Objektově-orientované testování

Radek Mařík

CA CZ, s.r.o.

September 14, 2007

CA Labs

Transforming IT Management

Obsah

1. Základy testování komponent
   - Definice
   - Specifikace komponent
   - Návrh testů
2. Testování objektově orientovaných systémů
   - Souhrn terminologie
   - Testovací strategie
   - Návrh testů
3. Optimalizace počtu testovacích případů
4. Příloha
   - Terminologie objektově orientovaných systémů

Kroky testování komponent [Mar95]

1. Vytvoř seznam testovacích požadavků
   1. Nalezni záchytné body.
   2. Rozšiř záchytné body na testovací požadavky
2. Navrhni testy
   1. zkombinuj testovacích požadavků do testovacích specifikací,
   2. zkontroluj testovací specifikace vůči typickým chybám testů.
3. Doplň testování s cílenou inspekcí kódu,
4. Implementuj podpůrný kód testu.
5. Implementuj testy.
6. Zhodnoť a vylepši testy.
   1. Změř pokrytí kódu.
   2. Použij výsledky k porozumění či nalezení chybějících záchytných bodů.
   3. Nalezení testovacích požadavků těchto záchytných bodů.
   4. Napiš více testovacích specifikací.

Elementy návrhu testů I [Mar95]

Záchytné body

(též výchozí bod, příznaky, náznaky) jsou zdrojem testovacích požadavků. Seznam záchytných bodů o tom, **co** se má testovat, redukuje možnosti, že tester opomine nějaký test.

- *Příklad:* SEZNAM, seznam řetězců.
- *Příklad:* element seznamu, řetězec znaků

Testovací požadavky

popisují množiny vstupů, které by měly být testovány. Tester musí rozhodnout, **jak** testovat záchytné body, aby se detekovaly případné chyby.

- *Příklad:* SEZNAM musí být jeden element (hodnota nespecifikována).
- *Příklad:* Použij prázdný řetězec jako prvek SEZNAMu.

Elementy návrhu testů II [Mar95]

Testovací specifikace

popisují přesné vstupy programu společně s přesnými výstupy. Než-li je test proveden, musí být proměnným testovacích požadavků přiřazeny **přesné hodnoty**.

- *Příklad:* SEZNAM má prázdný řetězec jako jeho jediný element; očekávaná návratová hodnota 5.

Podmínky ideálního selhání [Mar95]

Dosažitelnost Vstupní data musí být taková, aby navedla program na chybný příkaz, který způsobí selhání.

Nutnost Chybný příkaz vytvoří výsledek odlišný od výsledku správného příkazu.

- *Příklad:* $4 = 2 + 2$ vs. $4 = 2 * 2$

Propagace Nesprávný interní stav se transformacemi viditelně projevuje ve výsledcích programu.

Elementy specifikace [Mar95]

- Specifikace popisuje, co podsystém má dělat.
- Čím méně je informace ve specifikacích, tím méně se dá testovat, neboť specifikace tvoří hlavní zdroj záchytných bodů.
- Rozhranní podsystému může být popsáno výrazy vstupních a výstupních podmínek.
- **Vstupní podmínky** popisují odpovědnost volajícího před zavoláním podsystému.
- **Výstupní podmínky** popisují odpovědnost podsystému.

Vstupní podmínky [Mar95]

- Vstupní podmínky popisují případy chyb
  <Type>: <test>
  On failure: <failure effect>
- **Ověřované**
  - Pravdivost podmínky je kontrolována podsystémem.
  - Efekt selhání musí být úplným popisem - nic jiného by se nemělo změnit.
- **Předpokládané**
  - Volající je odpovědný za zajistění pravdivosti podmínky.
  - Podsystém podmínku nekontroluje, takže se nespecifikuje efekt selhání.
- Píší se jako fráze přirozeného jazyka, případně kombinované operátory *OR* a *AND*.
- Příklad:
  Validated: $(A > 0) AND (B > 0) AND (C > 0)$
  On failure: return string "error"
  Assumed: P is non-null *OR* *NULL_OK* is true

