

Životní cyklus a vývojový proces C#.NET aplikace

Filip Jurnečka, Martin Osovský

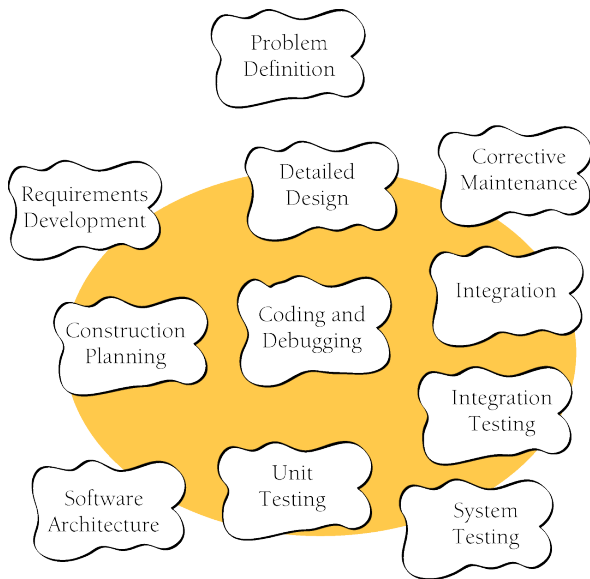
PV178

15. března 2011

- 1 Životní cyklus aplikace
 - Konstrukční aktivity
 - Metafory
 - Klíčová rozhodnutí při tvorbě softwaru
 - Plánování aplikace
 - Úrovně návrhu
- 2 Návrh softwaru v prostředí VS2010
 - Diagramy případů užití
 - Diagramy aktivit
 - Sekvenční diagramy
 - Diagramy komponent
 - Diagramy tříd
- 3 Analýza kódu
 - Architecture Explorer
 - Grafy závislostí
 - Code metrics
- 4 Developerské nástroje VS2010

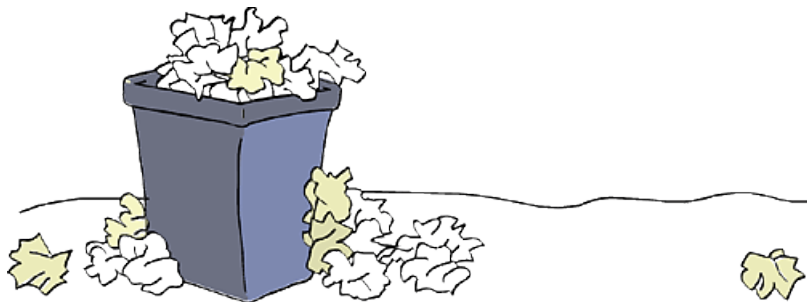
Co je vývoj softwaru

- Definice problému
- Požadavky na vývoj
- Plánování provedení
- Návrh softwaru
- Detailní návrh kódu
- Kódování a debugování
- Unit testy
- Integrační testy
- Integrace
- Systémové testy
- Opravná údržba



Obrázek: Konstrukční aktivity jsou ve žlutém kruhu.

Softwarové dopisy: psaní kódu



Obrázek: Psaní kódu dopisovým stylem naznačuje, že spoléháte na způsob pokus, omyl.

Softwarové konstrukce: budování kódu I



Obrázek: Chyba na jednoduché stavbě se dá většinou snadno opravit.

Softwarové konstrukce: budování kódu II



Obrázek: Složitější struktury vyžadují pečlivější plánování.

Klíčová rozhodnutí při tvorbě softwaru

- Výběr programovacího jazyka.
- Programovací konvence.
- Využití existujících technologií.
- Výběr konstrukčních praktik.

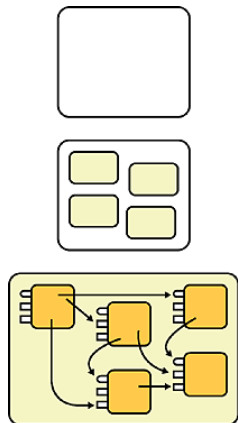
Výzvy plánování aplikace

- Model aplikace může být přesně definován až po jejím naplánování.
- Koncová aplikace má být bezchybná a dobře organizovaná. To ale neplatí pro její vývoj.
- Plánování je o kompromisech a prioritách.
- Plánování je nedeterministický \Rightarrow heuristický proces.

Požadované vlastnosti návrhu

- Minimální složitost (KISS - keep it simple, stupid).
- Minimální propojenost (SOLID - viz příští přednáška).
- Rozšiřitelnost (YAGNI - you ain't gonna need it).
- Znovupoužitelnost (DRY - don't repeat yourself).
- Štíhlost.
- Jednoduchost údržby.
- Přenositelnost.

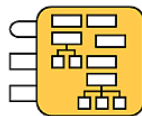
Úrovně návrhu



① Softwarový systém

② Rozdělení do podsystémů/balíčků

③ Rozdělení do tříd v balíčcích



④ Rozdělení do dat a metod v rámci třídy



⑤ Návrh vnitřních metod

Běžné podsystémy

Obchodní logika jsou právní zásady a postupy, které zabudováváte do svého softwaru. Např. mzdový systém.

