IV113 Validace a verifikace

Organizace kurzu, motivace, přehled technik

Jiří Barnat

Validace versus verifikace

Proces validace

- Ověření, že systém nabízí chtěnou/požadovanou funkcionalitu.
- Ověření, že model reality je v požadovaných aspektech věrný.

Proces verifikace

- Ověření, že poskytované funkce systému pracují dle očekávání.

Anotace

Cílem předmětu je seznámit studenty s

- Základním rozdělením verifikačních metod.
- Technikami testování.
- Vybranými technikami formální verifikace.
- Využitím řešičů SAT a SMT v oblasti formální verifikace.

Organizace kurzu

Místo a čas konání (podzim 2016)

- středa 16:00-17:40, A318

Ukončení předmětu

- Ústní zkouška
- Hodnocení A navíc vyžaduje vypracování všech domácích úloh.

Zdroje

Testování

- http://www.testingeducation.org/
- Cem Kaner, James Bach, Bret Pettichord:
  Lessons Learned in Software Testing
- Cem Kaner, Jack Falk, Hung Quoc NGuyen:
  Testing Computer Software

Formální verifikace

- Edmund M. Clarke, Orna Grumberg, Doron Peled:
  Model checking
- D. A. Peled:
  Software Reliability Methods
- ... a další ...

Motivace V&V

Vývoj kvalitních systémů

Marketing

- Lepší kredit a důvěra u zákazníků.
- Osobní uspokojení a prestiž.

Ekonomické dopady nekvalitního produktu

- Náklady spojené s nápravou chyb v době vývoje produktu.
- Náklady spojené s podporou po dodání zákazníkovi.
- Náklady na soudní výlohy v případě právních dopadů.
- ...

Vývoj kvalitních produktů

- Chyby ve vyvíjeném produktu snižují jeho kvalitu.
- Procedury pro detekci a prokázání absence chyb.
- **Validace a Verifikace**

Nejdražší/nejzávažnější počítačové chyby

Therac-25 (1985)

- Přistroj pro léčbu ozařováním
- Therac-25 ozařoval ve 2 režimech
  1) několika vteřinové ozařování o malé intenzitě
  2) násobný výboj o velké intenzitě
- při rychlém zadávání příkazů na klávesnici došlo k race-condition
- 5 lidí zemřelo po léčbě, kdy byli ozáření po dobu několika vteřin velkou intenzitou záření

Nejdražší/nejzávažnější počítačové chyby

Ariane 5 (1996)

- nosná raketa ESA, byla zničena 40 vteřin po startu
- následník Ariane 4, používala software z Ariane 4
- A5 dosahovala při startu 5x vyššího zrychlení než A4
- hodnoty se dostaly mimo očekávaný rozsah a při konverzi 64-bitového desetinného čísla na 16-bitové celé došlo k aritmetickému přetečení
- rutina, která měla tuto výjimku ošetřit, byla z důvodů efektivity kódu vypnuta
- A5 se sebe-zničila
- `http://www.youtube.com/watch?v=kYUrqdUyEpI`

Nejdražší/nejzávažnější počítačové chyby

Mars Climate Orbiter (1999)

- měl obíhat Mars ve výšce 150km
- klesl do výšky 57 km, kde byl zničen
- důvod: 2 spolupracující jednotky navigačního podsystému pracovaly jedna v metrických jednotkách a druhá v imperiálních

Nejdražší/nejzávažnější počítačové chyby

Výpadek sítě AT&T (1990)

- přetížil se jeden uzel sítě následkem čehož se "rebootoval"
- před každým "rebootem" však uzel informoval sousední uzly
- zpráva však díky softwarové chybě způsobila "reboot" adresáta
- výsledek: celá síť AT&T se s frekvencí 6 vteřin kompletně restartovala
- fix: inženýři nahráli předchozí verzi SW

Výpadek proudu v Severní Americe (2003)

- ztráty 6 miliard USD
- 50 milionů lidí bez dodávky elektřiny
- důvod: race condition

Nejdražší/nejzávažnější počítačové chyby

Pentium FDIV bug (1994)

- nesprávné výsledky při dělení desetinných čísel
- způsobeno nevyplněnou tabulkou v matematickém koprocesoru
- celková škoda 500 milionů USD

2001: Vesmírná odysea

- průzkumná mise k Saturnu
- palubní počítač měl nekonzistentní specifikaci svých úkolů
  - nesdělit členům posádky pravý účel cesty
  - nezatajovat posádce žádné informace
- počítač dospěl k logickému řešení nekonzistence

