

# **Vývoj programových systémů v jazyce Java**

---

## **Vývoj programových systémů v jazyce Java**

---

---

---

# Obsah

1. Úvod do předmětu, průběh, kritéria hodnocení. Architektury rozsáhlých aplikací. ....	1
Charakteristika a cíle předmětu a průběh výuky .....	1
Charakteristika .....	1
Průběh výuky .....	1
Úlohy .....	1
Projekty .....	1
Zkouška .....	2
Složky hodnocení .....	2
Kritéria hodnocení předmětu .....	2
Modely .....	2
Modely vícevrstevných aplikací .....	3
Vrstvy .....	3
.....	3
Komponenty .....	3
.....	3
Orientace na služby .....	3
.....	3
Správa .....	3
.....	3
Zabezpečení .....	3
.....	3
Kontejnery a rámce .....	3
.....	3
Komponentní vs. objektové programování .....	3
Objektové programování - principy .....	3
Objektové programování - terminologie .....	3
Objektové programování - přednosti .....	4
Objektové programování - nedostatky .....	4
Komponentní programování - principy .....	4
Co musí komponenta splňovat .....	4
Srovnání komponentního a objektového přístupu .....	5
Komponentní programování - výhody oproti OOP .....	5
Komponentní programování - souvislosti .....	5
Komponentní dokumenty .....	6
Architektury orientované na služby (Service-oriented Architectures - SOA) .....	6
Motivace k SOA .....	6
Definice SOA .....	6
Technologie SOA .....	6
Charakteristiky SOA .....	7
Vztah komponent a služeb .....	7
Odkazy .....	7
2. Metodiky vývoje - Agilní a extrémní programování. Nástroje správy vývoje a nasazení SW. ..	8
Agilní programování .....	8
Co je Agilní programování .....	8

Manifesto for Agile Software Development .....	8
Motivace .....	8
Principy .....	9
Zásada - stálý kontakt se zákazníkem - zadavatelem .....	9
Zásada - časté uvolňování funkčního SW .....	9
Zásada - vysoká kvalita .....	9
Zásada - netvořit do zásoby .....	9
Extremní programování (Extreme Programming, XP) .....	10
Motivace pro Extreme Programming (XP) .....	10
Co je XP .....	10
Charakteristika XP .....	10
Východiska řízení týmu podle XP .....	10
Hlavní zásady XP .....	11
Vedlejší zásady XP .....	11
Vývojové činnosti XP .....	11
Hlavní techniky XP .....	11
Fáze XP projektu .....	12
KISS (Keep It Simple, Stupid) .....	12
Princip KISS .....	12
Zásada - navrhovat jednoduše .....	13
Zásada - netvořit do zásoby .....	13
Refaktoring .....	13
Refaktoring - proč .....	13
Refaktoring - metody .....	13
Refaktoring - nástroje kom. ....	14
Refaktoring - nástroje o-s .....	14
Programování řízené testy (Test-drive Development, TDD) .....	14
Motivace .....	14
Principy .....	15
Kde nelze testovat? .....	15
Generický postup TDD .....	15
Odkazy .....	15
Systémy správy verzí .....	16
Motivace .....	16
Principy .....	16
Klasická řešení - RCS a CVS .....	16
Typické příkazy správy verzí .....	17
Klienti .....	17
Pro co se systémy nehodí? .....	17
Subversion .....	17
Subversion - klient Tortoise pro Windows .....	18
Tortoise -- vzdálený přístup .....	18
Správa sestavování - Ant .....	18
Charakteristika .....	18
Motivace .....	18
Struktura projektu .....	18
Příklad 1 .....	19
Závislosti .....	19
Příklad 2 .....	19

Maven .....	19
Motivace .....	19
Maven - charakteristika .....	20
Project Object Model (POM) .....	20
Projekt v Mavenu .....	20
Maven repository .....	21
Instalace a nastavení .....	21
Příklad POM (project.xml) .....	22
Příklad POM - pokračování .....	22
Struktura POM obecně .....	22
Proměnné (properties) v POM .....	23
Struktura repository .....	23
Cíle v Mavenu .....	23
Maven Plugins .....	24
Často používané cíle .....	24
Vytvoření projektu .....	24
Reporting .....	25
Rozšíření možností Mavenu .....	25
3. Webové aplikace. ....	
3. Datová vrstva javových aplikací - zajištění perzistence a správy dat. JDBC, JDO, Hibernate. ...	
Java Database Connectivity (JDBC) .....	
Co je JDBC .....	
Postup práce s JDBC .....	
Hlavní třídy (komponenty) JDBC .....	
Hlavní třídy (komponenty) JDBC (2) .....	
Příklad - zavedení ovladače DBMS .....	
Příklad - vytvoření spojení, příkazu a jeho spuštění .....	
Charakteristika JDBC rozhraní .....	
JDBC rozhraní - balíky .....	
Práce s databází .....	
Transakce .....	
Transakce - příklad .....	
Demo - HSQLDB .....	
HSQLDB - režimy práce .....	
Předpřipravené příkazy .....	
Předpřipravené příkazy - příklad .....	
Uložené procedury .....	
Modifikovatelné výsledky dotazu .....	
Metadata .....	
Zpřístupnění datového zdroje přes JNDI .....	
Java Data Objects (JDO) .....	
JDO - charakteristika .....	
JDO - forma .....	
JDO - motivace .....	
JDO - struktura .....	
JDO - implementace .....	
JDO - informační zdroje .....	
JDO - alternativy .....	
Serializace dat do XML .....	

