Kapitola 2

Metafora

Programátor píše *programy*. Designér navrhuje *design*. Co to ale vlastně znamená? Neexistují žádné televizní reality show nebo hollywoodské filmy, které by zachycovaly, jak ve skutečnosti pracujeme. A než něco řeknete, *sociální síť* se jako reprezentativní příklad z oboru nepočítá. Takže když se mě někdo zeptá, co dělám, často se uchýlím k analogii. Naše odvětví je jich plné. Právě pomocí nich popisujeme naši práci komukoli z venčí.

Šéfkuchař nemusí vymýšlet metafory pro vaření; že je vývar moc slaný, poznáte podle chuti. Muzikant nemusí popisovat písničky nějak oklikou; melodie je ohraná, protože jste ji už někdy slyšeli. To je lidem jasné. Taková práce vysvětluje sama sebe. Instalatéři nebo pokrývači mají svou pracovní činnost – vodoinstalaci a pokrývání střech – jasně zmíněnou už v názvu.

Jenže s programováním je to jinak. Běžný člověk neví, proč je kód elegantní nebo zmatečný. Naše odvětví je navíc *velmi* mladé. Zatímco lidstvo má tisícileté zkušenosti s vařením, skládáním hudby nebo stavebnictvím, na archeologický nález jeskynních maleb *Člověk píšící na stroji* se ještě čeká.

Proto se naším metajazykem musí stát metafora. Nejenže nám slouží k tomu, abychom srozumitelně vysvětlili unikátnost programování široké veřejnosti, ale často s její pomocí i přijímáme rozhodnutí, jak postupovat při řešení softwarových problémů.

Esej 1: S metaforou zacházejte opatrně

A tady se nám situace komplikuje a začíná být nebezpečná.

Hranice mezi metaforou a realitou se někdy smaže, a my pak kvůli metafoře můžeme přecenit skutečnosti, které nejsou důležité, zatímco ty podstatné podceníme.

Když v těchto přirovnáních zajdeme příliš daleko, nerozhodujeme se pro danou situaci nejlépe. Metafora má ďábelskou schopnost zamlžit před námi skutečnou pravdu. V kontextu metafory mohou naše rozhodnutí dávat perfektní smysl, když se však vrátíme zpět na úroveň samotné tvorby aplikací, snadno můžeme sejít z cesty. Někdy přespříliš spoléháme na domnělé výhody metafory, místo abychom se soustředili na realitu situace.

Pro vytváření softwaru například často používáme přirovnání k tradiční architektuře. Z ní koneckonců pochází většina názvů pracovních pozic. Proto si říkáme s*oftwaroví architekti, informační architekti, senior* a *junior vývojáři* a *projektoví manažeři*. Proto mnozí z nás stále nedají dopustit na skici webů, specifikace, workflow diagramy, Ganttovy diagramy a na vývoj podle vodopádového modelu. Velkou část procesů softwarového vývoje jsme vykradli z úplně jiného odvětví.

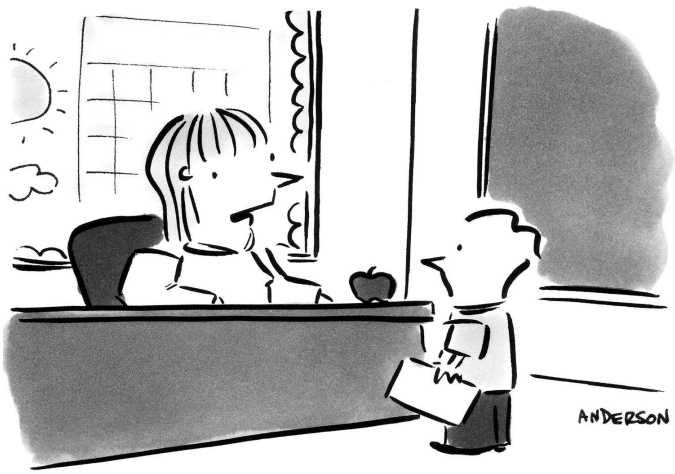
Avšak ačkoli jsou tyto koncepty pro jinou oblast nezbytně důležité a v našem světě mají také své zásluhy, velmi snadno nás můžou uvěznit. Zamyslíme-li se nad tím, proč některé problémy řešíme určitým způsobem, můžeme původ takového postupu najít v metafoře tradiční architektury (nebo jiné), které jsme se drželi příliš horlivě.

Čím nám metafora ubližuje? Pojďme se podívat na několik příkladů situací, kdy byl koncept z architektury nešikovně napasován na software.

