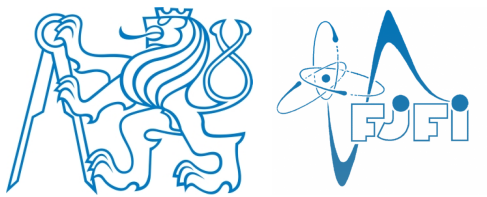
Klasická metodologie testování

Radek Mařík

ČVUT FEL
Katedra telekomunikační techniky, K13132

13. prosince 2017



Obsah

1. Základní terminologie testování
   - Softwarová chyba
   - Ekonomika softwarového procesu
   - Úrovně testování
   - Postupy návrhu testů
2. Základy testování jednotek
   - Principy
   - Architektura a nástroje pro testování jednotek
   - JUnit
3. Kategorie softwarových chyb
   - Chyby uživatelského rozhranní
   - Chyby omezení
   - Procesní chyby
   - Chyby vedení
   - Chyby požadavků
   - Strukturální chyby
   - Datové chyby

6 zásad testování softwaru [Kit95, Het88]

Proces testování softwaru vyjadřuje, jakým způsobem jsou lidé, metody, měření, nástroje a zařízení integrovány za účelem testování softwarového produktu.

1. *Kvalita* testovacího procesu určuje úspěch testovacího úsilí.
2. Zabraň *migraci defektů* použitím technik testování v počátečních fázích vývoje.
3. Je čas začít používat *softwarové testovací nástroje*.
4. Odpovědnost za vylepšování testovacího procesu musí být nesena *lidmi*.
5. Testování je *profesionální disciplína* vyžadující trénované lidi s odpovídajícími vědomostmi.
6. Testování vyžaduje kultivovaný *positivní* postoj týmu ke kreativní destrukci.

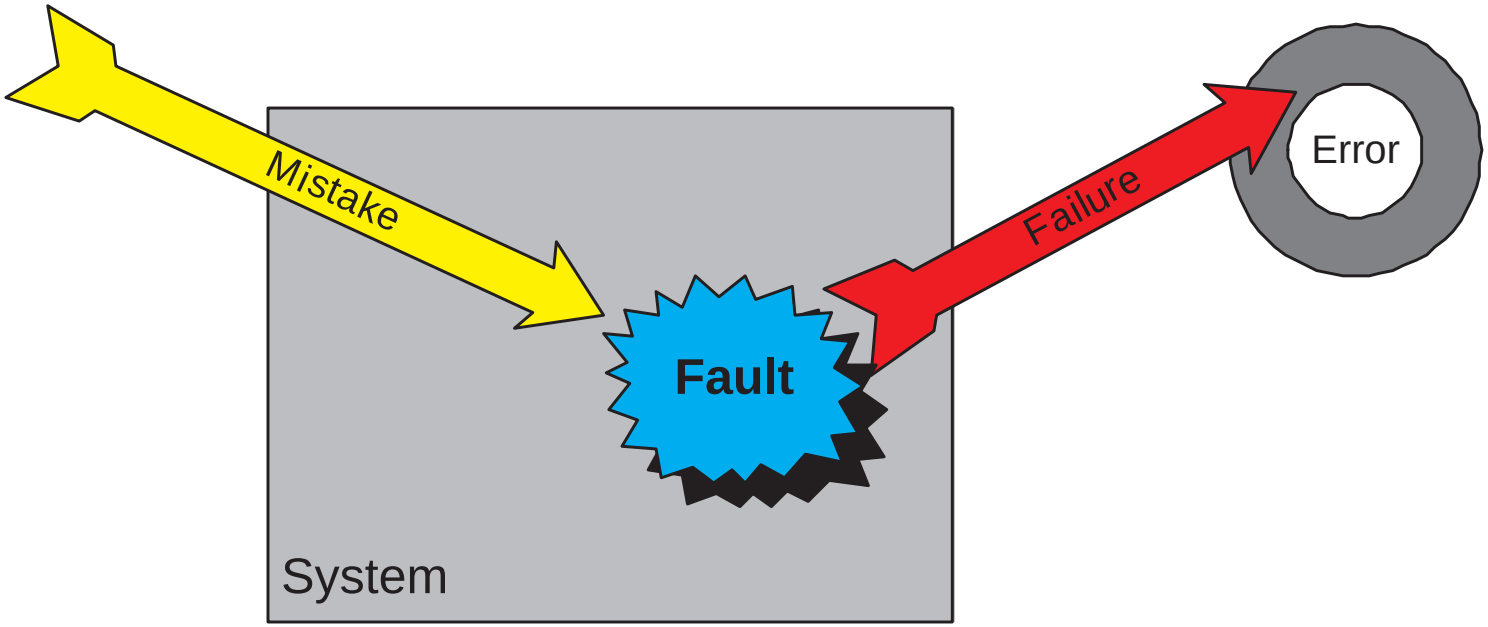
Principiální otázky testování [Kit95, Het88]

- Co by se mělo testovat?
- Kdy by mělo testování začít a kdy skončit?
- Kdo dělá testování?

Co je to softwarová chyba? [KFN93]

- Softwarová chyba je prezentace toho, že program nedělá něco, co jeho koncový uživatel předpokládá (Myers, 1976).
- Nemůže existovat absolutní definice softwarové chyby ani absolutní určení její existence. Míra přítomnosti chyb v programech odpovídá míře, podle které program přestává být užitečný. V základu lidská míra (Beizer, 1984).
- **ŠPATNĚ:** softwarová chyba je nesouhlas mezi programem a jeho specifikací.
  - Nesouhlas mezi programem a jeho specifikací je chybou pouze tehdy a jen tehdy, jestliže specifikace existují a jsou správné.

Softwarové chyby [Kit95]



Pochybení: Akce člověka, která produkuje nesprávný výsledek.

Vada: Nesprávný krok, proces nebo definice dat v počítačovém programu. Výsledek pochybení. Potenciálně vede k selhání.

