

Testování software

David Grochol

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
igrochol@fit.vutbr.cz



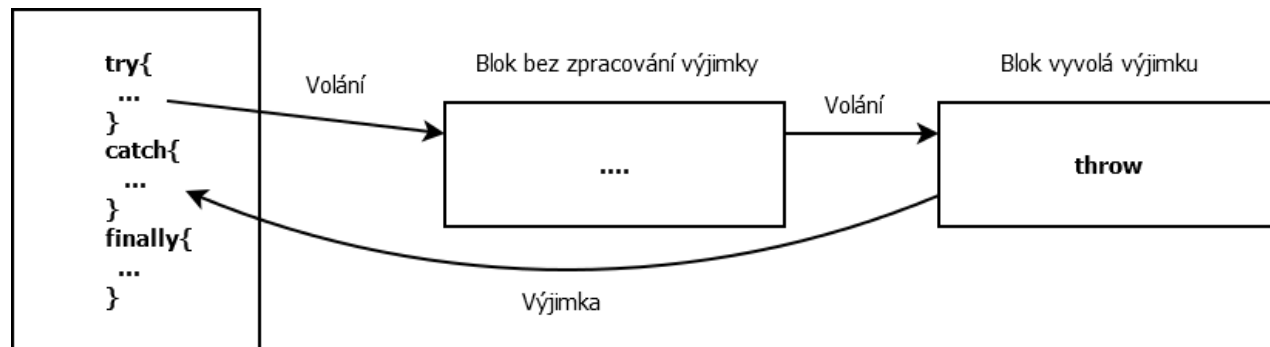
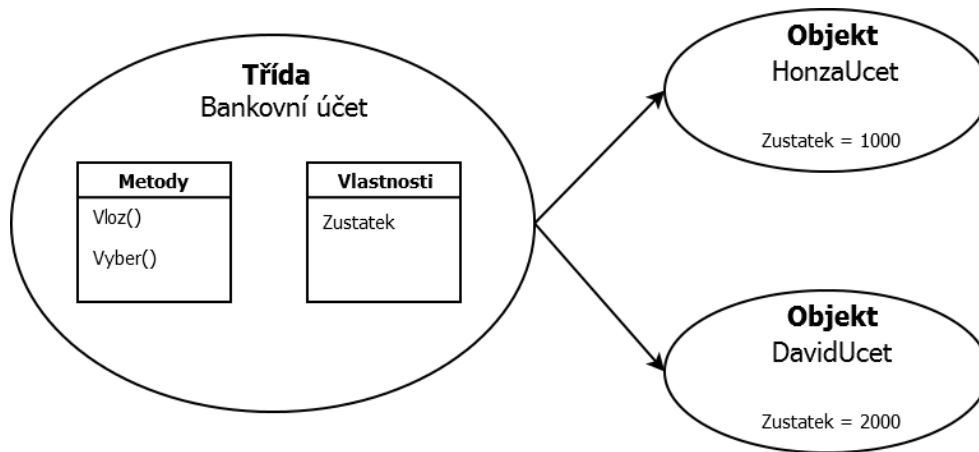
13.02.2018

- **Prerekvizity**
 - OOP
 - Chyby v SW
- **Testování**
 - Základní pojmy
 - Stupně testů
 - Strategie testování
 - TDD
 - BDD
 - Něco navíc
- **Zadání projektu**

PREREKVIZITY – OBJEKTIVĚ ORIENTO VANÉ PROGRAMOVÁNÍ

- **Objektově orientované programování (OOP)**
 - **Třída** – množina objektů se shodnými charakteristikami
 - Vytváříme z ní instance (objekty)
 - **Objekt** je jeden konkrétní jedinec (reprezentant, entita) příslušné třídy
 - Pro konkrétní objekt **nabývají vlastnosti** deklarované třídou **konkrétních hodnot**
 - **Proměnné** – datové typy zapouzdřené v objektech
 - **Metody** – funkce pracující převážně nad proměnnými objektu
 - **Konstruktor** – volá se při vytváření objektu
 - **Destruktor** – volá se při rušení objektu