Výstupní podmínky [Mar95]

- Výstupní podmínky popisují výsledky podsystému.
- Psány ve tvaru:
  IF <trigger> THEN <effect>
  or
  IF <trigger> THEN <effect> ELSE <effect>
- *IF – THEN – ELSE* příkazy nejsou vnořené.
- Jestliže se výstupní podmínka aplikuje vždy, pak se část ¡trigger¿ vypouští.
- Výstupní podmínky nemusí být vzájemně se vylučující.
- Příklad:

```
IF (A == B) AND (B == C)
    THEN return "equilateral"
IF ( (A == B) AND (B != C)
       OR (A == C) AND (B != C)
       OR (B == C) AND (A != B))
    THEN return "isosceles"
```

Příklad specifikace podsystému [Mar95]

- STR je pole bytů, které se má vyplnit. Výstupní parametr. RLEN je maximální počet bytů plnící STR.
- Vstupní podmínka:
  - Assumed: RLEN >= 0
  - Validated: RLEN is not 0.
    On failure:
    - *NREAD is 0
    - Návratová hodnota je 0
- Výstupní podmínka:
  - IF *ODD_DIGIT is initially >= 0
    THEN DIGIT[0] = the initial value of *ODD_DIGIT

Záchytné body [Mar95]

1. Vstupní a výstupní podmínky: program by měl zvládnout alespoň dva případy (splněno, nesplněno).
2. Přídej proměnné jako záchytné body.
3. Přídej záchytný bod pro každou větší operaci, kterou můžeš identifikovat ve specifikaci.
4. Přídej zachytné body pro mezivýsledky.
5. Z kódu:
   - Proměnné.
   - Globální proměnné
   - Typické (modelové) operace, např. “prohledání seznamu”
   - Volání funkcí

Příklad záchytných bodů [Mar95]

- Vstupní podmínka 1
- Vstupní podmínka 2
- Výstupní podmínka 1
- RLEN, čítač
- vyhledávání hodnoty sudé číslice

Testovací požadavky [Mar95]

- Jestliže ověřovaná vstupní podmínka je jednoduchá (bez AND nebo OR), uvažují se dva testovací požadavky (nesplnění, splnění).
- OR operátory jsou expandovány, uvažují se pouze TF, FT, FF.
- Při násobném použití AND, jeden z požadavků má všechny termy pravdivé. Potom se sestrojí N požadavků, které mají právě jediný term nepravdivý.
- Nemá smysl definovat ERROR požadavky pro předpokládané vstupní podmínky.
- Výstupní podmínky se až na nějaké vyjímky zpracovávají podobně jako vstupní podmínky.
- Záchytné body operací a proměnných vedou ke katalogizovaným testovacím požadavkům:
  - BOOLEAN
    - 1 (true) [IN, OUT]
    - 0 (false) [IN, OUT]
    - nějaká pravdivá hodnota, která není 1 [IN]

Příklad testovacích požadavků [Mar95]

- Vstupní podmínka:
  - Vstupní podmínka 7
    - RLEN není 0
    - RLEN = 0 . . . ERROR
- Výstupní podmínka . . .
- RLEN, čítač
  - RLEN = 1
  - RLEN = size of STR

Testovací specifikace [Mar95]

- Testovací požadavky by měly být konvertovány do malého počtu testovacích specifikací.
- Každá testovací specifikace pokrývá jeden či více testovacích požadavků.
- Testovací specifikace popisují přesně a úplně vstupy i výstupy.
- Pokud se testuje zpracování vyjímek, pak se každá chyba testuje zvlášť (maskovací efekt).

Příklad

- Test 1:
  - STR, 100-bytový bufer, inicializováno cele 1
  - RLEN = 0
  - *NREAD = 1000
  - EXPECT
    - STR nezměněn
    - *NREAD = 0
    - návratová hodnota 0

Striktní dědičnost [SPK+99]

- Vstupní podmínky dané metody v odvozené třídě musí být ty samé nebo slabší než podmínky pro tu samou metodu v bázové třídě.
- Výstupní podmínky dané metody odvozené třídy musí byt ty same nebo silnější než výstupní podmínky metody v bázové třídě.
- Invariant třídy musí být ten samý nebo silnější než invariant bázové třídy.

