Pet linija koda

CHRISTIAN CLAUSEN

MANNING

kompjuter
biblioteka

Izdavač:
kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: CHRISTIAN CLAUSEN

Prevod: Slavica Prudkov

Lektura: Nemanja Lukić

Slog : Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2022.

Broj knjige: 553

Izdanje: Prvo

ISBN: 978-86-7310-576-5

Five Lines of Code
CHRISTIAN CLAUSEN
2021
9781617298318

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.4’416
004.42.045

КЛАУСЕН, Кристијан, 1988-
Pet linija koda / Christian Clausen ; [prevod Slavica Prudkov]. - 1. izd. - Beograd : Kompjuter Biblioteka, 2022 (Zemun : Pekograf). - XXIV, 307 str. : ilustr. ; 24 cm. - (Kompjuter biblioteka ; br. knj. 553)

Autorova slika. - Prevod dela: Five Lines of Code. - Tiraž 500. - O autoru: str. XXIII. - Registar.

ISBN 978-86-7310-576-5

a) Софтвер - Рефакторисање
b) Објектно оријентисано програмирање

COBISS.SR-ID 66731017

KRATAK SADRŽAJ

SADRŽAJ

DEO 2

POGLAVLJE 7

POGLAVLJE 8

POGLAVLJE 9

POGLAVLJE 14

DODATAK A

KRATAK PREGLED OBRAZACA REFAKTORISANJA

- IZDVAJANJE METODA (P3.2.1) - Uzima deo jednog metoda i izdvaja ga u sopstveni metod.
- ZAMENA KODA TIPA KLASAMA (P4.1.3) - Transformiše enum u interfejs, a vrednosti nabrajanja postaju klase.
- UMETANJE KODA U KLASE (P4.1.5) - Prirodni nastavak ZAMENE KODA TIPA KLASAMA (P4.1.3), jer premešta funkcionalnost u klase.
- UMETNUT METOD (P4.1.7) - Uklanja metode koji više ne dodaju čitljivost programu.
- SPECIAJALIZOVAN METOD (P4.2.2) - Uklanja nepotrebnu i problematičnu uopštenost iz metoda.
- POKUŠAJ BRISANJA, A ZATIM KOMPILIRANJA (P4.5.1) - Uklanja nekorišćene metode iz interfejsa i klasa kada znamo njihov ceo opseg.
- OBJEDINJAVANJE SLIČNIH KLASA (P5.1.1) - Objedinjuje dve ili više klasa, koje se razlikuju jedna od druge, u skup konstantnih metoda.
- KOMBINOVANJE `if` ISKAZA (P5.2.1) - Smanjuje dupliranje spajanjem uzastopnih iskaza `if` koji imaju identična tela.
- UVOĐENJE STRATEŠKOG OBRASCA (P5.4.2) - Zamenjuje varijansu u iskazima `if` instanciranjem klasa.
- IZDVAJANJE INTERFEJSA IZ IMPLEMENTACIJE (P5.4.4) - Zamenjuje zavisnosti od klase interfejsom.
- ELIMINISANJE GET ILI SET METODA (P6.1.3) - Eliminiše get i set metode pomeranjem funkcionalnosti bliže podacima.
- ENKAPSULACIJA PODATAKA (P6.2.3) - Lokalizuje invarijante povezane sa promenljivama i koheziju čini jasnijom.
- NAMETANJE SEKVENCE (P6.4.1) - Omogućava kompajleru da garantuje da se sve dešava određenim redosledom.

Mojim univerzitetskim mentorima, koji su mi rekli

Svakodnevni naporan rad je ključ da budete konstantno briljantni.

— Olivije Danvi

i

Nedostaje ti poenta.

-Majer Goldberg

Hvala vam što ste me naučili da prestanem da pokušavam da uradim pravu stvar i da uradim pravu stvar.

PREDGOVOR

Da li ste ikada pročitali knjigu o softveru i pomislili da vam ništa nije jasno? Da li su u knjizi korišćeni nepoznat rečnik i preterano složeni koncepti za objašnjenja? Da li ste se zbog toga osećali kao da je napisana za neki elitni interni krug sveznalica u koji vi niste uključeni?

Ovo nije takva knjiga. Ova knjiga je prizemna, fokusirana i jasna.

Ova knjiga nije bukvar. Ne počinje od atoma i ne dosađuje vam osnovama programiranja i jezika. Ne pokušava da vas mazi i čuva. Garantujem da će ova knjiga biti izazov. Ali će vas izazvati bez zastrašivanja i bez vređanja vaše inteligencije.

Refaktorisanje je disciplina transformacije lošeg koda u dobar kod bez njegovog kršenja. Kada uzmemo u obzir da za svoje dalje postojanje čitava naša civilizacija sada zavisi od softvera, malo je verovatno da postoji tema vrednija proučavanja.

Možda mislite da je to hiperbolično. Nije. Pogledajte oko sebe. Koliko procesora sa pokrenutim softverom trenutno ima na vašem telu? Vaš sat, telefon, ključevi od automobila, slušalice... koliko ih ima na 30 metara od vas? Vaša mikrotalasna pećnica, šporet, mašina za pranje sudova, termostat, mašina za veš... a vaš automobil?

Ovih dana se u našem društvu ništa ne dešava bez softvera. Bez softvera ne možete ništa da kupujete ili prodajete, vozite ili letite bilo gde, kuvate hot-dog, gledate TV, ili pozivate nekoga telefonom.

I koliko je tog softvera zapravo dobar kod? Razmislite o sistemima na kojima trenutno radite. Da li su čisti? Ili su oni, kao i većina, haos kome je očajnički potrebno refaktorisanje?

Ova knjiga ne predstavlja vrstu sterilnog i pojednostavljenog refaktorisanja o kakvom ste možda ranije čuli ili čitali. Ova knjiga govori o *pravom* refaktorisanju. O refaktorisanju *pravih* projekata. O refaktorisanju zastarelih sistema. O refaktorisanju u tipovima okruženja sa kojima se svi suočavamo praktično svaki dan.

Štaviše, ova knjiga vas neće naterati da se osećate krivim jer nemate automatizovane testove. Autor shvata da je većina nasleđenih sistema rasla i evoluirala tokom vremena, a mi nismo te sreće da imamo takve testne pakete.

Ova knjiga postavlja skup jednostavnih pravila koja možete da pratite da biste pouzdano refaktorisali složene, neuredne, zapletene, netestirane sisteme. Učenjem i poštovanjem ovih pravila zaista možete da poboljšate kvalitet sistema koje održavate.

Nemojte me pogrešno shvatiti - to nije srebrni metak. Refaktorisanje starog, grubog, netestiranog koda nikada nije lako. Ali, naoružani pravilima i primerima iz ove knjige, moći ćete da zakoračite u gužvu i zbrku sistema koji su vas predugo opsedali.

Zato vam savetujem da pažljivo pročitate ovu knjigu. Proučite primere. Dobro razmislite o apstrakcijama i namerama koje autor predstavlja. Nabavite bazu koda koju nudi i refaktorišite je zajedno sa njim. Pratite njegovo putovanje kroz refaktorisanje od početka do kraja.