Nejdražší/nejzávažnější počítačové chyby

Pentium FDIV bug (1994)

- nesprávné výsledky při dělení desetinných čísel
- způsobeno nevyplněnou tabulkou v matematickém koprocesoru
- celková škoda 500 milionů USD

2001: Vesmírná odysea

- průzkumná mise k Saturnu
- palubní počítač měl nekonzistentní specifikaci svých úkolů
  - nesdělit členům posádky pravý účel cesty
  - nezatajovat posádce žádné informace
- počítač dospěl k logickému řešení nekonzistence
- **vyvraždil posádku**

Souvislost s oborem “PDS” @ FI

Pozorování

- Netriviální množství skutečně zákeřných a drahých chyb v systému vzniká v důsledku nespecifikovaných či nejasně specifikovaných spojení jednotlivých částí systému a následně pak zpracováním neočekávaných vstupních dat.

Paralelní/distribuované počítačové systémy

- Distribuované webové aplikace
- Systémy vystavěné z komponent
- Vícevláknové aplikace
- Řídící a operační systémy
- Bezpečnostní a jiné komunikační protokoly
- Vestavné systémy (Embedded systems, HW-SW co-design)
- ...

Kontext IV113 na FI

Verifikace

- IV022 Návrh a verifikace algoritmů
- IA159 Formal Verification Methods
- IA040 Modální a temporální logiky procesů
- IV101 Seminář z verifikace

Formální analýza, Modelování, Simulace, ...

Vývoj SW

Výzkum na FI související s verifikací

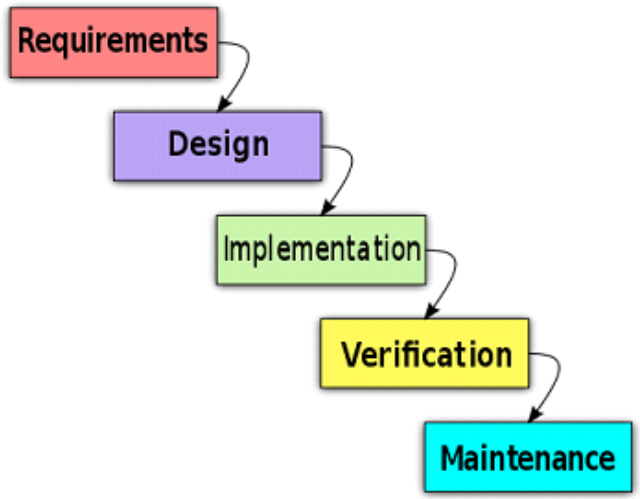
- Laboratoř paralelních a distribuovaných systémů (ParaDiSe)
- Laboratoř formálních metod (ForMeLa)

V&V ve vývojových modelech

Fáze vývoje systému

Fáze

- Sběr požadavků (Requirements)
- Design
- Implementace
- Testování
- Běh u zákazníka a údržba systému



Vztahy a produkty jednotlivých fází

- Fáze logicky navazují a probíhají v daném pořadí.
- Fáze nemusí být ve vývojovém modelu konkrétního výrobce explicitně identifikovatelné.
- Nezávislé činnosti mohou probíhat "mimo pořadí", např. vývoj testovacího prostředí může předcházet samotné implementaci.

Poučka o ceně za nekvalitní produkt

Pozorování

- Čím delší je doba mezi okamžikem zanesení chyby do systému a jejím odhalením, tím je případná realizace nápravy chyby nákladnější.

Relativní cena nápravy chyby

Chyba zanesena	Chyba odhalena				
	Req.	Design	Impl.	Test.	U zákazníka
Requirements	1	3	5	20	100
Design		1	10	25	100
Implementace			1	4	50
Testování				1	10

Testování v modelech vývoje software

Vodopád/V-model

- Testování je poslední fáze.
- Ve V-Modelu je Testování hierarchicky členěno
  - Testování modulů (jednotek) systému
  - Integrační testování
  - Systémové testování
  - "Acceptance" testování

Agilní metody (iterativní metody vývoje)

- Ranné testování
- Vývoj testů často předchází samotné implementaci
- Dominuje testování nových modulů
- Integrační a systémové testování utlumeno
- "Acceptance testy" při vstupu do další iterace

Sekce

Techniky pro zvyšování kvality

Procedurální postupy

Snížení rizika zanesení chyb

- Povinnost formalizace požadavků.
- Model Driven Development (automatické generování kódu).
- Programování ve dvou.
- "Štábní" kultura a disciplína.

Snížení rizika neodhalení zanesených chyb

- Předepsané verifikační kroky ve vývoji systému.
- Požadavky na výsledky verifikačních kroků.