- **Výjimka** – výjimečná situace, která může nastat za běhu programu (chyba). Objekt, který nese informaci o chybě.
 - Obdoba chybového kódu (např.: v C errno)
 - **Vyhození výjimky (throw)**
 - Pokud nastane chybová situace, je vyhozena výjimka (neexistující soubor, nedostatek paměti, ...).
 - **Zachycení výjimky (catch)**
 - Zpracování vyvolané výjimky; může existovat speciální objekt zpracovávající výjimky.
 - **Dokončení (finally)**
 - Uvolnění systémových zdrojů (i v případě výjimky).
 - Spustí se vždy ať už k výjimce došlo či nikoli.



- Třída:

```
class BankovniUcet
{
public:
    double zustatek;

    BankovniUcet();
    ~BankovniUcet();

    void vloz(double castka);
    void vyber(double castka);
};
```

```
BankovniUcet::BankovniUcet(){
    zustatek = 0;
}

BankovniUcet::~BankovniUcet(){
    zustatek = 0;
}

void BankovniUcet::vloz(double castka){
    if(castka <= 0){
        throw MyException1("castka musi byt vetsi nez 0!");
    }
    zustatek += castka;
}

void BankovniUcet::vyber(double castka){
    if(castka <= 0){
        throw MyException1("castka musi byt vetsi nez 0!");
    }
    else if((zustatek - castka) >= 0){
        zustatek -= castka;
    }
    else{
        throw MyException2("Nedostatek penez!");
    }
}
```

- Výjimky:

```
struct MyException1: public std::runtime_error
{
    MyException1(std::string const& message)
        : std::runtime_error(message)
    {}
};
```

```
struct MyException2: public std::runtime_error
{
    MyException2(std::string const& message)
        : std::runtime_error(message)
    {}
};
```

```
BankovniUcet ucet = BankovniUcet();

try{
    ucet.vyber(-100);
}
catch (MyException1& e){
    std::cout<<"vyjimka MyException1 zachycena: "<<e.what() <<std::endl;
}
catch (MyException2& e){
    std::cout<<"vyjimka MyException2 zachycena: "<<e.what() <<std::endl;
}
catch (std::exception& e){
    std::cout<<"vyjimka zachycena: "<<e.what() <<std::endl;
}
```


PREREKVIZITY – CHYBY V SOFTWARE

Platí alespoň jedno z následujících tvrzení:

1. SW nedělá něco, co by podle specifikace produktu dělat měl
2. SW dělá něco, co by podle specifikace produktu dělat neměl
3. SW dělá něco, o čem se produktová specifikace nezmiňuje
4. SW nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat
5. SW je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo – podle názoru testera SW – jej koncový uživatel nebude považovat za správný

