

Obsah

Úvod	11
O autorech.....	13
O odborných korektorech.....	14
Poděkování	15
Předmluva	16

Část I

DDT vs. TDD

Kapitola 1

Někdo to má pozpátku	19
Problémy, které řeší testování řízené návrhem	20
Vědět, kdy je hotovo, je obtížné.....	20
Ponechání testování na později je dražší	21
Testování špatně navrženého kódu je těžké.....	21
Je snadné opomenout testy na úrovni zákazníka.....	22
Vývojáři bývají samolibí	22
Testy někdy postrádají smysl.....	22
Rychlý přehled metodiky DDT bez ohledu na nástroje.....	22
Struktura testování řízeného návrhem.....	23
Metodika DDT v akci.....	25
Jak se metodiky TDD a DDT odlišují	26
Ukázkový projekt: představení webové aplikace Mapplet 2.0	28
Shrnutí	29

Kapitola 2

Úvodní příklad s metodikou TDD.....	33
10 nejdůležitějších vlastností metodiky TDD	34
10. Testy řídí návrh	34

9. Celkový nedostatek dokumentace.....	34
8. Vše je test jednotky	35
7. Testy metodiky TDD nejsou tak docela testy jednotek (nebo snad ano?).....	35
6. Testy přijatelnosti poskytují zpětnou vazbu vůči požadavkům.....	35
5. Metodika TDD propůjčuje důvěru k provádění změn.....	35
4. Návrh se vyvíjí inkrementálním způsobem.....	36
3. Nějaký předběžný návrh je v pořádku	36
2. Metodika TDD produkuje velké množství testů.....	36
1. Metodika TDD je nehorázně obtížná.....	37
Přihlašování implementované pomoci metodiky TDD.....	37
Seznámení s požadavkem.....	37
Přemýšlejte o návrhu	40
Nejdříve se napíše první test psaný jako první test.....	41
Vytvoření kódu pro kontrolu přihlášení, aby test prošel	44
Vytvoření mockobjektu	46
Refaktorizace kódu ukazující rozvoj návrhu.....	48
Testy přijatelnosti s metodikou TDD.....	54
Závěr: metodika TDD je opravdu nehorázně obtížná	54
Shrnutí	56

Kapitola 3

Úvodní příklad s metodikou DDT..... 57

10 nejdůležitějších vlastností metodiky ICONIX/DDT	58
10. Metodika DDT zahrnuje testy obchodních požadavků	58
9. Metodika DDT zahrnuje testy scénářů.....	58
8. Testy jsou odvozené z návrhu	58
7. Metodika DDT zahrnuje testy řadičů	59
6. Metodika DDT testuje chytřeji, ne obtížněji	59
5. Testy jednotek metodiky DDT jsou „klasickými“ testy jednotek.....	59
4. Testovací případy metodiky DDT lze transformovat do testovacího kódu	59
3. Testovací případy metodiky DDT vedou k testovacím plánům	60
2. Testy metodiky DDT jsou užitečné pro vývojáře i členy QA týmů.....	60
1. Metodika DDT dokáže eliminovat nadbytečné úsilí.....	60
Přihlašování implementované pomoci metodiky DDT.....	60
Krok 1: Vytvoření diagramu robustnosti	62
Krok 2: Vytvoření testovacích případů.....	66
Krok 3: Přidání scénářů	68
Krok 4: Transformace testovacích případů testů řadiče do tříd.....	70
Krok 5: Generování testovacího kódu řadičů.....	72
Krok 6: Nakreslení diagramu posloupností.....	75

Krok 7: Vytvoření testovacích případů pro testy jednotek.....	78
Krok 8: Doplnění testovacího kódu	82
Shrnutí	86