Esej 2: Dvakrát měř, jednou řež

V tradiční architektuře platí, že plánování je základ. Určité věci musí nutně předcházet jiným. Úchyty musí být upevněny před trubkami. Trubky se namontují před omítnutím zdí. Stěny musí být hotové před malbou. Operace Zpět, Vyjmout nebo Vrátit nejsou při stavbě mrakodrapů k dispozici.

Softwarové Zpět je Ctrl+Z. Softwarové Vyjmout je Ctrl+X. Softwarová operace Vrátit spočívá v uvedení kódu do původního stavu v nástroji správy zdrojových kódů.



„Zapomeňte na Ctrl+X, na to máte nůžky."

Ve stavebnictví nezbývá než se obejít bez luxusu, který poskytují tato jednoduchá, a přesto mocná gesta. Proto je třeba velmi detailní specifikace. Rozdíl mezi výdělečnou nemovitostí a katastrofickými palcovými titulky na přední straně novin je několik špatně odměřených centimetrů.

Představme si, že jsme tradiční architekti a že máme k dispozici všechny výhody softwarového vývoje. To by byl parádní svět! Materiály by byly k dispozici v nekonečném množství. Model budovy v životní velikosti by bylo možné postavit za pár týdnů. Testovací visuté mosty bychom mohli podrobovat zátěžovým testům znova a znova. Kdyby se most zřítil, komu by to vadilo? Hned bychom během pár minut mohli vyrobit deset nových kopií!

Samozřejmě tohle všechno je pouhá fantazie. Proto, pokud jsme se rozhodli pro stavbu mrakodrapu, dává rozepsání specifikace do nejmenšího detailu jasný smysl.

Na druhou stranu toto *jsou* výhody našeho odvětví. Softwarové komponenty nemusí čekat na doručení písmen a čísel z místní továrny. Píšeme, kompilujeme, testujeme a tak pořád dokola. Můžeme kód otestovat na skutečném produktu, ne na nějakém jeho modelu. Můžeme si dovolit ten luxus sledovat během vývoje, jak se naše visuté mosty stokrát rozlámou, na všech možných místech a za všech možných podmínek, aniž bychom se museli bát plýtvání materiálem nebo ohrožení lidských životů. Realizovat práci takovýmto způsobem je zcela proveditelné. Když dokončíme software, stejnou aplikaci můžeme se zanedbatelným úsilím tisíckrát zduplikovat.

Když v roce 2008 vznikalo prakticky identické dvojče hotelu Wynn v Las Vegas pojmenované Encore, nemohli stavitelé jen pohodlně zkopírovat a vložit předchozí verzi na sousední prázdný pozemek. Museli začít se specifikací a plánováním, jen aby mohli postavit téměř stejný objekt.

I v případech, kdy software znamenal převážet kód na discích, dávalo rozsáhlé plánování stále smysl. Naproti tomu u softwaru založeného na webových technologiích je tomu jinak. Plánování prostřednictvím velmi detailních specifikací, dříve než napíšete jediný řádek kódu, má stále své výhody, ale nedokáže vykřesat maximum z možností, které se u webového vývoje nabízí. Můžeme vydávat nové verze každý den, hodinu nebo kdykoli chceme, s velmi malou režií a z pohodlí měkkých kancelářských židlí.

Naštěstí se jako obor začínáme ze sevření metaforou vymaňovat. Agilní vývojová metodologie není revolučním zvratem; jenom nás odpoutává od metafory, která dnes nemá již takový význam jako v minulosti. Tím se ale nechce říct, že tradiční vodopádový model je již k ničemu. Stále má své uplatnění u větších a složitějších softwarových projektů. Pokud se však budeme řídit metaforou bez zamyšlení, můžeme snadno přehlédnout jiný přístup, který by do našeho pracovního prostředí zapadnul lépe.

Příměr „dvakrát měř a jednou řež" v našem případě přeceňuje čas, který strávíme pilováním plánů k dokonalosti, a podceňuje ten čas, který věnujeme samotnému psaní kódu.

Esej 3: Uvedení na trh není nic víc než jen vydání první z mnoha verzí

Tradičně jsme zvyklí přistupovat k termínu spuštění projektu jako ke stěžejnímu okamžiku v čase, kdy musí být software *hotový*. Není cesty zpět.