Návrh dle kontraktu & Defensivní návrh [SPK+99]

```
class Stack_C
{
  public:
    Stack_C();
    // Pre: None
    // Post: A stack instance exists, is empty,
    //       and has capacity.

    Boolean_T IsEmpty();
    // Pre: None
    // Post: returns (queue.Length() == 0)

    Push(Itemtype_T a) throws OVERFLOW;
    // Pre - Validated: a has a copy constructor
    // Post:
    //   IF queue.IsFull()
    //   THEN OVERFLOW thrown
    //   ELSE
    //     (queue'.length() == queue.length()+1)
    //     && (queue'.Contains(a))
};
```

Implikace pro testování - hierarchie tříd [SPK+99]

- Separace specifikace a implementace dovoluje oddělit vytvoření testovacích případů na funkce a strukturu.
- Posílání zpráv mezi objekty je podobné volání podprogramů nebo funkcí v procedurálních systémech.
- Změny v definici jedné třídy se mohou propagovat automaticky to několika dalších tříd.
- Některé násobně použité definice a deklarace mohou vzájemně reagovat s jinými definicemi a deklaracemi.
- Lze ušetřit hodně úsilí testování, pokud základní vztah “is-a” není porušen.

Důsledky polymorfizmu pro testování [SPK+99]

- Každá třída je balík metod a atributů, které musí být testovány jako jednotka.
- Vzhledem k polymorphismu je nutné uvažovat více druhů vazeb mezi objekty. Nelze uplatnit tradiční postupy statická analýzy.
- Vede na operování s mnohem více metodami. Každá metoda může posílat zprávy dalším objektům, což zvyšuje komplexitu systému.
- Každá metoda je menší než typická procedura/funkce/podprogram, což opět zvyšuje komplexitu systému.

Úrovně OO testování [SPK+99]

Testování tříd je objektově orientovaný ekvivalent jednotce testování procedurálního přístupu. Testování tříd kombinuje tradiční testování metodami černé a bílé skříňky s aspekty integračního testování.

Testování shluků se používá k testování množiny uzce svázaných spolupracujících tříd. Cílem jsou interakce mezi objekty tohoto shluku.

Testování systému se zaměřuje na testování požadované funkcionality a výkonnosti celého systému.

Regresní testování je opakované testování funkcionality systému, která může být narušena modifikacemi systému.

- Regresní testování je integrální součástí iterativně inkrementálního vývojového procesu, neboť každá inkrementace a iterace může změnit produkty dosud vytvořené.

Cíle OO testování [SPK+99]

- Prvním cílem je redukce počtu testovacích případů, které se musí vytvořit.
- Druhým cílem je minimalizace počtu testovacích případů, které se musí provést a jejichž výsledky muzí být ověřeny.
- Testovací případy objektově orientovaných systému mají tvar: *<initial state, message(s), final state>*

Konstrukce funkcionálního testovacího případu [SPK+99]

- Každá vstupní podmínka vymezuje patřičné testovací prostředí objektu.
- Každá výstupní podmínka je logický příkaz konstruovaný jako sekvence "IF-THEN" formulí. Formule mohou být svázány spojkami "OR" a "AND" (konjukce disjunkcí).
- Testovací případ se vytváří pro každou disjunkci.
- Testovací případ se vytváří pro každou vyjímku.
- Vytváří se testovací případy pro zjevné hraniční podmínky.
  - *Příklad:*
    - prázdný zásobník,
    - plný zásobník,
    - zásobník pouze s jedním elementem.
- Testovací případy interakce se identifikují pomocí přístupu ke společnému atributu nebo posíláním zpráv v rámci jednoho objektu.

