najbolje postupke za njegovu primenu, a naglašava i kada i gde treba započeti kopanje po kodu da biste ga poboljšali. Srž knjige čini obiman katalog refaktorisanja. Svako refaktorisanje objašnjava motivaciju i mehanizam dokazane transformacije koda. Neka refaktorisanja, kao što su Izdvajanje metoda i Pomeranje polja, možda deluju očekivano. Ali nemojte da vas to zavara. Razumevanje mehanike takvih refaktorisanja je ključ za disciplinovano razumevanje refaktorisanja. Refaktorisanja u ovoj knjizi će vam pomoći da svoj kôd menjate korak po korak, jer time umanjujete rizike koje nosi razvijanje vaše zamisli. Ubrzo ćete ova refaktorisanja i njihova imena dodati svom razvojnom rečniku.

Moje prvo iskustvo sa disciplinovanim refaktorisanjem, korak po korak, bilo je kada sam programirao u paru sa Kentom (Kent Beck) na visini od deset hiljada metara. On se pobrinuo da što se tiče refaktorisanja iz ove knjige primenimo princip korak po korak. Bio sam zapanjen kako je to davalo dobre rezultate. Ne samo da je poraslo moje poverenje u rezultujući kôd već sam bio i pod manjim stresom. Srdačno vam preporučujem da pokušate ova refaktorisanja: i vi i vaš kôd ćete se osećati znatno bolje.

—Erich Gamma Object Technology International, Inc.
January 1999

Uvodno poglavlje

Pre mnogo godina, jedan konsultant je došao da proveri kako napreduje projekat. Pogledao je ponešto od napisanog koda; u središtu sistema bila je hijerarhija klasa. Dok je lutao po hijerarhiji, konsultant je zapazio da je ona prilično neuredna. Klase viših nivoa su pravile određene pretpostavke o tome kako će klase raditi – to su, zapravo, pretpostavke koje su bile konkretizovane u nasleđenom kodu. Taj kôd, međutim, nije bio prilagođen svim potklasama i pretrpeo je puno izmena. Da je superklasa bila manje modifikovana, onda bi bilo i mnogo manje izmena. Na drugim mestima neke namere superklase nisu bile ispravno shvaćene, pa su funkcije koje postoje u klasi bile udvojene. Na još nekim mestima nekoliko potklasa je obavljalo isti zadatak, pa je kôd jasno mogao biti pomeren naviše u hijerarhiji.

Konsultant je preporučio rukovodiocima projekta da prouče kôd kako bi ga malo doterali, ali rukovodioci nisu bili oduševljeni time. Izgledalo je da kôd funkcioniše bez ikakvih većih poteškoća, a bilo je potrebno i izboriti se sa velikim pritiskom kako bi se ispunili zadati rokovi. Rukovodioci su rekli da će se time kasnije pozabaviti.

Konsultant je i programerima koji su pravili hijerarhiju pokazao šta se dešava. Progamerima je, nasuprot rukovodiocima, stalo do hijerarhije, pa su uočili problem. Znali su da to, zapravo, nije njihova greška; ponekad je potreban neko sa strane da bi se uočio problem. Tako su programeri proveli dan ili dva sređujući hijerarhiju. Kada su završili, uklonili su polovinu koda u hijerarhiji ne gubeći pritom ništa na funkcionalnosti. Bili su zadovoljni rezultatom. Zaključili su da je brže i lakše da dodaju nove klase hijerarhiji i da klase primene u ostatku sistema.

Rukovodioci projekta nisu bili zadovoljni. Rokovi su bili tesni, a čekalo ih je još mnogo posla. Ova dva programera su potrošila čitava dva dana radeći nešto što ne pomaže dodavanju novih karakteristika sistemu koji treba isporučiti za nekoliko meseci. Stari kôd je lepo radio. Međutim, projekat je postao malo čistiji i pregledniji. Cilj projekta jeste da se isporuči kôd koji radi, a ne kôd koji će zadovoljiti nekog akademika. Konsultant je predložio da se ovakvo sređivanje izvrši i na ostalim centralnim delovima sistema. Takva aktivnost je mogla ceo projekat da zaustavi na nedelju ili dve. Cela ta aktivnost je imala za cilj da kôd izgleda lepše, a ne da ga natera da radi nešto što nije već radio.