Část II

Metodika DDT v praxi: Mapplet 2.0, web pro cestovatele

Kapitola 4

Seznámení s Mapplet 91

10 nejdůležitějších osvědčených postupů metodiky ICONIX Process/DDT	92
10. Vytvořte architekturu	93
9. Dohodněte se na obchodních požadavcích a testujte vůči nim	94
8. Návrh odvoďte z problémové domény	96
7. Podle storyboardů uživatelského rozhraní napište případy užití	98
6. Pro ověření, že případy užití pracují, napište testy scénářů	100
5. Testujte vůči konceptuálnímu a podrobnému návrhu	104
4. Model pravidelně aktualizujte	104
3. Testy přijatelnosti udržujte synchronizované s požadavky	111
2. Mějte automatizované testy stále aktuální	112
1. Kandidáta na finální verzi porovnejte s původními případy užití.....	112
Shrnutí	116

Kapitola 5

Podrobný návrh a testování jednotek..... 117

10 nejdůležitějších „úkolů“ při testování jednotek.....	118
10. Začněte diagramem posloupností	119
9. Na základě návrhu identifikujte testovací případy.....	120
8. Pro každý testovací případ napište scénáře.....	122
7. Testujte chytřeji: nepište překrývající se testy.....	124
6. Testovací případy transformujte do jazyka UML.....	126
5. Napište testy jednotek a doprovodný kód.....	130
4. Napište testy jednotek jako testy „bílé skříňky“	133
3. Použijte framework pro mock objekty.....	137
2. Algoritmickou logiku testujte pomocí testů jednotek.....	140
1. Napište samostatnou sadu integračních testů	141
Shrnutí	142

Kapitola 6

Konceptuální návrh a testování řadičů 143

10 nejdůležitějších „úkolů“ při testování řadičů	145
10. Začněte diagramem robustnosti	145
9. Z řadičů odvoďte testovací případy	148
8. Pro každý testovací případ definujte jeden či více scénářů	151
7. Vyplňte pole Description, Input a Acceptance Criteria	154
6. Vygenerujte testovací třídy	155
5. Implementujte testy	159
4. Napište kód, který se snadno testuje	160
3. Napište testy řadičů jako testy „šedé skříňky“	162
2. Zřetězte testy řadičů dohromady	163
1. Napište samostatnou sadu integračních testů	165
Shrnutí	166

Kapitola 7

Testování přijatelnosti: rozvinutí scénářů

v případě užití..... 167

10 nejdůležitějších „úkolů“ při testování scénářů	169
Případy užití aplikace Mapplet	169
10. Začněte popisným případem užití	170
9. Proveďte transformaci na strukturovaný scénář	173
8. Ujistěte se, že všechny cesty mají kroky	174
7. Přidejte předběžné a následné podmínky	175
6. Vygenerujte diagram činností	175
5. Rozviňte „vlákna“ pomocí externích testů	176
4. Umístěte testovací případ do diagramu testovacích případů	178
3. Ponořte se do zobrazení testování nástroje Enterprise Architect	179
2. Do testovacích scénářů přidávejte podrobnosti	179
1. Vygenerujte dokument s plánem testů	180
Morální ponaučení	181
Shrnutí	184

Kapitola 8

Testování přijatelnosti: obchodní požadavky 185

10 nejdůležitějších „úkolů“ při testování požadavků	186
10. Začněte doménovým modelem	187
9. Napište testy obchodních požadavků	189
8. Požadavky vymodelujte a uspořádejte	189
7. Vytvořte testovací případy z požadavků	192

6. Projděte svůj plán se zákazníkem.....	194
5. Napište ruční popisy testů.....	198
4. Napište automatizované testy požadavků.....	198
3. Exportujte testovací případy.....	199
2. Testovací případy náležitě zpřístupněte.....	199
1. Zapojte svůj tým!	200
Shrnutí	200

