Čisto majstorstvo

discipline, standardi i etika

Robert C. Martin

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: Robert C. Martin

Prevod: Biljana Tešić

Recezent: Đorđe Radonić

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2021.

Broj knjige: 548

Izdanje: Prvo

ISBN: 978-86-7310-571-0

Clean Craftsmanship

Disciplines, Standards, and Ethics

Robert C. Martin

ISBN:978-0-13-691571-3

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.41

МАРТИН, Роберт Сесил, 1952-
Čisto majstorstvo: razvoj softvera: discipline, standardi i etika / Robert C. Martin ; [prevod Biljana Tešić]. - 1. izd. - Beograd: Kompjuter Biblioteka, 2021 (Zemun: Pekograf). - XXVIIII, 385 str.: ilustr. ; 24 cm. - (Kompjuter biblioteka; br. knj. 548)

Prevod dela: Clean Craftsmanship. - Autorova slika. - Tiraž 500. O autoru: str. [XXVII-XXVIII]. - Napomene uz tekst. - Registar.

ISBN 978-86-7310-571-0

a) Софтвер - Пројектовање
COBISS.SR-ID 54373385

Pohvala „Čistom majstorstvu“

U Bobovoj knjizi „Čisto majstorstvo“ odlično su objašnjeni svrha agilne tehničke prakse, zajedno sa detaljnom istorijskom osnovom o tome kako je ta praksa nastala, i pozicioniranje zbog kojeg će uvek biti važna. Učešće Ujke Boba u istoriji i formiranju agilnosti, temeljno razumevanje prakse i njena svrha veoma jasno se odražavaju u celoj knjizi.

Tim Ottinger,
poznati agilni kouč i autor

Bobov stil pisanja je odličan. Knjiga se lako čita i koncepti su savršeno detaljno objašnjeni, tako da može da ih prati i svaki novopečeni programer. Bob čak ima i neke smešne trenutke koji vas mogu razveseliti. Prava vrednost knjige je zaista u vapaju za promenom, za nečim boljim, u vapaju da programeri budu profesionalni, u spoznaji da je softver svuda. Osim toga, verujem da postoji velika vrednost u celoj istoriji koju Bob ispisuje. Drago mi je što ne gubi vreme okrivljujući nas da smo došli do ovoga gde smo sada. Bob poziva ljude na akciju, tako što traži od njih da preuzmu odgovornost za poboljšanje njihovih standarda i nivoa profesionalizma, iako to ponekad znači povlačenje.

Heather Kanser

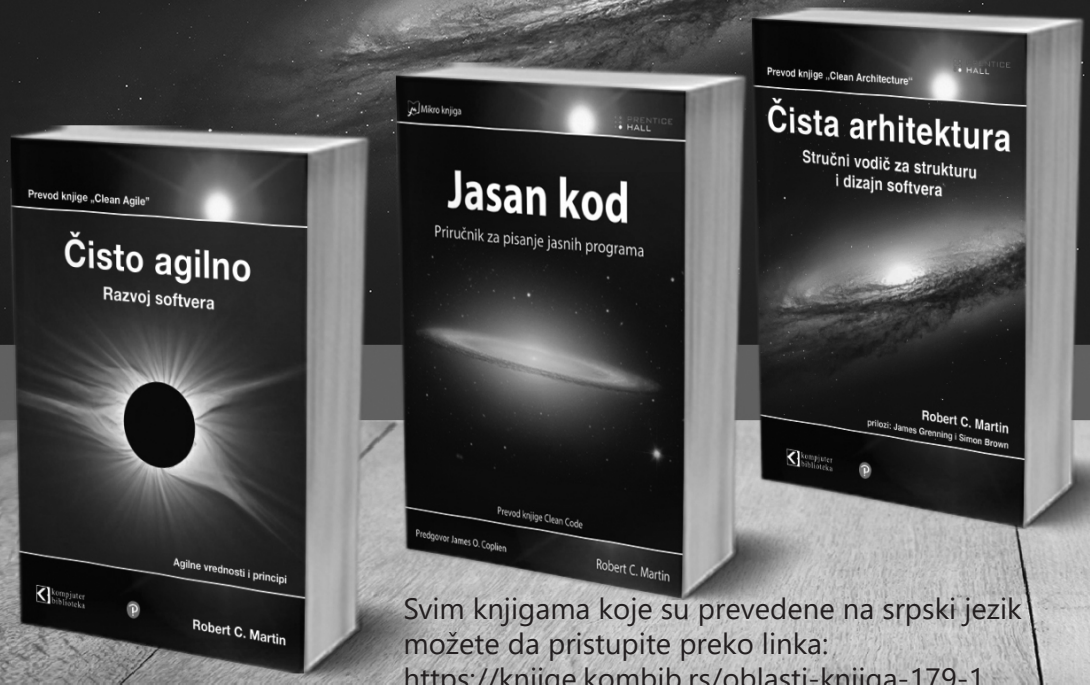
Kao softverski programeri, moramo stalno da rešavamo važne probleme za naše poslodavce, klijente i kolege. To što aplikacija funkcioniše nije dovoljno - to nas ne čini stručnjakom. Kada aplikacija funkcioniše, prošli ste *test rada aplikacije*. Možda možete da budete kreator aplikacija, ali ima još mnogo štošta da savladate. Na ovim stranicama Bob jasno opisuje tehnike i odgovornosti izvan *testa rada aplikacije* i pokazuje kako se postaje ozbiljan kreator softvera.

James Grenning, autor knjige
„Test-Driven Development for Embedded C“
i koautor Agilnog manifesta

Bob je jedan od retkih poznatih programera sa kojim bih voleo da radim na tehničkom projektu, ali ne zato što je Bob dobar programer, popularan ili dobar sagovornik, već zato što mi pomaže da budem bolji programer i član tima. Uočio je svaki veliki razvojni trend godinama ispred drugih i mogao je da objasni njegovu važnost, što me je ohrabrilo da učim. Još kada sam ja počeo da se bavim programiranjem, ideja o majstorstvu i etici potpuno je nedostajala u ovoj oblasti. Čini se da je to najvažnija stvar koju profesionalni programeri mogu naučiti, čak i pre samog pisanja koda. Srećan sam što vidim da je Bob ponovo „vođa puta“. Jedva čekam da čujem njegovu perspektivu i da je ugradim u svoju praksu.

Daniel Markham, direktor,
Bedford Technology Group, Inc

SERIJA ROBERTA C. MARTINA



Svim knjigama koje su prevedene na srpski jezik možete da pristupite preko linka:
https://knjige.kombib.rs/oblasti-knjiga-179-1

Kratak sadržaj

Sadržaj

POGLAVLJE 2

POGLAVLJE 3

POGLAVLJE 9

POGLAVLJE 10

POGLAVLJE 11

DEO III

POGLAVLJE 12

POGLAVLJE 13

Uvodna reč

Sećam se da sam upoznao Ujku Boba u proleće 2003. godine, ubrzo nakon što je Scrum predstavljen našoj kompaniji i tehnološkim timovima. Sećam se da sam kao skeptični, novopečeni ScrumMaster, slušao Boba kako nas uči o TDD-u i maloj alatki koja se zove FitNesse, i u sebi sam pomislio: „Zašto bismo *ikada* pisali testne slučajeve koji odmah nisu uspešni? Zar testiranje ne dolazi *nakon* pisanja koda?“ I dan danas se jasno sećam Bobovog primetnog entuzijazma kada je reč o razvoju koda, kao da je to bilo tek juče. Sećam se da nas je jednog dana, dok je gledao naš spisak grešaka (bug backlog), pitao zašto bismo, pobogu, donosili tako loše odluke o softverskim sistemima koji, zapravo, nisu u našem vlasništvu, napomenuvši: „Ovi sistemi su i*movina kompanije*, a ne vaša *lična imovina*.“ Njegova strast probudila je našu znatiželju, a godinu i po dana kasnije refaktorisali smo naš put, tako da smo imali oko 80 posto pokrivenosti automatskim testovima i čistu bazu koda koja je znatno olakšavala zaokret (pivot), pa su korisnici i timovi bili mnogo srećniji. Nakon toga smo se munjevito kretali, pridržavajući se naše definicije *završenog (definition of done)* poput oklopa koji nas štiti od „goblina“ u kodu koji uvek vrebaju; u suštini smo naučili kako da se zaštitimo od nas samih. Vremenom nam je Ujka Bob „prirastao za srce“ i zaista smo se osećali kao da nam je ujak - srdačan, odlučan i hrabar čovek, koji će nam vremenom pomoći da naučimo da se zalažemo za sebe i da činimo ono što je ispravno.