Kakva osećanja izaziva u vama ova priča? Da li mislite da je konsultant bio u pravu kada je savetovao dalje sređivanje? Ili ste od onih koji poštuju staru inženjersku izreku „ako radi, ne diraj ga"?

Moram priznati da sam u ovom slučaju pristrasan. Ja sam bio taj konsultant. Šest meseci kasnije projekat je doživeo neuspeh, dobrim delom zato što je kôd bio preterano kompleksan za otklanjanje grešaka, odnosno za doterivanje prihvatljivih performansi.

Doveden je konsultant Kent Beck da oživi projekat, a taj zadatak je podrazumevao ponovno pisanje gotovo celog sistema ispočetka. Nekoliko stvari je učinio drugačije, ali jedna od najvažnijih je bila što je insistirao na stalnom sređivanju koda pomoću refaktorisanja. Veća efikasnost tima koji je radio na tom projektu, kao i uloga koju je refaktorisanje igralo u tom uspehu, naveli su me da napišem ovu knjigu kako bih mogao da prenesem znanje koje su Kent i drugi stekli u primeni refaktorisanja za poboljšanje kvaliteta softvera.

Od tog trenutka refaktorisanje postaje sastavni deo programerskog rečnika. Staro izdanje ove knjige se veoma dobro pokazalo. Ipak, period od 18 godina je prava praistorija kada je u pitanju neka programerska knjiga, tako da sam osetio da je krajnje vreme da napišem novo izdanje knjige. Radeći na novom izdanju, izmenio sam gotovo svaku stranicu starog izdanja, ali se u suštini malo toga promenilo. Smisao samog refaktorisanja je isti i većina ključnih elemenata u njemu ostaje nepromenjena. Nadam se da će ovo novo izdanje knjige pomoći većem broju zainteresovanih ljudi da nauče kako da na efikasan način sprovedu u delo refaktorisanje.

Šta je refaktorisanje?

Refaktorisanje je proces menjanja softverskog sistema tako da se ne menja spoljno ponašanje samog koda, već se poboljšava unutrašnja struktura. To je disciplinovani način sređivanja koda, koji svodi na minimum šanse da se pojave greške. U suštini, pri refaktorisanju unapređujete projekat koda koji je već napisan.

„Unapređivanje dizajna koda koji je već napisan." To je čudno formulisana fraza. Po sadašnjem razumevanju razvoja softvera, verujemo da prvo dizajniramo, a zatim pišemo kôd. Prvo dođe dobar dizajn, a kodiranje dolazi posle toga. Vremenom će kôd biti modifikovan, a integritet sistema, njegova struktura kakvu nalaže predviđeni dizajn, postepeno bledi. Polako od inženjerskog pristupa tonemo prema hakerisanju.

Refaktorisanje predstavlja suštu suprotnost ovoj praksi. Kod refaktorisanja možete uzeti loše dizajniran kôd, čak i prilično haotičan, i to preraditi u dobro strukturiran kôd. Svaki korak je jednostavan, čak i krajnje pojednostavljen. Polje premeštate iz jedne klase u drugu, izvučete nešto koda iz metoda da biste načinili zaseban metod i neke segmente kôda premeštate gore ili dole u hijerarhiji. Ipak, kumulativni efekat ovih malih izmena može radikalno promeniti vaš projekat. To je čista suprotnost uobičajenom poimanju opadanja kvaliteta (zastarelosti) softvera.

Kod refaktorisanja se odnos delova posla menja. Utvrdićete da se dizajn ne pojavljuje unapred, već postepeno nastaje tokom razvoja. Iz građenja sistema učite kako da poboljšate dizajn. Rezultujuće međudejstvo vodi programu sa dizajnom koji ostaje dobar u nastavku razvoja.

Šta se nalazi u ovoj knjizi?

Ova knjiga predstavlja jednu vrstu vodiča za refaktorisanje; napisana je za profesionalne programere. Cilj mi je da pokažem kako da refaktorisanje obavljate kontrolisano i efikasno. Naučićete da refaktorišete tako da ne pravite nove greške u kôdu, već da metodično unapređujete strukturu.