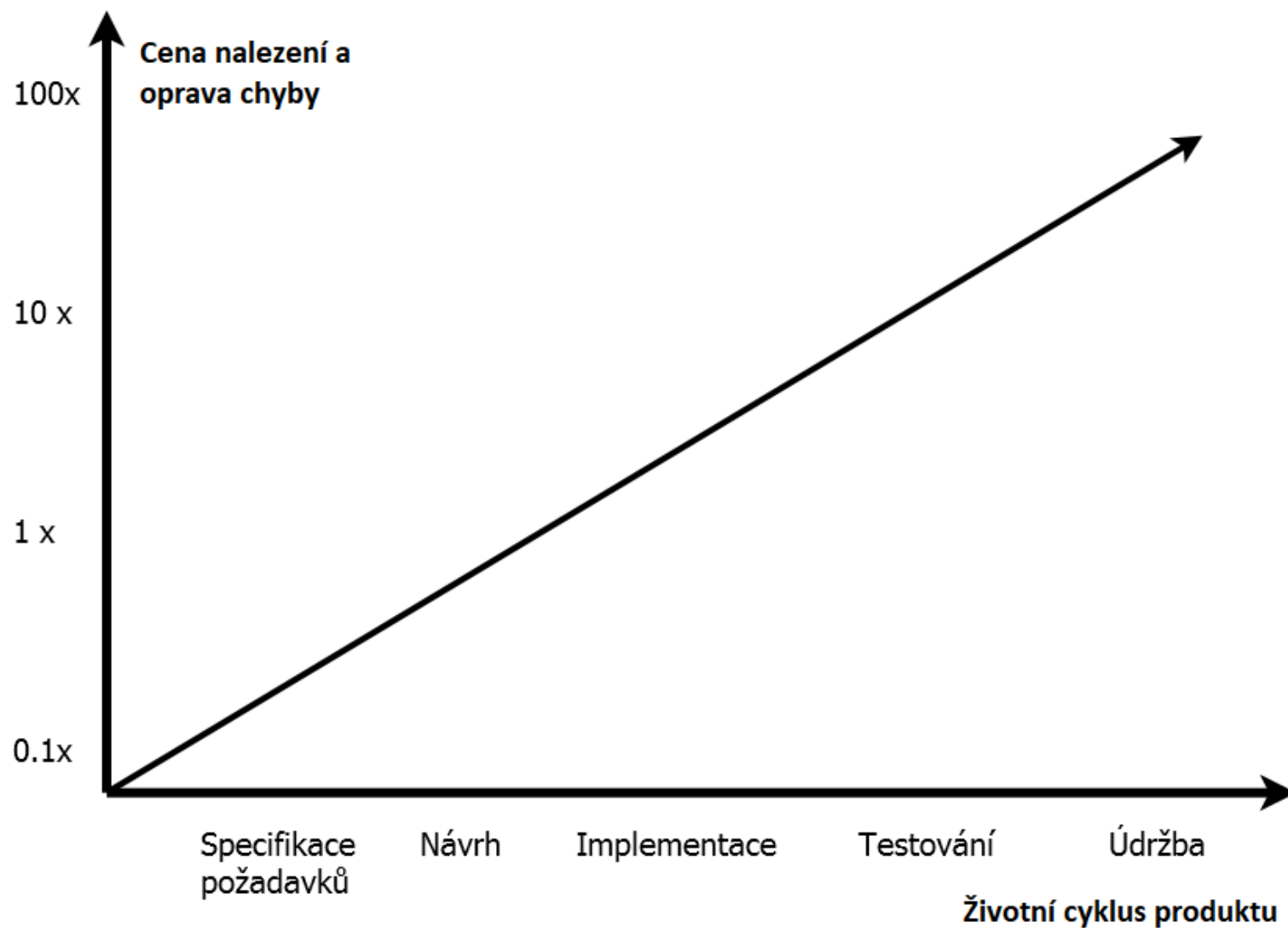
- **Důsledky chyby:**

- Od nepohodlí až po ztráty na životech

- **Dělení s pohyblivou řádovou čárkou Intel (1994)**
 - V některých případech vnikají neočekávané výsledky
 - Softwarová chyba, kterou se podařilo vypálit do počítačového čipu a v procesu výroby ji mnohokrát zopakovat
 - Náklady: přes 400 mil dolarů
- **Přistávací modul na Mars (1999)**
 - Chybné nastavení jednoho datového bitu (signalizace dosednutí). Problém nezjištěn při testech
 - Nebyl testován systém jako celek, ale jen jeho části
- **Rok 2000**
 - Zkracování ukládání data 1989 -> 89, ?? 2000 -> 00 ??
 - Náklady: stovky mld. dolarů

TESTOVÁNÍ





- Verifikace
 - kontrola, zda produkt odpovídá specifikaci
- Validace
 - kontrola, zda produkt odpovídá požadavkům zákazníka