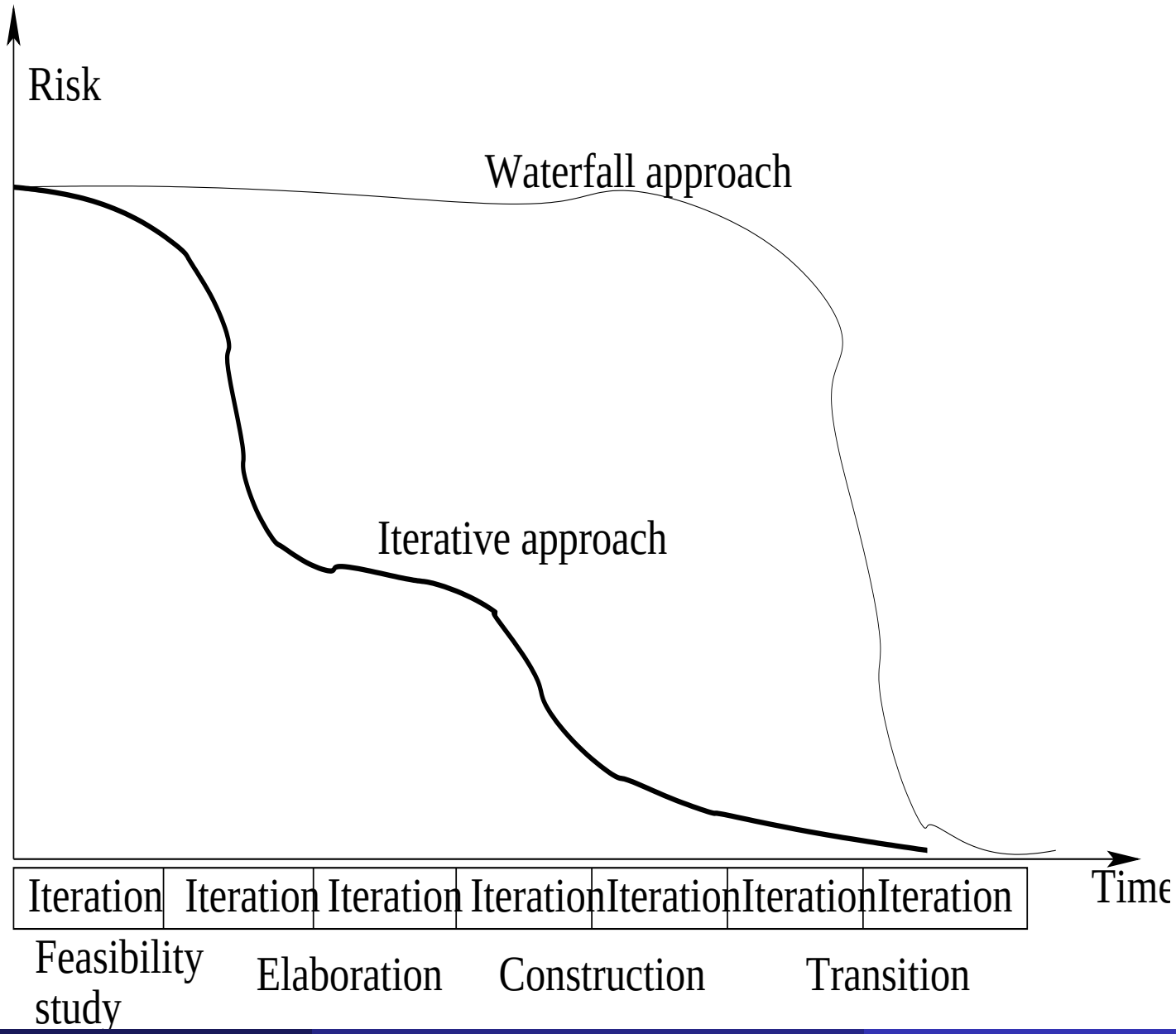
Selhání: Nesprávný výsledek. Projev vady.

Chyba: Kvantitativní vyjádření toho, na kolik je výsledek nesprávný.

Chybná víra testerů [Bei90]

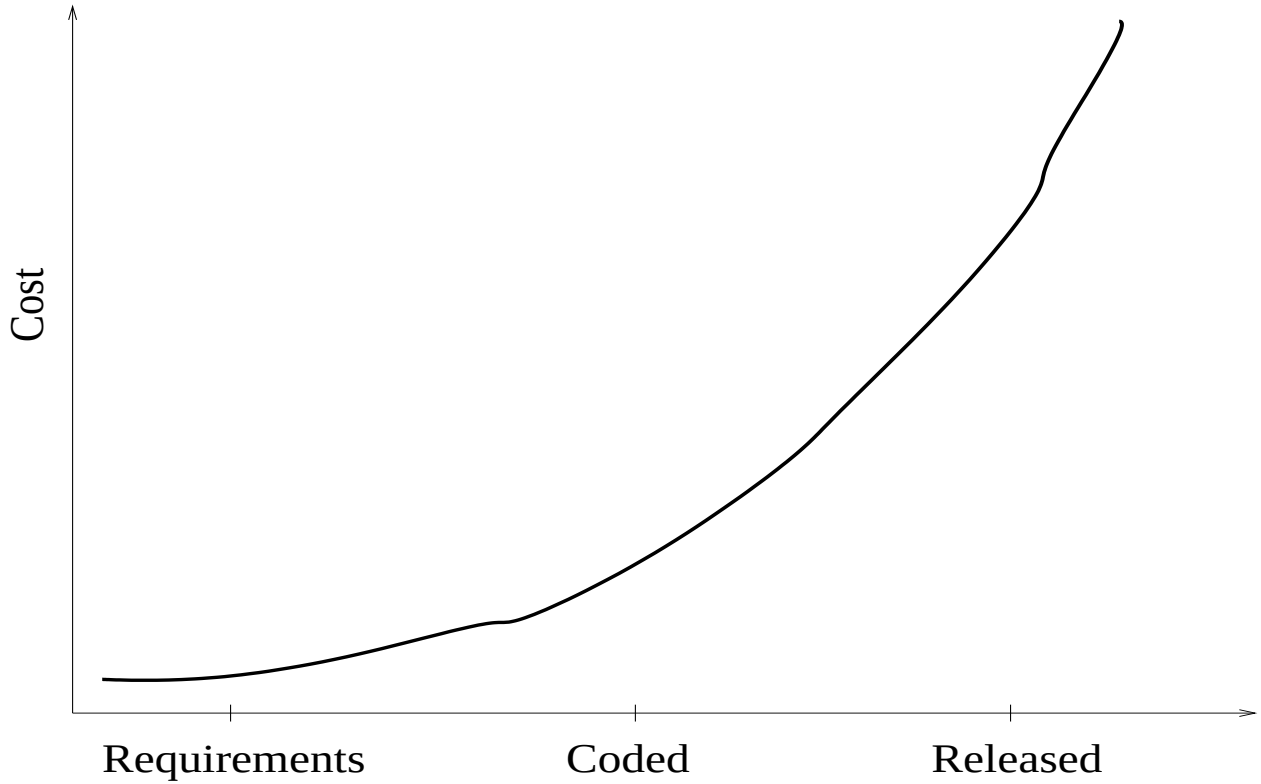
- **Hypotéza laskavých chyb:** chyby jsou krásné, bezduché a logické.
- **Hypotéza lokality chyb:** chyba objevená v nějaké komponentě ovlivňuje pouze chování této komponenty.
- **Dominance chyb v řízení:** chyby v řídicích strukturách převládají (vs. chyby v toku dat a datových struktur)
- **Oddělení kódu a dat:** chyby respektují oddělení kódu a dat.
- **Lingua Salvator Est:** syntaxe a sémantika jazyka eliminuje většinu chyb (vs. prevence).
- **Opravy přetrvávají:** opravená chyba zůstává opravena. (A,B ovlivněné, skutečná chyba je v C)
- **Univerzální všelék:** X (jazyk, návrhová metoda, atd.) zaručuje imunitu vůči chybám,
- **Sadismus postačuje:** k vyhlazení většiny chyb. Obtížné chyby vyžadují metodologii a techniky.
- **Testeři - andělé:** tester je lepší při návrhu testů než programátoři při navrhu kódu.