Potrebno je vreme. Biće frustrirajuće. To će biti izazov. Ali izaći ćete sa druge strane sa skupom veština koje će vam dobro služiti do kraja karijere. Takođe ćete izaći sa novom intuicijom i razumevanjem šta razlikuje dobar kod od lošeg i šta je to što kod čini čistim.

— Robert C. Martin (Uncle Bob)

O OVOJ KNJIZI

Otac me je naučio da kodiram kad sam bio veoma mlad, tako da sam o strukturama razmišljao otkako znam za sebe. Uvek sam bio motivisan da pomažem ljudima; zato sam ustajao ujutru. Stoga mi je nastava bila zanimljiva. Dakle, kada mi je ponuđeno mesto asistenta na univerzitetu, odmah sam prihvatio. Imao sam pregršt takvih poslova, ali nažalost, ponestalo mi je sreće i jednog semestra nisam mogao ništa da predajem.

Pošto sam preduzimljiv, odlučio sam da osnujem studentsku organizaciju u kojoj bi studenti podučavali jedni druge. Svako je bio dobrodošao da prisustvuje ili govori, a teme su varirale, od lekcija naučenih iz pratećih projekata do naprednih tema koje nisu obuhvaćene nastavnim planom i programom. Verovao sam da će mi to omogućiti da držim predavanja, i nisam pogrešio. Kako se ispostavilo, računarski naučnici su plašljivi, tako da sam morao da budem domaćin skoro 60 nedelja zaredom, da bih pokrenuo i ostale. Tokom tog perioda naučio sam mnogo, kako o temama koje sam predavao, tako i o podučavanju. Ti razgovori su takođe pokrenuli zajednicu radoznalih ljudi gde sam upoznao svoje najbolje prijatelje.

Neko vreme nakon što sam napustio fakultet družio sam se sa jednim od tih prijatelja. Bilo nam je dosadno, pa me je pitao da li mogu da improvizujem razgovor, jer sam ih odradio toliko. Odgovorio sam: “Hajde da saznamo.” Otvorili smo laptop i bez zaustavljanja otkucao sam ono što je u suštini sveobuhvatan primer prvog dela ove knjige.

Kada sam sklonio prste sa tastature, bio je zapanjen. Mislio je da je to demonstracija, ali ja sam imao drugačiju ideju. Hteo sam da ga naučim da refaktoriše kod.

Moj cilj je bio da posle jednog sata može da kodira kao da je glavni za refaktorisanje koda. Pošto su refaktorisanje i kvalitet koda prilično zamršene teme, bilo je očigledno da to moramo da simuliramo. Dakle, pogledao sam kod i pokušao da smislim pravila koja bi ga naterala da uradi pravu stvar, a da je lako zapamti. Tokom vežbe, iako smo samo simulirali, izvršio je stvarna poboljšanja koda. Rezultati su bili toliko obećavajući, a njegovo poboljšanje je bilo tako brzo, da sam, kada sam te večeri stigao kući, zapisao sve što smo obuhvatili. Ponovio sam vežbu kada smo zapošljavali mlade programere na poslu i polako sam sakupljao, gradio i usavršavao pravila i obrasce refaktorisanja za ovu knjigu.

Cilj: Izabrana pravila i obrasci refaktorisanja

Savršenstvo se dostiže, ne kada nema više šta da se doda, već kada više nema šta da se oduzme.

— Antoan de Sent Egziperi

U svetu postoje stotine obrazaca refaktorisanja; odlučio sam da uključim samo 13. Učinio sam to jer verujem da je duboko razumevanje vrednije od širokog poznavanja. Takođe sam želeo da napravim kompletnu, kohezivnu priču, jer pomaže u dodavanju perspektive i olakšava mentalno organizovanje predmeta. Isti argumenti važe za pravila.

Nema ničeg novog pod suncem.

— Knjiga Propovednika

Ne tvrdim da sam smislio mnogo novih stvari u ovoj knjizi, ali mislim da sam kombinovao stvari na način koji je i zanimljiv i koristan. Mnoga pravila su izvedena iz knjige Roberta C. Martina Jasan kod *(Clean Code,* Pearson, 2008), ali su modifikovana da bi bila jednostavnija za razumevanje i primenu. Mnogi obrasci refaktorisanja potiču iz knjige Martina Faulera Refaktorisanje *(Refactoring,* Addison-Wesley Professional, 1999), ali su prilagođeni tako da iskoriste prednosti kompajlera, umesto da se oslanjaju na stroge testne pakete.

Publika i putokaz

Ova knjiga se sastoji iz dva dela sa veoma različitim stilovima. U prvom delu gradimo čvrstu osnovu refaktorisanja i usmeren je na pojedince. Umesto na sveobuhvatnost, fokusiram se na lakoću učenja. Ovaj deo je za ljude koji tek treba da razviju čvrstu osnovu za refaktorisanje, kao što su studenti i mlađi ili samouki programeri. Ako pogledate izvorni kod knjige i pomislite: „Izgleda da je ovo lako poboljšati“, onda prvi deo knjige nije za vas.

U drugom delu se više fokusiram na kontekst i tim. Odabrao sam ono za šta verujem da su najvrednije lekcije o razvoju softvera u stvarnom svetu. Neke teme su uglavnom teorijske, kao što su „Saradnja sa kompajlerom“ i „Pratite strukturu u kodu“; a neke su prvenstveno praktične, kao što su „Volim brisanje koda“ i „Neka loš kod izgleda loše“. Stoga se ovaj deo primenjuje šire, pa čak i iskusni programeri bi trebalo da nauče nešto iz ovih poglavlja.

Pošto za sva poglavlja prvog dela koristimo jedan sveobuhvatan primer, ona su čvrsto povezana i trebalo bi da ih čitate jedno za drugim. Ali u drugom delu poglavlja su uglavnom samostalna, osim nekoliko referenci. Ako nemate vremena da pročitate celu knjigu lako možete da odaberete najuzbudljivije teme u drugom delu i čitate ih posebno.

O nastavi

Proveo sam mnogo vremena razmišljajući o podučavanju. Prenošenje znanja i veština predstavlja mnoge izazove. Nastavnik mora da stimuliše motivaciju, samopouzdanje i razmišljanje. Ali mozak učenika radije čuva energiju, pa stalno pokušava da odvrati pažnju od učenja.

Da bismo prevazišli ovaj problem mozga, prvo je potrebno da stimulišemo motivaciju. Obično to radim tako što postavljam vežbu jednostavnog izgleda; kada učenici shvate da ne mogu da je reše, javlja se prirodna radoznalost. To je svrha koda u prvom delu knjige. „Poboljšajte ovu bazu koda“ izgleda kao jednostavna instrukcija; međutim, kod je već kvalitetan i mnogi ljudi ne znaju kako da nastave izradu vežbe.