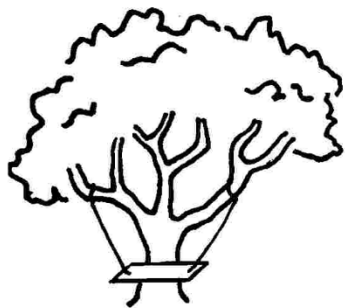
Co je požadováno



Co je uvedeno v projektu



Co navrhne senior analytik



Co je skutečně naprogramováno



Co je nainstalováno



Co zákazník skutečně chce

- Fakta o testování (životní moudra)
 - **Žádný program není možné otestovat kompletně**
 - Velký počet vstupů/výstupů
 - Velký počet cest v SW
 - **Velké riziko**
 - Co není pokryto testy? Co když je tam závažná chyba?
 - **Testování nikdy neprokazuje, že chyby neexistují**
 - Vždy existuje určitá kombinace vstupů, která není testována
 - **Čím více chyb najdete, tím více chyb je v SW**
 - Opakující se chyby, různě se projevující jedna chyba
 - **Ne všechny nalezené chyby se opraví**
 - Není čas, není to chyba, oprava je riskantní, nestojí za to

- **Jednotkové (Unit)**
 - Jednotlivá komponenta
- **Integrační**
 - Několik komponent jako skupina
 - Zaměřené na rozhraní podsystému
- **Systemové**
 - Celý systém s ohledem např. na spolehlivost či výkonnost
- **Akceptační**
 - Splňuje systém požadavky zákazníka?

- **Jednotka**
 - Záleží na paradigmatu daného jazyka - (funkce, objekt, ...)
 - Komponenta třetí strany
 - Testy jednotky probíhají v izolaci od ostatních
 - Cílem je zjistit, zda jednotka funguje podle specifikace
- Koncept **xUnit**

- Framework pro unit testy pro různé jazyky (platformy) postavené na designu SUnit pro Smalltalk.
- Poskytují konstrukce pro vytváření opakovaně spustitelných testů a ověřování očekávaného chování.
- Příklady:
 - cppUnit, Google Test – C++
 - jUnit – Java, Scala
 - NUnit - .NET
 - Python – pyUnit
 - http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

- Test Suite
 - Sada testovacích případů (test case)
 - Např. test všech jednotek v konkrétním systému
- Test Case
 - Shlukuje sadu testů (metod) testujících jednotku
 - Metoda *setup* – zajištění předpokladů (preconditions)
 - Spuštění samotného testu, který obsahuje jedno či více tvrzení (assertions), které mají platit.
 - Metoda *teardown* – uvedení do původního stavu tak, aby nebyly ovlivněny další testy

- Zajištění předpokladů
- Fixture
 - Fixture jsou všechna vstupní data pro xUnit taková, aby bylo možné provést test opakovaně a bez ohledu na aktuální kontext.
 - Množina dat a objektů
 - Tvoří konfiguraci testů
 - Může obsahovat:
 - Mock objekty
 - Vstupní a očekávané hodnoty
 - Inicializace databáze vhodnými hodnotami
 - Vstupní soubory apod.

- Reálné funkce využívají jiné funkce
- Analogicky to platí pro objekty v OOP
- Aby se zabránilo průniku chyb odjinud, nesmí se používat kód jiné jednotky
- Používají se simulace okolí:
 - *Stub (pahýl)* – simuluje jiný kód pro účely testů
 - Vrací předdefinovanou hodnotu
 - Může zaznamenávat své volání
 - *Mock (napodobenina)*
 - Ověřuje chování toho, kdo ho používá
 - Jsou ověřovány interakční scénáře

- EXPECT
 - Pokud test selže, pokračuje dale.
 - Příklad: Testuji různé vstupy, pokud selže pro určitý vstup, můžu pokračovat testováním dalšího vstupu.