Dok drugu decu njihovi ujaci uče da voze bicikl ili pecaju, naš Ujka Bob nas je naučio da ne ugrožavamo svoj integritet - i do danas su sposobnost i želja da se hrabro i radoznalo pokažem u svakoj situaciji ono najbolje što sam naučio u mojoj karijeri.

Nosio sam Bobove prve lekcije sa sobom na svoje putovanje dok sam odlazio u svet kao agilni kouč i brzo sam uvideo da su najbolji timovi za razvoj proizvoda osmislili kako da upakuju svoju najbolju praksu za svoj jedinstveni kontekst, za svoje određene klijente, u svojim industrijama. Setio sam se Bobovih lekcija kada sam primetio da su najbolje razvojne alatke na svetu dobre samo onoliko koliko i njihovi ljudski rukovaoci - timovi koji će osmisliti najbolju *primenu* tih alatki u okviru svojih oblasti. Primetio sam da, naravno, timovi mogu dostići visoke procente pokrivenosti jediničnim testovima kako bi zadovoljili metriku, a onda bi saznali da je veliki procenat tih testova nestabilan - metrika je zadovoljena, ali vrednost nije isporučena. Najbolji timovi nisu morali da brinu o metrici; imali su cilj, disciplinu, ponos i odgovornost, a metrike su, u svakom slučaju, „govorile" same za sebe. U knjizi „Čisto majstorstvo" sve lekcije i principi utkani su u praktične primere koda i iskustva kako bi bila ilustrovana razlika između pisanja nečega radi ispunjenja roka, s jedne strane, i izrade nečeg održivog za budućnost, s druge strane.

Knjiga „Čisto majstorstvo" nas podseća da nikada ne pristanemo na manje, da treba da hodamo Zemljom sa *nepokolebljivim znanjem*. Ova knjiga, poput starog prijatelja, podsetiće vas na ono što je važno - šta funkcioniše, šta nije važno, šta stvara rizik i šta umanjuje rizik. Ove lekcije su vanvremenske. Možda ćete otkriti da već koristite neke od tehnika sadržanih u ovim lekcijama i kladim se da ćete pronaći nešto novo, ili bar nešto što ste propustili, jer ste u nekom trenutku popustili pred rokovima ili drugim pritiscima u karijeri. Ako ste novajlija u svetu razvoja - bilo u poslovanju ili tehnologiji, učićete od najboljih, pa čak i oni koji su najviše uvežbani i koji su umorni od borbe naći će načine na koje će poboljšati sebe. Možda će vam ova knjiga pomoći da ponovo pronađete svoju strast, da obnovite želju za poboljšanjem svog zanata ili da preusmerite svoju energiju, u potrazi za savršenstvom, bez obzira na prepreke na vašem horizontu.

Softverski programeri vladaju svetom, a Ujka Bob je ponovo tu da nas podseti na profesionalnu disciplinu onih koji imaju takvu moć. On nastavlja tamo gde je stao u knjizi „Jasan kod"; pošto softverski programeri doslovno pišu pravila čovečanstva, Ujka Bob nas podseća da moramo da poštujemo strogi etički kodeks - da znamo kako ga ljudi koriste i da znamo gde se krši. Softverske greške koštaju ljude egzistencije, pa i života. Softver utiče na način na koji razmišljamo i na odluke koje donosimo, a, kao rezultat veštačke inteligencije i prediktivne analitike, utiče na ponašanje društva. Stoga, moramo biti odgovorni i postupati sa velikom pažnjom i empatijom - zdravlje i dobrobit ljudi zavise od toga. Ujka Bob nam pomaže da se suočimo sa ovom odgovornošću i *da postanemo profesionalci u meri u kojoj to od nas naše društvo očekuje*.

Pošto se Agilni manifest približava svom dvadesetom rođendanu nakon pisanja ovog predgovora, ova knjiga je savršena prilika da se vratimo osnovama: pravovremenom i smernom podsetniku na sve veću složenost našeg programerskog sveta i na našu obavezu prema nasleđu čovečanstva i nama samima - da poštujemo razvojnu etiku. Odvojite vreme da pročitate „Čisto majstorstvo". Neka vam ova knjiga bude uvek pri ruci. Neka vam ova knjiga bude stari prijatelj - vaš vodič, dok radoznalo i hrabro krčite sebi put u tom svetu.

Stacia Heimgartner Viscardi, CST i Agile Mentor

.

Predgovor

Najpre je potrebno da razmotrimo dva problema kako bismo bili sigurni da ćete, dragi čitaoci, razumeti referentni okvir u kojem je ova knjiga predstavljena.

O terminu majstorstvo (craftsmanship)

Početak ovog veka obeležen je kontroverzom o jeziku. Mi u softverskoj industriji videli smo svoj udeo u ovoj kontroverzi. Da budem otvoren, termin koji je često nazivan promašajem da bude inkluzivan je *kreator (craftsman*1*)*.

Dosta sam razmišljao o ovom problemu i razgovarao sa mnogim ljudima različitih mišljenja i došao sam do zaključka da nema boljeg termina koji mogu da upotrebim u kontekstu ove knjige.

Razmatrane su alternative za termin kreator - između ostalih, termini majstor i zanatlija. Ali, ni jedan od njih nema istorijsku težinu kao termin *kreator*. A ta istorijska težina važna je za poruku u ovoj knjizi.

Kreator implicira na osobu koja je veoma vešta i uspešna u određenoj aktivnosti, tj. na nekoga ko zna dobro da koristi alat i da se dobro snalazi u trgovini, ko se ponosi svojim radom i kome se može verovati da će se ponašati dostojanstveno i profesionalno u svojoj profesiji.

1 Prim. rec.: Engleska reč „craftsman“ je izrazito muškog roda („man“ – čovek, muškarac). Otud priča o inkluziji. Autor ih dalje poredi sa rodno neutralnijim rečima poput „craftperson“, „craftfolk“, „crafter“…

Možda se neki od vas neće složiti sa mojom odlukom. Razumem zašto se nećete složiti. Nadam se samo da to nećete protumačiti kao pokušaj isključivosti na bilo koji način - jer to ni u kom slučaju nije moja namera.

O jedinom pravom putu

Dok budete čitali knjigu „Čisto majstorstvo: discipline, standardi i etika“, možda ćete imati osećaj da je ovo *jedini pravi put do stručnosti*. Možda je to jedini pravi put *za mene*, ali ne i za vas. Ovu knjigu vam nudim kao primer *mog puta*. Naravno, vi ćete morati da izaberete svoj put.