Kao što to tradicija nalaže, uobičajeno je da knjige počinju uvodom. Mada se slažem sa tim principom, nije mi lako da napravim uvod o refaktorisanju nekakvom opštom diskusijom ili definicijama. Zato počinjem primerom. U poglavlju 1 se razmatra mali program sa uobičajenim propustima u dizajnu, a zatim se on refaktoriše u prihvatljiviji i razumljiviji program. Usput vidimo i proces refaktorisanja i primenu nekoliko pojedinačnih refaktorisanja. To je ključno poglavlje koje morate da pročitate ako želite da razumete šta je zapravo refaktorisanje.

U poglavlju 2 sam pisao više o opštim principima refaktorisanja, nekim definicijama i razlozima zašto se refaktorisanje vrši. Posebno ću istaći neke probleme koji se javljaju prilikom refaktorisanja. U poglavlju 3, Kent Beck mi pomaže da opišem kako treba namirisati sumnjiva mesta u kodu i kako ih srediti refaktorisanjem. Testiranje igra veoma važnu ulogu u refaktorisanju, zato se u poglavlju 4 opisuje kako da napravite testove u samom kodu.

Srž knjige, katalog refaktorisanja, zauzima preostali deo. To nikako ne znači da je taj katalog sveobuhvatan. Naime, on pokriva ključna refaktorisanja koja će biti potrebna većini developera tokom rada na projektima. Ovaj katalog obuhvata refaktorisanja s kraja devedesetih godina prošlog veka, koja sam do sada napisao baveći se ovom oblašću. Ono što je zanimljivo jeste da i dalje koristim ove beleške pošto ih ne znam sve napamet. Kada želim nešto da učinim, kao što je *Faza deljenja (154)*, katalog me podseća kako da to učinim bezbedno, korak po korak. Nadam se da je to deo knjige kojem ćete se često vraćati.

Prva knjiga koja se bazira na Veb-u

Veb je izvršio ogroman uticaj na naše društvo, a posebno utiče na to kako prikupljamo informacije. Kada sam napisao ovu knjigu, većina saznanja o razvoju softvera je bila prenošena putem štampanja. Danas većinu svojih informacija prikupljam onlajn. Ovo je bio izazov za autore poput mene: postoji li još uvek uloga koja pripada knjigama i kako bi one morale da izgledaju?

Verujem da još uvek postoji neka uloga za knjige poput ove, ali se one moraju promeniti. Vrednost knjige jeste u tome što ona predstavlja veliki izvor znanja, jer je sastavljena na kohezivan način. Pri pisanju ove knjige pokušao sam da pokrijem mnogo različitih refaktorisanja, kao i da ih organizujem na dosledan i integralan način.

Ali takva integralna celina predstavlja apstraktno književno delo koje je tradicionalno zastupljeno u štampanom formatu i za kojim neće biti potrebe u budućnosti.

Većina ljudi u izdavačkoj industriji gleda štampanu verziju knjige kao primaran način predstavljanja nekog sadržaja. Iako smo sa oduševljenjem prihvatili elektronske knjige, one samo predstavljaju elektronsku verziju originalnog rada koji se zasniva na strukturi štampane knjige.

JavaScript primeri

Kao što je to slučaj i u većini tehničkih oblasti softverskog razvoja, primeri sa kodom su veoma bitni radi ilustracije samih koncepata. Ipak, refaktorisanje uglavnom izgleda isto u različitim programskim jezicima. Ponekad se možete naći u situacijama kada morate obratiti pažnju na određene stvari, pošto to od vas zahteva programski jezik, ali u suštini najbitniji elementi refaktorisanja ostaju isti. Što se tiče ovog drugog izdanja knjige, izabrao sam JavaScript kako bih pokazao refaktorisanja pošto sam osetio da će ovaj programski jezik odgovarati većini čitalaca. Imajte na umu da nije toliko teško da prilagodite data refaktorisanja bilo kom drugom jeziku koji trenutno koristite. Trudiću se da ne koristim komplikovane segmente programskog jezika, tako da ćete biti u stanju da pratite refaktorisanja čak i u slučaju da posedujete samo elementarna znanja JavaScript-a. Moje korišćenje JavaScript-a u ovoj knjizi nikako ne znači i favorizovanje ovog programskog jezika.