- ASSERT
 - Pokud test selže, ukončí ho.
 - Příklad: Selhalo otevření souboru -> ukončím, nemůžu s ním pracovat.

```
class osobniUcet : public ::testing::Test
{
protected:
    BankovniUcet *ucet;

    virtual void SetUp() {
        ucet = new BankovniUcet();
    }
    virtual void TearDown() {
        delete ucet;
    }
};
```

```
TEST_F(osobniUcet, vlozeni){
    EXPECT_NO_THROW(ucet->vloz(100));
    EXPECT_EQ(ucet->zustatek, 100);
}

TEST_F(osobniUcet, vlozeni_err){
    EXPECT_ANY_THROW(ucet->vloz(-100));
}

TEST_F(osobniUcet, vyber_err){
    EXPECT_ANY_THROW(ucet->vyber(1000));
    EXPECT_EQ(ucet->zustatek, 0);
    EXPECT_ANY_THROW(ucet->vyber(-100));
    EXPECT_EQ(ucet->zustatek, 0);
}

TEST_F(osobniUcet, operace){
    ASSERT_NO_THROW(ucet->vloz(200));
    ASSERT_NO_THROW(ucet->vyber(100));
    EXPECT_EQ(ucet->zustatek, 100);
}
```


Ukázka příkladů

PŘÍKLADY

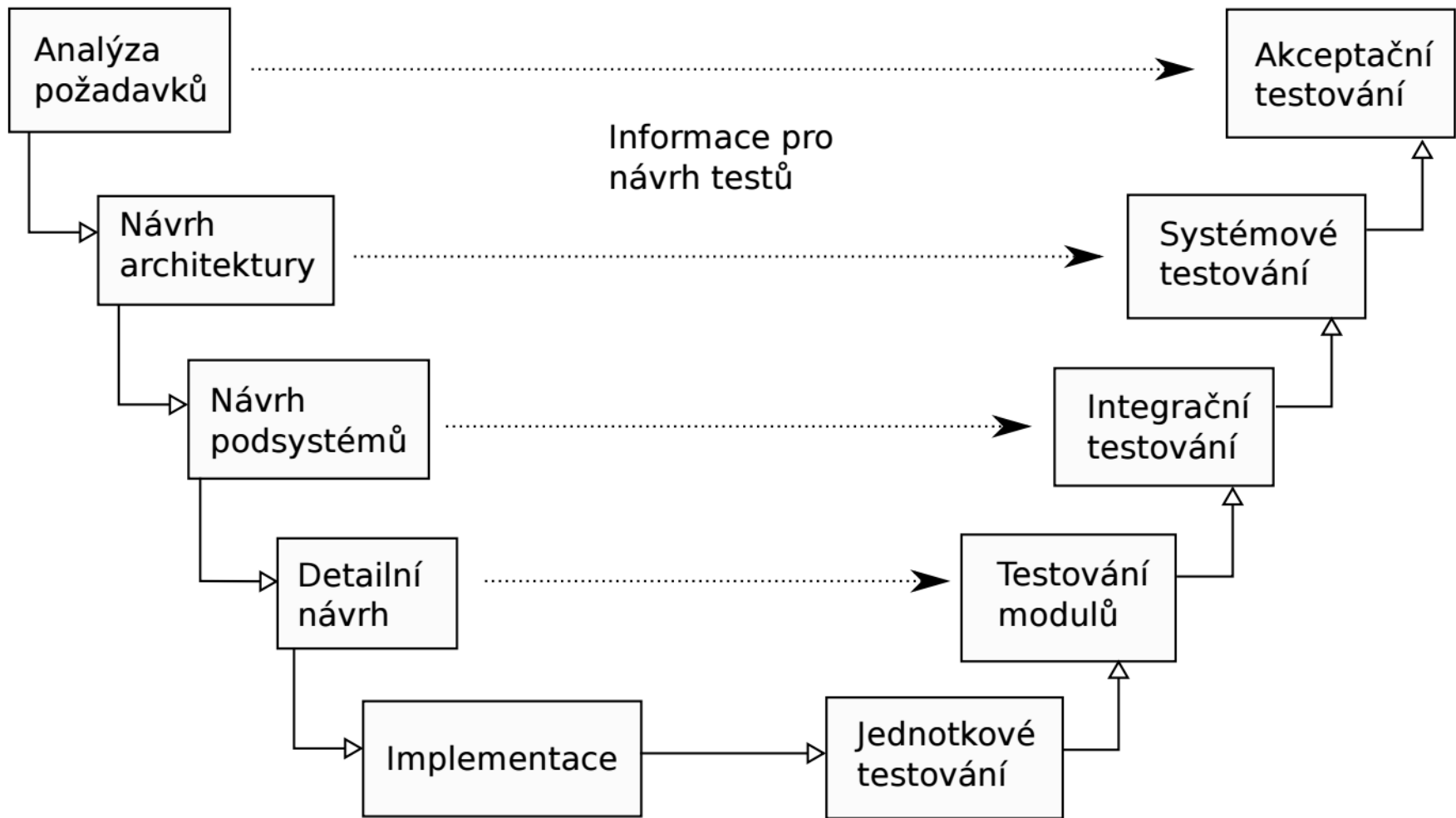
- Testování prováděné po změnách v kódu (RefaktORIZACE)
- Potvrzení, že nový kód odpovídá specifikaci
- Potvrzení, že jiná funkcionality nebyla změnou ovlivněna
- „Black box“ testy zde vlastně hrají roli specifikace
- Vhodné jsou automatizované testy