Riskové řízení [Kru99, Rat99]

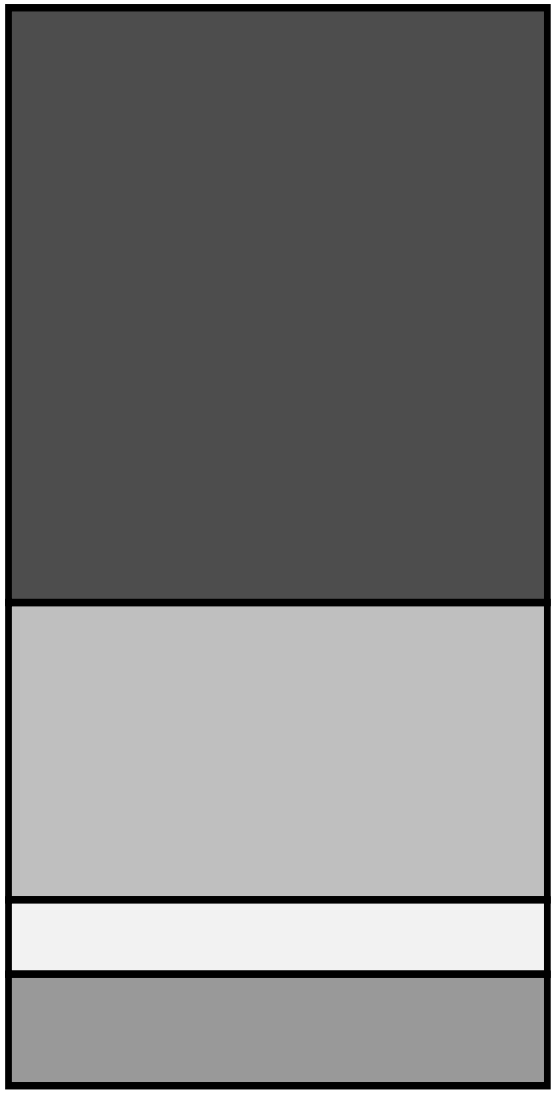


Cena nalezení a opravy chyb [KFN93]

Čím dříve je chyba nalezena a opravena, tím je levnější.



Distribuce chyb



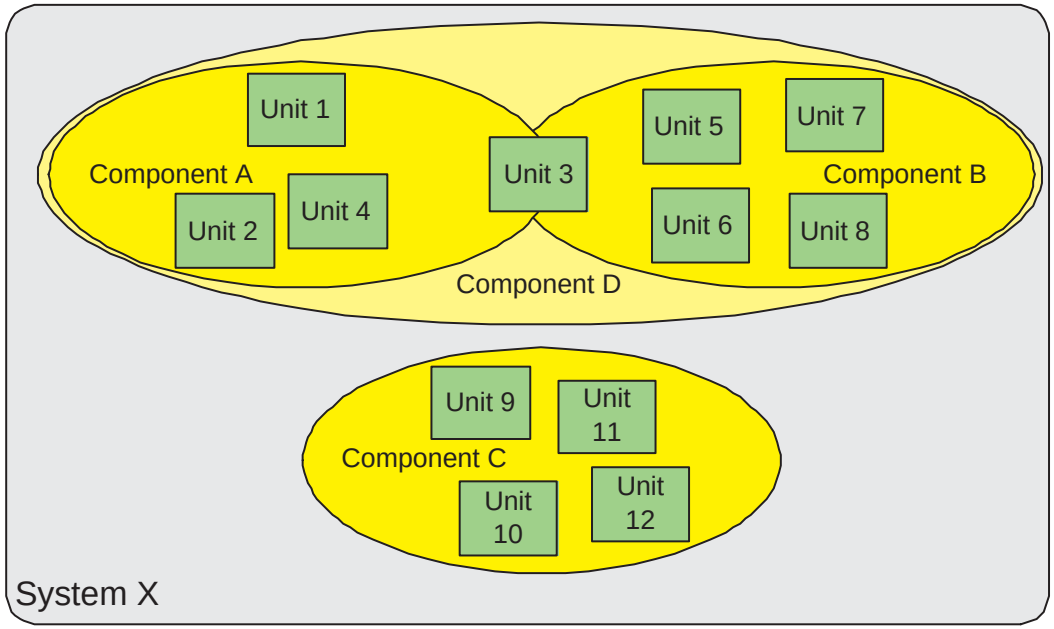
56 % Requirements

27 % Design

7 % Code

10 % Other

Co lze testovat? [Bei90]



Jednotka je nejmenší testovatelný kus softwaru. Znamená to, že může být přeložen, sestaven, spuštěn a řízen testovacím přípravkem nebo řadičem.

Komponenta je integrovaný agregát jedné a více jednotek.

Systém je velká komponenta obvykle odpovídající celému produktu.

Úrovně testování [Bei90]

Testování jednotek - funkční a strukturní požadavky na úrovni jednotky,

Testování komponent - požadavky na úrovni komponenty,

Integrační testování - za předpokladu funkčních komponent testování kombinace komponent,

Testování systému - zabývá se problematikou chování, ke kterému dochází v plně integrovaném systému.

Typy testování [Het88]

Formální testování je proces provádění testovacích aktivit a hlášení výsledků testů podle odsouhlaseného testovacího plánu.