Da li će nam na kraju biti potreban *jedini pravi put*? Ne znam. Možda. Kao što ćete pročitati na ovim stranicama, pritisak da se pronađe izričita definicija softverske profesije raste. Možda će nam biti potrebno samo nekoliko različitih puteva, u zavisnosti od važnosti softvera koji treba da kreiramo. Međutim, kao što ćete pročitati u nastavku, možda neće biti toliko lako odvojiti važan softver od nevažnog.

U jedno sam siguran. Dani „sudija“2 su prošli. Više nije dovoljno da svaki programer radi ono što on smatra ispravnim. *Pojaviće* se neke discipline, standardi i etika. Odluka koja je pred nama danas je da li ćemo ih mi programeri sami definisati ili će nam ih nametnuti oni koji nas ne poznaju.

Uvod u knjigu

Ova knjiga je napisana za programere i za menadžere programera. Drugim rečima, ova knjiga je napisana za celo ljudsko društvo. Jer, mi, programeri, smo se nenamerno našli na samom uporištu tog društva.

2 Odnosi se na Knjigu o sudijama iz Starog zaveta.

Za sebe

Ako ste programer sa višegodišnjim iskustvom, verovatno vam je poznat osećaj zadovoljstva zbog primene i rada sistema. Postoji određeni ponos koji osećate, jer ste imali udela u uspehu. Ponosni ste jer ste objavili sistem.

Međutim, da li ste ponosni na *način* na koji ste taj sistem objavili? Da li je vaš ponos - ponos završetka? Ili je vaš ponos - ponos izrade? Da li ste ponosni jer je sistem primenjen? Ili ste ponosni na način na koji ste izradili taj sistem?

Kada odete kući nakon napornog dana tokom kojeg ste pisali kod, da li se pogledate u ogledalo i kažete: „Danas sam dobro obavio posao“ ili prvo morate da se istuširate?

Previše nas se oseća prljavo na kraju dana. Previše nas se oseća zarobljeno u obavljanju nekvalitetnih poslova. Previše nas smatra da je nizak kvalitet očekivan i da je neizbežan ukoliko želimo veoma brzo da radimo. Previše nas misli da su produktivnost i kvalitet obrnuto srazmerni.

U ovoj knjizi nastojim da prekinem takav način razmišljanja. Ovo je knjiga o *dobrom radu*. Ovo je knjiga o dobrom obavljanju posla. Ovo je knjiga u kojoj su opisane discipline i prakse koje bi svaki programer trebalo da zna da bi radio brzo, da bi bio produktivan i da bi bio ponosan na ono što piše svakog dana.

Za društvo

Naše društvo je, zbog svog opstanka, u ovom veku prvi put u istoriji čovečanstva postalo zavisno od tehnologije koja praktično nema disciplinu ili kontrolu. Softver je zahvatio sve aspekte savremenog života, od kuvanja jutarnje kafe do večernje zabave, od pranja odeće do vožnje automobilima, od povezivanja sa svetskom mrežom do društvene i političke podele. Doslovno ne postoji aspekt života u savremenom svetu kojim ne dominira softver. Međutim, mi koji izrađujemo ovaj softver ipak nismo ništa više od gomile stvaralaca koji jedva da imaju pojma šta rade.

Da smo mi programeri bolje razumeli šta radimo, da li bi rezultati predizbora u Ajovi za 2020. godinu bili spremni kada su obećani? Da li bi 346 ljudi poginulo u dve nesreće aviona 737 Max? Da li bi kompanija „Knight Capital Group“ izgubila 460 miliona dolara u roku od 45 minuta? Da li bi 89 ljudi izgubilo živote u nesrećama nenamernog ubrzanja „Tojotinih“ automobila?

Svakih pet godina broj programera u svetu se udvostruči. Ti programeri su vrlo malo učili o svom zanatu. Prikazuju im se alatke, daje im se nekoliko eksperimentalnih projekata koje treba da razviju, a zatim se „ubacuju“ u eksponencijalno rastuću radnu snagu kako bi odgovorili na eksponencijalno rastuću potražnju za sve više softvera. Svakog dana, „kula od karata“ koju nazivamo softver sve dublje se uvlači u našu infrastrukturu, naše institucije, naše vlade i naše živote. I svakim danom raste rizik od katastrofe.

O kojoj katastrofi govorim? To nije kolaps naše civilizacije, niti nagli raspad svih softverskih sistema odjednom. To je „kula od karata“ koja zbog kolapsa nije sastavljena od samih softverskih sistema. Umesto toga, krhki temelj poverenja javnosti je u opasnosti.

Previše je nesreća aviona 737 Max, „Tojotinih“ nenamernih ubrzanja, grešaka u kompaniji „Volkswagen California EPA“ ili grešaka na predizborima u Ajovi (previše novih slučajeva kvara softvera visokog profila ili zloupotreba) i nedostatak discipline, etike i standarda naći će se u fokusu intresovanja nepoverljive ili razbesnele javnosti.

A onda će uslediti propisi: propisi koje niko od nas ne bi trebalo da priželjkuje, koji će ugroziti našu sposobnost da slobodno istražujemo i širimo veštinu razvoja softvera i koji će postaviti ozbiljna ograničenja rastu tehnologije i ekonomije.

Svrha ove knjige nije da se zaustavi bezglava jurnjava ka sve većem usvajanju softvera, niti da se uspori brzina proizvodnje softvera. Takvi ciljevi bi bili uzaludan trud. Našem društvu je potreban softver i dobiće ga, bez obzira na sve. Pokušaj da se umanji ta potreba neće zaustaviti nadolazeću katastrofu koja se odnosi na poverenje javnosti.

Umesto toga, cilj nam je da se ovom knjigom programerima softvera i njihovim menadžerima nametne potreba za disciplinom i da oni nauče discipline, standarde i etiku koji su najefikasniji u povećanju njihove sposobnosti da proizvode softver koji je robustan, otporan na greške i efikasan. Tek promenom načina na koji mi programeri radimo, tako što će se povećati disciplina, etika i standardi, „kula od karata“ može biti ojačana i zaštićena od rušenja.

Struktura ove knjige

Ova knjiga je napisana u tri dela, u kojima su opisana tri nivoa: discipline, standardi i etika.

Discipline su najniži nivo. Ovaj deo knjige je pragmatičan, tehnički i upotrebljiv. Programeri svih vrsta će imati koristi od njegovog čitanja i razumevanja. Na stranicama ovog dela nalazi se nekoliko referenci za video-zapise. Ovi video-zapisi prikazuju ritam disciplina razvoja vođenog testovima i refaktorisanja u realnom vremenu. Na stranicama knjige pokušavamo da „uhvatimo“ i taj ritam, ali ništa ne služi toliko dobro za tu svrhu kao video-zapisi.

Standardi su srednji nivo. U ovom delu su opisana očekivanja koja svet ima od naše profesije. Bilo bi dobro da ga pročitaju menadžeri kako bi znali šta mogu da očekuju od profesionalnih programera.

Etika je na najvišem nivou. U ovom delu opisan je etički kontekst profesije programiranja u obliku zakletve ili skupa obećanja. Etika je prožeta velikom istorijskom i filozofskom diskusijom. Ovaj deo bi trebalo da pročitaju i programeri i menadžeri.

Napomena za menadžere

Ove stranice sadrže mnogo informacija koje će vam biti korisne. One takođe sadrže dosta tehničkih podataka koji vam verovatno nisu potrebni. Moj savet je da pročitate uvod svakog poglavlja i da prestanete da čitate kada sadržaj postane previše tehnički za vas. Zatim, pređite na sledeće poglavlje i počnite ponovo.