Druga faza je da učenicima date samopouzdanje da eksperimentišu i primenjuju nova znanja ili veštine. Koliko je to važno shvatio sam tokom vannastavnih časova francuskog jezika. Kada je naša učiteljica htela da nas nauči novoj frazi, prolazila je kroz iste korake:

1. Zamolila je svakog od nas da doslovno ponovimo frazu. Ovaj čisti korak imitacije bi nas naterao da jednom izgovorimo frazu.
2. Svakom od nas je postavila pitanje. Nismo uvek razumeli pitanje, ali je intonacija jasno govorila da je to pitanje. Pošto nismo imali druge alate na raspolaganju, ponovo smo ponovili frazu. To ponavljanje je izgradilo samopouzdanje i dalo nam prvi deo konteksta za frazu. Tad je počelo razumevanje.
3. Zamolila nas je da koristimo frazu u razgovoru. Biti u stanju sintetizovati nešto novo je cilj nastave i zahteva razumevanje i samopouzdanje.

Naučio sam da ovaj pristup prati japanski *Shuhari* koncept iz borilačkih veština, koji je sve popularniji. Sastoji se iz tri dela: „Šu” je imitacija, bez pitanja i razumevanja; „ha“ je varijacija, raditi nešto novo; a „ri“ je originalnost, koja u potpunosti odstupa od poznatog.

Šuhari obuhvata sve u prvom delu. Preporučujem da prvo sledite pravila bez razumevanja; onda, kada shvatite njihovu vrednost, možete da smišljate varijacije. Konačno, kada ih savladate, možete da pređete na mirise koda. Za obrasce refaktorisanja ću pokazati kako nešto da uradite u stvarnom kodu, a čitalac bi trebalo da prati (imitacija). Zatim ću pokazati isti obrazac refaktorisanja u drugom kontekstu (varijacija). Na kraju ću predstaviti još jedno mesto za primenu obrasca; ovde, podstičem čitaoca da to pokuša sam da uradi (sinteza).

Možete da koristite knjigu da verifikujete proces, a Git oznake za verifikaciju koda. Ako ne pratite kod, činiće vam se da je previše ponavljanja, pa zato preporučujem da pročitate prvi deo sa rukama na tastaturi.

O kodu

Ova knjiga sadrži mnogo primera izvornog koda, kako u numerisanim listama, tako i umetnuto u normalan tekst. U oba slučaja izvorni kod je formatiran `Ovakvim fontom fiksne širine` da bi bio odvojen od običnog teksta. Sintaksa koda je istaknuta postavljenim ključnim rečima ispisanim **`podebljano`**, što strukturu koda čini razumljivijom.

U mnogim slučajevima, originalan izvorni kod je ponovo formatiran; dodali smo prelome linija i preradili uvlačenje da bismo prilagodili kod raspoloživom prostoru stranica u knjizi. Pored toga, komentari u izvornom kodu su često uklonjeni iz programskog koda kada je kod opisan u tekstu. Napomene koda prate mnoge programske kodove naglašavajući važne koncepte.

Kod za primere u ovoj knjizi dostupan je za preuzimanje na Manning veb sajtu (https://www.manning.com/books/five-lines-of-code) ili u mom GitHub skladištu (https://github.com/thedrlambda/five-lines).

Forum za diskusiju liveBook

Kupovina knjige *Pet linija koda* omogućava besplatan pristup privatnom veb forumu koji vodi Manning Publications, gde možete da ostavite komentar na knjigu, da postavljate tehnička pitanja i dobijate pomoć od autora i drugih korisnika. Da biste pristupili forumu, otvorite sledeću stranicu: https://livebook.manning.com/#!/book/five-lines-of-code/discussion. Takođe, možete da saznate više o Manning forumima i pravilima ponašanja na stranici https://livebook.manning.com/#!/discussion.

Manning-ova posvećenost svojim čitaocima obavezuje nas da obezbedimo mesto gde se može održati smislen dijalog između čitalaca, i između čitalaca i autora. To nije posvećenost bilo kakvom određenom iznosu učešća od strane autora, čiji doprinos forumu ostaje dobrovoljan (i neplaćen). Predlažemo da pokušate da mu postavite neka izazovna pitanja da njegovo interesovanje ne bi "isparilo"! Forum i arhiva prethodnih diskusija biće dostupni sa veb stranice izdavača sve dok je knjiga u štampi.

Bonus projekat

Da bih vam pomogao da dodatno shvatite kako da koristite pravila i obrasce refaktorisanja iz ove knjige, postavio sam bonus projekat. Ovaj projekat je malo napredniji i nije obezbeđeno rešenje; možete da ga preuzmete sa GitHub skladišta: https://github.com/thedrlambda/bomb-guy. Srećno!

PRIZNANJA

Prvo, ne bih bio osoba kakva jesam, i ne bih napisao ovu knjigu, da nije bilo dvoje ljudi kojima je ova knjiga posvećena: Olivijea Danvija i Majera Goldberga. Ne mogu dovoljno da zahvalim svakome od vas. Naučili ste me teoriju tipova i lambda račun, koji čine samu osnovu ovog rada. Ali kao svaki odličan učitelj, uradili ste mnogo više. Danviju: Znam da je za vas bilo iznenađenje, ali za mene nije iznenađenje što ste vi osoba najvrednija hvale. Zaslužili ste to nudeći savete koji su odmah primenljivi i koji i dalje mogu biti korisni godinama kasnije. Majeru: Vaš neiscrpni entuzijazam, strpljenje i metod podučavanja proizvoljno složenih tema u programiranju oblikovali su način na koji razmišljam i predajem programiranje.

Takođe, želim da izrazim veliku zahvalnost Robertu C. Martinu; ako neko smatra da je ova knjiga inspirativna, kakvom ja smatram vašu, biću srećan. Takođe sam neverovatno zahvalan što ste odvojili vreme da pregledate ovu knjigu i odlučili da napišete predgovor.

Poslednja osoba koja je doprinela ovoj knjizi je moj grafički dizajner: hvala vam Li Mekgori. Vaša kreativnost i kompetencija podigli su kvalitet grafike na nivo sadržaja.

Iskreno zahvaljujem svima u mom Manning timu. Moj akvizicioni urednik, Endru Voldron, ponudio je fantastične povratne informacije i entuzijazam, zbog čega sam odlučio da radim sa Manning izdavačkom kućom. Moja razvojna urednica, Helen Stergius, bila je moj vodič tokom ogromnog poduhvata koji je bio potreban da napišem knjigu poput ove. Bez njenog ohrabrenja i odličnih povratnih informacija ova knjiga ne bi dostigla ovaj nivo kvaliteta. Moj fantastičan urednik tehničkog razvoja bio je Mark Elston, čiji su komentari uvek bili veoma pronicljivi i tačni; njegov pogled na teme savršeno dopunjuje moj. Takođe, hvala uredniku kopija, marketinškom timu i Manning izdavačkoj kućina saradnji i strpljenju.

Zahvaljujem i ljudima koji su mi bili mentori u mom radnom životu. Jakob Blom: Naučili ste me svojim primerom kako da budem tehnički briljantan konsultant bez žrtvovanja sebe ili svojih vrednosti. Vaša strast prema onome što radite je očigledna zbog činjenice da ste mogli da prepoznate i da se setite koda na kome ste radili 10 godina ranije – nešto što me još uvek zbunjuje. Klaus Noregard: Vaš nivo unutrašnjeg mira i dobrote je nešto čemu težim svaki dan. Johan Abildskov: Nikada nisam sreo osobu koja ima toliku tehničku širinu i dubinu u isto vreme, kojima može da parira samo vaša ljubaznost. Bez vas ova knjiga nikada ne bi napustila moj hard disk. Takođe, zahvaljujem svim ljudima kojima sam bio mentor ili sa kojima sam blisko sarađivao.