Iako u svojim primerima koristim JavaScript, to nikako ne znači da su prikazane tehnike refaktorisanja u ovoj knjizi ograničene samo na ovaj programski jezik. U prvom izdanju ove knjige primere sam pisao u programskom jeziku Java i takav pristup je bio veoma koristan velikom broju programera koji nikada pre toga nisu napisali nijednu Java klasu. Dokumentovao sam na brojnim programskim jezicima da ova kompatibilnost u pogledu refaktorisanja jeste moguća, ali sam smatrao da bi to čitaocima ove knjige bilo previše konfuzno. Ipak, ova knjiga je napisana za programere bilo kog programskog jezika. Osim odeljaka u kojima su dati primeri, ne pravim nikakve pretpostavke o programskom jeziku. Očekujem da čitaoci ove knjige shvate moje opšte komentare i da ih primene na onaj programski jezik koji oni koriste u svom radu. Upravo to i očekujem od čitalaca, da koriste moje JavaScript primere kako bi ih prilagodili željenom programskom jeziku.

Ovo znači da osim u slučajevima kada objašnjavam specifične primere, na svakom drugom mestu kada govorim o klasama, modulu, funkcijama i tako dalje mislim na njih u opštem programerskom smislu, a ne kao na specifične termine modela JavaScript programskog jezika.

Činjenica da koristim primere sa JavaScript-om takođe podrazumeva da pokušavam da izbegnem JavaScript stilove koji su nepoznati onima koji nisu JavaScript programeri. Ovo nije knjiga o refaktorisanju u JavaScript-u, već je to knjiga o refaktorisanju uopšte. Postoje brojni interesantni primeri refaktorisanja koji su specifični za JavaScript (kao što su refaktorisanje za callback, promises pa sve do async/wait metoda), ali oni nisu predmet ove knjige.

Ko treba da pročita ovu knjigu?

Ova knjiga je namenjena profesionalnom programeru, nekome ko živi od pisanja softvera. Primeri i rasprave obuhvataju mnogo koda koji treba pročitati i razumeti. Primeri su svi u JavaScript-u, ali se mogu primeniti na većinu programskih jezika. Očekuje se da programer ima određeno iskustvo kako bi pratio ovu knjigu, ali ništa preterano.

Iako su ciljna publika ove knjige programeri koji žele da nauče refaktorisanje, ova knjiga je od neprocenjive važnosti i za one koji već znaju nešto o refaktorisanju – može se koristiti kao pomoćno sredstvo i u nastavi. Uložio sam dosta truda kako bih u ovoj knjizi objasnio različite tehnike refaktorisanja, tako da iskusni programer može da koristi ovaj materijal prilikom mentorskog rada sa svojim kolegama.

Mada je fokusirano na kôd, refaktorisanje ima veliki uticaj na dizajn sistema. Od izuzetne je važnosti za starije projektante i arhitekte da razumeju principe refaktorisanja i primene ih u svojim projektima. Refaktorisanje će najbolje primeniti istaknuti i iskusni programeri. Takav učesnik u razvoju najbolje može da razume principe koji stoje iza refaktorisanja i da ih primeni na specifično polje rada. To je naročito slučaj ako koriste programski jezik koji nije JavaScript, jer treba prilagoditi i primeniti date primere na drugi jezik.

Evo kako da vam ova knjiga što više koristi a da je ne pročitate celu:

- **Ako želite da razumete šta je refaktorisanje**, pročitajte poglavlje 1; trebalo bi da primer učini proces jasnijim.
- **Ako želite da razumete zašto treba da refaktorišete**, pročitajte prva dva poglavlja. Ona će vam reći šta je refaktorisanje i zašto ga treba primeniti.
- **Ako želite da nađete gde da refaktorišete**, pročitajte poglavlje 3. Ono vas upućuje na znake koji sugerišu potrebu za refaktorisanjem.
- **Ako želite da izvršite samo refaktorisanje**, pročitajte prva četiri poglavlja u celosti. Onda samo prođite kroz katalog na brzinu. Od kataloga pročitajte samo onoliko koliko je potrebno da biste stekli predstavu šta je u njemu. Nije nužno da razumete sve detalje. Kada dođete u situaciju da stvarno refaktorišete, detaljno pročitajte o potrebnim refaktorisanjima i služite se njima. Katalog je referentni deo knjige, pa verovatno nema potrebe da ga pročitate odjednom.