Pročitajte Deo II „Standardi“ i Deo III „Etika“.

Obavezno pročitajte uvode u svaku od pet disciplina.

Da biste registrovali knjigu, potrebno je da posetite sajt izdavača originala, Pearson:

https://www.informit.com/store/clean-craftsmanship-disciplines-standards-and-ethics-9780136915713

Nakon što ste pristupili strani origilanog izdanja knjige, kliknite:

Register your product

Sistem će vam postaviti pitanje.

Pitanje kopirajte i pošaljite na naš mobilni telefon 063635069 kao poruku.

Ukoliko poruku pošaljete radnim danima od 10-15 časova, odgovorićemo vam najbrže što možemo.

Ukoliko poruku pošaljete u subotu ili nedelju, najkasnije ćemo vam odgovoriti u ponedeljak.

Zahvalnice

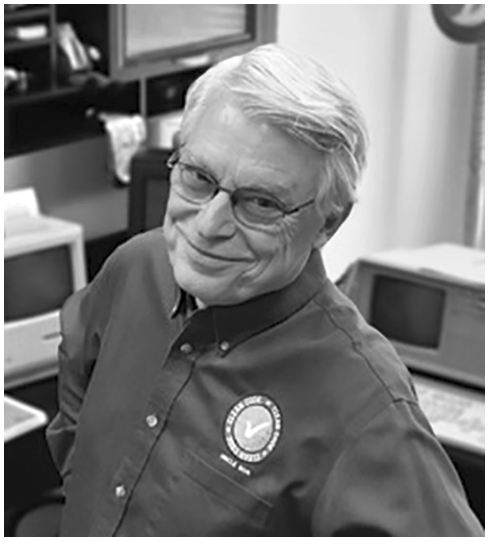
Hvala mojim neustrašivim recenzentima: Damonu Poolu, Ericu Crichlowu, Heather Kanser, Timu Ottingeru, Jeffu Langru i Stacii Viscardi. Spasili su me od mnogih nesigurnih koraka.

Hvala i Juliei Phifer, Chrisu Zahnu, Menka Mehtau, Carol Lallier i svima onima u izdavačkoj kući „Pearson“ koji se neumorno trude da ove knjige budu dobre.

Kao i uvek, hvala mojoj kreativnoj i talentovanoj ilustratorki Jennifer Kohnke. Njene slike me uvek nasmeju.

I, naravno, hvala mojoj ljupkoj supruzi i našoj divnoj porodici.

O autoru



Robert C. Martin (Ujka Bob) je svoj prvi red koda napisao kao dvanaestogodišnjak 1964. godine. Radi kao programer od 1970. godine. Suosnivač je sajta cleancoders.com, nudi online video-obuku za programere i osnivač je kompanije „Uncle Bob Consulting LLC“, koja pruža konsultacije o softverima, obuku i usluge razvoja veština velikim korporacijama širom sveta. Radio je, kao vrhunski stručnjak, u čikaškoj konsultantskoj firmi za softver „8th Light, Inc“.

Martin je objavio desetine članaka u raznim stručnim časopisima i redovan je predavač na mnogobrojnim međunarodnim konferencijama i sajmovima. Takođe je tvorac priznate obrazovne video-serije na sajtu cleancoders.com.

Martin je autor i urednik mnogih knjiga, uključujući sledeće:

Designing Object-Oriented C++ Applications Usingthe Booch Method

Patterns Languages of Program Design 3

More C++ Gems

Extreme Programming in Practice

Agile Software Development: Principles, Patterns, and Practices

UML for Java Programmers

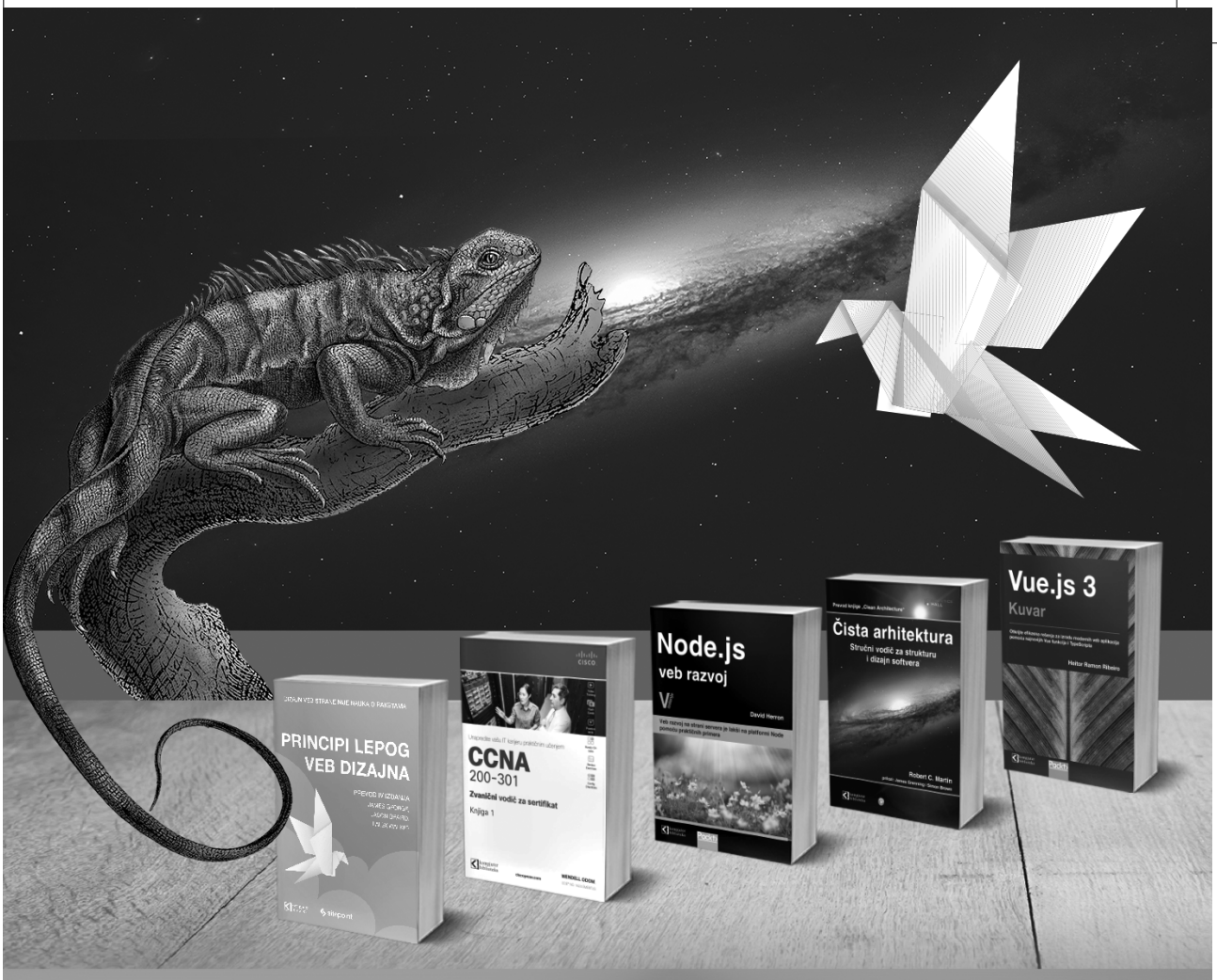
„Jasan kod“"

„TheCleanCoder“

„Čista arhitektura“

„Čisto agilno“

Lider u industriji razvoja softvera, Martin je tri godine bio glavni urednik časopisa *C++ Report*, a bio je i prvi predsednik Agilne alijanse.



Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popustai učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

1 Majstorstvo

San o letenju je skoro sigurno star koliko i čovečanstvo. Starogrčki mit koji opisuje let Dedala i Ikara datira iz približno 1550. godine pre nove ere. U milenijumima koji su usledili veliki broj hrabrih, iako budalastih duša vezivao je nezgrapne naprave za svoja tela i skakao sa litica i kula u propast, u potrazi za ostvarenjem sna o letenju čoveka.

Stvari su počele da se menjaju pre oko pet stotina godina, kada je Leonardo da Vinči nacrtao skice mašina koje su, iako zaista neupotrebljive za letenje, bile smislene. On je shvatio da je let moguć, jer otpor vazduha deluje u oba smera. Otpor izazvan guranjem vazduha nadole daje istu silu potiska.To je mehanizam koji se koristi za savremene avione.

Ideje Leonarda da Vinčija bile su zaboravljene sve do sredine 18. veka, kada je počelo skoro mahnito istraživanje mogućnosti leta. Osamnaesti i devetnaesti vek bili su vreme intenzivnih vazduhoplovnih istraživanja i eksperimenata. Prototipovi bez motora su izrađivani, isprobavani, odbacivani i poboljšavani. Nauka o vazduhoplovstvu počela je da poprima svoj oblik. Sile podizanja, vuče, potiska i gravitacije su identifikovane i shvaćene. Neke hrabre duše su pokušale.

I, neki su se srušili i poginuli.

Poslednjih godina osamnaestog veka i tokom prve polovine veka koji je usledio George Cayley, otac moderne aerodinamike, izrađivao je eksperimentalne platforme, prototipove i modele u punoj veličini, što je kulminiralo prvim letom jedrilice sa ljudskom posadom.

I, neki su se ipak srušili i poginuli.

Zatim je došlo doba pare i usledila je mogućnost motorizovanog leta sa posadom. Izrađeno je na desetine prototipova i eksperimenata. Clement Ader je 1890. godine leteo 50 metara u letelici težoj od vazduha.

I, neki su se ipak srušili i poginuli.

Ali, motor sa unutrašnjim sagorevanjem je zaista promenio „igru“. Po svoj prilici, prvi motorizovan i kontrolisan let sa posadom 1901. godine izveo je Gustave Whitehead. Međutim, braća Wright su 17. decembra 1903. godine u Kil Devil Hilsu, u Severnoj Karolini, izveli prvi istinski održiv, motorizovan i kontrolisan let sa posadom u letelici koja je teža od vazduha.

I, neki su se ipak srušili i poginuli.

Ali se svet promenio preko noći. Jedanaest godina kasnije, 1914. godine, dvokrilni avioni su se borili u vazduhu iznad Evrope.

I mada su se mnogi pojedinci u letelicama srušili i poginuli pod neprijateljskom vatrom, približan broj se srušio i poginuo samo učeći da leti. Principi leta su možda bili savladani, ali *tehnika* leta je jedva shvaćena.

Dve decenije kasnije, strašni lovci i bombarderi u Drugom svetskom ratu pustošili su Francusku i Nemačku. Leteli su na ekstremnim visinama. Imali su razornu moć.

Tokom rata izgubljeno je 65.000 američkih aviona. Ali je samo 23.000 izgubljeno u borbi. Piloti su leteli i ginuli u borbi, a češće su leteli i ginuli kada niko nije pucao. Još uvek se nije znalo *kako* da se leti.

Još jednu deceniju kasnije mogli smo da vidimo letelice na mlazni pogon, probijanje zvučnih barijera i eksploziju komercijalnih avio-kompanija i civilnih avio-letova. Bio je to početak doba mlaznih aviona, kada su imućni ljudi (tzv. džet set) mogli za nekoliko sati da se prebace iz grada u grad i iz države u državu.

A mlazni avioni su se razdirali na komadiće i rušili u zastrašujućem broju. Bilo je nepoznanica o pravljenju i letu aviona.

To nas dovodi do pedesetih godina prošlog veka. Boeing 707 je krajem te decenije prevozio putnike od jedne do druge tačke širom sveta. Posle još dve decenije pojavio se prvi džambo mlaznjak širokog tela, 747.

Vazduhoplovstvo i vazdušni saobraćaj postali su najbezbedniji i najefikasniji način putovanja u istoriji. Trajalo je dugo i koštalo je mnogo života, ali je konačno naučeno kako da se bezbedno grade avioni i kako da se leti njima.[1]

Chesley Sullenberger rođen je 1951. godine u Denisonu, u Teksasu. Naučio je da leti kada je imao 16 godina i na kraju je leteo avionom F4 Phantoms za ratno vazduhoplovstvo. Postao je pilot US Airwaysa 1980. godine.

Neposredno nakon poletanja sa aerodroma LaGuardie, 15. januara 2009. godine, njegov Airbus A320 sa 155 osoba naleteo je na jato gusaka i izgubio oba mlazna motora. Kapetan Sullenberger, sa iskustvom od više od dvadeset hiljada sati u vazduhu, upravljao je oštećenim avionom do sletanja na vodu - u reku Hadson i, zahvaljujući savršenoj nepokolebljivoj veštini, spasio svih 155 saputnika. Kapetan Sullenberger se istakao u svom zanatu. Kapetan Sullenberger je bio stručnjak.

San o brzom, pouzdanom računanju i upravljanju podacima star je skoro koliko i čovečanstvo. Brojanje na prste, štapiće i perle datira od pre nekoliko hiljada godina. Ljudi su izrađivali i koristili računaljke pre više od četiri milenijuma. Mehanički uređaji korišćeni su za predviđanje kretanja zvezda i planeta pre oko dve hiljade godina. Logaritmari su izmišljeni pre približno četiri veka.

Početkom 19. veka Charles Babbage je počeo da gradi računarske mašine sa napajanjem. To su bili pravi digitalni računari sa memorijom i aritmetičkom obradom. Međutim, bilo ih je teško izraditi korišćenjem tadašnje tehnologije obrade metala i mada je Babbage izradio nekoliko prototipova, oni nisu bili komercijalno uspešni.

Sredinom 19. veka Babbage je pokušao da izradi mnogo moćniju mašinu koja bi radila na parni pogon i bila sposobna da izvršava prave programe. Nazvao ju je *analitička mašina*.

Kći lorda Bajron Ada, grofica od Lavlejsa, prevela je beleške sa predavanja koje je održao Babbage i uvidela činjenicu koja očigledno nikome u to vreme nije pala na pamet: *brojevi u računaru uopšte ne moraju da predstavljaju brojeve, već stvari u realnom svetu*. Zbog toga, tu groficu često nazivaju prvim pravim programerom na svetu.

1 Bez obzira na 737 Max.

Problemi precizne obrade metala nastavili su da frustriraju Babbagea, pa je na kraju njegov projekat doživeo neuspeh. Nikakav dalji napredak nije postignut na digitalnim računarima tokom ostatka 19. i početkom 20. veka. Za to vreme, međutim, mehanički *analogni* računari dostigli su svoj vrhunac.

Alan Turing je 1936. godine pokazao da ne postoji opšti način da se dokaže da zadata Diofantska jednačina[2] ima rešenja. Ovaj dokaz je konstruisao tako što je zamišljao jednostavan, beskonačan, digitalni računar, a zatim dokazao da postoje brojevi koje ovaj računar ne može da izračuna. Zahvaljujći ovom dokazu, izumeo je mašine konačnih stanja, mašinski jezik, simbolički jezik, makroe i osnovne potprograme. On je izumeo ono što danas nazivamo softver.