Část III

Pokročilá metodika DDT

Kapitola 9

Antivzory při testování jednotek..... 205

Chrám zkázy (neboli kód).....	206
Celkový pohled.....	207
Třída HotelPriceCalculator.....	208
Podpůrné třídy.....	209
Servisní třídy.....	210
Antivzory.....	212
10. Komplexní konstruktor.....	212
9. Stratosférická hierarchie tříd.....	213
8. Statický spouštěč.....	215
7. Statické metody a proměnné.....	217
6. Návrhový vzor Singleton.....	218
5. Těsně svázaná závislost.....	221
4. Obchodní logika v kódu uživatelského rozhraní.....	223
2. Servisní objekty deklarované jako finální.....	225
1. Nedomyšlené funkce dobromyslného programátora.....	225
Shrnutí	225

Kapitola 10

Návrh s ohledem na snazší testování 227

Seznam deseti nejdůležitějších úkolů pro „návrh s ohledem na testování“.....	228
Chrám zkázy – důkladně pročištěný.....	229
Případ užití – pochopení toho, co vlastně chceme dělat.....	230
Identifikování testů řadičů.....	231
Test výpočtu celkové ceny.....	232
Test načtení poslední ceny.....	233

Návrh s ohledem na snazší testování	234
10. Inicializační kód udržujte mimo konstruktor	235
9. S dědičností šetřete	236
8. Vyvarujte se statických inicializačních bloků	237
7. Používejte metody a proměnné na úrovni objektu	238
6. Nepoužívejte návrhový vzor Singleton	238
5. Udržujte své třídy oddělené	239
4. Nedávejte obchodní logiku do kódu uživatelského rozhraní	241
3. Použijte testování černé skříňky a šedé skříňky	246
2. Modifikátor „final“ používejte pro konstanty a obecně jím neoznačujte komplexní typy, jako jsou servisní objekty	246
1. Držte se případů užití a návrhu	247
Podrobný návrh pro případ užití Quote Hotel Price	247
Test řadiče pro výpočet celkové ceny	249
Test řadiče pro načtení poslední ceny	249
Předělaný návrh a kód	250
Shrnutí	251

Kapitola 11

Automatizované integrační testování 253

Seznam 10 nejdůležitějších „úkolů“ pro integrační testování	254
10. Testovací vzory hledejte ve svém konceptuálním návrhu	255
9. Nezapomeňte na testy zabezpečení	256
8. Rozhodněte, která „úroveň“ integračních testů se má psát	258
7. Integrační testy na úrovni řadičů či jednotek odvodte z konceptuálního návrhu	259
6. Testy scénářů odvodte ze scénářů případů užití	262
5. Napište celkové testy scénářů	263
4. Použijte testovací framework „přátelský k obchodnímu hledisku“	267
3. Kód grafického uživatelského rozhraní testujte v rámci testů scénářů	270
2. Nepodceňujte obtížnost integračního testování	270
1. Nepodceňujte hodnotu integračních testů	274
Hlavní body při psaní integračních testů	274
Shrnutí	276

Kapitola 12

Testování algoritmů 277

10 nejdůležitějších „úkolů“ při testování algoritmů.....	278
10. Začněte řadičem z konceptuálního návrhu.....	279
9. Rozvíňte řadiče do návrhu algoritmu.....	281
8. Diagram volně propojte s doménovým modelem.....	282
7. Rozdělte rozhodovací uzly obsahující více než jednu kontrolu.....	284
6. Pro každý uzel vytvořte testovací případ.....	284
5. Pro každý testovací případ definujte testovací scénáře.....	285
4. Vytvořte vstupní data z nejrůznějších zdrojů.....	288
3. Přidejte logický tok do jednotlivých metod a tříd.....	289
2. Napište testy jednotek jako testy „bílé skříňky“.....	293
1. Aplikujte metodiku DDT na další návrhové diagramy.....	303
Shrnutí.....	304

