Stručný obsah

Část I
Mechanismus změn

Část II
Modifikace softwaru

Část III
Techniky rušení závislostí

Obsah

Úvodní slovo Roberta C. Martina

„...tehdy to začalo...“

V předmluvě k této knize používá Michael Feathers uvedený slovní obrat, aby popsal začátek své vášně pro software.

„...tehdy to začalo...“

Znáte ten pocit? Můžete ukázat na jediný okamžik svého života a říci: „...tehdy to začalo...“? Existovala jediná událost, která změnila běh vašeho života a eventuálně vás přivedla k tomu, že jste vzali tuto knihu do rukou a začali číst tuto předmluvu?

Když se to stalo mně, byl jsem v šestém ročníku. Zajímal jsem se o vědu a vesmír a vše technické. Moje matka našla v katalogu nějaký plastický počítač a objednala mi jej. Nazýval se *Digi-Comp*. Po čtyřiceti letech zaujímá tento malý plastický počítač čestné místo v mé polici na knihy. Byl to katalyzátor, který zažehl moji trvalou vášeň pro software. Dal mi první tušení o tom, jaké uspokojení přináší psaní programů, které umí řešit lidem jejich problémy. Byly to pouhé tři klopné obvody a šest hradel realizujících logickou funkci „and“ – a to stačilo, splnilo to svou funkci. Tehdy... to pro mě... začalo...

Ale radost, kterou jsem pociťoval, byla brzy mírněna zjištěním, že softwarové systémy se téměř vždy propadají do chaosu. Co začíná v programátorově mysli jako jasný a krystalický návrh, často shnije jako kus špatného masa. Úhledný a malý systém, který jsme sestavili minulý rok, se dalším rokem změnil v příšernou bažinu spletitých funkcí a proměnných.

Proč se to stává? Proč systémy degenerují? Proč nemohou zůstat čisté?

Někdy obviňujeme zákazníky. Někdy je obviňujeme z toho, že mění požadavky. Uklidňujeme sami sebe vírou, že kdyby zákazníci byli spokojeni s tím, co řekli, že potřebují, návrh by byl dobrý. Je to chyba zákazníka, že změnil požadavky.

Dobrá, zde je nový záblesk: *změna požadavků.* Návrhy, které neumožňují změny požadavků, jsou hned od počátku špatné. Cílem každého kompetentního vývojáře softwaru je vytvářet takové návrhy, které umožní změny. To se zdá být nepoddajný a těžko řešitelný problém. Ve skutečnosti tak těžký, že téměř každý systém, který kdy byl vytvořen, trpí pomalou a oslabující degenerací. Ta je natolik všudypřítomná, že jsme přišli se zvláštním názvem degenerovaných programů. Nazýváme je *zděděný kód*.

Zděděný kód. Fráze, která způsobuje v srdcích programátorů znechucení. Vyvolává představu pachtění temnou bažinou spletitého podrostu dole s pijavicemi a s bodavým hmyzem nad nimi. Vyvolává zápach temnoty, bahna, stagnace a odpadu. Ačkoliv vaše první radost z programování může být intenzivní, utrpení z práce se zděděným kódem často postačuje, aby tento žár vyprchal.

Mnoho z nás již zkusilo nalézt způsob, jak *zabránit* tomu, aby se z kódu stal „zděděný kód“. Napsali jsme knihy o principech, schématech a postupech, které mohou pomáhat programátorům, aby své systémy udržovali čisté. Ale Michael Feathers má náhled, který mnozí z nás postrádají. Prevence je nedostatečná. I ty nejdisciplinovanější týmy vývojářů, ovládající nejlepší pravidla, používající nejlepší schémata a držící se těch nejlepších praktik, vytvoří

čas od času něco zmateného. Degenerace se stále hromadí. Nestačí se pokoušet jí zabraňovat – musíte být schopni obrátit směr.

O tom tato kniha je. Je o změně směru degenerace. Je o tom, jak vzít komplikovaný, neprůhledný a spletitý systém a pomalu, postupně, krok za krokem jej změnit na jednoduchý, úhledný, strukturovaný a dobře navržený systém. Je to o změně entropie.