Skoro u isto vreme, Alonzo Church je konstruisao potpuno drugačiji dokaz za isti problem i kao posledicu toga razvio je lambda račun, tj. osnovni koncept funkcionalnog programiranja.

Konrad Zuse je 1941. godine izradio prvi elektromehanički programabilan digitalni računar Z3, koji se sastojao od više od 2.000 releja i pokretan je pri radnom taktu od 5 do 10 Hz. Mašina je koristila binarnu aritmetiku organizovanu u 22-bitne reči.

Tokom Drugog svetskog rata Turing je angažovan da pomogne „genijalcima“ u Blečli Parku u dešifrovanju kodova nemačke mašine „Enigma“.

Mašina „Enigma“ bila je jednostavan digitalni računar koji je nasumično birao znakove tekstualnih poruka koje su obično emitovane pomoću radio-telegrafa. Turing je pomogao u izradi elektromehaničkog digitalnog pretraživača za pronalaženje ključeva tih kodova.

Posle rata, Turing je odigrao ključnu ulogu u izradi i programiranju jednog od prvih svetskih elektronskih računara sa vakuumskim cevima – automatskih računarskih mašina (Automatic Computing Engine) ili ACE-a. Originalan prototip koristio je 1.000 vakuumskih cevi i manipulisao binarnim brojevima brzinom od milion bitova u sekundi.

2 Jednačina celih brojeva.

Nakon što je napisao neke programe za ovu mašinu i istražio njene mogućnosti, Turing je 1974. godine održao predavanje na kome je dao ove proročanske izjave:

> *Biće nam potreban veliki broj matematičara koji mogu da pretvore probleme u formu za računanje.*
>
> *Jedna od poteškoća biće nam održavanje odgovarajuće discipline da ne bismo izgubili trag o tome šta radimo.*

I svet se promenio preko noći.

U roku od nekoliko godina razvijena je osnovna memorija. Mogućnost da stotine hiljada, ako ne i milioni delova memorije budu dostupni u roku od nekoliko mikrosekundi, postala je stvarnost. U isto vreme je masovna proizvodnja vakuumskih cevi učinila računare jeftinijim i pouzdanijim. Ograničena masovna proizvodnja postala je stvarnost. Do 1960. godine kompanija IBM je prodala 140 računara modela 70x. To su bile ogromne mašine sa vakuumskim cevima, vredne milione dolara.

Turing je svoju mašinu programirao u binarnom formatu, ali svi su shvatili da to nije praktično. Grace Hopper je 1949. godine osmislila reč *kompajler* i do 1952. godine kreirala je prvi A-0. Krajem 1953. godine John Bachus je podneo prvu FORTRAN specifikaciju. ALGOL i LISP su usledili 1958. godine.

Prvi primenljivi tranzistor kreirali su John Bardeen, Walter Brattain i William Shockley 1947. godine. Tranzistori su uvedeni u računare 1953. godine. Zamena vakuumskih cevi tranzistorima potpuno je promenila „igru". Računari su postali manji, brži, jeftiniji i mnogo pouzdaniji.

Do 1965. godine kompanija IBM je proizvela 10.000 računara modela 1401. Iznajmljivani su za 2.500 USD mesečno. Ovo je bilo u dometu srednjih preduzeća. Potražnja za programerima počela je da raste.

Ko je programirao sve ove mašine? Nije bilo univerzitetskih kurseva. Niko nije išao u školu za programiranje 1965. godine. Programeri su imali svoja preduzeća. Bili su to zreli ljudi koji su neko vreme radili u svojim preduzećima. Bili su u tridesetim, četrdesetim i pedesetim godinama.

Do 1966. godine kompanija IBM je proizvodila 1.000 računara modela 360 svakog meseca. Ove mašine su imale veličinu memorije koja je dostigla i nadmašila 64 kB. Mogle su da izvrše stotine hiljada instrukcija u sekundi.

Iste godine, radeći na Univacu 1107 u Norveškom računarskom centru, Ole-Johan Dahl i Kristen Nygard izmislili su Simula 67, proširenje ALGOL-a. Bio je to prvi objektno-orijentisan jezik.

Od predavanja Alana Turinga prošle su samo dve decenije!

U martu 1968. godine Edsger Dijkstra je napisao svoje čuveno pismo *Communications of the ACM (CACM)*. Urednik je tom pismu dao naslov „Go To Statement Considered Harmful". Rođeno je strukturirano programiranje.

U kompaniji „Bell Labs" u Nju Džerziju 1972. godine su Ken Thompson i Dennis Ritchie radili na dva projekta. Tražili su više vremena za PDP 7 od drugog projektnog tima i izmislili su UNIX i C.

Sada se tempo vrtoglavo ubrzao. Navešću vam nekoliko ključnih datuma. Za svaki od njih zapitajte se koliko računara ima u svetu? Koliko programera ima u svetu? A odakle su došli ti programeri?

1970. – Kompanija Digital Equipment Corporation je proizvela 50.000 računara PDP-8 od 1965. godine.

1970. – Vinston Roice je napisao rad o softverskom procesu „waterfall", koji se zvao „Managing the Development of Large Software Systems".

1971. – Kompanija „Intel" je predstavila mikroračunar sa jednim čipom 4004.

1974. – Kompanija "Intel" je predstavila mikroračunar sa jednim čipom 8080.

1977. – Kompanija „Apple" je predstavila Apple II.

1979. – Kompanija „Motorola" je predstavila 68000, 16-bitni mikroračunar sa jednim čipom.

1980. – Bjarne Stroustrup je izumeo *C sa klasama* (pretprocesor koji čini da C izgleda kao Simula).

1980. – Alan Kay je izumeo Smalltalk.

1981. – Kompanija IBM je predstavila IBM PC.

1983. – Kompanija „Apple" je predstavila 128K Macintosh.

1983. – Stroustrup je preimenovan u C sa klasama u jeziku C++.

1985. – Ministarstvo odbrane SAD usvojilo je „waterfall" kao zvaničan softverski proces (DOD-STD-2167A).

1986. – Stroustrup je objavio knjigu „The C++ Programming Language" (Addison-Wesley).

1991. – Grady Booch je objavio knjigu „Object-Oriented Design with Applications" (Benjamin/Cummings).

1991. –James Gosling je izumeo jezik Java (tada se zvao *Oak*).

1991. – Guido Van Rossum je predstavio Python.

1995.– Erich Gamma, Richard Helm, John Vlissides i Ralph Johnson napisali su knjigu „DesignPatterns: Elements of Reusable Object-Oriented Software" (Addison-Wesley).

1995. – Yukihiro Matsumoto je predstavio Ruby.

1995. – Brendan Eich je kreirao JavaScript.

1996. – Kompanija „SunMicrosystems" je predstavila Javu.

1999. – Kompanija „Microsoft" je izumela jezik C#/. NET (tada se zvao *Cool*).

2000.–Y2K! Milenijumska greška (The Millennium Bug).

2001. – Napisan je Agilni manifest.

Između 1970. i 2000. godine brzine radnog takta računara porasle su za tri puta. Gustina se povećana za četiri puta. Prostor na disku povećan je za šest ili sedam puta. Kapacitet RAM-a je povećan za šest ili sedam puta. Troškovi su pali sa jednog dolara po bitu na jedan dolar po gigabitu. Teško je zamisliti promenu hardvera, ali samo sabiranje svih povećanja koje sam pomenuo dovodi nas do povećanja sposobnosti za tridesetak puta.