Takođe, želim da zahvalim svim ljudima koji su pomogli da ova knjiga postane ono što jeste kroz povratne informacije i bezbrojne tehničke diskusije. Odlučio sam da provodim vreme sa vama jer mi život činite boljim. Hanibal Keblovski: Vaša radoznalost je iznedrila originalnu ideju za ovu knjigu. Mikel Kringelbah: Hvala vam što ste mi pomogli svaki put kada sam to zatražio, što ste me izazvali intelektualno i podelili svoj uvid i iskustva, što je značajno koristilo knjizi. Mikel Brun Jakobsen: Vaša strast i kompetencija u izradi softvera me inspirišu i podstiču da budem bolji. Hvala svima koji

ste u odvojili svoje slobodno vreme i u svakom trenutku sebe smatrali delom zajednice nastavnika; vaša neutaživa žeđ za znanjem učinila je da nastavim da predajem. Posebno zahvaljujem: Sune Ort Sorensenu, Matijasu Voreiter Pedersenu, Jens Jensenu, Kasperu Freksenu, Matijasu Baku, Frederiku Brink Trulsenu, Kentu Grigou, Džonu Smedegardu, Ričardnu Monu, Kristoferu Nodebo Knudsenu, Kenetu Hansenu, Rasmusu Bucholdtu i Kristoferu Just Andersenu.

Na kraju, zahvaljujem svim recenzentima: Benu McNamari, Biliju O'Calaganu, Bonnie Maleci, Brentu Honadelu, Čarlsu Lamu, Kristianu Hasselbalch Toudalu, Klajvu Harberu, Danielu Vaskuezu, Davidu Trimu, Gustavu Filipe Ramos Gomesu, Džefu Neumanu, Džoelu Kotarskom, Džonu Guthrieu , Džonu Norkotu, Karthikeiarajanu Rajendranu, Kim Kjersulfi, Luisu Mouku, Marcelu van den Brinku, Mareku Petaku, Matijsu Afourtitu, Orlandu Mendez Moralesu, Paulu Nuinu, Ronaldu Haringu, Šonu Mehafiu, Sebastianu Larsonu, Sergiu Popau, Tan Viu, Tailoru Dolezalu, Tomu Maddenu, Tajleru Kovallisu i Ubaldou Pescatoreu - vaši predlozi su pomogli da ova knjiga bude bolja.

O AUTORU

CHRISTIJAN CLAUSEN magistrirao je računarstvo. Specijalizovao je programske jezike, tačnije kvalitet softvera i kodiranje bez grešaka. Bio je koautor dva recenzirana rada na temu kvaliteta softvera, objavljena u nekim od najprestižnijih časopisa i konferencija. Kristijan je radio kao softverski inženjer na projektu pod nazivom Coccinelle za istraživačku grupu u Parizu. Predavao je uvodne i napredne teme programiranja u objektno orijentisanim i funkcionalnim programskim jezicima na dva univerziteta. Kristijan je pet godina radio kao konsultant i tehnološki vođa.

O ILUSTRACIJI NASLOVNE STRANICE

Figura na naslovnoj strani knjige *Pet linija koda* je pod naslovom „Femme Samojede en habit d'Ete", ili "Samojedska žena u letnjoj odeći". Ilustracija je preuzeta iz kolekcije kostima iz raznih zemalja, Žaka Grasea de Sen Sovera (1757–1810), pod nazivom *Costumes Civils Actuels de Tous les Peuples Connus*, objavljene u Francuskoj 1788. godine, Svaka ilustracija je fino nacrtana i ručno obojena. Bogata raznolikost kolekcije Grassea de Sen-Sovera nas živo podseća na to koliko su pre samo 200 godina gradovi i regioni u svetu bili kulturno odvojeni. Izolovani jedni od drugih, ljudi su govorili različitim dijalektima i jezicima. Na ulicama ili na selu, samo po njihovoj odeći bilo je lako prepoznati gde žive i šta je njihov zanat ili položaj u životu.

Način na koji se oblačimo se od tada promenio, a raznolikost po regionima, tako bogata u to vreme, je nestala. Sada je teško razlikovati stanovnike različitih kontinenata, a kamoli različitih gradova, regiona ili zemalja. Možda smo zamenili kulturnu raznolikost za raznovrsniji lični život - svakako za raznovrsniji i brzi tehnološki život.

U vreme kada je teško razlikovati jednu kompjutersku knjigu od druge, Manning slavi inventivnost i inicijativu kompjuterskog poslovanja koricama knjiga zasnovanim na bogatoj raznolikosti regionalnog života od pre dva veka, koje je oživeo Grase de Sen- Sover na svojim slikama.

1

REFAKTORISANJE REFAKTORISANJE

Ovim poglavljem obuhvaćene su sledeće teme:

- Razumevanje elemenata refaktorisanja
- Uključivanje refaktorisanja u svakodnevni rad
- Važnost bezbednosti za refaktorisanje
- Predstavljanje sveobuhvatnog primera za prvi deo knjige

Dobro je poznato da visok kvalitet koda dovodi do jeftinijeg održavanja, manje grešaka i zadovoljnijih programera. Najčešći način da dobijete visok kvalitet koda je refaktorisanje. Međutim, način na koji se refaktorisanje obično uči — pomoću *mirisa koda* i *jediničnim testiranjem* - nameće nepotrebno visoku barijeru za ulazak. Verujem da svako, uz malo vežbe, može bezbedno da izvrši jednostavne obrasce refaktorisanja.

U razvoju softvera probleme postavljamo negde na dijagramu prikazanom na slici 1.1, što ukazuje na nedostatak veštine, kulture, alata ili njihove kombinacije. Refaktorisanje je sofisticiran poduhvat i stoga se nalazi u sredini. Zahteva svaku komponentu:

- *Veština* - Veština nam je potrebna da znamo koji je kod loš i zahteva refaktorisanje. Iskusni programeri to mogu da utvrde poznavanjem mirisa koda. Ali granice mirisa koda su mutne (zahtevaju rasuđivanje i iskustvo) ili su podložne tumačenjima i stoga ih nije lako naučiti; a mlađem programeru razumevanje mirisa koda može izgledati više kao šesto čulo nego kao veština.

- *Kultura* - Potrebni su nam kultura i radni tok koji podstiču odvajanje vremena za refaktorisanje. U mnogim slučajevima ova kultura se sprovodi preko poznate *crveno-zeleno-refaktor* petlje koja se koristi u razvoju vođenom testovima. Međutim, po mom mišljenju, razvoj vođen testovima je mnogo teži zanat. Petlja crveno-zeleno-refactor takođe ne omogućava jednostavno refaktorisanje zastarele baze koda.
- *Alati* - Potrebno nam je nešto što će nam pomoći da osiguramo da je ono što radimo bezbedno. Najčešći način da se to postigne je automatizovano testiranje. Ali kao što je već pomenuto, učenje efikasnog automatizovanog testiranja je samo po sebi teško.