Značajan deo pisanja ove knjige posvećen je i dodeljivanju naziva različitim refaktorisanjima. Terminologija nam pomaže da komuniciramo na pravi način. Tako, na primer, kada jedan programer savetuje drugoga da izvuče jedan deo koda u funkciju ili da neka izračunavanja podeli u više zasebnih etapa, obojica znaju na šta se odnose termini *Extract Function* i *Split Phase*. Ovakav vokabular takođe pomaže prilikom odabira automatizovanog refaktorisanja.

Nadograđivanje tuđih osnova

Treba da kažem odmah na početku da za ovu knjigu puno dugujem onima koji su radom u poslednjoj dekadi 20. veka razvili polje refaktorisanja. Učenje iz njihovog iskustva dalo mi je inspiraciju i znanje da napišem prvo izdanje ove knjige. Iako je od ovog izdanja prošlo mnogo godina, ipak su oni postavili temelj. Bilo bi idealno da je neko od njih napisao ovu knjigu, ali je ispalo da sam ja taj koji je našao vremena i energije.

Dva vodeća zagovornika refaktorisanja su **Ward Cunningham** i **Kent Beck**. Oni su ga primenjivali kao centralni deo svog razvojnog procesa u ranim danima i prilagodili su svoje razvojne procese tome da od njega izvuku korist. Saradnja sa Kentom mi je posebno ukazala na značaj refaktorisanja i bila je neposredna inspiracija za pisanje ove knjige.

Ralph Johnson vodi na Univerzitetu Ilinoisa u Urbana-Šampejnu grupu, koja je čuvena po praktičnom doprinosu objektnoj tehnologiji. Ralph je dugo bio prvak u refaktorisanju, a time se bavilo i nekoliko njegovih studenata. **Bill Opdyke** je napisao prvi detaljan rad o refaktorisanju u svojoj doktorskoj tezi. **John Brant** i **Don Roberts** su pisanu reč nadmašili pisanjem prve automatizovane alatke pod nazivom *Refactoring Browser*, koja služi refaktorisanju Smalltalk programa.

Od izlaska prvog izdanja ove knjige, veliki broj ljudi radi na unapređivanju oblasti refaktorisanja. Zapravo, možemo reći da su život programerima u velikoj meri olakšali oni koji su programerskim alatima dodali automatizovano refaktorisanje. Meni je sada jednostavno da uzmem zdravo za gotovo preimenovanje neke funkcije samo u par koraka, a ovo su omogućili napori koje su u svoj rad uložili IDE timovi.

Zahvalnost

Iako sam se oslanjao na razne radove, i dalje mi je trebalo puno pomoći u pisanju ove knjige. Prvo izdanje se u velikoj meri oslanja na iskustvo i podršku koje sam dobio od **Kent Beck-a**. On me je prvi uveo u oblast refaktorisanja i ohrabrio da pravim beleške u vezi sa refaktorisanjem. Kent je prvi došao na ideju da „nešto u samom kodu zaudara". Ne mogu da se otmem utisku da bi on sam bolje napisao prvo izdanje ove knjige da nismo zajedno radili na pisanju knjige o ekstremnom programiranju.

Svi autori tehničkih knjiga koje poznajem su izjavili da veliku zahvalnost duguju zapravo stručnim recenzentima ovih knjiga. Svi mi možemo da napišemo knjige, ali neke od njih mogu imati i veće nedostatke koje jedino mogu uočiti naše kolege koje se bave stručnom recenzijom. Što se mene lično tiče, smatram da nisam dovoljno dobar da se bavim stručnom recenzijom, tako da se divim onima koji uživaju u tome. Stručna recenzija neke knjige jeste čin darežljivosti.