Serializace obecně - připomenutí .....	
Serializace .....	
Serializace - metody .....	
Serializace do XML - vazba XML na Javu .....	
Jednoduché nástroje serializace - XStream .....	
Jak funguje XStream .....	
XStream - příklad, krok 1 .....	
XStream - příklad, krok 2 .....	
Hibernate .....	
Hibernate .....	
4. Aplikační vrstva javových aplikací - řízení toku, správa komponent. Pokročilé zásady a metody návrhu aplikační logiky - Design by Contract, Inversion of Control, Aspect-oriented Programming. Kontejnery, aplikační servery. ....	
Aplikační vrstva javových aplikací .....	
Aplikační vrstva javových aplikací .....	
Design by Contract .....	
Design by Contract - návrh podle kontraktu .....	
DBC - jak dosáhnout .....	
DBC - nástroj jass .....	
Postup při práci s jass <b>InstantWeb</b> .....	
Odkazy .....	
Inversion of Control (IoC) .....	
Nezbytné pojmy z komponentních systémů .....	
IoC - Motivace .....	
Tradiční řízení životního cyklu komponent .....	
IoC - Hlavní princip .....	
IoC - Možné podoby .....	
Interface Injection .....	
Setter Injection - komponenta .....	
Setter Injection - popis komponenty .....	
Setter Injection - výhody/nevýhody .....	
Constructor Injection .....	
Constructor Injection - příklad komponenty .....	
Použití IoC - kontejnery .....	
Aspect Oriented Programming (AOP) .....	
AOP - Motivace .....	
AOP - Motivační příklad .....	
AOP - Principy .....	
Kontejnery a aplikační servery .....	
Kontejnery a aplikační servery .....	
Java Management extension (JMX) .....	
Co je JMX .....	
Co řídí JMX .....	
Principy JMX .....	
Který objekt (komponentu) jako JMX? .....	
Úrovně JMX modelu .....	
Ovládané zdroje .....	
Jak se zdroje ovládají .....	

MBean .....	
Co obsahují/zpřístupňují rozhraní MBean .....	
Typy MBean .....	
Aplikační rámec Tammi - případová studie .....	
Charakteristika Tammi .....	
Příklad jednoduché komponenty typu MBean .....	
Skriptování v javovém prostředí - BSF .....	
Co je skriptování? .....	
Proč skriptovat? .....	
Proč skriptovat právě teď? .....	
Bean Scripting Framework .....	
BSF - co nabízí .....	
BSF - typické použití .....	
BSF - download a další info .....	
Skriptování v javovém prostředí - Groovy .....	
Groovy - motivace .....	
Stažení .....	
Instalace .....	
Spuštění .....	
Příklad - iterace .....	
Příklad - mapa .....	
Příklad - switch .....	
Řízení a sledování aplikací - protokolování .....	
Protokolování (logging) .....	
Protokolování - výhody .....	
Protokolování - možnosti v Javě .....	
Protokolování - API .....	
Protokolování - příklad .....	
5. Prezentační vrstva javových webových aplikací. Přístupy - šablony, dekorátory. Velocity, Web-Macro, Sitemesh, Freemarker. Nové přístupy - AJAX .....	
Prezentační vrstva webových aplikací .....	
Prezentační vrstva webových aplikací .....	
AJAX - Asynchronous JavaScript and XML .....	
Webové aplikace - důvody pro .....	
Webové aplikace - důvody proti .....	
Přístup AJAX - Asynchronous JavaScript and XML .....	
Výhody .....	
Kde je použito? .....	
Jak pracuje? .....	
AJAX ve spojení s J2EE .....	
6. Komplexní podpora webových aplikací. Webové rámce. ....	
Webové aplikační rámce .....	
Osnova .....	
Webový rámec .....	
Třívrstvá architektura .....	
Koncepce MVC .....	
Přednosti MVC .....	
Javové webové aplikace .....	
Javové webové aplikace .....	