I sve to za nešto više od 50 godina od predavanja Alana Turinga.

Koliko programera sada ima? Koliko linija koda je napisano? Koliko je dobar sav taj kod?

Uporedite ovu vremensku liniju sa vremenskom linijom vazduhoplovstva. Da li uviđate sličnost? Da li uočavate postepeno bogaćenje teorije, žurbu i neuspeh entuzijasta, postepeno povećanje znanja? Decenijama ne znamo šta radimo?

S obzirom da samo postojanje našeg društva zavisi od naših veština, pitamo se da li imamo kapetane Sullenberger koji su potrebni? Da li smo pripremili programere koji razumeju svoj posao isto koliko i današnji piloti? Da li imamo stručnjake koji će nam sigurno biti potrebni?

Majstorstvo je znati kako nešto učiniti dobro i rezultat je dobrog tutorstva i dugogodišnjeg iskustva. Sve donedavno, softverska industrija je imala premalo i od jednog i od drugog. Programeri su obično na programiranje gledali kao na „odskočnu dasku" za menadžment, pa se nisu dugo zadržavali u toj profesiji. To znači da je bilo malo programera koji su stekli dovoljno iskustva da druge poduče zanatu. Da situacija bude gora, broj novih programera koji ulaze „na teren" udvostručuje se, približno, na svakih pet godina, pa je broj iskusnih programera premali.

Većina programera nikada nije naučila discipline, standarde i etiku koji bi trebalo da definišu njihov zanat. Tokom relativno kratke karijere pisanja koda oni ostaju nestručne novajlije. A to, naravno, znači da je veliki deo koda koji su proizveli neiskusni programeri nekvalitetan, loše struktuiran, nebezbedan, sa greškama i u generalnom neredu.

U ovoj knjizi opisujem standarde, discipline i etiku koje bi svaki programer trebalo da zna i da sledi kako bi postepeno stekao znanje i veštinu koji su potrebni u njegovom zanatu.

Discipline

Šta je disciplina? Disciplina je skup pravila koji se sastoji od dva dela: suštinski deo i proizvoljan deo. Suštinski deo je ono što disciplini daje moć; to je razlog njenog postojanja. Proizvoljan deo je ono što disciplini daje njen oblik i sadržaj. Disciplina ne može da postoji bez proizvoljnog dela.

Na primer, hirurzi peru ruke pre operacije, i to na veoma specifičan način. Hirurg ne pere ruke sapunom pod mlazom tekuće vode, kao što bismo vi i ja mogli da uradimo. Umesto toga, on sledi ritualizovanu disciplinu pranja ruku. Jedna takva rutina koju sam video je, delimično, sledeća:

- koriste odgovarajući sapun
- koriste odgovarajuću četku
- za svaki prst primenjuju
  - 10 poteza preko vrha
  - 10 poteza preko leve strane
  - 10 poteza preko donje strane
  - 10 poteza preko desne strane
  - 10 poteza preko noktiju
- i tako dalje

Suštinski deo discipline treba da bude očigledan. Ruke hirurga moraju da budu veoma čiste. Međutim, da li ste primetili proizvoljan deo? Zašto 10 poteza, umesto osam ili 12? Zašto se prst deli na pet delova? Zašto ne na tri ili sedam delova?

To je sve proizvoljno. Ne postoji pravi razlog za te brojke, osim što se smatraju dovoljnim.

U ovoj knjizi proučavamo pet disciplina izrade softvera. Neke od ovih disciplina stare su pet, a neke samo dve decenije. A sve su pokazale svoju korisnost tokom tih decenija. Bez njih, sam pojam „veština izrade softvera“ bio bi praktično nezamisliv.

Svaka od ovih disciplina ima svoje važne i proizvoljne elemente. Dok čitate, možda ćete primetiti da vam um prigovara jednoj ili više disciplina. Ako se to dogodi, razlučite da li se prigovor odnosi na suštinske elemente ili samo na proizvoljne elemente disciplina. Ne dozvolite da vas proizvoljni elementi pogrešno usmere. Fokusirajte se na suštinske elemente. Kada usvojite suštinu svake discipline, proizvoljni oblik će verovatno biti manje važan.

Na primer, Ignaz Semmelweis je 1861. godine objavio svoja otkrića o primeni discipline pranja ruku lekara. Rezultati njegovog istraživanja bili su zapanjujući. Uspeo je da pokaže da, kada su lekari temeljno oprali ruke izbeljivačem na bazi hlora pre pregleda trudnica, stopa smrtnosti tih žena od naknadne sepse pala je sa jedne od 10 na skoro nulu.

Međutim, tadašnji lekari nisu odvojili važno od proizvoljnog kada su razmatrali Semmelweisovu predloženu disciplinu. Izbeljivač na bazi hlora je bio proizvoljan deo. Pranje je bilo važno. Odbijala ih je neprijatnost pranja izbeljivačem na bazi hlora, pa su odbacili dokaz o važnosti pranja ruku.

Prošlo je mnogo decenija pre nego što su lekari počeli da peru ruke.

Ekstremno programiranje

Winston Royce je 1970. godine objavio rad koji je proces razvoja „waterfall“ uveo u mejnstrim. Bilo je potrebno skoro 30 godina da se ta greška ispravi.

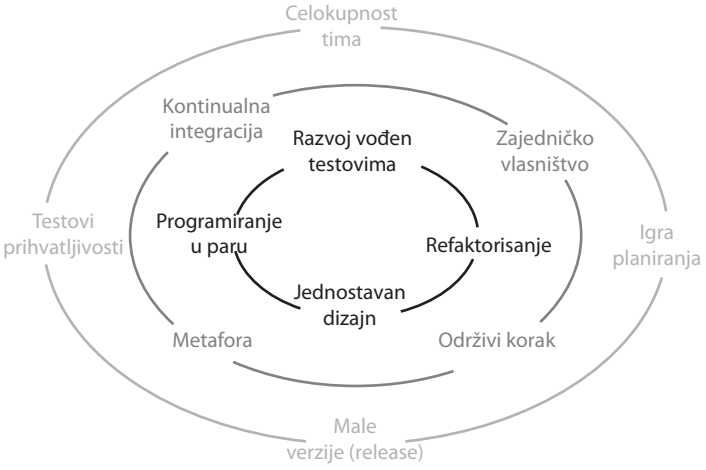
Do 1995. godine stručnjaci za softver su počeli da razmatraju drugačiji, dodatni pristup. Predstavljeni su procesi kao što su Scrum, razvoj vođen funkcijama (FDD - Feature-Driven Development), metod razvoja dinamičkih sistema (DSDM - Dynamic Systems Development method) i Crystal metodologije. Međutim, malo se čega promenilo u softverskoj industriji uopšte.

Zatim je Kent Beck 1999. godine objavio knjigu „Extreme Programming Explained“ (Addison-Wesley). Ekstremno programiranje (XP) je izgrađeno na idejama iz tih prethodnih procesa, ali je dodato nešto novo. XP je dodao *inženjerske prakse*.

Entuzijazam za XP je eksponencijalno rastao između 1999. i 2001. godine. On je iznedrio i pokrenuo agilnu revoluciju. Do danas XP ostaje najbolje definisan i najkompletniji od svih agilnih metoda. Inženjerske prakse u svojoj osnovi su u fokusu ovog dela o disciplinama.

Krug života

Na slici I.1 vidite *Krug života* Rona Jeffriesa, u kojem su prikazane prakse XP-a. U ovoj knjizi razmatramo četiri discipline u centru i jednu disciplinu krajnje levo.