Akceptační testování je formální testování prováděného za účelem stanovit, zda systém splňuje akceptační kritéria a umožňuje zákazníkovi určit zda přijme systém či nikoliv.

Systémové testování je proces testování integrovaného systému za účelem ověření, zda vyhovuje specifikovaným požadavkům.

Regresní testování je částečné testování s cílem ověřit, že provedené modifikace nezpůsobují nechtěné vedlejší efekty nebo že modifikovaný systém stále splňuje požadavky.

Hodnocení výkonnosti - určení dosažení efektivnosti operativní charakteristiky.

Revize [KFN93]

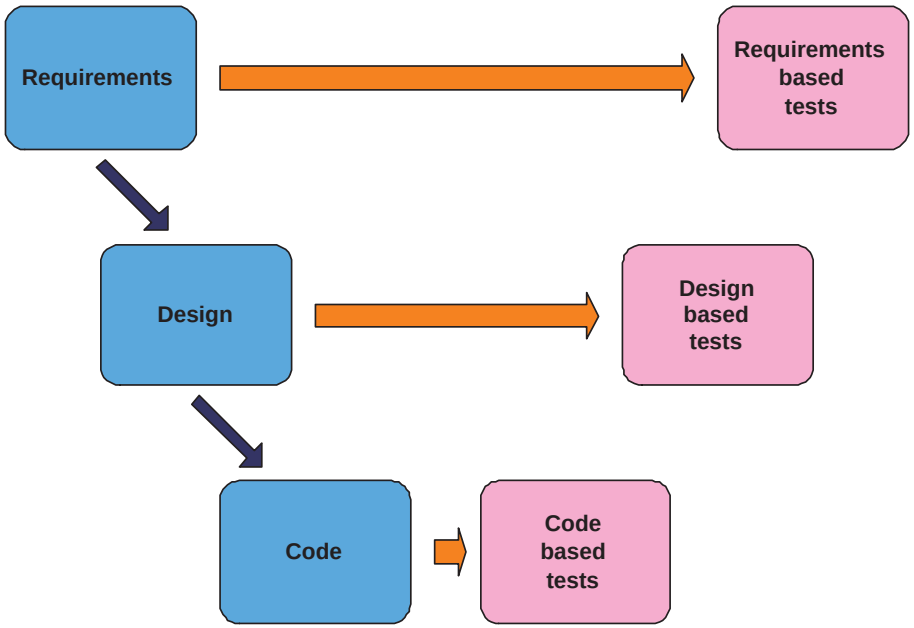
- identifikace problémů v návrhu,
- okolo 7 lidí.

Inspekce - formální hodnotící technika zahrnující detailní prozkušování člověkem či skupinou jiným než autorem. Inspektoři kontrolují každou řádku návrhu proti každé položce kontrolního seznamu.

Demonstrace - inspekční proces, při kterém návrhář ukazuje ostatním pomocí simulace část návrhu nebo kódu, který napsal.

Technická porada - každý přinese seznam problémů. Účelem schůzky je vytvořit seznam problémů a zajistit, aby návrháři všemu rozuměli. Konečná rozhodnutí nejsou součástí této schůzky.

Vstupy návrhu testů [Het88, KFN93, Bei95]



Návrh testů

založený na požadavcích ... z externí specifikace,

založený na návrhu ... z architektury softwaru,

založený na kódu ... ze zakódované logiky a datových struktur.

Návrh testů [Het88, KFN93, Bei95]

Testování černé skříňky funkcionální testování:
strategie testování chování založené na požadavcích, program se chápe jako černá skříňka.

- Testování funkcí: funkce jsou testovány předložením vstupů a prověřováním jejich výstupů. Interní struktura programy se uvažuje pouze zřídka.

Testování bílé skříňky testování skleněné skříňky:
strategie testování struktur odvozených ze struktur testovaných objektů. Programátor využívá znalosti a přístup ke zdrojovému kódu k vývoji testovacích případů.

- Strukturální testování: Hlavní důraz je kladen na vhodný výběr cest skrz program nebo podprogram, které se procházejí při provádění sady testů.

Terminologie přípravy testů [Het88, Bei90]

Požadavek - podmínka nebo schopnost, kterou uživatel potřebuje k řešení problému nebo vyřešení úlohy.

Specifikace - vyjádření množiny požadavků, kterým by měl produkt vyhovět.

Testovací plán - dokument popisující zvolený přístup k zamýšleným testovacím aktivitám.

Testovací případ - specifická množina testovacích dat společně s očekávanými výsledky vztažené k vybranému cílu testu.

Návrh testu - výběr a specifikace množiny testovacích případů, které splňují úlohu testu nebo kritéria pokrytí.

Dobrý test - nezanedbatelná pravděpodobnost detekce dosud neobjevené chyby.

Úspěšný test - detekuje dosud neobjevenou chybu.

Terminologie testování [Het88, Kit95]

Testovací data - vstupní data a podmínky pro soubory asociované s daným testovacím případem.

Očekávané výsledky - predikované výstupní data a podmínky souborů asociované s daným testovacím případem.

Orákulus je jakýkoliv program, proces nebo objem dat, které specifikují očekávaný výsledek množiny testů, pokud jsou aplikovány na testovaný objekt.

Testovací procedura - dokument definující kroky směřující k pokrytí alespoň části testovacího plánu nebo běhu množiny testovacích případů.

Záznam testu - chronologický záznam všech význačných podrobností testovací aktivity.

Platnost testu - stupeň, jak dalece test dosahuje specifického cíle.

První kolo testování [KFN93]

1. Začni se zřejmým a **jednoduchým testem**.
2. Poznamenej si, co dále je potřeba testovat:
   - Hledej **hraniční podmínky**.
   - Typicky se chyby nacházejí v blízkosti hranic.
3. Zkontroluj **platné případy** a pozoruj, co se děje.
4. Proveď testování "za letu".
   - Vždy si zapisuj, co jsi udělal a co se děje, pokud provádíš **průzkumné testy**.
5. **Shrň**, co víš o programu a jeho problémech:
   - zpracování chyb,
   - datové typy,
   - skryté hranice.

Plán systémových testů I, Fáze 2, Krok 5 SPH [KJ96]

Příručka softwarového testování (SPH - software process handbook)

- definuje přístup rozložený na fáze,
- Příručka by měla být tak krátká, aby se dala přečíst během jedné hodiny.