Varuji vás před přílišným nadšením. Zpětný chod není nic jednoduchého a rychlého. Techniky, schémata a nástroje, které v této knize Michael Feathers předvádí, jsou efektivní, ale vyžadují práci, čas, výdrž a *zájem*. Tato kniha není kouzelnou hůlkou. Neřekne vám, jak z vašeho systému eliminovat všechny nahromaděné prvky degenerace přes noc. Spíše popisuje sadu závazných pravidel, pojetí a postojů, kterých byste se měli držet po zbytek své kariéry a které *vám budou pomáhat měnit systémy, které podlehly postupné degeneraci, na systémy, které se postupně zkvalitňují.*

29. června 2004

Robert C. Martin
Spolutvůrce agilního programování,
autor mnoha knih o softwarovém
inženýrství, zakladatel a generální
ředitel firmy Object Mentor Inc.

Předmluva

Vzpomínáte si na první program, který jste napsali? Já si vzpomínám na ten svůj. Byl to malý grafický program, který jsem napsal na jednom z prvních osobních počítačů. Začal jsem s programováním později než většina z mých přátel. Samozřejmě, že jsem viděl počítače, když jsem byl ještě dítě. Vzpomínám si, jak jsem byl opravdu ohromen minipočítačem, který jsem viděl v kanceláři, ale léta jsem neměl šanci si k němu ani sednout. Později, když jsem byl teenager, si někteří moji přátelé koupili první počítače TRS-80. Zajímalo mne to, ale byl jsem také trochu znepokojený. Věděl jsem, že když si začnu s počítači hrát, pohltí mne to. Vypadalo to příliš super. Nevím, proč jsem to o sobě věděl tak dobře, ale držel jsem se zpátky. Později na univerzitě měl můj spolubydlící počítač a já jsem si koupil kompilátor jazyka C, takže jsem se mohl učit programování. Tehdy to začalo. Byl jsem vzhůru noc co noc a zkoušel jsem různé věci a podrobně studoval zdrojové kódy na editoru emacs, který byl součástí kompilátoru. Bylo to jako droga, byla to výzva a já jsem tomu vášnivě propadl.

Předpokládám, že máte podobnou zkušenost – pouhou obyčejnou radost z provádění různých kousků na počítači. Téměř každý programátor, kterého se ptám, ji má. Tato radost je součástí něčeho, co nás do této práce dostalo, ale kde je režimu denodenním životě?

Před několika lety jsem jednoho večera po práci telefonoval svému příteli Eriku Maedeovi. Věděl jsem, že Erik právě začal konzultační práci s novým týmem, a tak jsem se ho zeptal, jak se jim vede. Odpověděl: „Píšou zděděný kód, příteli.“ Tohle byl jeden z mála okamžiků v mém životě, kdy jsem v odpovědi svého kolegy dostal zcela nečekaný úder. Cítil jsem to v žaludku. Erik vyřkl slova a navodil přesný pocit, který jsem často míval při první návštěvě týmů. Vyvíjejí velké úsilí, ale ke konci dne, kvůli časovému tlaku, tlakům z minulosti nebo nedostatku jakéhokoliv lepšího kódu, se kterým by porovnali své výsledky, vytvářejí zděděný kód.

Co je to zděděný kód? Použil jsem tento termín, aniž bych ho definoval. Podívejme se na přesnou definici: Zděděný kód je kód, který jsme obdrželi od někoho jiného. Možná, že ho naše společnost získala od jiné společnosti, možná, že lidé z původního týmu přešli k jiným projektům. Zděděný kód je kódem někoho jiného. Ale v programátorské řeči znamená tento termín mnohem více. Termín *zděděný kód* časem získal mnoho dalších odstínů ve významu i důležitosti.

Co vám přijde na mysl, když slyšíte termín *zděděný kód?* Pokud jste jako já, přemýšlíte o zmatené, nepochopitelné struktuře, o kódu, který máte modifikovat, ale kterému ve skutečnosti nerozumíte. Přemýšlíte o probděných nocích, kdy zkoušíte přidávat vlastnosti, jejichž přidání by mělo být jednoduché, a uvažujete o demoralizaci, o pocitu, že každý v týmu je natolik znechucen kódem aplikace, že se zdá, že je to ani nezajímá – druh kódu dobrý akorát do stoupy. Část z vás se cítí špatně již při pouhém pomyšlení, že by to mohli vylepšovat. Zdá se, že je to nehodné vašeho úsilí. Tato definice zděděného kódu nemá nic společného s tím, kdo jej napsal. Kód může být znehodnocen mnoha způsoby a řada z nich nemá nic společného s tím, zda byl vytvořen jiným týmem.