Snížení rizika opomenutí nápravy odhalených chyb

- Nástroje pro sledování nalezených chyb (Bugzilla, Trac).

Metody zvyšování kvality produktu

Neformální metody

- Technická revize
- Ladění a desk-checking
- Simulace
- Runtime analýza
- Testování

Formální metody

- Deduktivní verifikace (Dokazování)
- Statická analýza a Abstraktní Interpretace
- Symbolická exekuce
- Model Checking

Technická revize

Technická revize

- Člověkem prováděná analýza daného artefaktu za určitým předem stanoveným cílem.
- Revize se (ideálně) účastní několik osob v různých rolích.
- Hlavní role jsou moderátor, sekretář a tým recenzentů.

Studované Artefakty

- Popis specifikace, část zdrojového kódu, výsledek provedeného testu, ...

Možné cíle revize

- Detekce chyb či jiných důvodů nízké kvality produktu.
- Nesoulad se standardem či se specifikací, ...

Průběh technické revize

Sezení

- Samotná revize probíhá na sezení.
- Jednotlivé námitky recenzentů jsou postupně prezentovány na sezení a diskutují se s autory analyzovaného artefaktu a ostatními recenzenty.
- Cílem revize není hledat řešení na nalezené problémy, pouze poukázat na jejich existenci. Po dobu sezení se analyzovaný artefakt nesmí měnit.
- Nalezené problémy jsou sekretářem zaznamenány ve zprávě o revizi.

Mimo sezení

- Před sezením by měli recenzenti mít dostatek času na analýzu revidovaného artefaktu.
- Po sezení je možné stanovit priority nalezených problémů a případně stanovit termín další revize.

Varianty technické revize

Walkthrough

- Autor revidovaného artefaktu je přítomen sezení.
- Autor moderuje sezení a prochází analyzovaný artefakt.

Inspekce

- Opakování walkthrough dokud není dosaženo přijatelně nízkého počtu nově objevených problémů.
- Každé sezení končí rozhodnutím, zda je třeba další iterace.

Audit

- Nezávislá revize provedená externí skupinou.
- Objektivní, úplná, systematická a dokumentovaná.
- Cílem auditu je sběr podkladů, na základě kterých lze argumentovat, že revidovaný artefakt splňuje stanovená kritéria auditu.

Výhody a nevýhody technické revize

Výhody revizí

- Aplikovatelné na artefakty, které nelze podrobit automatizované analýze.
- Lidská kreativita v identifikaci problémů.

Nevýhody revizí

- Úspěšně provedená revize negarantuje skutečný stav věci.
- Drahá metoda.

Automatizovaná podpora

- Revize je manuální činnost, nástrojová podpora je zejména procedurální, tj. možná pro roli sekretáře.

Desk-Checking

Desk-Checking

- Metoda spojená s kontrolou algoritmických aspektů vyvíjeného produktu.
- Označuje aktivitu, kdy autor mentálně vykonává jednotlivé kroky algoritmu nad konkrétní sadou vstupních dat.
- Walkthrough, ve kterém si je autor sám sobě recenzentem.

Ladění

- Desk-Checking s využitím softwarové podpory – debugger.
- Typicky se používá pro odhalení původu neočekávaného chování.

Simulace

Simulace

- Imitace chování vyvíjeného produktu s využitím modelu.
- Pozorování jednoho běhu systému za účelem potvrzení očekávaného chování, a nebo pro účely zjištění nových charakteristik chování systému v dané situaci.

Vlastnosti techniky

- Provedení vyžaduje imitační prostředí a nástroje.
- Model musí popisovat behaviorální složku produktu.
- Typické pro vývoj HW částí vestavných systémů.
- Simulací nelze prokázat korektnost, pouze přítomnost chyby.

Runtime analýza

Runtime analýza

- Pozorování chování systému v době jeho skutečného běhu.
- Umožňuje zachytit aspekty, které se neprojevují přímo na vnějším chování (obtížně se testují).
- Realizuje se vložením pozorovatelů přímo do spustitelného kódu (augmentace).

Typické cíle runtime analýzy

- Nekorektní použití dynamicky alokované paměti.
- Omezené možnosti detekce Race-Condition a nesprávného zamykání unikátních zdrojů.
- Detailní profilování výkonu aplikace.
- Kontrola invariantů (assertů), kontrola platnosti vstupních a výstupních podmínek a jejich vztahů.