- Jednotky tvoří podsystémy
- Integrační testy testují podsystémy
- Napodobeniny (Mock) jsou nahrazeny skutečnými implementacemi
- Cílem je detekovat chyby na rozhraní jednotek a jejich interakce

- Testování systému jako celku
- Ujistění se, že funguje dle požadavků

- Typy systémových testů
 - Funkční
 - Výkonnostní
 - Zátěžové
 - Bezpečnostní
 - Konfigurační
 - Zaměřené na zotavení

- Prakticky systémové testy při předávání zákazníkovi
- Ověřují splnění požadavků zákazníka
- Ověřují se uživatelské scénáře
- Black box



- **Černá skříňka (black box)**
 - Bez znalosti vnitřní struktury
 - Důležité jsou pouze vstupy / výstupy
 - Ohled pouze na specifikaci / funkci
- **Průhledná skříňka (white / glass box)**
 - S ohledem na vnitřní strukturu
 - Prakticky testování napsaného kódu
 - Snaha pokrýt všechny větve
 - Může být časově náročné; příliš detailní
 - Také musí odpovídat specifikaci

- **Statická analýza** zkoumá vlastnosti SW bez jeho provádění.
- **Dynamická analýza** ověřuje vlastnosti SW na základě provádění kódu.

- **Formální verifikace** s pomocí formálních metod ověřuje, že systém odpovídá specifikaci.
- **Testování** zkoumá SW jeho spouštěním za účelem zvýšení jeho kvality.

- Jednou z důležitých technik je sledování, nakolik je aplikace pokryta testy.
- Můžeme sledovat, kolik případů užití máme pokrytých, kolik zákaznických požadavků, ale nejpřesnější, alespoň co se týče pokrytí funkčnosti, je pokrytí kódu.
- Při zjišťování pokrytí kódu se zaznamenává, jaký kód byl při testování spuštěn a jaký naopak otestován ještě nebyl.
- Nejjednodušší, ale zároveň nedostačující a zavádějící, je **pokrytí příkazů - řádků**.
- O stupeň pokročilejší je **pokrytí hran – rozhodnutí**.
- Nejvyšší stupeň pokrytí je **pokrytí cest**.

- Testy řízený vývoj, někdy také „programování řízené testy.“
- Proces vývoje SW, kde je hlavní důraz kladen na testy.
- Řadí se k agilním procesům.
- Testování se snaží přenést co nejvíce na začátek projektu (nezapomene se na něj).
- Těží z automatizace spuštění testů (typicky unit testy použitelné nejen pro ověření nové funkcionality, ale i jako regresní testy).
- Předem napsané testy nejsou ovlivněny jakoukoli znalostí implementace.

1. Napsat sadu testů
 - Prakticky návrh rozhraní (API)
2. Spustit testy a ujistit se, že žádný test neprojde.
3. Napsat kód tak, aby testy prošly
 - Minimální kód pro splnění testů
4. Ujistit se, že projde celá sada testů
 - Potom způsob, jakým byl kód napsán, nenarušil kód ověřovaný jinými testy
5. RefaktORIZACE (pročistění) kódu s průběžným ověřováním pomocí testů.