Uživatelské rozhraní je podsystém, ve kterém se snažíte izolovat komponenty uživatelského rozhraní od ostatní logiky softwaru tak, aby se mohlo vyvíjet nezávisle na zbytku programu.

Přístup k externím datům se snaží odstínit zbytek programu od manipulace s externími službami či nízkoúrovňovými strukturami jako jsou databáze.

Architektornické nástroje ve VS2010

Visual Studio 2010 plně podporuje UML verze 2.1.2. Přímo předpřipravených UML diagramů je pět:

- Diagramy případů užití.
- Diagramy aktivit.
- Sekvenční diagramy.
- Diagramy komponent.
- Diagramy tříd.

Diagram případů užití (use case diagram)

- Popisuje lidi, kteří budou aplikaci používat a co všechno s ní budou dělat.
- Popisuje vztahy mezi požadavky, uživateli a hlavními komponentami systému.

Hlavní elementy:

Název	Popis
Actor	značí osobu nebo externí systém, který zahajuje nebo se podílí na případě užití.
Use Case	označuje interakci mezi systémem a případem použití.
Subsystem	reprezentuje část systému.
Association	spojuje aktéra s případem použití.
Dependency, Include, Extend, Generalization	popisují závislosti mezi případy užití.

Diagram aktivit (activity diagrams)

- Popisuje softwarový proces jako řídicí tok (work flow) prostřednictvím série akcí.
- Popisuje vztahy mezi požadavky, uživateli a hlavními komponentami systému.
- Umožňuje zobrazení souběžných toků.
- Pomocí komponenty *Artifact* je možno jej přidat do diagramu případu užití.

Příklad ObjednatProdukt.

Hlavní elementy:

Název	Popis
Initial Node	značí počátek aktivity.
Action	označuje krok k němuž v dané aktivitě dochází.
Object Node	reprezentuje objekt participující na aktivitě.
Decision, Merge Node	rozděluje a spojuje alternativní cesty toku.
Fork, Join Node	rozděluje a spojuje paralelní cesty toku.
Send Signal, Accept Signal Action	označuje odeslání a čekání na signál.
Input, Output Pin	označuje data, která aktivita požaduje a produkuje.
Connector	označuje tok nebo spojení mezi prvky diagramu.

Sekvenční diagram (sequence diagrams)

- Popisuje interakce (např. výměny zpráv) mezi různými objekty.
- Shora dolů vyjadřuje tok času.
- Zleva doprava vyjadřuje tok řízení od jednoho prvku k druhému.

Hlavní elementy:

Název	Popis
Lifeline	reprezentuje účastníka v interakční sekvenci.
Synchronous	značí zprávu, kde odesílatel čeká na odpověď.
Asynchronous	značí zprávu, kde se na odpověď nečeká.
Create	zobrazuje vytvoření nové instance cíle.

Příklad Objednavka.

Diagram komponent (component diagram)

- Popisuje nejen jednotlivé komponenty aplikace, ale i ostatní části systému (např. GUI, web service, ...).
- Znázorňuje vztahy mezi jednotlivými komponentami.
- Komponenty mohou být spustitelné soubory, knihovny nebo celé systémy.
- Komponenta by měla být modulární nahraditelná jednotka systému.
- Každý element má vlastnosti (např. Qualified Name, Work Items, Is Abstract, ...), které více zpřesňují daný diagram.

Příklad eShopComponents.

Hlavní elementy:

Název	Popis
Dependency	určuje, že jedna komponenta závisí na druhé.
Component	značí komponentu systému.
Delegation	označuje závislost mezi portem na vnější a rozhraním na vnitřní komponentě.
Provided, Required Interface	určuje, že daná komponenta nabízí nebo požaduje dané rozhraní.
Generalization	určuje, jak se jedna komponenta odvozuje od druhé.
Connector	označuje základní vztah mezi obrazy.

Diagram třídy (class diagram)

- Popisuje jednotlivé třídy v aplikaci a vztahy mezi nimi.
- Soustředí se na logiku tříd na místo jejich implementace.
- Primárně rozlišujeme tři typy elementů

Typy jsou třída, rozhraní nebo výčet.

Atributy jsou hodnoty, které mohou být přidruženy k instanci třídy nebo rozhraní.

Operace jsou metody nebo funkce, které mohou být vykonány instancí třídy nebo rozhraní.

- Každý element má vlastnosti (např. Qualified Name, Is Abstract, Visibility ...), které více zpřesňují daný diagram.

Příklad ZboziClass.