- **Účel:** Identifikovat a popsat testy požadované k tomu, aby produkt splnil funkční požadavky, pracoval tak, jak je specifikováno v dokumentaci produktu, a vyhověl jeho technickým omezením.
- **Vstupy:** Specifikace softwarových požadavků, osnova dokumentu.
- **Tým:**
  - Primární - skupina zajištění kvality softwaru,
  - Sekundární - techničtí vedoucí projektu, manažér projektu, manažér dokumentace.

Plán systémových testů II, Fáze 2, Krok 5 of the SPH [KJ96]

- **Úlohy:**
  1. Identifikuj hardwarovou a softwarovou konfiguraci testovacího prostředí.
  2. Popiš instalaci produktů, které se budou testovat a které budou řídit testovací prostředí.
  3. Popiš jednotlivé testovací případy.
  4. Odhadni časový plán a zdroje potřebné pro
     1. vytvoření testovacích případů,
     2. provedení testů,
     3. údržbu testovacích případů a příslušných testů.
  5. Vytvoř návrh plánu systémových testů.
  6. Zreviduj plán.
  7. Identifikuj potencionální problémy.
  8. Vyřeš tyto problémy.
  9. Vytvoř referenční plán systémových testů.
- **Reference:** Příloha F - Vzor plánu systémových testů
- **Výstup:** Plán systémových testů
- **Výstupní kritéria:** Revize and podepsání odpovědnou osobou

Prohlubování testovacího plánu pomocí seznamů [KFN93]

Seznamy je jednoduché vytvořit, problémem bývá úplnost.

- Seznam zpráv a obrazovek vstupů dat.
- Seznam vstupních a výstupních proměnných.
- Seznam vlastností a funkcí.
- Seznam chybových hlášek.
- Seznam souborů programu.
- Seznam kompatibilního hardwaru.
- Seznam kompatibilního softwaru.
- Seznam kompatibilních operačních prostředí.
- Seznam komponent, které nalezne zákazník v krabici.
- Seznam veřejných dokumentů.

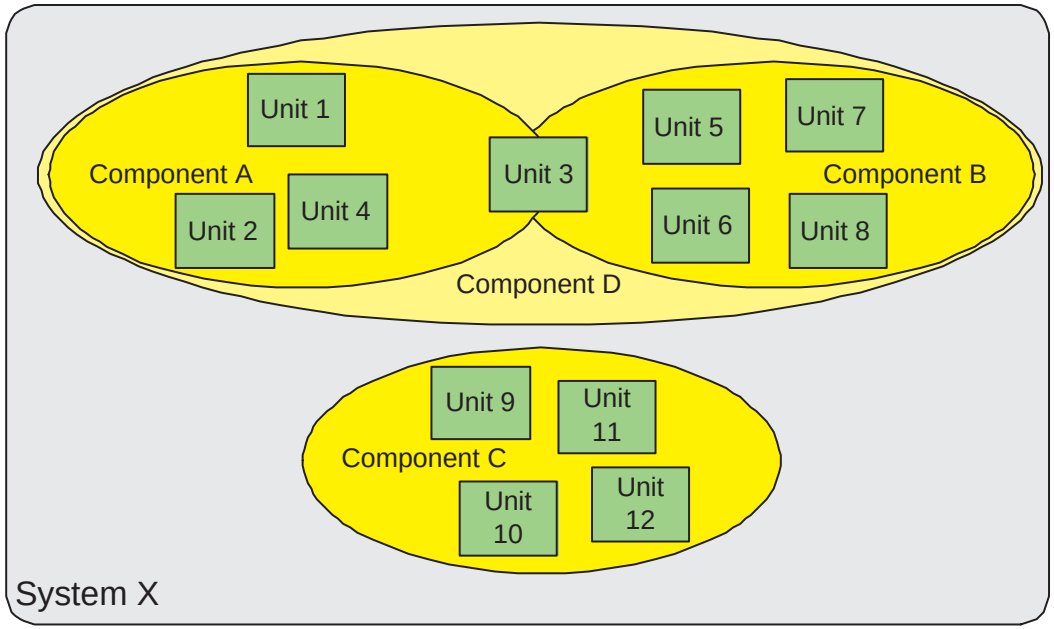
Prohlubování testovacího plánu pomocí tabulek [KFN93]

Tabulky dobře charakterizují vztahu.

- Tabulka zpráv.
- Tabulka vstupních a výstupních proměnných.
- Tabulka vztahu vstupů a výstupů.
- Rozhodovací tabulky a stromy.
- Tabulka kompatibility hardwaru/softwaru.

IF	Rozlišující kód = 3	Y Y Y Y N N N N
	Označeno "Odloženo"	Y Y N N Y Y N N
	Vyřešeno v červnu	Y N Y N Y N Y N
THEN	Zahrň do červnové zprávy	Y N Y N Y N N N
	Zahrň do přehledové zprávy	Y Y Y Y Y Y N N

Terminologie



Jednotka je nejmenší testovatelný kus softwaru. Znamená to, že může být přeložen, sestaven, spuštěn a řízen testovacím přípravkem nebo řadičem.

- **procedurální programování:** program, funkce, procedura.
- **objektově-orientované programování:** třída

Komponenta, Modul je integrovaný agregát jedné a více jednotek.

Cíl

- Cílem je
  - izolování každé části programu,
  - prokázání, že jednotlivé individuální části jsou správné.
  - **otestování dané individuální metody v izolaci.**
- Každý testovací případ je nezávislý na ostatních.
- Testy zaměřující se na chování jiné než určené signaturou metody nejsou považovány za jednotkové.
- Každý test určuje striktní explicitně popsaný kontrakt, který daný kód musí splnit.
- Eliminace interakcí mezi jednotkami umožňuje jejich testování za podmínek, že potřebné další jednotky nejsou jestě implementovány.

Strategie

- Testování jednotek typicky provádí **vývojáři**.
- Testování jednotek je základním pilířem **extrémního programování** (XP).
- Automatizované, opakovatelné, a proaktivní testování.
- **Vývoj řízený testy**
  - Pokud napíšete nejprve testy, pak se hned z počátku ztotožňujete s rolí zákazníka.
  - "Zákazník má vždy pravdu"- prioritní pohled zákazníka.