U budov a dalších staveb je takový přístup klíčový. V oblasti softwaru dávala jeden čas tato metafora také smysl: Když jsme distribuovali software na disketách a CD, vše muselo být v naprostém pořádku. Jakákoli chyba měla velký dopad na finance a čas. Projekty se zpožďovaly kvůli dolaďování k dokonalosti nebo kvůli zavádění dalších vylepšení. O tom, co takový postup dělá s pracovní morálkou, se zmíním v příští kapitole.

Dnes se aplikace založené na webu pouze nevydávají, dnes se nahrávají na server, spouští a nahrazují. Software je živoucí organismus, který postupem času dozrává.

Jakmile je aplikace vydaná, verze číslo 2, 3 a 20 mohou přijít již za pár dní, či dokonce hodin. I samotný koncept formálního vydávání verzí u softwaru je zastaralý. Dával smysl v dobách, kdy se software distribuoval na fyzických nosičích, ale ty už jsou dávno pryč.

Dnes nepřetržitě integrujeme a neustále iterujeme. V naší branži není na rozdíl od automobilového průmyslu důvod stahovat fyzické výrobky z prodeje. Kritickou chybu je dnes možné opravit záplatou, otestovat a okamžitě zprovoznit upravenou verzi. Už to není verze 2.0, ale 2.0.12931. Nebo jednodušeji řečeno je to *dnešní* verze. Opravdu to veřejnost ještě stíhá sledovat?

Společnost si na iterační model také postupně zvyká. Viděli jste novou galerii fotografií na Facebooku? Všimli jste si nejnovější funkce našeptávání ve vyhledávači Google? A co nový vzhled na Twitteru? Nikdo nás nevaroval měsíc trvající reklamní kampaní. Nové změny se dnes jednoduše objeví. Populární aplikace pro 3D chatování IMVU[2] se pyšní více než sty miliony registrovaných uživatelů a vydává nové verze padesátkrát *denně*.

V dnešní situaci by počáteční uvedení na trh nemělo být chápáno jako událost, na které všechno stojí a padá, jako tomu bylo dříve a v mnoha jiných odvětvích je i nyní. Je to jen vydání jedné z mnoha mini-verzí, které během životního cyklu softwaru přijdou. Budete-li se tohoto pohledu držet, vyhnete se psychickému tlaku při prvotním spuštění.

Podobné uvažování je však snadno zneužitelné. Nepoužívejte tuto změnu přístupu jako omluvu pro lenost nebo nedotaženou práci. Aplikace by měla být při prvním uvedení do ostrého provozu ve velmi dobrém stavu. Podstatné věci musí fungovat. Odpovídající bezpečnost musí být zaručena. Avšak drobnosti, které je možné opravit později, by neměly spuštění softwaru zdržovat. Divili byste se, jak často věci, které vám připadaly důležité, když jste vydávali první verzi, najednou důležité nejsou – teď, když je software venku.

I nadále byste měli první uvedení na trh nebo spuštění oslavovat. Vezměte svůj tým do oblíbené restaurace. Ale nespotřebujte veškerý emoční kapitál na svatbu, po ní na vás čeká dlouhý vztah, který se svým softwarem budete pěstovat.

2 *www.imvu.com*

Bude čas udělat úpravy, přidat celou rodinu nových prvků a napravit nedostatky.

Vydání první verze představuje jen jeden okamžik v životě softwaru. A není jediný a osudový.

Esej 4: Architekt schovaný ve „slonovinové věži" je mýtus

Nikdy jsem nebyl nakloněn myšlence, že softwaroví architekti mají přestat s psaním kódu.

Architekti budov jsou usazeni ve svých slonovinových věžích a zabráni jen a jen do plánování. Nezatloukají hřebíky ani nepájí spoje. Chtít po architektech, aby dělali fyzickou práci vrtání děr a pokládání betonu, by bylo jednoduše nepraktické. Architektonický návrh a realizace stavby jsou dvě odlišné kariérní dráhy.

Kariérní postup vede k menšímu množství kódu

V našem odvětví se do role technického architekta propracujeme přes vlastní realizaci softwaru – „fyzickou" tvorbou aplikací. Avšak ve většině organizací s kariérním postupem v prostředí softwarového vývoje také píšeme čím dál méně kódu. Jsme více zabráni do plánování než do objevování zádrhelů „tam na frontě". Více se staráme o celkový obraz a méně o nejjemnější detaily kódu. Náš obor se přimknul k představě, že architekti mají plánovat a vývojáři mají vyvíjet kód.