V průmyslu je *zděděný kód* často používán jako slangový výraz pro kód, který je těžké měnit a kterému nerozumíme. Ale během let práce s týmy, kterým jsem pomáhal překonat vážné problémy s kódem, jsem dospěl k jiné definici.

Pro mne je *zděděný kód* jednoduše kód bez testování. Tato definice mi způsobila trápení. Co mají testy společného se špatným kódem? Pro mne je odpověď přímočará a je to ústřední téma, které rozvíjím ve všech částech knihy:

Z praxe

Neotestovaný kód je špatný kód. Nezáleží na tom, jak dobře je napsaný. Nezáleží na tom, jak je úhledně nebo objektově orientovaný nebo jak dobře je zapouzdřený. S pomocí testů můžeme změnit chování kódu rychle a ověřitelným způsobem. Bez nich opravdu nevíme, zda náš kód spěje k lepšímu nebo k horšímu.

Možná si myslíte, že je to tvrdé. Co takhle bezchybný kód? Pokud je kód aplikace dobře čitelný a dobře strukturovaný, copak to nestačí? Dobrá, neudělejte chybu. Mám rád bezchybný kód. Mám jej raději než většina lidí, které znám, ale i když je bezchybný kód dobrý, nestačí to. Týmy podstupují vážná rizika, když se snaží provádět velké změny bez testů. Je to jako vzdušná akrobacie bez záchytné sítě. Vyžaduje neuvěřitelnou obratnost a jasné pochopení toho, co může nastat při každém kroku. Přesná znalost toho, co nastane, když změníte několik proměnných, je často totéž jako vědět, zda vás druhý gymnasta chytne za ruce poté, co provedete salto. Pokud pracujete v týmu s čitelným kódem, jste v lepší situaci, než je většina programátorů. Při své práci jsem si všiml, že se týmy s takovým stupněm bezchybnosti celého kódu nevyskytují často. Zdá se, že jde o statistickou anomálii. A víte co? Pokud neprovádějí podpůrné testy, i tak se zdá, že jejich modifikace kódu je pomalejší než u týmů, které je provádějí.

Ano, týmy se zlepšují a začínají psát čitelnější kód, ale udělat starý kód čitelnějším zabírá mnoho času. V mnoha případech se to kompletně nepodaří nikdy. Z tohoto důvodu nemám problém definovat zděděný kód jako kód bez testů. Je to dobrá pracovní definice a ukazuje na řešení.

Již jsem řekl něco o testech, ale o testování tato kniha není. Tato kniha je o schopnosti provádět spolehlivě změny v jakémkoli kódu aplikace. V následujících kapitolách popisuji techniky, které můžete použít pro porozumění kódu, k jeho otestování, refaktorování a přidávání vlastností.

Jedná z věcí, kterých si při čtení této knihy všimnete, je, že nepojednává o přehledném kódu. Příklady, které používám v této knize, jsou umělé, protože se svými klienty pracuji na základě smluv, které neumožňují sdílení informací. Ale v mnoha příkladech jsem se pokusil zachovat ducha kódu, který jsem viděl v praxi.

Nebudu tvrdit, že tyto příklady jsou vždy reprezentativní. V terénu jsou jistě ostrůvky dobrého kódu, ale upřímně řečeno, jsou tam i části kódu horšího než cokoliv, co bych v této knize mohl použít. I kdyby nebylo třeba zachovávat důvěryhodnost vůči klientovi, nemohl bych takový kód v této knize použít, aniž bych vás do morku kostí nudil a pohřbil důležitá témata v množství detailů. Výsledkem je, že mnoho příkladů je relativně krátkých. Pokud se na některý z nich podíváte a pomyslíte si „ne, on tomu nerozumí – moje metody jsou mnohem rozsáhlejší, než je tohle, a mnohem horší", zvažte prosím moji radu – zda je použitelná, i když se tento příklad zdá být jednodušší.