Typická pravidla

1. Napiš nejprve test, který je možné přeložit (ale ne více).
   - vede k testovatelnému kódu,
   - vede k cílově-orientované implementaci kódu,
2. Nikdy nepiš test, který je bezprostředně úspěšný po jeho napsání.
   - každý test by měl ověřovat novou, ještě neimplementovanou vlastnost,
   - ověření správnosti testu reakcí na implementovanou vlastnost,
3. Začni s prázdným případem, či s něčím, co nepracuje.
4. Neboj se něčeho, co vede na triviální věci, které splní test.
   - testy, které ověřují hraniční jednoduché hraniční podmínky
5. Eliminace interakcí podporuje testovatelnost.
   - vede na udržovatelný kód, se kterým je možné pracovat i za podmínek, že okolní části kódu chybí,
6. Používej imitačních objektů.
   - poskytují jasný pohled na průběh interakcí mezi komponentami.
7. Chodící testy se neodstraňují, tj. jsou spojeny s implementovanou funkcionalitou.

Výhody I

- Umožňuje změny implementace
  - kontroluje stabilitu funkcionality při refaktorování kódu,
  - jednotkové testy odráží zamýšlené použití kódu,
  - dobrý návrh pokrývá všechny použitelné cesty.
- Zesiluje separaci rozhraní od implementace
  - Zaměření pouze na jednotku vyžaduje minimalizaci interakce jednotky s okolím.
  - Případná rozhraní jsou explicitně definována, aby bylo možné prostředí jednotky vytvořit pomocí náhrad.
  - Nalezení a eliminování nadbytečných závislostí mezi jednotkami.

Výhody II

- Zjednodušuje integraci
  - při návrhu a implementaci zdola-nahoru zjednodušuje integrační testy.
- Poskytuje "živou" dokumentaci
  - vývojář může vyčíst funkcionality z testovacího kódu,
  - umožňuje porozumění API jednotky,
  - identifikuje jak
    - positivní chování, tj. cílenou funkcionalitu,
    - tak negativní chování, např. při nevhodných parametrech či zpracování výjimek
  - na rozdíl od běžné dokumentace se vyvíjí současně s implementací kódu.
- Chování identifikované testy podporuje lepší komunikaci s ostatními programátory
  - chování a jeho protokoly jsou explicitně zachyceny,
  - při žádostech o změny selhávající testy ihned identifikují problémy,
  - eliminuje případy použití s vedlejšími skrytými účinky,

Nevýhody, Omezení

- Nechytí všechny chyby programu.
- Testuje pouze funkcionalitu jednotek. Nenalezne chyby
  - integrační,
  - výkonostní,
  - systémové.
- Jako všechny ostatní formy testování
  - může pouze ukázat **přítomnost chyb**,
  - ale nemůže prokázat jejich **absenci**.
- K identifikaci příčin chyb je nutné podpořit systémem řízení změn.

Testovací prostředí

- angl. unit testing framework,
- je software zajišťující testování jednotek,
  - spouštění celých sad testů, případně vybraných částí,
  - spuštění vybraného testu.
- vytváří proměnlivé podmínky testů,
- monitoruje chování jednotek a jejich výstupů,
  - průběžné sledování běhu testů,
  - generovaní reportů
- umožňuje analýzu výsledků,
- skládá se z exekučního modulu a sady testovacích scriptů

Koncepty

Příslušenství (angl. fixture) - sada objektů, které jsou testovány.

Testovací případ (angl. test case)

- třída, která definuje příslušenství pro řadu testů,
- definuje proměnnou pro každou položku příslušenství,
- zaručuje vytvoření a likvidaci příslušenství.

Nastavení (angl. setup) - metoda použitá pro inicializaci proměnných před každým testem či sadou testů.

Úklid (angl. tearDown) - metoda pro uvolnění zdrojů alokovaných nastavením.

Testovací sada (angl. test suite) - soubor testovacích případů.

Řešení chybějících jednotek

Náhrada (angl. test double) - obecný pojem použitý pro jakýkoliv objekt, jehož účelem je doručit funkcionalitu reálného objektů pro účely testování.

Prázdný object (angl. dummy object) - objekt se předává, typicky jako parametr, ale není de facto použit.

Padělek (angl. fake object) - plná funkcionalita řádné implementaci, typicky nevhodná pro nasazení v reálné aplikaci,

- nahrazení reálné databáze jednoduchou databází v paměti.

Imitátor (angl. mock object) - objekty plní případně kontrolují vybranou specifikaci jejich volání,

- typicky specifikují sekvenci volání, tj. verifikují chování

Zástupce (angl. stub) - je schopen dodat odpovědi v omezených případech cílených jednotlivými testy, tj. verifikuje stav.

- mohou zaznamenávat i průběh volání.

Přehled prostředí - vybrané příklady

- Smalltalk
  - Kent Beck publikoval myšlenku prostředí pro testování jednotek v roce 1998. Tato myšlenka a navržený protokol byl pak převzat a implementován řadou dalších (10^2) programovacích prostředí.
  - SUnit
- Java
  - TestNG
  - JUnit
- Python
  - PyUnit
- C++
  - CppUnit
  - CxxUnit
- .NET
  - NUnit
  - Visual Studio Team Edition
- Delphi
  - DUnit

JUnit

- http://www.junit.org
- jednotkové testování pro jazyk Java
- podpora vývojovými prostředky
  - Ant,
  - Maven,
  - NetBeans
  - Eclipse.
- JUnit 3.8, lokalizace testů založena na
  - dědičnosti tříd,
  - reflexi,
  - konvenci jmen.
- JUnit 4.x, založen na vlastnostech Java 5
  - anotace
  - statický import
- JUnit 5.x (September 2017), založen na vlastnostech Java 8
  - meta-annotations, composing of annotations

Architektura JUnit 3.8

- *Příkazová šablona* pro definici testů
  - *TestCase* je příkazový objekt, který obsahuje implementace testovacích metod
  - *testXXX()* definuje testovací metodu (začíná "test"),
  - *assert()* metodu a řadu jejích variant lze použít pro porovnání očekávaných a aktuálních výsledků.
  - *setUp()* a *tearDown()* metody inicializují a ruší společné objektu příslušenství pro každou testovací metodu zvlášť.
- *Kompoziční šablona* pro vytvoření hierarchie testů
  - *TestSuite* definuje hierarchii testů,
  - vytváření testů obvykle automatizováno užitím reflexe a konvencí jmen,
  - postupy se velmi liší,
- *Běh testů*
  - **Textové rozhraní:** java junit.textui.TestRunner junit.samples.AllTests
  - **Grafické rozhraní:** java junit.swingui.TestRunner junit.samples.AllTests