Tak vniká falešný dojem, že jakmile dosáhneme určité úrovně ve firemní hierarchii, programování přestane být oblastí, kde jsou nejvíce potřeba naše schopnosti. Jen ať

špinavou práci dělají vývojáři na juniorských pozicích! Tento klam zároveň odrazuje programátory na nižších pozicích od přemýšlení nad celkovými cíli a směřováním projektu. Chce se po nich, aby se zaměřili jen na implementaci. Model architekt–vývojáři způsobuje, že obě strany jsou méně odpovědné za aplikaci jako celek.

Když si rozdělíme role podle této poněkud umělé hierarchie na ty, kteří hloubají nad technickým „celkovým pohledem", a ty, kdo přemýšlí jen v konstrukcích `if` a `for`, odtrhneme od sebe dvě disciplíny, jež patří těsně dohromady.

Softwaroví architekti mohou odhadnout nejvhodnější architekturu, nejpříhodnější návrhový vzor nebo nejlepší postup. Ale jen tehdy, když se brodí po kolena ve zdrojových kódech, dokážou rozpoznat, kde jsou skutečné výzvy. Stejně tak vývojáři, kteří nemají volnost v přemýšlení na vyšší úrovni návrhu, nedostanou možnost vyjádřit protinázor. Přitom často právě člověk, jenž má na starosti vlastní implementaci, je tím, kdo na zvolené cestě nejlépe vidí hrboly.

V analogii s architekturou jsme zašli moc daleko. Kariérní žebříček v odvětví vývoje softwaru potřebuje lepší analogii.

Při stavbě budov architekti navrhují a stavitelé staví. Tradiční architekti vědí, jak vytvořit propracované plány a detailní specifikace. Ale nestaví. Jednoduše to není rozumné. Rozdělení na ty, kteří přemýšlí o vysokoúrovňových záležitostech, a ty, kdo zastanou samotné fyzické práce, vzniklo hlavně z důvodů praktičnosti.

U softwaru to tak však být nemusí. Zkušení vývojáři můžou pracovat v obou rovinách současně. Softwaroví architekti sice jistě stráví většinu času prací na vyšších stupních

návrhu, ale i tak by se měli zapojit do vlastního vývoje, aby si dokázali udělat celkový obrázek.

Udělejte si čas na kód

V mnoha technických organizacích však to, co jsem právě navrhnul, není proveditelné. Většina softwarových architektů tráví pracovní dobu na poradách s ostatními skupinami ve firmě. Často jsou voláni k telefonickým hovorům s klienty, aby s nimi probrali různá technická úskalí, kterým softwarový projekt musí čelit. Kde vůbec vzít čas na psaní kódu?

Několik měsíců poté, co jsem začal pracovat v jednom ze svých zaměstnání na plný úvazek jako programátor pro web, najala naše firma nového softwarového architekta na seniorskou pozici. Přišel mezi nás Adam a úplně změnil atmosféru v naší skupině mladých vývojářů. I přes všechny běžné povinnosti, které na sebe ve své funkci vzal, bylo zjevné, že jeho vášní je kód. Hned jsem si připadal, že mluvím jen s dalším programátorem, ačkoli o hodně chytřejším a moudřejším, než jsem já. Z našeho pracovního vztahu architekta s vývojářem se stal vztah mentora s žákem.

U našeho prvního projektu, kterým byl extranet pro přední právnickou firmu, se Adam zmínil o automatickém generování kódu. Mým uším to znělo trochu jako sci-fi. Avšak jak jsem měl brzy zjistit, podpůrné objekty na straně serveru, dotazy a metody, které jsem se chystal napsat ručně, byly převážně algoritmické. Bylo možné je z velké části odvodit z databázového schématu extranetu. Adam navrhnul, abych se místo pachtění s vývojem metodou hrubé síly

zaměřil na tvorbu uživatelských formulářů a dialogů, zatímco on začal psát generátor kódu. Tvorbě generátoru se věnoval po několik týdnů vždy během své hodinové jízdy vlakem každý den do práce a z práce. Po pár týdnech jsme měli jednoduchý, ale mocný nástroj, který dokázal vygenerovat vše, co bych býval psal ručně.

Veškerý čas, který jsme ztratili odbočkou do automatického generování, jsme si rychle vynahradili, jakmile jsme generátor začali používat. Pokaždé, když se měnilo naše databázové schéma, spustil Adam svůj čarovný prográmek a ten přepsal veškerý kód, který jsem při tvorbě aplikace potřeboval změnit. Netrvalo dlouho a projevily se výsledky úsilí, jež Adam při tvorbě nástroje vyvinul. Náklady na jeho napsání se více než vrátily. A navíc to byl nástroj, který jsme mohli používat znovu a znovu.