Kada sam krenuo ozbiljno da radim na ovoj knjizi, sastavio sam čitavu mail listu savetnika koji će mi davati povratne informacije u toku pisanja same knjige. Kako sam odmicao sa pisanjem, slao sam ovoj grupi ljudi napisani materijal i tražio

njihovo mišljenje. Stoga bih hteo ovom prilikom da se zahvalim sledećim ljudima: **Arlo Belshee**, **Avdi Grimm**, **Beth Anders-Beck**, **Bill Wake**, **Brian Guthrie**, **Brian Marick**, **Chad Wathington**, **Dave Farley**, **David Rice**, **Don Roberts**, **Fred George**, **Giles Alexander**, **Greg Doench**, **Hugo Corbucci**, **Ivan Moore**, **James Shore**, **Jay Fields**, **Jessica Kerr**, **Joshua Kerievsky**, **Kevlin Henney**, **Luciano Ramalho**, **Marcos Brizeno**, **Michael Feathers**, **Patrick Kua**, **Pete Hodgson**, **Rebecca Parsons**, i **Trisha Gee.**

Iz ove grupe ljudi posebno bih želeo da izdvojim **Beth Anders-Beck**, **James-a Shore-a** i **Pete-a Hodgson-a,** koji su mi posebno pomogli sa JavaScript-om.

Kada sam završio sa prvim nacrtom knjige, poslao sam ga na stručnu recenziju jer sam želeo da stručnjaci još jednom pogledaju to što je napisano. Posebno su iscrpne komentare i sugestije dali **William Chargin** i **Michael Hunger**. Takođe, dobio sam veliki broj korisnih komentara i od **Bob-a Martin-a** i **Scott-a Davis-a**. Opsežnu recenziju prvog nacrta uradio je i **Bill Wake**. Moje kolege iz ThoughtWorks-a predstavljaju stalan izvor ideja i povratnih informacija kada je u pitanju moje pisanje. Brojna pitanja, komentari i razmišljanja su bili okidač za pisanje ove knjige. Jedna od prednosti rada u kompaniji ThoughtWorks jeste i u tome što mi je dopušteno da provedem veliki deo svog radnog vremena pišući ovu knjigu. Posebno želim da izrazim svoju zahvalnost našem tehnološkom direktoru **Rebeki Parsons,** jer sam u stalnim razgovorima sa njom dobio pregršt ideja.

Moj glavni urednik **Greg Doench** iz izdavačke kuće Pearson je zaslužan za prevazilaženje brojnih poteškoća pre nego što je ova knjiga ugledala svetlost dana. Želim da izrazim zahvalnost i svojoj tehničkoj urednici **Julie Nahil**. Izuzetno mi je drago što sam ponovo imao priliku da radim sa **Dimitrijem Kirsanovim,** koji je radio na pregledu čitave knjige, i **Alinom Kirsanovom,** koja je bila zadužena za sastav knjige i indeksiranje.

Poglavlje 1

Refaktorisanje – prvi primer

Kako da započnem pisanje o refaktorisanju? Tradicionalni način da počnete priču o nečemu jeste da opišete istoriju, šire principe i tome slično. Kada na nekoj konferenciji neko to uradi, meni se pridrema. Moj mozak započinje pozadinski proces u kome proverava kada će govornik preći na konkretan primer. Primeri me razbude jer kod njih mogu da vidim šta se zbiva. Kod principa je preterano lako napraviti generalizaciju, a suviše teško razaznati kako da se nešto primeni. Primer razjašnjava stvari.

Zato ću knjigu započeti primerom refaktorisanja. Tokom procesa ću vam reći dosta o tome kako refaktorisanje funkcioniše i stvoriti vam osećaj za proces refaktorisanja. Tada mogu da napravim uobičajeni uvod o principima u narednom poglavlju.

Međutim, kada počinjem primerom, nailazim na veliki problem. Ako odaberem veliki program, opišem ga i onda objasnim kako je refaktorisan – to je preterano komplikovano za čitaoca. (Pokušao sam sa dva primera, ali se njihovo objašnjavanje proteglo na stotinak strana.) S druge strane, ako odaberem program dovoljno mali da bi bio razumljiv, stiče se utisak da refaktorisanje nije vredno.