Slika 1.1 Veština, kultura i alati

U sledećim odeljcima detaljno ćemo opisati svaku od ovih oblasti i kako možemo da započnemo putovanje refaktorisanja iz mnogo jednostavnije osnove, bez testiranja i apstraktnih mirisa koda. Učenje refaktorisanja na ovaj način može da omogući podizanje kvaliteta koda mlađih programera, studenata i entuzijasta programiranja, na viši nivo. Tehnički lideri takođe mogu da koriste metode iz ove knjige kao osnovu za uvođenje refaktorisanja u timove koji to ne rade rutinski.

1.1 Šta je refaktorisanje?

Mnogo detaljnije ću odgovoriti na pitanje „Šta je refaktorisanje?“ u sledećem poglavlju, ali je od pomoći da steknete intuiciju za to unapred, pre nego što zaronimo u različite *načine* refaktorisanja. U svom najjednostavnijem obliku, *refaktorisanje* znači „promena koda bez promene onoga što on izvršava“. Počnimo od primera refaktorisanja, da bi bilo jasno o čemu govorim. Ovde zamenjujemo izraz lokalnom promenljivom.

Programski kod 1.1 Pre

```
return pow(base, exp / 2) * pow(base, exp / 2);
```

Programski kod 1.2 Posle

```
let result = pow(base, exp / 2);
return result * result;
```

Postoji mnogo mogućih razloga za refaktorisanje:

- Brži kod (kao u prethodnom primeru)
- Smanjivanje koda
- Uopšteniji ili višekratni kod
- Olakšavanje čitanja ili održavanja koda

Poslednji razlog je toliko važan i presudan da ga izjednačavamo sa dobrim kodom.

DEFINICIJA *Dobar kod* je čitljiv, lak za održavanje i ispravno obavlja ono što je zamišljeno da uradi.

Pošto refaktorisanje ne sme da promeni ono što kod izvršava, u ovoj knjizi se fokusiramo na čitljiv kod koji je lak za održavanje. O ovim razlozima za refaktorisanje ćemo detaljnije govoriti u poglavlju 2. U ovoj knjizi razmatramo samo refaktorisanje koje rezultira dobrim kodom; dakle, definicija koju koristimo je sledeća.

DEFINICIJA *Refaktorisanje* - Promena koda da bi bio čitljiviji i lakši za održavanje, bez promene onoga što izvršava.

Takođe bi trebalo da napomenem da se tip refaktorisanja koji razmatramo u velikoj meri oslanja na korišćenje objektno-orijentisanog programskog jezika.

Mnogi ljudi misle o programiranju kao o pisanju koda; međutim, većina programera provodi više vremena čitajući i pokušavajući da razume kod, nego što ga piše, zato što radimo u složenom domenu, a promena nečega bez razumevanja može da dovede do katastrofalnih neuspeha.

Dakle, prvi argument za refaktorisanje je čisto ekonomski: vreme programera je skupo, pa ako bazu koda učinimo čitljivijom, oslobađamo vreme za implementaciju novih funkcija. Drugi argument je da ako je kod jednostavniji za održavanje to podrazumeva manje grešaka koje se lakše ispravljaju. Treće, dobra baza koda je jednostavno zabavnija. Kada čitamo kod, u glavama gradimo model onoga što kod izvršava; što više toga moramo da držimo na umu u jednom trenutku, više nas iscrpljujuje. Zbog toga je mnogo zabavnije početi od nule - i zato otklanjanje grešaka može da bude strašno.

1.2 Veština: Šta refaktorisati?

Prva prepreka koju je potrebno da prevaziđete je da znate šta bi trebalo da refaktorišete. Obično se refaktorisanje uči zajedno sa nečim što nazivamo *miris koda*. Ovi „mirisi“ su opisi nečega, koji bi mogli da sugerišu da je kod loš. Iako su moćni, oni su takođe apstraktni i veoma je teško započeti učenje na njima, a potrebno je vreme da se razvije osećaj za njih.

Ova knjiga ima drugačiji pristup i predstavlja lako prepoznatljiva, primenljiva pravila za određivanje šta je potrebno da refaktorišete. Ta pravila su laka za korišćenje i brzo se uče. Takođe su ponekad previše stroga i zahtevaju da ispravite kod koji ne „miriše“. U retkim prilikama možemo da poštujemo pravila i da i dalje imamo mirišljav kod.

Kao što je ilustrovano na slici 1.2, preklapanje između mirisa i pravila nije savršeno. Moja pravila nisu sve i kraj svog dobrog koda. Ona su početak na putu ka razvijanju osećaja za to šta je dobar kod. Pogledajmo primer razlike između mirisa koda i pravila u ovoj knjizi.

Slika 1.2 Pravila i mirisi koda

1.2.1 Primer mirisa koda

Dobro poznati miris koda je sledeći: funkcija bi trebalo da izvršava *jedan* zadatak. To je odlična smernica, ali nije lako znati koji je taj *jedan zadatak*. Pogledajte ponovo prethodni kod: da li ima miris? Kod deli, eksponira, a zatim množi. Da li to znači da izvršava tri zadatka? S druge strane, vraća samo jedan broj i ne menja nijedno stanje, pa da li izvršava samo jedan zadatak?

```
let result = pow(base, exp / 2);
return result * result;
```

1.2.2 Primer pravila

Uporedite prethodni miris koda sa sledećim pravilom (detaljno obrađenim u poglavlju 3): metod nikada ne bi trebalo da ima više od *Pet linija koda*. To možemo da utvrdimo na prvi pogled, bez dodatnih pitanja. Pravilo je jasno, sažeto i lako za pamćenje - pogotovo što je to i naslov ove knjige.

Zapamtite, pravila predstavljena u ovoj knjizi su poput točkova za treniranje. Kao što je ranije rečeno, pravila ne mogu da garantuju dobar kod u svakoj situaciji; a u nekim prilikama bi moglo da bude pogrešno slediti ih. Međutim, pravila su korisna ako ne znate odakle da počnete, a motivišu lepo refaktorisanje koda.

Imajte na umu da su svi nazivi pravila navedeni u apsolutnim terminima, korišćenjem reči poput *nikad*, pa ih je lako zapamtiti. Ali detaljni opisi često navode izuzetke: kada da *ne* primenite pravila. Opisi takođe navode namere pravila. Na početku učenja refaktorisanja potrebno je da koristimo samo apsolutne nazive; kada se oni odomaće, možemo da počnemo da učimo i izuzetke, nakon čega možemo da počnemo da koristimo namere - tada ćemo biti gurui kodiranja.

1.3 Kultura: Kada refaktorisati?

Refaktorisanje je kao tuširanje.