Příloha

Alenka v říši případů užití..... 305

Úvod.....	306
Část 1.....	306
Alenka při čtení usnula.....	307
Příslib vývoje řízeného případu užití.....	307
Model analýzy spojuje text případů užití s objekty.....	308
Jednoduché a přímočaré.....	308
Stereotypy <<includes>> a <<extends>>.....	308
Máme zpoždění! Musíme začít programovat!.....	309
Alenka se podivuje, jak se dostat od případů užití ke kódu.....	309
Abstraktní... podstata.....	309
Až příliš abstraktní?.....	310
Teleocentricita... ..	310
Skutečně je nutné to vše specifikovat pro každý případ užití?.....	311
Část 2.....	312
Alenka dostala řízení.....	312
Alenka pocituje mdloby.....	313
Imagine... (s omlouvou Johnu Lennonovi).....	313
Párové programování znamená, že se nikdy nezaznamenají požadavky.....	314
Na zaznamenání požadavků není čas.....	315
Stejně tak můžete říci, že „kód je návrh“.....	315
Koho zajímají případy užití?.....	316
Projekt C3 ukončen.....	317

Jen a pouze jedinkrát?.....	318
Alenka odmítá začít programovat bez zapsaných požadavků.....	318
Spáchala jsi VPN... ..	320
Model CMM je mrtvý! Pryč s její hlavou!	321
Seriózní refaktorování návrhu.....	321
Část 3	321
Alenka se probouzí	322
Uzavření díry mezi „Co“ a „Jak“	323
Statické a dynamické modely jsou spojené dohromady.....	323
Přidělování chování se odehrává na diagramu posloupností	323
Ponaučení, které z toho plyne	324

Závěr

Byl smažno a lepě svyhlé testy..... 325

Rejstřík 331

Úvod

Jim je jako programový manažer pro mapovací softwarový produkt ArcGIS společnosti ESRI odpovědný za každodenní sestavování 20 milionů řádků kódu. Dříve spravovat dopravní a logistické oddělení pro ESRI Professional Services, kde se mu podařilo dokončit bez překročení termínu a rozpočtu řadu softwarových projektů v hodnotě milionů dolarů, přičemž používal techniky návrhu popsané v této knize.

Spousta lidí má řadu vyhraněných názorů na prakticky všechny stránky testování softwaru. Viděl jsem nebo slyšel o různých metodikách, systémech, procedurách, procesech, vzorech, nadějích, modelitbách i prostém štěstí, že se některé cesty kódu nikdy neprovedly. S veškerou touto pomocí musíme přece vymyslet, jak dodávat výjimečný, neprůstřelný a bezchybný software, ne? Přesto s každým novým vydáním další skvělé, nejstabilnější revize našeho softwarového produktu dělají testovací inženýři stále takový ten bolestivý obličej (znáte to) a při otázce „Máš příslušné testy napříč všemi vrstvami softwaru?“ se stahují zpět.

Příčinou bolestivého obličej je fakt, že pravdivou odpověď na tuto otázku je téměř vždy: „Myslím, že ano, ale ve skutečnosti neznám všechny oblasti, pro které bych měl mít testy.“ A software se i tak vydá, převít jeden. To je pak pracovní jistota pro tým technické podpory, protože chyby ve vydaném produktu se vrátí přímo k vašemu týmu pro další servisní balíček nebo nouzovou rychlou opravu. Jde to ale dělat mnohem lépe a tato kniha vám ukáže, jak.

Tato kniha vás provede osvědčeným procesem vývoje softwaru s názvem ICONIX Process a zaměří se na tvorbu a údržbu testů jednotek a testů přijatelnosti založených a řízených návrhem softwaru. Jedná se o testování řízené návrhem (Design-Driven Testing – DDT). Svůj návrh tak využijete pro přesné vymezení míst, kde je nutné založit kritické testy na návrhu a chování objektů. Nejedná se o návrh řízený testy (Test-Driven Design – TDD), kdy se nejdříve píšou testy jednotek, a to ještě před dokončením návrhu a zahájením programování. Nevím, jak je to u vás, ale předvídání budoucnosti je myslím docela obtížné, a přimět softwarové inženýry, aby naprogramovali něco, co „pasuje“ do určité sady testů, je dokonce ještě těžší.