Testování

- Pozorování chování výsledného produktu, nebo části výsledného produktu nad vybranou množinou vstupních dat.
- Realizováno s různými cíli, například s cílem detekovat chyby.
- Nejčastěji používaná technika verifikace, přestože neschopnost prokázat chybu neznamená, že se v produktu chyba nevyskytuje.

Deduktivní verifikace

Dokazování

- Matematické prokazování vlastností algoritmů.
- V případě úspěšného sestrojení důkazu je dokázaná vlastnost algoritmu skutečně garantována.

Nevýhody

- Nelze algoritmizovat (problém zastavení je nerozhodnutelný).
- Existují vlastnosti programů, které nelze dokázat ani vyvrátit.
- Vyžaduje vysoce kvalifikovaný personál.
- Pokud se nepodaří dokázat nějakou vlastnost programu, není jasné, zda tuto vlastnost program má, či nemá, a pokud ji nemá, není žádné vodítko k důvodu, proč tomu tak je.
- Neadresuje chyby vzniklé implementací.
- Obtížné nasazení nad rámec algoritmizace.

Abstraktní interpretace – Statická analýza programů

Statická analýza programů

- Zjišťování vlastností programů na základě jejich popisu v daném programovacím jazyce, tj. bez nutnosti skutečného spuštění programu.
- Zjištění vlastností jednotlivých uzlů Control-Flow grafu.

Typické použití – překladač

- Detekce mrtvého kódu.
- Detekce použití nedefinovaných proměnných.
- Typová kontrola.

Výhody a nevýhody

- Jednostranně přesné výsledky (neodhalené $\times$ falešné chyby).
- Pokrývá pouze vlastnosti programů, které jsou nezávislé na konkrétních hodnotách vstupních proměnných.
- Aplikovatelná na rozsáhlé projekty.

Symbolická exekuce

Symbolická exekuce

- Vykonávání programu dle Control-Flow grafu s tím, že hodnoty vstupních proměnných jsou označeny a manipulovány symbolicky (constraint solving).
- Pro jeden konkrétní průchod Control-Flow grafem lze postupně vystavět omezení na hodnoty vstupních proměnných, které k tomuto průchodu vedly – **podmínka cesty**.

Typické použití a omezení metody

- Detekce porušení invariantů, assertů, ...
- Syntéza hodnot vstupních proměnných, které vynutí vykonání programu určitým způsobem.
- Počet možných průchodů roste exponenciálně s počtem větvení (path-explosion), to omezuje velikost projektu, na který lze symbolickou exekuci smysluplně aplikovat.

Model Checking

Model Checking

- Verifikovaný systém se abstrahuje do modelu s konečně stavovou sémantikou, pro který se **algoritmicky** prokáže platnost dané vlastnosti systému.
- Prokázání je realizováno prohledáním celého stavového prostoru modelovaného systému.
- Dává garanci, nebo vrací protipříklady.

Omezení

- Nevhodné pro dokazování významových funkcí programů.
- Velikost stavového prostoru roste exponenciálně vzhledem k velikosti domén vstupních proměnných a počtu paralelně prováděných procesů.
- Aplikovatelné pouze na problémy do určité velikosti.

Ukázka runtime verifikace
(valgrind)

Valgrind

Nástroj pro ladění a profilování linuxových programů.

Jak pracuje

- Kód programu se přeloží do HW-neutrálního formátu (intermediate representation, IR)
- Kód v IR se označkuje, tj. doplní se kód, který pozoruje původní kód.
- Označkovaný IR se přeloží do proveditelného kódu dle odpovídající architektury.
- Původní program je vykonáván nativním HW, ale je za běhu pozorován.

Výhody

- Různé moduly valgrindu, mohou provádět různá pozorování.
- Kód je nativně prováděn nikoliv simulován.
- Aplikovatelné na rozsáhlé projekty, např. OpenOffice.

Valgrind – memcheck

Co umí detekovat:

- Neinicializovanou paměť.
- Přístup před, či za alokovaný blok.
- Nepárové volání `malloc/free` a `new/delete`.
- Pokus o uvolnění nealokované paměti.
- Přístup do již uvolněných paměťových bloků.
- Přístup na neoprávněná místa zásobníku.
- Předávání neinicializovaných hodnot systémovým voláním.
- Nesprávné použití `memcpy`.

Valgrind – použití, callgrind, kcachegrind

Spuštění aplikace:

```
my_app arg1 arg2
```

Valgrind a memcheck:

```
valgrind my_app arg1 arg2
```

valgrind a callgrind:

```
valgrind -tool=callgrind my_app arg1 arg2
```

analýza call-grafu:

```
kcachegrind callgrind.out.xxx
```