Kroky návrhu testovacího případu [SPK+99]

1. Navrhni funkcionální testovací soubor, který úplně pokrývá specifikaci třídy.
2. Navrhni stavově založené testovací případy pro všechny přechody dynamického modelu.
3. Navrhni testovací případy interakce mezi metodami uvnitř třídy a mezi třídami.
4. Navrhni strukturální testovací případ pokrývající každou řádku kódu a každou podmínku.

Testování podtříd [SPK+99]

- Konstrukce testovacích případů začíná na vrcholu hierarchie dědičnosti a pokračuje ke speciálním třídám.
- Jak se rozhranní třídy zvětšuje, roste počet testovacích případů.
- Pokud je to možné, testovací případy se násobně využívají.

Hierarchické inkrementální testování [SPK+99]

- Hierarchical Incremental Testing (HIT)
- **Odvození na základě dědičnosti**
  - Pokud nevznikne nový kód, nevznikají nové testovací případy.
  - Pokud nenastanou interakce, není potřeba provádět testy.
- **Případ nové metody**
  - Nový kód požaduje vytvoření nových funkcionálních, strukturálních a interakčních testovacích případů.
  - Proveď všechny testovací případy za účelem ověření nové implementace.
- **Případ přepsání**
  - Nový kód může vyžadovat nové testovací případy, avšak není potřeba vytvářet nové funkcionální testovací případy, pokud výstupní podmínky jsou shodné.
  - Proveď všechny testovací případy za účelem ověření nové implementace.

Testy shluků [SPK+99]

- **Shluk** je množina úzce svázaných tříd, které mezi sebou spolupracují více než s ostatními třídami.
- Specifikace shluku může mít ten stejnou formu jako pro třídu.
- Existuje pouze omezené množství zpráv přicházejících do shluku z externího prostředí. Odpovídají metodám specifikace shluku.
- Vstupní a výstupní podmínky mohou být odvozeny pro každou metodu rozhraní použitím podmínek metod odpovídajících tříd.
- Testovací případy se odvozují podobně jako pro třídu.

Počet OO testovacích případů

- volající objekt
  - protokol chování objektu,
  - předpokládané vstupní podmínky,
  - ověřované vstupní podmínky - zpracování vyjímek.
- volaný objekt
  - protokol chování objektu,
  - splnění výstupních podmínek.
- členy třídy
- parametry
- stavy

PŘÍLIŠ MNOHO TESTŮ

- selekce testovacích případů tak, aby se získalo maximální pokrytí s minimem počtu testovacích případů.

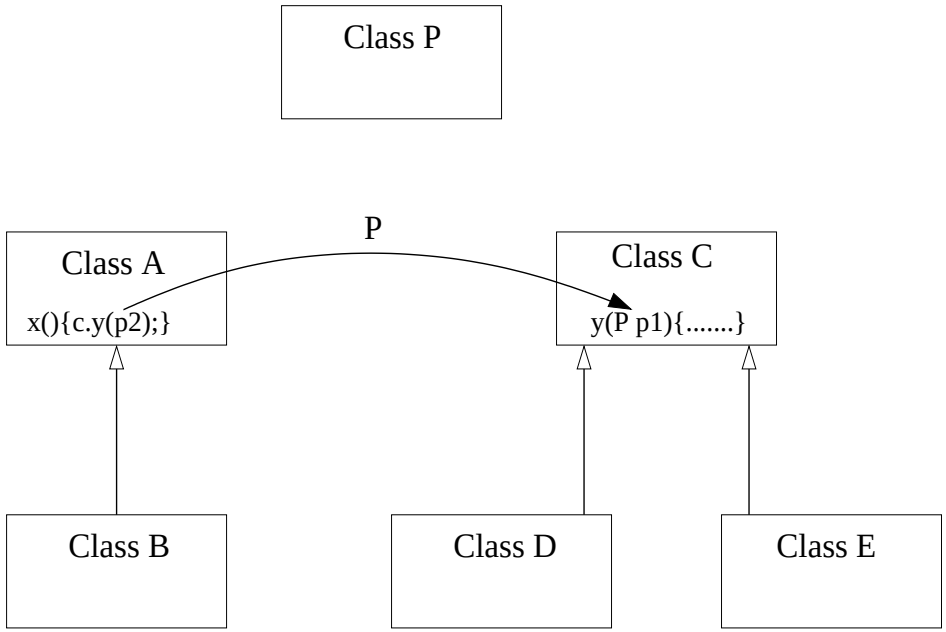
Ortogonální pole [Mon91, Ros88, LW89, SPK+99]