Javové webové aplikace (2)	.....
Javové webové aplikace (3)	.....
Javové webové kontejnery	.....
Funkcionalita webových rámců	.....
Oddělení prezentační vrstvy	.....
Oddělení prezentační vrstvy (2)	.....
Oddělení prezentační vrstvy (3)	.....
Funkcionalita webových rámců	.....
Správa zabezpečení - příklad	.....
Validace uživatelských vstupů	.....
Udržování informací o relaci	.....
Řízení toku aplikace	.....
Příklad řízení toku - Struts	.....
Zotavení z chyb	.....
Záznamy o událostech (logging)	.....
Internacionalizace a lokalizace	.....
Internacionalizace a lokalizace - příklad	.....
Zajištění perzistence objektů	.....
Škálovatelnost a distribuovanost	.....
Škálovatelnost a distribuovanost	.....
Podpora vývojovými prostředími	.....
Přehled rámců	.....
Shrnutí	.....
7. Enterprise JavaBeans	.....
Enterprise JavaBeans	.....
Enterprise JavaBeans	.....
8. Systémy řízení zpráv, Java Messaging Service (JMS) API	.....
Systémy řízení zpráv, Java Messaging Service (JMS) API	.....
Systémy řízení zpráv - motivace	.....
Systémy řízení zpráv - výhody	.....
Systémy řízení zpráv - principy	.....
Systémy řízení zpráv - standardy	.....
JMS - kdy použít?	.....
JMS - co nabízí	.....
JMS - architektura	.....
JMS - domény	.....
Point-to-point Messaging	.....
Publish/Subscribe	.....
Komponenty JMS API	.....
Zpracování zprávy	.....
Demo JMS s použitím Sun Java System Message Queue	.....
Použití Sun Java System Message Queue	.....
Co je třeba pro překlad klienta	.....
Spuštění serveru (Message Brokeru)	.....
Test běhu SJSMQ	.....
Kostra jednoduché aplikace	.....
9. Webové služby	.....
Webové služby	.....
Webové služby	.....

Webové služby typu REST .....	
Motivace .....	
Co je REST .....	
Principy .....	
Co REST aplikace používají .....	
Charakteristika REST služeb .....	
Co rozmyslet před návrhem REST služby .....	
Postup návrhu .....	

---

## Seznam obrázků

31..1. Použití servletu .....	
31..2. Životní cyklus servletu .....	
8.1. Spolupráce JMS a JNDI (z tutoriálu J2EE Sun) .....	
8.2. Mechanismus Point-to-Point (z tutoriálu J2EE Sun) .....	
8.3. Mechanismus Publish/Subscribe (z tutoriálu J2EE Sun) .....	
8.4. Programovací model JMS API (z tutoriálu J2EE Sun) .....	
8.5. Spuštění SJS Message Queue .....	
8.6. Test běhu SJS MQ .....	

---

## Seznam příkladů

31..1. HelloClientServlet.java .....	
32..1. web.xml .....	
32..2. build.properties .....	

---

# Kapitola 1. Úvod do předmětu, průběh, kritéria hodnocení. Architektury rozsáhlých aplikací.

## Charakteristika a cíle předmětu a průběh výuky

### Charakteristika

Předmět PA165 Vývoj programových systémů v jazyce Java [<http://is.muni.cz/predmety/predmet.pl?id=272777>] je magisterským předmětem určeným pro zájemce o hlubší proniknutí do principů výstavby rozsáhlejších programových systémů na platformě Java.

Cílem je seznámit s principy:

- kvalitního návrhu rozsáhlých systémů (jak navrhnout architekturu, jakou zvolit infrastrukturu/rámec...),
- jejich tvorby (jaké zásady, nástroje,...)
- testování (při vývoji, zavádění a v provozu),
- refaktoringu (=zlepšování kódu bez změny funkcionality),
- ladění výkonu (optimalizace)

### Průběh výuky

Výuka probíhá jako:

- Dvouhodinová přednáška - Po od 14.00 v A107
- Dvouhodinová cvičení v nové učebně B130 (v přízemí "za halou")
- Nedílnou součástí je řešení projektu - ve cvičení s možností konzultací nebo ve volném čase

### Úlohy

Těžištěm předmětu je samostatná práce na projektech. Společně zadávaných úloh je proto minimum.

### Projekty

Projekty jsou koncipovány jako týmové (4členné týmy).

Tematicky vždy pokrývají netriviální část z celkového obsahového záběru předmětu a musí tomu být i technologicky přizpůsobeny.

Zadání projektu stanovuje nebo schvaluje cvičící.

## Zkouška

Zkouška bude ústní.