Zatímco spousta lidí má na testování různé názory, je tu jedna věc, na které se všichni asi shodneme: testování je často velice obtížné a složité. Jako programový manažer velkého vývojového týmu vím, jak rychle se může testování vymknout z rukou nebo prostě pozastavit celý projekt. Organizace vydávají na testování velmi rozdílné prostředky a naneštěstí existují i velké rozdíly, co se týče návratnosti takto investovaných prostředků. Je možné provádět příliš mnoho testování a investované prostředky tak promarnit. Avšak pravděpodobnější je, že se provede příliš málo testování (samozřejmě s představou, že se udělalo více než dost), v nesprávných oblastech softwaru a s nedostatečnými investicemi. K tomu může dojít proto, že prostě nevíte, kam testy umístit pro vyvážené investice a získat tak správné testovací pokrytí.

Tato kniha vám na příkladu návrhu a vybudování skutečné mapové webové aplikace ukáže, jak docílit této vyváženosti a optimalizovat návratnost investic do testování. Při použití metodik ICONIX Pro-

cess a DDT je naprosto jasné, jaké testy jsou nutné a kde mají být. Ba co víc, řadu z těchto testů za vás automaticky vygenerují používané nástroje (v tomto případě nástroj Enterprise Architect od společnosti Sprax Systems), které (kromě toho, že jde o naprosto skvělé nástroje) jsou pro váš projekt nesmírně cenné. Pokud tedy potřebujete vytvořit skvělý software s použitím agilního procesu, kdy se vám testy vygenerují téměř zadarmo, pak je tato kniha právě pro vás.

Jim McKinney
Programový manažer produktu ArcGIS, společnost ESRI

0 autorech

Matt Stephens je softwarový konzultant s finančními organizacemi v Centrálním Londýně a zakladatel nezávislého knižního vydavatelství Fingerpress (www.fingerpress.co.uk). Psal pro mnoho časopisů a webů, například pro *The Register* a *Application Development Trends*. Na Internetu jej najdete na webu společnosti Software Reality (<http://articles.softwarereality.com>).

Doug Rosenberg v roce 1984 ve svém obývacím pokoji založil společnost ICONIX (www.iconixsw.com). Po několika letech práce na tvorbě CASE nástrojů začal školit společnosti v objektově orientované analýze a návrhu, která se specializuje na školení v oblasti jazyků UML a SysML a nabízí jak místní kurzy, tak i kurzy s otevřeným zápisem. V roce 1993 vyvinul jednotný Booch/Rumbaugh/Jacobson přístup k modelování, což bylo o několik let dříve, než se objevil jazyk UML, a kolem roku 1995 začal psát knihy.

Testování řízené návrhem je šestou knihou z oblasti softwarového inženýrství (a čtvrtou s Mattem Stephensem). Kromě toho vytvořil bezpočet multimediálních tutoriálů, jako je například *Enterprise Architect for Power Users*, a několik elektronických knih, mezi něž patří kupříkladu *Embedded Systems Development with SysML*.

Když nepíše a neučí, tak rád vytváří panoramatické fotografie virtuální reality, které si můžete prohlédnout na jeho webu pro cestovatele VResorts.com.

0 odborných korektorech

Jeffrey P. Kantor je manažer projektu správy dat teleskopu LSST (Large Synoptic Survey Telescope – velký synoptický přehledový teleskop). V této funkci je odpovědný za implementování výpočetních a komunikačních systémů zajišťujících kalibraci, hodnocení kvality, zpracování, archivaci a přístup koncových uživatelů a externích systémů k astronomickým obrazům a inženýrským datům produkováným teleskopem LSST.

Po čtyřech letech v Americké armádě, kde působil jako specialista zpravodajské služby na ruský jazyk a signály, začal v roce 1980 jako programátor na základní úrovni, načež zastával pozice na všech úrovních IT organizací v řadě průmyslových odvětví, včetně obrany a letectví, výroby polovodičů, geofyziky, poradenství v oblasti softwarového inženýrství, řízení domácností a budov, zboží dlouhodobé spotřeby, prodeje a elektronické komerce.