- **Faktor:** entita, parametr, na kterém závisí testovací případ.
- **Úroveň:** každý faktor má konečný počet možných hodnot nazývaných úrovněmi.
- *Příklad:*
  - tři faktory A, B, C a každý faktor má tři úrovně $1, 2, 3$.
  - existuje $27 = 3^3$ možných kombinací (testovacích případů)
  - existuje pouze 9 párových kombinací testující interakce vzájemné interakce faktorů.

trial	A	B	C
1	1	1	3
2	1	2	2
3	1	3	1
4	2	1	2
5	2	2	1
6	2	3	3
7	3	1	1
8	3	2	3
9	3	3	2

Testování interakce tříd [SPK+99]

- Otestuj interakci tříd "*A* hierarchie" se třídami "*C* hierarchie" při zprávě *y* s parametrem třídy *P*.



Třída	A	B	C	D	E	P
# stavů	2	3	2	3	3	2

Testování pomocí ortogonálních polí I [SPK+99]

1. Orthogonal Array Testing (OATS)
2. Zobrazení hierarchií tříd do ortogonálních polí:
   - Vysílající hierarchie je jeden faktor.
   - Přijímací hierarchie je druhý faktor.
   - Každý parametr zprávy generuje další faktor.
   - Možné stavy faktoru zdvojnásobují počet faktorů.
   - *Příklad:* 6 faktorů = **A** hierarchie, **P** hierarchie, a **C** hierarchie a stav objektu pro každou třídu.
3. Výběr úrovní:
   - Maximální počet možných hodnot každého faktoru definuje počet úrovní.
   - *Příklad:*
     - 1 faktor má 1 úroveň (P),
     - 2 faktory mají maximálně 2 úrovně (A, C),
     - 3 faktory mají maximálně 3 úrovně (stavy).

Testování pomocí ortogonálních polí II [SPK+99]

1. Použití standardních polí:
   - Vyber nejmenší standardní pole, které řeší problém.
   - *Příklad:*
     - 6 faktory s maximálně 3 úrovněmi,
     - standardní pole L_{18}, které má 1 faktor s 2 úrovněmi a 7 faktorů se 3 úrovněmi.
     - $2^1 \times 3^7$
2. Vytvoř zobrazení mezi standardním polem a daným problémem:
   - Úrovně každého faktoru jsou zobrazeny do čísel standardního pole.
   - *Příklad:*
     - Hierarchie třídy *A*:

Hodnoty domény	A	B
Hodnoty pole	1	2

     - Stavy hierarchie třídy *P*

Hodnoty domény	P,State1	P, State2	P, State2
Hodnoty pole	1	2	3

Vzorky ortogonálních polí [Mon91, Ros88, LW89, SPK+99]

$L_4(2^3)$

	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

$L_8(2^7)$

	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Ortogonální pole $L_{18}(2^1 \times 3^7)$ [Mon91, Ros88, LW89, SPK+99]

$L_{18}(2^1 \times 3^7)$

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	2	2	2	2	2	2
3	1	1	3	3	3	3	3	3
4	1	2	1	1	2	2	3	3
5	1	2	2	2	3	3	1	1
6	1	2	3	3	1	1	2	2
7	1	3	1	2	1	3	2	3
8	1	3	2	3	2	1	3	1
9	1	3	3	1	3	2	1	2
10	**2**	**1**	**1**	**3**	**3**	**2**	**2**	**1**
11	2	1	2	1	1	3	3	2
12	2	1	3	2	2	1	1	3
13	2	2	1	2	3	1	3	2
14	2	2	2	3	1	2	1	3
15	2	2	3	1	2	3	1	3
16	2	3	1	3	2	3	2	1
17	2	3	2	1	3	1	2	3
18	2	3	3	2	1	2	3	1