JUnit 3.8 příklad: testovací metody

```
import junit.framework.TestCase;
public class AdditionTest extends TestCase {
private int x = 1;
private int y = 1;

@Override protected void setUp() {
y = 2;}

public void testAddition() {
int z = x + y;
assertEquals(3, z);}

protected void tearDown() {
System.gc();}
......
}
```

JUnit 3.8 příklad: testovací sada

```
import junit.framework.*;
public class AdditionTest extends TestCase {

.......

public static Test suite(){
return new TestSuite(AdditionTest.class);
}

public static void main(java.lang.String[] argList){
junit.textui.TestRunner.run(suite());
}
}
```

Architektura JUnit 4.5

- *Příkazová šablona* pro definici testů
  - testovací třída se neodvozuje z TestCase,
  - *assert()* metodu a její varianty lze použít podobně jako v JUnit 3.8.
  - *@Test* dekorace definuje testovací metodu,
  - *@Before* dekorace označuje metody inicializují společné objektu příslušenství pro každou testovací metodu zvlášť.
  - *@After* dekorace označuje metody rušící společné objektu příslušenství po každé testovací metodě.
  - *@BeforeClass* dekorace označuje veřejné statické metody běžící před třídou.
  - *@AfterClass* dekorace označuje veřejné statické metody běžící po třídě.
  - *@Ignore* dekorace označuje ignorovaný test.

Architektura JUnit 4.5

- *Kompoziční šablona* pro vytvoření hierarchie testů
  - *@RunWith(Suite.class)* dekorace specifikuje třídu testovací sady
  - *@SuiteClasses(TestA.class, TestB.class)* dekorace specifikuje třídy testů patřící do příslušné sady
- *Běh testů*
  - přímá podpora v IDE nebo **ant**
  - lze použít podpory vytvořené v JUnit 3.8 pomocí adaptérů

JUnit 4.5 příklad: testovací metody

```
import org.junit.*;
import static org.junit.Assert.*;
public class AdditionTest {
private int x = 1;
private int y = 1;

@Before public void setUp() {
y = 2;}

@Test public void testAddition() {
int z = x + y;
assertEquals(3, z);}

@After public void tearDown() {
System.gc();}
......
```

JUnit 4.5 příklad: testovací sada

```
import junit.framework.JUnit4TestAdapter; //from 3.8
public class AdditionTest {

.......

public static junit.framework.Test suite() {
return new JUnit4TestAdapter(AdditionTest.class);
}

public static void main(java.lang.String[] argList){
junit.textui.TestRunner.run(suite());
}
}
```

Chyby uživatelského rozhranní - funkcionalita [KFN93]

Funkčnost

- Program má problém s funkčností, jestliže
  - nedělá něco, co by měl dělat nebo
  - to dělá nevhodně, zmatečným způsobem či neúplně,
  - lze některé operace provést obtížně,
- Konečná definice, co se "předpokládá" od programu, žije pouze v mysli uživatele.
- Všechny programy mají problémy s funkčností vzhledem k různým uživatelům.
- Funkční problém je chybou, pokud očekávání uživatele jsou rozumná.

Chyby uživatelského rozhranní - vstupy [KFN93]

Komunikace

- Jak lze nalézt, jak používat daný program?
- Jaká je nápověda, pokud uživatel udělá chybu či spustí $< Help >$?

Struktura příkazů

- Jak snadné je ztratit se v programu?
- Jaké chyby uživatel dělá, kolik je to stojí času a proč?

Chybějící příkazy

- Co chybí?
- Nutí program uživatele přemýšlet nějakým pevným, nepřirozeným nebo neefektivním způsobem?

Chyby uživatelského rozhranní - výstupy [KFN93]

Výkonnost

- Rychlost je základem interaktivního softwaru.
- Cokoliv vyvolává v uživateli pocit, že program pracuje pomalu, je problém.

Výstup

- Získá užitel, co potřebuje?
- Mají výstupní reporty smysl?
- Může uživatel přizpůsobit výstup svým potřebám?
- Lze přesměrovat výstup podle výběru uživatele na monitor, tiskárnu, či do souboru daného formátu?

Chyby omezení [KFN93]

Chyby zpracování vyjímek zahrnují neschopnost

- předvídat možnost chyby bránit se jim,
- zpozorovat podmínky chyby,
- zpracovat detekovanou chybu rozumným způsobem.

Chyby hraničních podmínek

- Nejjednodušší hranice jsou numerické.
- Mezní nároky na paměť, za kterých program může pracovat.

Výpočetní chyby

- Chyby aritmetiky jsou časté a obtížně detekovatelné.
- Program ztrácí přesnost během výpočtu vlivem zaokrouhlovacích chyb a chyb ořezávání.
- Výpočetní chyby způsobené chybnými algoritmy.

Procesní chyby - sekvenční [KFN93]

Počáteční a jiné speciální stavy

- Funkce mohou selhat při prvním použití, např. chybějící inicializační informace či soubory.
- Nastaví se skutečně vše do výchozího bodu, vymažou se všechna data, jestliže uživatel provede reset programu?

Chyby řízení

- Chyba řízení nastane, pokud program provede chybný příští krok.
- Extrémní chyba nastane, pokud se program zastaví či naopak vymkne řízení.

Procesní chyby - paralelní [KFN93]

Chyby souběhu (angl. race errors)

- Jsou jedny z nejméně testovaných.
- Nastávají v multiprocesorových systémech a v interakčních systémech.
- Velmi obtížně se opakují.

Zátěžové podmínky

- Program se začně chovat chybně, pokud se přetíží.
- Spadají sem chyby:
  - velkého *objemu*, tj. hodně práce za dlouhou dobu.
  - velkého *stresu*, tj. hodně práce v daném okamžiku.
- Všechny programy mají své limity. Je však důležité vědět, co nastane.

Chyby vedení I [KFN93]

Hardware

- Programy posílají chybná data na zařízení, ignorují chybové kódy přicházející zpět a zkouší použít zařízení, která neexistují či jsou právě vytížená.

Řízení zdrojů a verzí

- Staré problémy se opět objevují, pokud programátor zakomponuje do programu nějakou starou verzi komponenty.
- Ujistěte se, že program má správný copyright, vstupní obrazovky a čísla verzí.

Chyby vedení II [KFN93]

Dokumentace:

- Slabá dokumentace může způsobit, že uživatel přestane věřit, že software pracuje správně.

Chyby testování:

- Chyby udělané testery jsou nejčastějšími chybami objevenými během testování.
- Jestliže program navádí většinu uživatelů ke způsobení chyb, pak program není správně navržen.

