**SW 25 – Testování software, ošetření chyb**

* **Kompilovaný jazyk** – Zdrojový kód nejprve přeložen pomocí překladače (kompilátoru) do strojového kódu a až poté je možné ho spustit (C#)
* **Interpretovaný jazyk** – Program zvaný interpreter přímo spouští kód step-by-step bez nutnosti předběžného překladu (Python)

**Chyba**

* Nedostatek v kódu
  + Buď porušuje pravidla zápisu programu
  + Nebo při svém běhu vyústí v nečekané/špatné chování programu

1. Hardwarové chyby – Chyby způsobené selháním hardwaru
2. Uživatelské chyby – Chyby způsobené nekorektními vstupy od uživatele
   * Některé může programátor snadno ošetřit, *např. když uživatel do kolonky hmotnost zadá textový řetězec namísto čísla*
   * Na jiné se reaguje složitěji, např. *když uživatel do kolonky hmotnost zadá hodnotu v librách namísto kilogramů*
3. Programátorské chyby – Chyby způsobené nesprávně napsaným kódem programu

**Programátorské chyby**

**Syntaktická chyba**

* Syntax – Symbolová reprezentace
* Syntaktická chyba – Kód neodpovídá pravidlům zápisu daného jazyka
* U kompilovaných jazyků je hlášena během překladu, většina IDE na chybu upozorní, popřípadě nabídne možné opravy
* Chyby v zápisu klíčového slova (překlepy) se občas nazývají lexikální

**Sémantická chyba**

* Sémantika – Význam
* Dva programy napsané v různých jazycích mohou dělat stejnou věc (sémantika), ale symboly použité pro jejich zápis se mohou lišit (syntax)
* Sémantická chyba – Kód sice splňuje pravidla zápisu jazyka, ale jeho význam je neproveditelný nebo jiný, než autor zamýšlel
* Sémantickou chybu lze zkompilovat, jelikož je syntakticky správně, program se ale bude chovat nečekaně (špatný výsledek, výjimka, pád)
* Vývojové prostředí dokáže odhalit jen některé sémantické chyby

int i = "20";

if (podmínka); { /\*Kód\*/ }

* První řádek je syntakticky správně – datový typ, název, přiřazení, hodnota, středník
  + Sémantika (význam) je ale chybná, snažíme se přiřadit textový řetězec do číselné proměnné
  + Odhaleno typovou kontrolou již při kompilaci (VS C# – *"20"* podtrženo červeně)
* Druhý řádek také neporušuje pravidla syntaxe, nedává ale příliš smysl
  + Kód uvnitř závorek se provede nehledě na hodnotu podmínky
  + (VS C# – středník podtržen zeleně)

**Debugging**

* Hledání a oprava chyb
* Oprava překlepů (syntax) či chybných výsledků (sémantika)
* Sémantické chyby nemusí být na první pohled zjevné, s nalezením jejich příčin mohou pomoci nástroje pro ladění:

**Breakpoint (zarážka)**

* Zastavení běhu aplikace na daném řádku

**Krokovací nástroje**

* Lze pomocí nich postupovat v kódu řádek po řádku
* VS – *Krokovat s vnořením* či *Krokovat s přeskočením*

**Náhled do paměti**

* Výpis lokálních proměnných v momentálně procházeném podprogramu
* Lze zobrazit i paměť celého procesu, data ale budou pravděpodobně v méně čitelné podobě

**Logování**

* Logování buď výpisem do souboru nebo do IDE logovací konzole (Trace.WriteLine();)

**Výjimka**

try {

// "Nebezpečný kód"

}

catch(DivideByZeroException) {

// Obsluha výjimky DivideByZeroException

}

catch {

// Obsluha všech ostatních výjimek

}

finally {

// Provede se vždy – úklidové okno (zavírání souborů, mazání hodnot)

}

* Softwarové přerušení, nastává při běhu programu již po kompilaci a přeložení
* Standardní způsob předávání informací o chybě nebo nestandardním stavu
* Objekt nesoucí informaci o chybovém stavu
* V dobrém programu by měly být všechny výjimky ošetřeny
* Výjimku lze vyvolat operátorem throw, např. pro upozornění testera když část programu není doimplementována
* throw new DivideByZeroException("vlastní zpráva");
* Je možné si definovat i vlastní výjimky, všechny dědí z System.Exception

**Testování**

* Šetření kvality softwaru

**Jednotkové testování**

[Theory]

[InlineData(2, 3, 5)] // Projde

[InlineData(2, 3, 6)] // Neprojde

public void Add(double x, double y, double expected)

{

double actual = Calculator.Add(3, 2); // Metoda, kterou testujeme

Assert.Equal(expected, actual); // Porovnání s očekávanou hodnotou

}

* Testování individuálních jednotek zdrojového kódu
* Automatizované testy napsané a využívané programátory
* Zjištění, zdali jednotlivé sekce programu (jednotky) fungují, jak bylo zamýšleno
* Metodám se předávají různé vstupy a zkouší se, zdali jsou výstupy korektní
* Nejdříve by se měl napsat test a poté až začít pracovat na metodě ("přechod z červené do zelené")
* Ve VS testování v odděleném projektu, různé balíčky – NUnit, xUnit, ...

**Integrační testování**

* Jednotlivé softwarové moduly (které jsou prověřeny jednotkovými testy) jsou zkombinovány a testovány jako celek
* Testování spolupráce mezi jednotlivými službami aplikace
* Testování, jestli lze nové vlastnosti zaintegrovat do stávajícího kódu

**Systémové testování**

* Ověření, zdali je software jako celek funkční

**Akceptační testování**

* Testování z hlediska uspokojení uživatelů – ověření zákazníkem

**Pilotní testování**

* Zkušební provoz