Tako sam u klasičnoj zamci u kojoj se nađe svako ko želi da opiše tehnike koje su korisne programerima. Iskreno, nema efekta raditi refaktorisanje na malom programu kao što je ovaj koji ću koristiti. Ali ako je kôd koji prikazujem deo većeg sistema, onda refaktorisanje ubrzo postaje značajno. Zato vas molim da pogledate ovaj primer i zamislite ga u kontekstu mnogo većeg sistema.

Polazna tačka

U prvom izdanju ove knjige prvi program koji sam koristio bio je program za izračunavanje i štampanje izveštaja o zaduženjima članova video kluba. Sada će se dosta vas pitati: „Zašto baš video klub?" Umesto davanja odgovora na ovo pitanje, zamenio sam primer iz prvog izdanja primerom koji je star, ali i dalje aktuelan.

Zamislite jednu grupu pozorišnih glumaca koji idu na razna mesta gde će odigrati predstave. Obično mušterija zahteva nekoliko predstava i pozorište će za njih odrediti cenu u zavisnosti od veličine publike, kao i vrste predstave koja se izvodi. Trenutno se na repertoaru nalaze dve vrste predstava koje se izvode: tragedije i komedije.

Pored toga što pozorište izdaje karte za predstavu, njihovi kupci dobijaju i kartice lojalnosti koje mogu da iskoriste za ostvarivanje popusta za buduće predstave – to je neka vrsta mehanizma za nagrađivanje lojalnosti gledalaca.

Podaci o predstavama se čuvaju u jednostavnom JSON fajlu, koji izgleda ovako:

plays.json…

```
{
  "hamlet": {"name": "Hamlet", "type": "tragedy"},
  "as-like": {"name": "As You Like It", "type": "comedy"},
  "othello": {"name": "Othello", "type": "tragedy"}
}
```

Podaci za njihove račune dobijaju se iz JSON fajla:

invoices.json…

```
[
  {
    "customer": "BigCo",
    "performances": [
      {
        "playID": "hamlet",
        "audience": 55
      },
      {
        "playID": "as-like",
        "audience": 35
      },
      {
        "playID": "othello",
        "audience": 40
      }
    ]
  }
]
```

Zatim kôd štampa račun koristeći sledeću jednostavnu funkciju:

```
function statement (invoice, plays) {
  let totalAmount = 0;
  let volumeCredits = 0;
  let result = `Statement for ${invoice.customer}\n`;
  const format = new Intl.NumberFormat("en-US",
                       { style: "currency", currency: "USD",
                       minimumFractionDigits: 2 }).format;
  for (let perf of invoice.performances) {
    const play = plays[perf.playID];
    let thisAmount = 0;

    switch (play.type) {
    case "tragedy":
      thisAmount = 40000;
```

```
      if (perf.audience > 30) {
        thisAmount += 1000 * (perf.audience - 30);
      }
      break;
    case "comedy":
      thisAmount = 30000;
      if (perf.audience > 20) {
        thisAmount += 10000 + 500 * (perf.audience - 20);
      }
      thisAmount += 300 * perf.audience;
      break;
    default:
        throw new Error(`unknown type: ${play.type}`);
    }

    // dodati veličinu bonusa
    volumeCredits += Math.max(perf.audience - 30, 0);
    // dodati dodatni bonus za sve koji su odgledali 10 komičnih predstava
    if ("comedy" === play.type) volumeCredits += Math.floor(perf.audience / 5);

    // štampanje reda za ovu porudžbinu
    result += ` ${play.name}: ${format(thisAmount/100)} (${perf.audience} seats)\n`;
    totalAmount += thisAmount;
  }
  result += `Amount owed is ${format(totalAmount/100)}\n`;
  result += `You earned ${volumeCredits} credits\n`;
  return result;
}
```

Izvršavanje tog koda na fajlovima za testiranje podataka koji su gore prikazani dovodi do sledećeg rezultata:

```
Statement for BigCo
  Hamlet: $650.00 (55 seats)
  As You Like It: $580.00 (35 seats)
  Othello: $500.00 (40 seats)
Amount owed is $1,730.00
You earned 47 credits
```

Komentari polaznog programa

Kakvi su vaši utisci u pogledu konstruisanja ovog programa? Na prvom mestu, rekao bih da se ovaj program može tolerisati, jer tako kratak program ne zahteva neku dublju strukturu da bi bio razumljiv. Ali setite se šta sam ranije rekao – moram da imam primere koji su mali. Zamislite da ovaj program ima na stotine linija koda. Na takvom nivou bilo bi teško razumeti i običnu umetnutu funkciju.