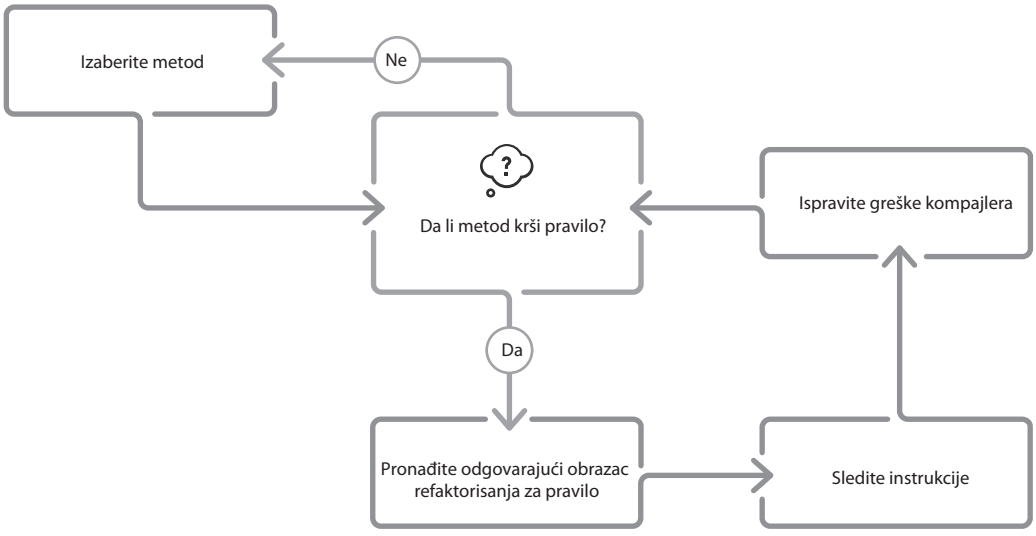
—Kent Bek

Refaktorisanje najbolje funkcioniše — i košta najmanje — ako to radite redovno. Dakle, ako možete, preporučujem vam da uključite refaktorisanje u svoj svakodnevni rad. Većina literature preporučuje tok rada crveno-zeleno-refaktor; ali kao što je ranije pomenuto, ovaj pristup vezuje refaktorisanje sa razvojem vođenim testiranjem — a u ovoj knjizi želimo da ih razdvojimo i da se posebno fokusiramo na deo koji se odnosi na refaktorisanje. Stoga preporučujem opštiji tok rada u šest koraka za rešavanje bilo kog programskog zadatka, kao što je prikazano na slici 1.3:

1. *Istraživanje*. Često nismo potpuno sigurni šta je potrebno da izgradimo od samog početka. Ponekad kupac ne zna šta želi da izgradimo; a ponekad su zahtevi napisani u dvosmislenoj prozi; ponekad čak i ne znamo da li se zadatak može rešiti. Dakle, uvek počnite eksperimentisanjem. Implementirajte nešto na brzinu, a zatim možete da potvrdite sa klijentom da se slažete šta im je potrebno.
2. *Specifikovanje*. Kada znate šta je potrebno da izgradite, učinite to eksplicitnim. Optimalno, to rezultira nekim oblikom automatizovanog testa.
3. *Implementiranje*. Implementirajte kod.
4. *Testiranje*. Uverite se da kod prolazi specifikaciju iz koraka 2.
5. *Refaktorisanje*. Pre isporuke koda, uverite se da sledeća osoba može lako da ga koristi (a ta sledeća osoba možda ćete biti vi).
6. *Isporuka*. Postoji mnogo načina za isporuku koda; najčešći su putem zahteva za povlačenjem ili guranjem na određenu granu. Najvažnije je da kod dođe do korisnika. Inače, koja je poenta?

Slika 1.3 Tok rada

Pošto radimo refaktorisanje *zasnovano na pravilima,* tok rada je jednostavan i lak za početak. Na slici 1.4 prikazan je 5. korak: *refaktorisanje*.



Slika 1.4 Detaljan prikaz koraka refaktorisanja

Dizajnirao sam pravila tako da se lako pamte i da je lako uočiti kada ih koristiti, bez ikakve pomoći. To znači da je pronalaženje metoda koji krši pravilo obično trivijalno. Svako pravilo takođe ima nekoliko obrazaca refaktorisanja koji su povezani sa njim, što olakšava tačno saznavanje kako da rešite problem. Obrasci za refaktorisanje imaju eksplicitna uputstva korak po korak kako biste osigurali da slučajno nešto ne prekinete. Mnogi obrasci refaktorisanja u ovoj knjizi namerno koriste greške kompajliranja kako biste bili sigurni da nećete unositi greške. Nakon malo vežbe, i pravila i obrasci refaktorisanja postaće vam potpuno jasni.

1.3.1 Refaktorisanje u zastarelom sistemu

Čak i ako polazimo od velikog zastarelog sistema, postoji pametan način da ugradimo refaktorisanje u svakodnevni rad, bez potrebe da sve zaustavimo i prvo refaktorišemo celu bazu koda. Jednostavno sledite ovaj sjajan citat:

Prvo olakšajte promenu, a zatim izvršite laku promenu.

—Kent Bek

Kad god se spremamo da implementiramo nešto novo, počinjemo od refaktorisanja, tako da je lako dodati novi kod. To je slično pripremanju svih sastojaka pre nego što počnete da pečete.

1.3.2 Kada ne bi trebalo da vršite refaktorisanje?

Uglavnom je refaktorisanje odlično, ali ima nekoliko nedostataka. Refaktorisanje može oduzeti mnogo vremena, posebno ako ga ne radite redovno. A kao što je ranije pomenuto, vreme programera je skupo.

Postoje tri tipa baza koda gde refaktorisanje verovatno nije vredno:

- Kod koji ćete napisati, pokrenuti samo jednom, a zatim ga izbrisati. To je poznato kao *spike* u zajednici ekstremnog programiranja.

- Kod koji je u režimu održavanja pre nego što će biti povučen.
- Kod sa strogim zahtevima za performansama, kao što su ugrađeni sistem ili vrhunski fizički mehanizam u igri.

U svakom drugom slučaju, tvrdim da je ulaganje u refaktorisanje pametan izbor.

1.4 Alati: Kako refaktorisati (bezbedno)

Volim automatizovane testove. Međutim, učenje kako da efikasno testirate softver je samo po sebi komplikovana veština. Dakle, ako već znate kako da radite automatizovano testiranje, slobodno ga koristite u ovoj knjizi. Ako ne znate, ne brinite.

O testiranju možemo razmišljati na sledeći način: automatizovano testiranje je za razvoj softvera ono što su kočnice za automobile. Automobili nemaju kočnice jer želimo da idemo polako - oni imaju kočnice, tako da se osećamo bezbedno kada vozimo brzo. Isto važi i za softver: automatizovani testovi čine da se osećamo bezbedno ako se krećemo brzo. U ovoj knjizi učimo potpuno novu veštinu, pa nema potrebe da idemo brzo.

Umesto toga, predlažem da se više oslanjate na druge alate, kao što su:

- Detaljni strukturirani obrasci refaktorisanja, korak po korak, slični receptima
- Kontrola verzija
- Kompajler

Verujem da ako su obrasci refaktorisanja pažljivo dizajnirani i izvedeni u malim koracima, moguće je refaktorisati bez prekidanja bilo čega. To je posebno tačno u slučajevima kada IDE može da izvrši refaktorisanje umesto nas.