Zde uvedené techniky byly testovány na podstatně větších objemech kódu. Příklady jsou kratší jen kvůli omezení formátu knihy. Zejména když vidíte tři tečky (...) v části kódu, jako je tento, můžete je číst jako „vložte sem 500 řádků ošklivého kódu“:

```
m_pDispatcher->register(listener);
...
m_nMargins++;
```

Jestliže tato kniha nepojednává o pěkném kódu, tím méně se zabývá pěkným návrhem. Dobrý návrh by měl být cílem nás všech, ale v případě zděděného kódu je to něco, k čemu dospíváme po oddělených krocích. V některých kapitolách popisuji způsoby přidávání nového kódu do existujícího kódu aplikace a ukazuji, jak jej přidat a mít při tom na paměti dobrý návrh. Můžete začít s rozšiřováním oblastí velmi dobrého kódu vysoké kvality ve zděděném kódu aplikace, ale nebuďte překvapeni, když některé z kroků, které provedete, některé části kódu o něco zhorší. Tato práce je jako chirurgie. Musíme provést incize, musíme pracovat ve střevech a vypnout některá estetická hodnocení. Mohly by být některé hlavní orgány a vnitřnosti pacienta lepší než jsou? Ano. Tedy zapomeneme na jeho současný problém, zašijeme jej a řekneme mu, aby jedl správným způsobem a trénoval na maraton? Mohli bychom, ale co skutečně potřebujeme udělat, je brát pacienta takového, jaký je. Opravit, co je špatné, a změnit jeho zdravotní stav k lepšímu. Třeba se nikdy nestane olympijským atletem, ale nesmíme dopustit, aby „nejlepší“ bylo nepřítelem „lepšího“. Kód aplikace se pro další práci může stát zdravějším a jednodušším. Když se pacient cítí trochu lépe, často nastává vhodný čas, abyste mu pomohli udělat předsevzetí pro zdravější životní styl. To je věc, za kterou bojujeme v případě zděděného kódu. Pokoušíme se dostat k bodu, ve kterém obvykle dosáhneme zjednodušení, a snažíme se, abychom vytvářeli změny kódu jednodušeji. Pokud v týmu tuto myšlenku obhájíme, návrh bude lepší.

Techniky, které popisuji, jsou ty, které jsem se svými spolupracovníky studoval a učil se v průběhu let, kdy jsem pracoval se svými klienty, abych se pokusil dostat pod kontrolu nepoddajný kód aplikace. Druhotně jsem se dostal k problematice zděděného kódu.

Když jsem začal poprvé pracovat pro firmu Object Mentor, hlavní objem mé práce spočíval v pomoci týmům v zásadním úkolu rozvinout jejich schopnosti a interakce k bodu, kdy by mohly regulérně odevzdávat kvalitní kód. Často jsme používali praktiky extrémního programování, abychom týmům pomohli, aby dostaly svoji práci pod kontrolu, intenzívně kooperovali a odevzdávali práci. Často jsem měl pocit, že extrémní programování není ani tak způsob vývoje softwaru, jako spíše způsob, jak vytvořit dobře sehraný pracovní tým, který prostě jen tak odevzdává každé dva týdny úžasný software.

Ale od počátku tu byl problém. Mnoho z projektů XP (extrémního programování) bylo postaveno na zelené louce. Klienti, které jsem potkával, měli podstatně rozsáhlejší kódy aplikací a měli potíže. Potřebovali nějakým způsobem získat kontrolu nad svými úkoly a začít odevzdávat práci. Po čase jsem zjistil, že jsem s klienty dělal stále stejné věci. Tato myšlenka kulminovala během jakéhosi úkolu, na kterém jsem pracoval s týmem v oboru finančnictví. Než jsem přijel, zjistili, že jednotkové testování softwaru bylo dobrou myšlenkou, ale testy, které používali, byly všezahrnující testy, které se vícenásobně připojovaly k databázi a využívaly značnou část kódu. Napsat testy bylo obtížné a tento tým je nepoužíval příliš často, protože trvaly příliš dlouho. Jakmile jsem si s nimi sedl, abychom zrušili vazby a testovali menší části kódu, měl jsem strašný pocit „déjà vu“. Zdálo se mi, že jsem tuto práci prováděl s každým týmem, který jsem potkal, a byl to ten druh problému, o kterém nikdo nechtěl vážně přemýš-

let. Byla to jen nečistá práce, kterou děláte v případě, že chcete začít pracovat se svým kódem řízeným způsobem, pokud víte, jak na to. Rozhodl jsem se, že stojí za to uvažovat nad tím, jak jsme tyto problémy řešili, a zapsat je na papír, aby týmy získaly výhodu a mohly začít své aplikační kódy psát jednodušší.