Hlavní elementy:

Název	Popis
Class	označuje typ třídy.
Interface	označuje typ rozhraní.
Enumeration	označuje typ výčtu.
Package	označuje organizační složku pro typy stejného namespace.
Association	určuje, jak jeden element interaguje s jinými.
Aggregation	říká, že zdrojový typ odkazuje na části cílového typu.
Composition	říká, že zdrojový typ obsahuje části cílových typů.
Dependency	určuje, jak jeden typ závisí na jiném.
Inheritance	říká, že typ dědí nebo realizuje komponenty cílového typu.
Package Import	určuje, jak balíček importuje typy definované v jiném balíčku.

Analýza kódu pomocí funkce Architecture Explorer

- Architecture Explorer je komponenta VS2010 dovolující alternativní prohledávání existujícího kódu, .dll knihoven a .exe souborů, výběr jejich části a její zobrazení jako graf závislostí.
- Pouze přítomen ve verzi Visual Studio 2010 Ultimate (naleznete v menu *View* → *ArchitectureExplorer*).
- Navigace umožňuje:
 - Výběr více objektů najednou pomocí CTRL (nebo SHIFT).
 - Filtrování podle jmen nebo parametrů na každé úrovni.
 - Filtrování dle dalších parametrů na pomocí speciálního tlačítka “Types”, “Members”,
- Pomocí tlačítek na levém okraji můžete:
 - Zobrazit aktuálně vybrané struktury v grafu závislostí.
 - Přidat vybrané struktury do právě zobrazeného grafu závislostí.
 - Uložit graf jako “*.dgml” soubor.
 - Rychle zrušit veškerý výběr.
 - Uložit právě vybranou cestu jako “*.dgql” soubor. Tyto pak najdete v prvním sloupci v skupině “Saved DGQL Queries”.

Grafy závislostí

- Grafy závislostí vizualizují informace z Architecture Exploreru.
- Zobrazují pouze kód, který lze úspěšně přeložit.
- Dále jsou nabízeny 4 typy generování grafu z celých “Assemblies”:
 - By Assembly** vytvoří jeden uzel za každý assembly element v daném solution.
 - By Namespace** vytvoří jeden uzel za každý namespace.
 - By Class** vytvoří jeden uzel za každou třídu.
 - Custom**

Příklad MySolution.

Metriky kódu

- VS 2010 generuje metriky kódu pro celou solution nebo jednotlivé projekty.
- Metriky kódu odhadují míru udržitelnosti kódu pomocí následujících hodnot.

Maintainability Index	reprezentuje relativní jednoduchost udržitelnosti kódu.
Cyclomatic Complexity	měří strukturální složitost kódu. Je vytvořena na základě výpočtu různých cest v toku programu.
Depth of Inheritance	označuje počet tříd, které rozšiřují kořenové třídy hierarchie.
Class Coupling	měří míru propojenosti jednotlivých tříd, metod, rozhraní,
Lines of Code	označuje přibližné množství řádků kódu.

Nástroje a funkce pro developery

Test impact analysis kontroluje, které unit testy byly ovlivněny změnou zdrojového kódu.

Improved code analysis dovoluje developerům vyhledat v kódu běžné problémy a bezpečnostní chyby.

Profiler enhancements výrazně vylepšuje uživatelské rozhraní profileru. Navíc je doplněno profilování client-side JavaScriptu.

Database extensibility dovoluje třetím stranám vývoj a správu jiných databázových systémů, než SQL Serveru.

Advanced debugging přináší funkci “IntelliTrace”, která sleduje množství zajímavých akcí během běhu programu a ty pak dovoluje znovu přehrát a prozkoumat.

Novinky a vylepšení Visual Studia 2010 I

- Podpora více monitorů.
- Debugging
 - okno “Threads”
 - nabízí filtrování,
 - prohledávání stacků,
 - seskupování
 - okno “Breakpoints”
 - nabízí vyhledávání,
 - pojmenování jednotlivých breakpointů,
 - sdílení breakpointů s ostatními vývojáři

Novinky a vylepšení Visual Studia 2010 II

- “IntelliTrace”
 - zobrazuje eventy, které se udály v minulosti spolu s jejich kontextovou informací
 - nabízí post-crash debugging ze souborů .iTrace, které IntelliTrace nebo Test Manager
 - umožňuje debugovat “nereprodukovatelné” chyby
 - v defaltním nastavení sbírá pouze “IntelliTrace eventy”
 - Debugger eventy - start aplikace a stavy, kdy aplikace vejde do stavu “break state”.
 - Exception eventy - pro handled i unhandled výjimky.
 - Framework eventy - nastávají v .NET knihovnách, např. přístup k souboru, zaškrnutí checkboxu,

Novinky a vylepšení Visual Studia 2010 III

- Team Foundation Version Control
 - základní version control system,
 - několik typů diagramů změn (Diagram of Merges, Diagram of Branch Hierarchy), ve kterých lze provádět základní operace
- Deployment
 - “Build Definitions” může kompilovat kód, spustit testy, automaticky nahrát binárky na cílovou lokaci
 - můžete přizpůsobit build proces pomocí “Windows Workflow Designer”
- IntelliSense pro JavaScript

Otázky?
Na shledanou!

Otázky?
Na shledanou!