Vytvářel, upravoval na míru, aplikoval a auditoval softwarové procesy pro široké spektrum organizací v průmyslu, vládě a školství. U některých z těchto organizací byl odpovědný za dosažení certifikace ISO9000 a úrovně CMM Level 2 institutu SEI. Dále poskytoval poradenství a školení více než 30 společnostem v oblasti objektově orientované analýzy a návrhu, modelování pomocí jazyka UML, testování řízené případy užití a řízení softwarových projektů.

Rád tráví čas se svou rodinou, u fotbalu (jako hráč, sudí a trenér) a ježděním na kole po horách.

David Putnam pracoval posledních 25 let ve vývoji softwaru a od začátku tohoto století je jedním z nejhorlivějších zastánců agilního přístupu v Anglii.

Od začátku bylo jeho zaměření velmi různorodé: vytvářel databázové systémy pro stavební průmysl, pracoval ve fitness odvětví a také přednášel na univerzitě. Během té doby publikoval řadu článků o vývoji softwaru a je známou tvář na konferencích o agilním vývoji softwaru. Nyní pracuje jako nezávislý poradce v oblasti agilního vývoje softwaru a poskytuje neocenitelné rady několika z nejznámějších společností v Anglii.

Poděkování

Autoři by rádi poděkovali:

Týmu společnosti ESRI odpovědnému za projekt Mapplet (Wolfgang Hall, Prakash Darbhamulla, Jim McKinney a Witt Mathot) za jeho práci na ukázkovém projektu v této knize.

Lidem ze společnosti Sparx Systems (Geoff Sparks, Ben Constable, Tom O'Reilly, Aaron Bell, Vimal Kumar a Estelle Gleeson) za vývoj doplňku Agile/ICONIX a editoru strukturovaných scénářů.

Alanahu Stephensovi, který byl naprosto nejlepší „Alenkou“, jaká kdy byla, a Michelle Stephensové, za její trpělivost během dalšího knižního projektu.

Jeffu Kantorovi a Robynu Allsmanovi z projektu LSST za jejich pomoci se specifikací expandéru vláken případu užití.

Miku Farnsworthovi a lidem ze společnosti Virginia DMV, kteří se podíleli na předběžném modelování aplikace Mapplet. Barbaře Rosi-Schwartzové za její zpětnou vazbu ohledně metodiky DDT a Jerry Hambymu za sdílení jeho odborných znalostí technologie Flex a zvláště za jeho pomoc se zpětnou analýzou kódu jazyka MXML.

A v neposlední řadě našim redaktorům v nakladatelství Apress: Jonathanu Gennickovi, Anitě Castrové a Mary Ann Fugatové.

Předmluva

Střež se nadšeného povyku kolem agilnosti

*Bylo smažno, když se YAGNI dní kód
desetkrát za den sám sestavil.
Indexové kartičky ve své křehkosti,
odklizené, refaktorované návrhy.*

*Ó synu, střež se nadšeného povyku kolem agilnosti,
více kódu s pachem, více chyb na odchycení.
Refaktorování bylo mnohem zábavnější,
než byly schopnosti tvé na znak poražené.*

*Své chopil testy jednotek,
proti litým chybám bojoval.
Zmatený přístupem TDD,
zdravý rozum vyhledal.*

*Ale v jeho dřevěném okně časovém,
určeném hrou kde vládl plán,
ony testy zezelenaly a vše bylo májové,
dokud konečný termín nenastal.*

*Je půl třetí, máme hotovo,
je čas na svačinku chutnou.
Že testy zčervenaly, pěl jsi, mátoho,
vše rychlým hackem utnou!*

*A hle, šokující myšlenka jak sen,
ne testy, ale nejdříve návrh!
V této báječný den,
jednodušší proces byl zaveden.*

*Bylo smažno, když se YAGNI dní kód
desetkrát za den sám sestavil.
Indexové kartičky ve své křehkosti,
odklizené, refaktorované návrhy...*