Příklad použití OATS [SPK+99]

- $L_{18}(2^1 \times 3^7)$
- Zobrazení (sloupce . . . faktory):
  1. Hierarchie třídy A,
  2. Stavy hierarchie třídy A,
  3. Stavy třídy P,
  4. Hierarchie třídy C,
  5. Stavy hierarchie třídy C,
  6. poslední tři sloupce jsou ignorovány.
- 10: 2 1 1 3 3 2 2 1
- použij instanci třídy B ve stavu 1 k poslání zprávy s instancí třídy P ve stavu 1 instanci třídy E ve stavu 3.

Literatura I

N. Logothetis and H. P. Wynn.

Quality through Design, Experimental Design, Off-line Quality Control and Taguchi's Contributions.
Clarendon Press, Oxford, 1989.

Brian Marick.

The Craft of Software Testing, Subsystem Testing, Including Object-Based and Object-Oriented Testing.
Prentice Hall, 1995.

Douglas C. Montgomery.

Design and Analysis of Experiments.
John Wiley and Sons, third edition, 1991.

Phillip J. Ross.

Taguchi Techniques for Quality Engineering, Loss Function, Orthogonal Experiments, Parameter and Tolerance Design.
McGraw-Hill Book Company, 1988.

Hans Schaefer, Martin Pol, Tim Koomen, Gualtiero Bazzana, Lee Copeland, Hans Buwalda, Geoff Quentin, Mark Fewster, Lloyd Roden, Ruud Teumissen, and Erik Jansen.
Software testing training week.
SQE Europe, Tuesday 28 September to Friday 1 October 1999, Residence Fontaine Royale, Amstelween, Netherlands, Oct 1999.

Objektově-orientované systémy [SPK+99]

Objekt je specifická entita nebo koncept, který má význam v doméně aplikace.

Třída je definice množiny možných objektů sdílející stejné atributy, chování a vztahy.

Atribut je datová hodnota definující objekt a mající význam v doméně aplikace.

Chování je služba poskytována objektem.

Metoda je implementace chování v daném objektově-orientovaném programovacím jazyce.

Zpráva je poslána objektu za účelem vyvolání metody.

Abstraktní třída reprezentuje důžitý koncept aplikační domény na abstraktní úrovni. Jako taková, specifikuje rozhraní (protokol), které ostatní třídy musí dodržovat.

Konkrétní třída je třída, jejíž každá metoda má implementaci, ať už definovaná v této třídě nebo získané z jiné třídy.

OO vztahy [SPK+99]

Vazba je fyzická nebo koncepční strukturální vztah mezi objekty.
- *Příklad:* Mark **vlastní** Spota.

Asociace je definice množiny vazeb se stejným jménem, strukturou a významem.
- Vazba je instance asociace.
- *Příklad:* Osoba **vlastní** psa.

Kardinalita specifikuje, kolik objektů jedné třídy je asociováno s jedním objektem jiné třídy.

Agregace je vztah celku a části mezi objekty.

Zobecnění znamená odstranění detailní informace, méně častých vlastností z popisu.

Specializace znamená přidání detailní informace, více specifické vlastnosti do popisu.

Dědičnost je vztah zobecnění/specializace mezi třídami.
- Třída definovaná pomocí jiných tříd.
- **Striktní dědičnost** neporušuje vztah **is-a**.

Invariant třídy [SPK+99]

- Tvrzení, které o třídě platí stále.
- *Příklad:*
  - Položky umístěné v nějakém pořadí do fronty jsou vráceny nezměněné ve stejném pořadí.

Polymorphismus [SPK+99]

- **Polymorphismus** je schopnost poslat zprávu objektu, aniž je známa specifická třída cílového objektu.
- Každý objekt zná svou třídu. Takto ví, jak se má zachovat zprávy jemu zaslané.