I pored toga, program radi. Da nije to samo estetski sud, averzija prema ružnom kodu? Na kraju krajeva, kompajler nije zainteresovan za to da li je kôd ružan ili uredan. Ali pri promeni sistema uključeni su ljudi, a oni jesu zainteresovani. Loše projektovan sistem je težak za menjanje. Težak je zato što je teško uočiti gde treba načiniti izmene i kako će te nove izmene funkcionisati u postojećem kodu da bismo ostvarili željeno ponašanje. Ako je teško uočiti šta treba da se menja, velike su šanse da će programer pogrešiti.

Stoga ako bih bio suočen sa situacijom da modifikujem program koji ima na stotine linija koda, radije bih ga strukturirao pomoću grupe funkcija i ostalih elemenata programa koji bi mi dopustili da na jednostavniji način shvatim šta zapravo radi taj program. Ako programu nedostaje struktura, obično mi je lakše da prvo dodam programu strukturu, pa da potom izvršim neophodne izmene.

Kada treba dodati neku odliku programu, a kôd nije strukturiran na način pogodan za to, prvo refaktorišite program kako biste se uverili da se odlika može lako dodati, a zatim to i učinite.

U ovom slučaju postoji par izmena koje korisnici žele da naprave. Prvo, žele da saopštenje bude ispisano u HTML-u. Samo razmislite koliki uticaj će imati ta izmena. Primoran sam da dodam uslove za svaku metodu koja dodaje string rezultatu. Na ovaj način će funkcija biti prilično kompleksna. U takvoj situaciji, većina programera pribegava kopiranju metode i menja je kako bi je prikazala u HTML-u. Pravljenje kopije se ne čini tako teškim zadatkom, ali to donosi brojne probleme u budućnosti. Ako dođe do bilo koje izmene u logici koda, onda sam primoran da ažuriram obe metode, ali i da se osiguram da će se one uvek ažurirati. Ako bih pisao program koji se više nikada neće menjati, onda bi bio opravdan pristup copy-paste. Međutim, ako je to program koji treba da bude dugo vremena aktivan, onda dupliranje predstavlja ozbiljnu pretnju.

To me dovodi do druge izmene. Glumci bi želeli da igraju u predstavama koje su različitog žanra; oni bi hteli da se repertoaru dodaju sledeće predstave: istorijske, pastoralne, pastoralno-komične, istorijsko-pastoralne, tragično-istorijske, tragikomične-istorijsko-pastoralne. Nije odlučeno šta bi želeli da igraju i kada. Ova izmena će uticati i na način naplaćivanja samih predstava, kao i na način kako se izračunava dodatni popust za kartice lojalnosti. Kao iskusan developer, jedino u šta mogu biti siguran u vezi sa njihovim izmenama jeste da će se za šest meseci ponovo nešto menjati. Na kraju krajeva, kada dobijemo zahteve za neke nove odlike (karakteristike) programa, oni ne dolaze pojedinačno, već uvek u velikom broju.

Metod `statement` je mesto gde treba načiniti izmene da biste se izborili sa promenom klasifikacije i načina obračuna. Međutim, ako metod za račun iskopiramo u metod za HTML račun, moramo da se uverimo da su izmene potpuno konzistentne. Osim toga, kako pravila postaju složenija, biće sve teže utvrditi gde treba načiniti izmene i načiniti ih bez pravljenja grešaka.

Želim da naglasim da su zapravo ove izmene glavni razlog zašto je potrebno izvršiti refaktorisanje. Ako kôd radi i nema potrebe da se ikada menja, onda je u