Da bismo ispravili činjenicu da u ovoj knjizi ne govorimo o testiranju, koristimo kompajler i tipove da bismo uhvatili mnogo uobičajenih grešaka koje bismo mogli da napravimo. Uprkos tome, preporučujem da redovno otvarate aplikaciju na kojoj radite i da proveravate da nije potpuno prekinuta. Kad god to potvrdite, ili kada znate da je kompajler zadovoljan, potvrdite to, tako da ako se u nekom trenutku aplikacija prekine i ne znate odmah kako da je ispravite, lako možete da se vratite na vreme kada je poslednji put funkcionisala.

Ako radimo na sistemu iz stvarnog sveta bez automatizovanih testova, i tako možemo da izvršimo refaktorisanje, ali moramo odnekud da steknemo samopouzdanje. Samopouzdanje može proizići iz upotrebe integrisanog razvojnog okruženja (IDE) za izvođenje refaktorisanja; ručnog testiranja; izvršavanjem zaista malih koraka; ili nečeg drugog. Međutim, dodatno vreme koje bismo potrošili na ove aktivnosti verovatno čini automatizovano testiranje isplativijim.

1.5 Alati koji su vam potrebni za početak

Kao što sam ranije rekao, tipovima refaktorisanja o kojima se govori u ovoj knjizi potreban je objektno-orijentisan jezik. To je osnovna stvar koja vam je potrebna da biste pročitali i razumeli ovu knjigu. Kodiranje i refaktorisanje su zanati koje izvodimo prstima. Zato ih je najbolje naučiti tako što ćete pratiti primere, eksperimentisati i zabavljati se, dok vaše ruke uče rutine. Da biste pratili knjigu, potrebni su vam alati opisani u nastavku. Uputstva za instalaciju pogledajte u dodatku.

1.5.1 Programski jezik: TypeScript

Svi primeri kodiranja predstavljeni u ovoj knjizi napisani su jezikom TypeScript. Odabrao sam TypeScript iz više razloga. Što je najvažnije, izgleda i deluje slično kao najčešće korišćeni programski jezici - Java, C#, C++ i JavaScript - i stoga bi ljudi koji su upoznati sa bilo kojim od tih jezika trebalo da budu u mogućnosti da čitaju TypeScript bez ikakvih problema. TypeScript takođe pruža način da se pređe od potpuno „neobjektno orijentisanog" koda (tj. koda bez i jedne klase) do visoko objektno orijentisanog koda.

> **NAPOMENA** Da bismo bolje iskoristili prostor u štampanoj knjizi koristimo stil programiranja koji izbegava prelome linija, a i dalje je čitljiv. Ne zagovaram da koristite isti stil - osim ako slučajno pišete knjigu koja sadrži mnogo TypeScript koda. To je takođe razlog zašto su uvlake i zagrade ponekad drugačije formatirane u knjizi nego u kodu projekta.

Ako niste upoznati sa jezikom TypeScript, objasniću sve nedostatke dok se pojavljuju, u okvirima kao što je sledeći.

U jeziku TypeScript...

Koristimo identitet (`===`) da proverimo jednakost, jer deluje više kao ono što očekujemo od jednakosti nego od dvostruke jednakosti (`==`). Uzmite u obzir sledeće:

- `0 == ""` je `true`.
- `0 === ""` je `false`.

Iako su primeri napisani jezikom TypeScript, svi obrasci i pravila refaktorisanja su opšti i primenjuju se na bilo koji objektno-orijentisan jezik. U retkim slučajevima, TypeScript nam pomaže ili nas ometa; ti slučajevi su eksplicitno navedeni, a govorićemo o tome kako da postupamo u ovim situacijama u drugim uobičajenim jezicima.

1.5.2 Uređivač koda: Visual Studio Code

Ne pretpostavljam da koristite određeni uređivač koda; međutim, ako nemate afinitet, preporučujem Visual Studio Code. Dobro funkcioniše za jezik TypeScript. Takođe, podržava pokretanje komande `tsc -w` u pozadinskom terminalu koji izvršava kompajliranje, tako da ne zaboravimo da to uradimo.

> **NAPOMENA** *Visual Studio Code* je potpuno drugačiji alat od razvojnog okruženja *Visual Studio*.

1.5.3 Kontrola verzija: Git

Iako se od vas ne zahteva da koristite kontrolu verzija da biste sledili vežbe za ovu knjigu, toplo je preporučujem, jer vam mnogo olakšava da poništite nešto ako se negde izgubite.

Vraćanje na referentno rešenje

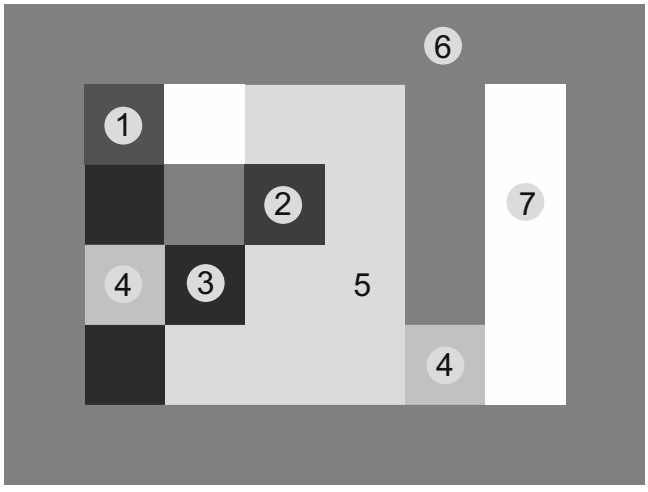
Na početku većeg odeljka u bilo kom trenutku možete da pređete na kod, kako bi trebalo da izgleda, korišćenjem komande kao što je:

```
git reset --hard section-2.1
```

Oprez: Izgubićete sve izmene koje ste napravili.

1.6 Sveobuhvatni primer: 2D slagalica

Na kraju, hajde da razgovaramo o tome kako ću podučavati o svim ovim divnim pravilima i neverovatnim obrascima refaktorisanja. Knjiga je izgrađena oko jednog sveobuhvatnog primera: 2D slagalica koja gura blokove, slična klasičnoj igri Boulder Dash (slika 1.5).



Slika 1.5 Snimak ekrana igre

To znači da imamo jednu značajnu bazu koda sa kojom možemo da se igramo u prvom delu knjige. Korišćenje jednog primera štedi vreme jer ne moramo da upoznajemo novi primer u svakom poglavlju.

Primer je napisan realističnim stilom, slično onom koji se koristi u industriji. To nikako nije laka vežba, osim ako imate veštine predstavljene u ovoj knjizi. Kod se već pridržava *DRY* (Don‘t Repeat Yourself) *KISS* (Keep It Simple, Stupid) principa; čak i tako, nije prijatnije od „suvog poljupca“.

Odabrao sam kompjutersku igricu jer prilikom ručnog testiranja lako može da se uoči da li se nešto ponaša pogrešno: imamo intuiciju kako bi trebalo da se ponaša. Takođe je malo zabavnije testirati nego gledati nešto poput evidencije iz finansijskog sistema.