Poznámka k příkladům: použil jsem příklady v několika různých programovacích jazycích. Podstatná část příkladů je napsána v jazycích Java, C++ a C. Vybral jsem si Javu, protože je to velmi rozšířený jazyk a zahrnul jsem C++, protože v prostředí zděděného kódu představuje určitou výzvu. Vybral jsem jazyk C, protože zvýrazňuje mnoho problémů, které jdou ruku v ruce s procedurálním zděděným kódem. Ale i když nejsou v příkladech použity jazyky, které používáte, podívejte se na ně. Mnohé techniky, které zahrnují, mohou být použity i v jiných jazycích, jako je např. Delphi, Visual Basic, COBOL nebo FORTRAN.

Programování může být velmi prospěšná a radostná práce. Pokud radost z programování během každodenní práce nepociťujete, doufám, že techniky, které v této knize nabízím, vám ji pomohou nalézt a ve vašem týmu rozvinout.

Poděkování

Především mám veliký dluh vůči své manželce Anně a svým dětem, Deboře a Ryanovi. Jejich láska a podpora umožnila vznik této knihy a také veškeré studium, které jí předcházelo. Rád bych také poděkoval „strýčku Bobovi“ Martinovi, presidentovi a zakladateli firmy Object Mentor. Jeho pečlivý přístup k vývoji a návrhu k oddělování zásadního od nedůležitého a mi dal něco, čeho jsem se mohl před deseti lety chytit – v době, kdy jsem si myslel, že se utopím v množství nerealistických rad. A díky, Bobe, za možnost vidět během minulých pěti let více kódu a pracovat s více lidmi, než bych si kdy pomyslil, že je možné.

Rovněž musím poděkovat Kentu Beckovi, Martinu Fowlerovi, Ronu Jeffries a Wardu Cunninghamovi, že mi v různých dobách nabídli radu a hodně mě naučili o týmové práci, návrhu a programování. Zvláštní dík všem lidem, kteří zhlédli moji pracovní verzi. Oficiální recenzenti byli Sven Gorts, Robert C. Martin, Erik Meade a Bill Wake, neoficiální Dr. Robert Koss, James Grenning, Lowell Lindstrom, Micah Martin, Russ Rufer, Silicon Valley Patterns Group a James Newkirk.

Rovněž děkuji recenzentům nejranějších pracovních verzí, které jsem umístil na Internet. Jejich odezva významně ovlivnila nasměrování knihy poté, co jsem reorganizoval její strukturu.

Omlouvám se dopředu všem těm z vás, které jsem vynechal. První recenzenti byli: Darren Hobbs, Martin Lippert, Keith Nicholas, Phlip Plumlee, C. Keith Ray, Robert Blum, Bill Burris, William Caputo, Brian Marick, Steve Freeman, David Putman, Emily Bache, Dave Astels, Russel Hill, Christian Sepulveda a Brian Christopher Robinson.

Rovněž děkuji Joshuovi Kerievskymu, který mi poskytl první klíčový posudek, a Jeffu Langrovi, který mi pomáhal radou a průběžnými komentáři během celého procesu.

Recenzenti mi do značné míry pomohli vybrousit pracovní verzi, ale pokud tam zůstaly nějaké chyby, jsou jen moje.

Děkuji lidem, jako je Martin Fowler, Ralph Johnson, Bill Opdyke, Don Roberts a John Brant, za jejich práci v oblasti refaktorizace. Byla inspirativní.

Zvláštní dluh mám také vůči Jay Packlickovi, Jacques Morelovi a Kelly Mower z firmy Sabre Holdings a Grahamu Wrightovi z firmy Workshare Technology za jejich podporu a odezvu.