- Všechny produkční kód je testovaný
- Je implementována pouze potřebná funkcionality
- Průběžným spouštěním testů je možno zamezit dlouhým úpravám kódu bez znalosti stavu věcí
- Občas může být výhodnější vrátit se do stavu, kdy testy prošly, než ladit.

- Rozšíření TDD
- Vývoj je řízen popisem chování systému
- Slovní popis testů, tudíž snadněji srozumitelný
- Testy skutečně popisují očekávané chování, tedy specifikaci kódu
- Testy odpovídají „příběhům uživatelů“, které jsou často výchozím bodem agilních metod vývoje
- **Cbehave**, Igloo
- RSpec (Ruby), NSpec (.Net), JBehave (Java)

- Používá behaviorální specifikaci zahrnující poloformálně zapsané scénáře:
 1. Z pohledu koho ...
 2. co je cílem, při ...
 3. počátečních podmínkách pro provedení scénáře,
 4. ve kterém stavu systému se scénář provádí
 5. a jaký je očekávaný výsledek nebo stav systému.

```
#include "cbehave.h"
```

```
FEATURE(1, "ucet")
```

```
  SCENARIO("Vlozeni 100,- na ucet")
```

```
    GIVEN("Vytvoreni ucetu pro noveho klienta.")
```

```
      BankovniUcet ucet = new BankovniUcet();
```

```
    GIVEN_END
```

```
    WHEN("Pouzijeme metodu vloz pro vlozeni 100,- na ucet")
```

```
      ucet.vloz(100);
```

```
    WHEN_END
```

```
    THEN("na uctu ma byt aktualni zustatek 100,-")
```

```
      SHOULD_INT_EQUAL(ucet.zustatek, 100);
```

```
    THEN_END
```

```
  SCENARIO_END
```

Features are as belows:

Feature: ucet

Scenario: Vytvoreni ucetu pro noveho klienta.

Given: Vytvoreni ucetu pro noveho klienta.

When: Pouzijeme metodu vloz pro vlozeni 100,- na ucet

Then: na uctu ma byt aktualni zustatek 100,-

Summary:

total features: [1]

failed features: [0]

total scenarios: [1]

failed scenarios: [0]

- Když to musí být formálně:
 - Kosmický program, letectví, apod.
 - Formální analýza a verifikace (FAV)
 - Funkční verifikace číslicových systémů (FVS)
- RDD – Readme Driven Development - Testování README
 - Testují se příklady v README
 - Vhodné zejména u veřejných knihoven
 - „Pokud nefunguje příklad v README, knihovnu většinou nepoužiji“

- Urychlení vývoje a hlídání spolupráce v týmu.
- Principy:
 - Centralizované repozitáře
 - Zdrojové soubory na jednom místě (např. GIT)
 - Automatická kompilace
 - Po nahrání změn do repozitáře, se provede build
 - Automatické testování
 - Automaticky se spustí testy
 - Kontrola kvality kódu
 - Kontrola pojmenování metod a tříd, definice komentářů, délka kódu tříd a metod, definice a použití proměnných apod.
 - Automatické zveřejňování nových verzí

- PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002, xiv, 313 s. : il. ISBN 80-7226-636-5.
- BURNSTEIN, I. *Practical software testing*. New York: Springer, 2003, 709 s. ISBN 0-387-95131-8.
- BATH, Graham a Judy MCKAY. *The software test engineer's handbook*. Santa Barbara: Rocky Nook, 2008, xviii, 397 s. ISBN 978-1-933952-24-6.
- STEPHENS, Matt a Doug ROSENBERG. *Testování softwaru řízené návrhem*. Brno: Computer Press, 2011, 336 s. : il., portréty. ISBN 978-80-251-3607-2.
- <https://github.com/google/googletest>
- <https://github.com/bigwhite/cbehave>

ZADÁNÍ 1. PROJEKTU

http://ivs.fit.vutbr.cz/ivs_project_1.zip