Chyby požadavků, vlatností a funkčnosti [Bei90]

Požadavky a specifikace:

- neúplné, nejednoznačné, nebo vzájemně si odporující,
- hlavní zdroj drahých chyb.

Chyby vlastností:

- chybějící, chybné, nebo nevyžádané vlastnosti,

Interakce vlastností:

- nepredikovatelné interakce (např. přesměrování telefonních volání ve smyčce)

Preventivní opatření proti chybám ve specifikacích a vlastnostech:

- problémy s komunikací člověk-člověk,
- jazyky formálních specifikací poskytují krátkodobé řešení, avšak neřeší problém chyb v dlouhodobém horizontu.

Strukturální chyby I [Bei90]

Chyby v řízení a sekvencích:

- příkazy GOTO, kód ala špagety, kód ala pačinko,
- většina chyb řízení (v novém kódu) se dá snadno testovat a je chycena během testování jednotek,
- neupravený starý kód může mít řadu chyb v řídicím toku,
- stlačování za účelem kratšího prováděcího času nebo menšího nároku na paměť je špatná praktika.

Chyby zpracování:

- zahrnuje chyby vyhodnocení aritmetických, algebraických, či matematických funkcí, výběru algoritmu.
- řada problémy v této oblasti se váže k nesprávným konverzím z jedné reprezentace dat na druhou.

Strukturální chyby II [Bei90]

Chyby logiky:

- neporozumění jak se selekční či logické operátory chovají samostatně nebo v kombinacích,
- neporozumění sémantice uspořádání logických výrazů a jeho vyhodnocení specifickými překladači,
- chyby datového toku: nevztahují se k chybám v řízení,
- chyby toku řízení: část logického výrazu, která je použita pro ovládání toku řízení.

inicializační chyby:

- typické chyby: opominutí inicializace pracovního prostoru, registrů, nebo oblastí dat.

Strukturální chyby III [Bei90]

Chyby a anomálie v toku dat:

- Anomálie toku dat nastane, pokud existuje cesta, při které se udělá s daty něco neodůvodněného, např. použití neinicializované proměnné, použití proměnné, která ještě neexistuje.
- Anomálie datového toku jsou stejně tak důležité jako anomálie toku řízení.

Datové chyby I [Bei90]

Obecně:

- datové chyby lze nalézt ve specifikacích datových objektů, jejich formátů, počtu objektů nebo jejich počátečních hodnotách,
- software se vyvíjí k tabulkám obsahujících řídicí a procesní funkce.
- trendy v programování vedou k zvýšenému používání nedeklarovaných, interních, speciálních programovacích jazyků.

Dynamické versus statické:

- protože efekt poškození dynamických dat se může projevit velmi vzdáleně od příčiny, nalézají se takovéto chyby jen velmi obtížně.
- základní problém zbytků ve sdílených zdrojích (např. vyčištění po použití uživatelem, sdílené čištění pomocí ovladače zdroju, žádné čistění).

Datové chyby II [Bei90]

Informace, parametr, řízení:

- údaj plní jednu ze tří rolí: jako parametr, jako řízení, jako zdroj informace.
- informace je obvykle dynamická s tendencí lokality pro danou transakci (nedostatek ochranného kódu validace dat)
- neadekvátní validace dat často vede k ukazování prstem.

Obsah, struktura, atributy:

- obsah - aktuální bitový vzor, řetězec znaků, nebo číslo vložené do datové struktury,
- struktura - velikost, tvar a počty popisující datové položky.
- atributy - specifikace významu (sémantika),
- základem je explicitní dokumentace obsahu, struktury a atributů všech datových objektů.

Chyby kódování [Bei90]

Charakteristiky

- dobrý překladač chytne syntaktické chyby, nedeklarovaná data, nedeklarované procedury, nedefinovaný kód a mnoho inicializačních problémů,
- častou chybou kódu jsou dokumentační chyby (komentáře).
- úsilí programování je dominováno údržbou.

Chyby správy paměti [Pur99]

Charakteristiky

- nejobtížnější chyby z hlediska lokalizace,
- nejdůležitější chyby z hlediska opravy,
- projevy nesprávného obsahu paměti jsou nepredikovatelné,
- chyby v obsahu paměti se typicky projevují vzdáleně od jejich příčiny.
- chyby zůstávají často nedetekované dokud nejsou náhodně spuštěny.

Typy chyb

- chyby hranic polí,
- přístup přes nedefinovaný ukazatel,
- čtení z neinicializované paměti,
- chyby alokace paměti,
- chyby ztráty paměti (memory leaks).

Slabá místa výkonnosti [Qua99]

- kolekce vyčerpávající přesné množiny dat pro výkonnostní testy programu a každé jeho komponenty (profilování).
- zaměření se na kritická data,
- sběr správně vybraných dat:

  řádka ... počítání kolikrát se každá řádka provedla během běhu programu. Poskytuje nejvíce přesné a detailní údaje, ale vyžaduje nejvíce času ke sběru.

  funkce ... tato úroveň poskytuje méně podrobné údaje než čítání řádek. Je užitečné, pokud se nezabýváme přesnou výkonností jednotlivých řádek.

  čas ... data se sbírají z údajů časovaných běhů funkcí. Data jsou správná pro daný běh, ale závislá na stavu mikroprocesoru a paměti. Nejméně náročná na sběr.

Literatura I

Boris Beizer.
Software Testing Techniques.
Van Nostrand Reinhold, New York, 2 edition, 1990.

Boris Beizer.
Black-Box Testing, Techniques for Functional Testing of Software and Systems.
John Wiley & Sons, Inc., New York, 1995.

Bill Hetzel.
The Complete Guide to Software Testing.
John Wiley & Sons, Inc., second edition, 1988.

Cem Kaner, Jack Falk, and Hung Quoc Nguyen.
Testing Computer Software.
International Thomson Computer Press, second edition, 1993.

Edward Kit.
Software Testing in the Real World.
Addison-Wesley, 1995.

Raymond Kehoe and Alka Jarvis.
ISO 9000-3, A Tool for Software Product and Process Improvement.
Springer, 1996.

Philippe Kruchten.
The Rational Unified Process.
Addison-Wesley, 1999.

Literatura II


Gettin ahead with Rational Purify, pinpoint and eliminate run-time errors.
Rational Software Corporation, 1999.


Gettin ahead with Rational Visual Quantify, pinpoint and eliminate application performance bottlenecks.
Rational Software Corporation, 1999.


Rational software symposium 1999.
Unicorn, Praha, Czech Republic, February 1999.