Zvláštní poděkování si zaslouží rovněž tito lidé: Paul Petralia, Michelle Vincenti, Lori Lyons, Krista Hansing a zbytek týmu z Prentice-Hall. Děkuji ti, Paule, za veškerou pomoc a povzbuzení, které tento nový autor potřeboval.

Zvláštní dík si rovněž zaslouží Gary a Joan Feathers, April Roberts, Dr. Raimund Ege, David Lopez de Quintana, Carlos Perez, Carlos M. Rodriguez, a později Dr. John C. Comfort za pomoc a povzbuzení v průběhu let. Rovněž musím poděkovat Brianu Buttonovi za příklad v kapitole 21, *Měním stejný kód na všech místech.* Napsal tento kód během asi hodiny, když jsme společně vyvíjeli a refaktorizovali, a tento kód se pak stal mým oblíbeným příkladem pro výuku.

Zvláštní dík si rovněž zaslouží Janik Top, jehož instrumentální skladba *De Futura* mi posloužila jako zvuková stopa během několika posledních týdnů práce na této knize.

Nakonec bych rád poděkoval všem, se kterými jsem během několika minulých let pracoval a jejichž náhled a připomínky podpořily materiál v této knize.

Michael Feathers
mfeathers@objectmentor.com
www.objectmentor.com
www.michaelfeathers.com

Poznámka redakce českého vydání

Nakladatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
redakce PC literatury
Holandská 8
639 00 Brno

nebo

knihy@cpress.cz

Další informace a případné opravy českého vydání knihy najdete na internetové adrese *http://knihy.cpress.cz/k1618*. Prostřednictvím uvedené adresy můžete též naší redakci zaslat komentář nebo dotaz týkající se knihy. Na vaše reakce se srdečně těšíme.

Úvod

Jak používat tuto knihu

Zkoušel jsem několik různých formátů, než jsem se rozhodl pro ten stávající. Mnoho různých technik a postupů, které jsou užitečné v práci se zděděným kódem, se těžko vysvětluje izolovaně. Nejjednodušší změny se často provádějí snadněji, pokud můžete nalézt švy, vytvořit imitace objektů nebo zrušit závislosti s pomocí několika k tomu určených technik. Rozhodl jsem se, že nejjednodušší způsob, jak vytvořit knihu přístupnou a užitečnou, je převést její převážnou část (Část II, *Modifikace softwaru*) do formátu FAQ (často kladené otázky). Protože specifické techniky často vyžadují použití dalších technik, kapitoly s často kladenými otázkami jsou vzájemně hustě propojeny. Téměř v každé kapitole naleznete odkazy (s čísly stránek) na jiné kapitoly a sekce, které popisují konkrétní techniky a refaktorizaci. Omlouvám se, pokud budete muset při hledání odpovědí na své otázky hodně v knize listovat, ale předpokládal jsem, že raději uděláte tohle, než byste četli knihu od začátku do konce a zkoušeli porozumět najednou, jak všechny ty techniky pracují.

V části *Modifikace softwaru* jsem se pokusil vypořádat se s velmi obecnými otázkami, které souvisejí s prací se zděděným kódem. Každá z těchto kapitol je pojmenována podle specifického problému. To poněkud prodlužuje nadpisy kapitol, doufám ale, že vám to umožní nalézt rychle sekci, která vám pomůže s konkrétním problémem, který právě máte.

Část *Modifikace softwaru* je zakončena souborem úvodních kapitol (*Část I, Mechanismus změn*) a katalogem refaktorizací, které jsou velmi užitečné v práci se zděděným kódem (*Část III, Techniky rušení závislostí*). Přečtěte si prosím úvodní kapitoly, zvláště Kapitolu 4, *Model švů*. Tyto kapitoly poskytují kontext a názvosloví všech následujících technik. Pokud naleznete termín, který není v kontextu popsán, podívejte se do glosáře.

Metody refaktorizace v kapitole *Techniky rušení závislostí* jsou zvláštní v tom, že se předpokládá, že budou bez testů, ale budou sloužit k přípravě testů. Doporučuji vám přečíst si každou z nich, abyste viděli další možnosti, když začnete zpracovávat váš zděděný kód.