Korisnik kontroliše kvadrat igrača pomoću tastera sa strelicama. Cilj igre je da se kutija (označena sa 2 na slici 1.5) dovede u donji desni ugao. Iako boje nisu prikazane u štampanoj knjizi, elementi igre su različitih boja, i to:

1. Crveni kvadrat je igrač.
2. Smeđi kvadrati su kutije.
3. Plavi kvadrati su kamenje.
4. Žuti kvadrati su ključevi *ili* brave - to ćemo ispraviti kasnije.
5. Zelenkaste kvadrate nazivamo *fluks*.
6. Sivi kvadrati su zidovi.
7. Beli kvadrati su vazduh (prazni).

Ako kutija ili kamen nisu ničim poduprti, oni padaju. Igrač može da gura jedan po jedan kamen ili kutiju, pod uslovom da nije zaprečen ili da ne pada. Putanja između kutije i donjeg desnog ugla u početku je blokirana bravom, tako da igrač mora da uzme ključ da bi je uklonio. Ako igrač nagazi na fluks, fluks može da bude „pojeden“ (uklonjen).

Sada bi bilo sjajno vreme da preuzmete igru i malo se poigrate:

1. Otvorite konzolu na kojoj želite da sačuvate igru.
    a. Komanda `git clone https://github.com/thedrlambda/five-lines` preuzima izvorni kod za igru.
    b. Komanda `tsc -w` kompajlira TypeScript u JavaScript svaki put kada se promeni.
2. Otvorite index.html u pretraživaču.

Moguće je da promenite nivo u kodu, pa se slobodno zabavite kreiranjem sopstvenih mapa ažuriranjem niza u promenljivoj `Map` (na primer, pogledajte dodatak):

1. Otvorite direktorijum u uređivaču koda Visual Studio Code.
2. Izaberite Terminal, a zatim NewTerminal.
3. Pokrenite komandu `tsc -w`.
4. TypeScript sada kompajlira promene u pozadini i možete da zatvorite terminal.
5. Svaki put kada izvršite promenu, sačekajte trenutak dok se TypeScript kompajlira, a zatim osvežite pretraživač.

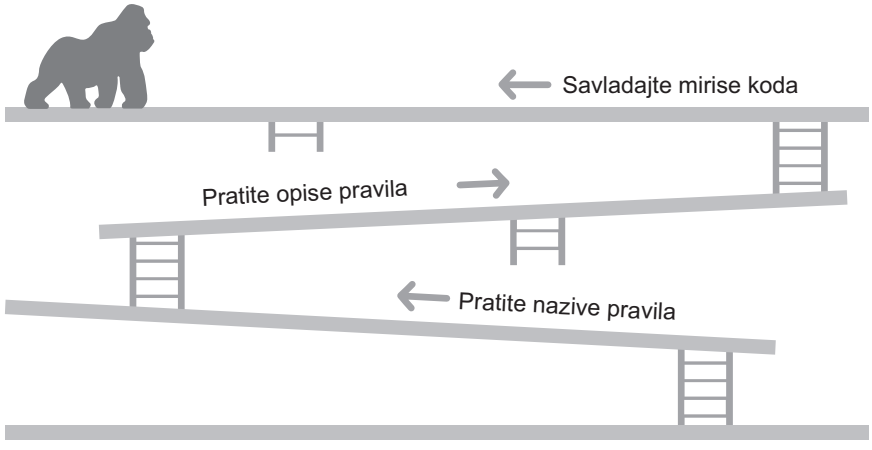
Ovo je isti postupak koji ćete koristiti kada kodirate zajedno sa primerima u prvom delu. Međutim, pre nego što dođemo do toga, u sledećem poglavlju ćemo izgraditi detaljniju osnovu refaktorisanja.

1.6.1 Praksa čini savršen kod: Druga baza koda

Pošto čvrsto verujem u praksu, napravio sam još jedan projekat, bez rešenja. Možete da koristitetaj projekat za ponovno čitanje knjige, ako želite izazov; ili kao vežbe za učenike, ako ste nastavnik. Ovaj projekat je 2D akciona igra. Obe baze koda koriste isti stil i strukturu, imaju iste elemente i potrebni su isti koraci za njihovo refaktorisanje. Iako je ova druga baza koda malo naprednija, pažljivo praćenje pravila i obrazaca refaktorisanja trebalo bi da daju željeni rezultat. Da biste preuzeli ovaj projekat, koristite iste korake koji su opisani za URL `https://github.com/thedrlambda/bomb-guy`.

1.7 Napomena o softveru u stvarnom svetu

Važno je da ponovim da je fokus ove knjige predstavljanje refaktorisanja. Fokus *nije* na pružanju specifičnih pravila koja možete da primenite na proizvodni kod u svim okolnostima. Način korišćenja pravila je da prvo naučite njihove nazive i da ih sledite. Kada vam ovo bude postalo lako, naučite opise sa njihovim izuzecima; konačno, iskoristite to znanje da biste izgradili razumevanje osnovnog mirisa koda. Ovo putovanje je ilustrovano na slici 1.6.



Slika 1.6 Kako da koristite pravila

Ovo takođe daje odgovor na pitanje zašto ne možemo da napravimo automatski program za refaktorisanje. (Možda ćemo moći da napravimo plagin za isticanje *potencijalno problematičnih* oblasti u kodu na osnovu pravila.) Svrha pravila je izgradnja razumevanja. Ukratko: pridržavajte se pravila dok ne naučite više.

Takođe, imajte na umu da s obzirom na to da se fokusiramo samo na učenje refaktorisanja i imamo bezbedno okruženje, možemo da radimo bez automatizovanih testova - ali to verovatno nije tačno za stvarne sisteme. To radimo jer je mnogo lakše naučiti automatizovano testiranje i refaktorisanje odvojeno.

Rezime

- Izvođenje refaktorisanja zahteva kombinaciju *veštine,* da znate šta da refaktorišete, *kulture,* da znate kada treba da refaktorišete i *alata,* da znate kako da refaktorišete.
- Konvencionalno, mirisi koda se koriste za opis onoga što je potrebno refaktorisati. Mladim programerima je teško da ih internalizuju jer su nejasni. U ovoj knjizi opisana su konkretna pravila za zamenu mirisa koda tokom učenja. Pravila imaju tri nivoa apstrakcije: vrlo konkretni nazivi, opisi koji dodaju nijansu u vidu izuzetaka i, na kraju, namera mirisa iz kojih su izvedeni.
- Verujem da automatizovano testiranje i refaktorisanje mogu da se nauče odvojeno kako bi se dodatno smanjila barijera za ulazak. Umesto automatskog testiranja koristimo kompajler, kontrolu verzija i ručno testiranje.
- Tok rada refaktorisanja povezan je sa razvojem vođenim testiranjem u crveno-zeleno-refaktor petlji. Ali, to opet implicira zavisnost od automatizovanog testiranja. Umesto toga, predlažem da koristite tok rada u šest koraka (istraživanje, specifikovanje, implementiranje, testiranje, refaktorisanje, isporučivanje) za novi kod, ili da izvršite refaktorisanje neposredno pre promene koda.
- U prvom delu ove knjige koristimo uređivač koda Visual Studio Code, jezik TypeScript i Git da bismo transformisali izvorni kod 2D slagalice.