Slika I.1 Krug života: Prakse XP-a

Četiri discipline u centru su inženjerske prakse XP-a: razvoj vođen testovima (TDD), refaktorisanje, jednostavan dizajn i programiranje u paru (koje ćemo nazvati *kolaborativno programiranje*). Praksa na krajnjoj levoj strani, koja se zove testovi prihvatljivosti, je najviše fokusirana na tehniku i inženjerstvo, među poslovnim praksama XP-a.

Ovih pet praksi su među osnovnim disciplinama majstorstva izrade softvera.

Razvoj vođen testovima

TDD je glavna disciplina. Bez njega, druge discipline su ili nemoguće ili nemoćne. Zbog toga, dva predstojeća odeljka u kojima je opisan TDD predstavljaju skoro polovinu ove knjige i intenzivno su tehnička. Ova organizacija može izgledati neuravnoteženo. Zaista, i meni se tako čini i mučio sam se šta da uradim. Moj zaključak je, međutim, da je neravnoteža reakcija na odgovarajuću neravnotežu unutar naše industrije. Samo mali broj programera dobro poznaje ovu disciplinu.

TDD je disciplina koja upravlja načinom na koji programer radi iz sekunde u sekundu. To nije ni prva, ni naknadna disciplina. TDD je u toku i direktan. Ne postoji način da se uradi delimičan TDD; to je disciplina po principu sve ili ništa.

Suština TDD discipline je veoma jednostavna. Mali ciklusi i testovi su na prvom mestu. Testovi su na prvom mestu u svemu. Prvi se pišu. Prvi se čiste. U svim aktivnostima testovi su na prvom mestu. I sve aktivnosti su podeljene na najmanje cikluse.

Vreme ciklusa se meri u sekundama, a ne u minutima. Meri se znakovima (characters), a ne linijama. Povratna sprega se zatvara skoro doslovno čim se otvori.

Cilj da se pomoću TDD-a kreira paket testova kome ćete verovati. Ako je paket testova uspešan, trebalo bi da bezbedno primenite (deploy) kod.

Od svih disciplina TDD je najteži i najkompleksniji. Najteži je, jer dominira svime. To je prva i poslednja stvar o kojoj razmišljate. To je ograničenje koje se nalazi u svemu što radite. To je vladar koji održava stabilan tempo, bez obzira na pritisak i stresove u okruženju.

TDD je složen (complex), jer je kod složen. Za svaki oblik ili formu koda postoji odgovarajući oblik ili forma TDD-a. TDD je složen, jer testovi moraju da budu dizajnirani tako da odgovaraju kodu, a da ne budu spojeni sa njim, kao i da pokrivaju skoro sve, a da se, ipak, izvršavaju u roku od nekoliko sekundi. TDD je podrobna i složena veština koja se veoma teško osvaja, ali neizmerno nagrađuje.

Refaktorisanje

Refaktorisanje je disciplina koja omogućava da napišemo čist kod. Ono je teško, ako ne i nemoguće, bez TDD-a.[1] Stoga je pisanje čistog koda jednako teško ili nemoguće bez TDD-a.

Refaktorisanje je disciplina kojom pretvaramo loše strukturirani kod u kod sa boljom strukturom - *bez uticaja na ponašanje*. Taj poslednji deo je kritičan. Kada garantujemo da se ne utiče na ponašanje koda, garantujemo da će poboljšanja strukture biti *bezbedna*.

Razlog zbog kojeg ne čistimo kod (razlog zbog kojeg softverski sistemi trule tokom vremena) je činjenica da se bojimo da će čišćenje koda narušiti ponašanje. Međutim, ako imamo bezbedan način na koji ćemo očistiti kod, onda *ćemo* očistiti kod i naši sistemi neće istruliti.

Na osnovu čega garantujemo da naša poboljšanja neće uticati na ponašanje? Imamo testove iz TDD-a.

Refaktorisanje je takođe složena disciplina, jer postoji mnogo načina za kreiranje loše strukturiranog koda. Postoji mnogo strategija za čišćenje tog koda. Štaviše, svaka od ovih strategija mora da se glatko i istovremeno uklopi u TDD ciklus prvo-test (test-first). Ove dve discipline su toliko čvrsto isprepletene da su praktično neodvojive. Skoro je nemoguće refaktorisati bez TDD-a i praktično je nemoguće primeniti TDD bez primene refaktorisanja.

1 Možda postoje i druge discipline koje bi mogle da podrže refaktorisanje kao i TDD. Kent Beckova komanda test && commit || revert je mogućnost. Međutim, u vreme pisanja ovog teksta, nije usvojena u velikoj meri i ostaje više kao akademski kuriozitet.

Jednostavan dizajn

Život na Zemlji mogao bi se opisati slojevima. Na vrhu je ekologija, proučavanje sistema živih bića. Ispod toga je fiziologija, proučavanje unutrašnjih mehanizama života. Sledeći donji sloj može biti mikrobiologija, proučavanje ćelija, nukleinskih kiselina, proteina i drugih makromolekularnih sistema. Taj donji sloj je opisan hemijom, koju opisuje kvantna mehanika.

Kada proširimo tu analogiju na programiranje, ako je TDD kvantna mehanika programiranja, onda je refaktorisanje hemija, a jednostavan dizajn mikrobiologija. Ako nastavimo tu analogiju, SOLID principi, objektno-orijentisan dizajn i funkcionalno programiranje su fiziologija, a arhitektura je ekologija programiranja.

Jednostavan dizajn je skoro nemoguć bez refaktorisanja. Zaista, to je krajnji cilj refaktorisanja, a refaktorisanje je jedino praktično sredstvo za postizanje tog cilja. Taj cilj je proizvodnja jednostavnih atomičnih delova dizajna koji se dobro uklapaju u veće strukture programa, sistema i aplikacija.

Jednostavan dizajn nije složena disciplina. Vođen je pomoću četiri veoma jednostavna pravila. Međutim, za razliku od TDD-a i refaktorisanja, jednostavan dizajn je neprecizna disciplina. Oslanja se na rasuđivanje i iskustvo. Kada je dobro obavljen, to je prvi pokazatelj koji odvaja šegrta koji poznaje pravila od stručnjaka koji razume principe. To je početak onoga što je Michael Feathers nazvao *smisao za dizajn*.

Kolaborativno programiranje

Kolaborativno programiranje je i disciplina i umetnost zajedničkog rada u softverskom timu. Uključuje poddiscipline, kao što su programiranje u paru, programiranje u grupi (mob), pregled koda (code review) i „mozganje“ (brainstorming). Kolaborativno programiranje uključuje sve u timu, podjednako i programere i one koji nisu programeri. To je primarno sredstvo pomoću kojeg delimo znanje, obezbeđujemo konciznost i pretvaramo tim u funkcionalnu celinu.

Od svih disciplina, kolaborativno programiranje je najmanje tehničko i najmanje propisano. Ipak, možda je ono i najvažnija od pet disciplina, jer je sastavljanje efikasnog tima i retko i dragoceno.

Testovi prihvatljivosti

Testiranje prihvatljivosti je disciplina koja povezuje tim za razvoj softvera sa poslovanjem (business). Poslovna svrha je specifikacija željenih ponašanja sistema. Ta ponašanja su kodirana u testove. Ako su testovi uspešni, sistem se ponaša onako kako je specificirano.

Testovi moraju da budu takvi da ih mogu čitati i pisati predstavnici poslovanja. Upravo pisanjem i čitanjem ovih testova, i gledanjem kako uspešno prolaze, poslovanje zna šta softver radi, i zna da radi to što je poslovanju potrebno da on radi.