Takže ačkoli jsem byl stále v podstatě hlavním vývojářem, Adam měl na vývoji velký podíl. Kdykoli jsem potřeboval nějaké jiné úseky algoritmického kódu, zapracoval během jízdy vlakem domů na přidání odpovídající funkčnosti do generátoru. Příští den jsem měl před sebou čerstvou sadu zdrojových kódů, které už jsem nikdy nemusel znovu ručně psát. Spoustu poznatků, které jsem se během těch prvních pár měsíců naučil, využívám dodnes.

Až se budete propracovávat na vyšší a vyšší programátorské pozice, od řadového vývojáře k architektovi, nezapomeňte, že kód je tím lepidlem, které všechny tyto funkce pojí dohromady. Nemusí to být během jízdy vlakem. Možná se najde hodinka nebo dvě, které můžete v práci věnovat čistě psaní kódu. V Eseji 23 *Zaveďte v týmu „vyhrazený čas“*

na straně 84 najdete rady, jak si rezervovat část dne jen pro psaní kódu. Nakonec ať už jste ve vývojářské hierarchii jakkoli vysoko, nepřestávejte programovat. Protože to je oblast, kde je vás nejvíce potřeba.

Esej 5: Zahoďte starý kód

Nedávno mě Mustafa, jeden z mých kolegů, upozornil, že to „dělám zase". Schovával jsem si kód *na potom*: zakomentoval jsem si část, kterou jsem neměl v plánu už nikdy použít. Jednoduše jsem neměl to srdce ho na místě smazat, i když veškeré naše zdrojové kódy ukládáme pomocí verzovacího systému (a vy byste měli dělat *naprosto* totéž). Protože jsem se stejně vždy mohl k tomu starému kusu kódu vrátit, nemělo smysl ho zakomentovávat, když jsem ho mohl prostě smazat.

Schovávání si kódu „na potom" je jedním z těch zvyků, které navenek působí jako dobrý nápad. Je to zvyk převzatý z jiných důležitých inženýrských zásad, které však nejsou pro oblast programování skutečně relevantní. Kdybychom sestavovali auto, nejspíš bychom uschovali veškerý kovový odpad, protože by se dal později znovu použít. Bylo by hloupé ho jen tak vyhodit. V tradičních technických oborech jsou práce a materiál opravdu kritické. Ve fyzickém světě je daleko jednodušší pracovat s něčím, co je skoro dobře, než všechno zahodit a začít znovu.

Při programování máme tendence těmto dvěma prvkům přikládat příliš velkou váhu. Vyžaduje naše zaměstnání opravdu hodně fyzické práce? Ani ne. Psaní na klávesnici

není tak namáhavé. A co materiál? Když jsem to naposledy kontroloval, nedostatkem úhozů jsme opravdu netrpěli.

Křečkování starého kódu nám navíc přidělává práci. Realita je většinou taková, že kód, který jsem před pár dny nebo týdny zakomentoval, už nikdy neodkomentuji. Místo toho po celém kódu, který právě píšu, raší fleky jednobarevné hatmatilky. Pohled na ně je hrozně otravný. Pokaždé, když pracuji se starým kódem, odvádí mou pozornost od toho, co je právě teď důležité.

I když nastane situace, že se rozhodnu znovu implementovat něco, co jsem napsal už dříve, zakomentovaný kód obvykle stejně nesedí, jak má. Možná jsem odpovídající kus logiky přesunul někam jinam. Objekty nebo metody, které jsem používal v původním kódu, se mohly změnit. Pokus o resuscitaci starého kódu znamená, že strávím víc času překopáváním, než kolik by trvalo přepsání onoho úseku správně a čistě. Ta verze mé osoby, která napsala kód z minulého týdne, znala jen podobu aplikace z minulého týdne.

Mazáním kódu místo jeho věčného schovávání do komentářů udržujeme zdrojové kódy v dobré kondici. Obsah stránky kódu by měl odrážet, jak přesně aplikace momentálně funguje. Nic víc, nic míň. Zbavíme-li se starého kódu *teď hned*, nebudeme muset uprostřed programovaní přeskakovat nesouvisející kousky hatmatilky. A později si nebudeme muset lámat hlavu nad tím, jestli ta obří hromada zakomentovaného, ale důležitě vypadajícího kódu je opravdu stále tak důležitá.