Student při ní musí prokázat dobrý přehled ve studované problematice s tím, že principy a technologie, které aplikoval při řešení projektu musí zvládnout do větší hloubky.

## Složky hodnocení

Hodnocení předmětu je stanoveno podle celkového součtu bodů z jednotlivých kritérií:

- Úlohy - 15 bodů
- Projekt - 50 bodů
- Ústní zkouška - 35 bodů

## Kritéria hodnocení předmětu

K ukončení zkouškou potřebujete:

- A 94 - 100 bodů
- B 88 - 93 bodů
- C 82 - 87 bodů
- D 76 - 81 bodů
- E 70 - 75 bodů
- F 0 - 69 bodů

K ukončení zápočtem potřebujete:

- 60 bodů

## Modely

## Modely vícevrstevných aplikací

## Vrstvy

## Komponenty

## Orientace na služby

## Správa

## Zabezpečení

## Kontejnery a rámce

## Komponentní vs. objektové programování

### Objektové programování - principy

Obecné, známé principy OO jazyků:

- možnost (a nutnost) *zapouzdření* dat a metod nad nimi pracujících do podoby objektu,
- *dědičnost* (inheritance) - odvozování nových typů s děděním vlastností typů rodičovských,
- *polymorfizmus* (mnohotvarost) - využitelnost objektů podděděných typů v roli předků

### Objektové programování - terminologie

Na obecných principech OOP není mezi teoretiky a vývojáři (v různých jazycích) až taková shoda, jak by se mohlo zdát:

- bezspornou charakteristikou OOP je *zapouzdření*,
- další char. jako *dědičnost* (inheritance) je už v různých jazycích pojímána různě, příp. chybí - a přesto můžeme ještě hovořit o OO jazyku,

- *polymorfizmus* (mnohotvarost)

## Objektové programování - přednosti

Výhody OOP (za předpokladu kvalitního návrhu) jsou dobře známy:

- dobrá čitelnost a udržitelnost kódu,
- rozšiřitelnost,
- určitá vnitřní nahraditelnost částí - např. třídu lze nahradit potomkem
- aspoň teoreticky - využitelnost i jako "black-box"/černá skříňka - bez znalosti zdrojového kódu

## Objektové programování - nedostatky

Ale tak skvělé to zase není:

- bez zdrojového kódu lze dobře znovupoužít jen kvalitní objektový kód,
- kód je znovupoužitelný většinou jen na mikroúrovni - jednotlivé třídy
- k znovupoužití je třeba detailní znalosti celého prostředí programu, rozhraní větších programů bývá většinou komplikované - i proto, že komponenty (objekty) jsou příliš malé, jemné

## Komponentní programování - principy

V zásadě vychází z objektového, ale:

- komponenty jsou VELKÉ (coarse-grained) objekty zapouzdřující ucelený kus aplikační logiky nebo dat,
- komponenty jsou nezdědky přístupné i vzdáleně (tj. po síti, často i z jiných platform/jazyků),
- komponenty mohou (za podpory middleware) existovat relativně samostatně
- vazby k ostatním jsou dobře kontrolovatelné a obecně spíše volné
- jako protokoly, datové formáty atd. jsou přednostně použity standardní, byť třeba ne tak efektivní

## Co musí komponenta splňovat

Komponenta musí být:



- dobře dokumentovaná, s jasně definovaným API, nejlépe i formálně
- důkladně testovaná
- robustní a odolná vůči chybám uživatele (validace vstupů)
- musí mít podrobné hlášení chyb
- musí se chovat defenzivně - být odolná vůči nesprávnému použití - je nutné s ním počítat

## Srovnání komponentního a objektového přístupu

Přestože komponenty jsou ve většině případů realizovány jako objekty - a tedy komponentní programování je v zásadě pokračováním objektového - je tu jeden zásadní rozdíl.

- Při návrhu objektů v podstatě přímočaře reprezentujeme strukturu a chování výseku reality.

Předpokládá se soulad chápání koncového uživatele (zákazníka) a návrháře ohledně této reprezentace.

- Návrh komponent naproti tomu zcela podřizujeme účelu, proč komponenta existuje, a znovupoužitelnosti.

Vnitřní struktura uživatele (zákazníka) nezajímá.

## Komponentní programování - výhody oproti OOP

Oproti OOP to umožňuje:

- daleko lepší znovupoužitelnost - slabší vazby, vyšší autonomie
- vyšší robustnost komponentních systémů
- snazší vyměnitelnost (nefunkční, zastaralé) komponenty
- možnost souběžného použití různých verzí komponent
- lepší testovatelnost komponent vzhledem ke slabším vazbám
- vývoj mohou dělat nezávislejší mikro-týmy

## Komponentní programování - souvislosti

V javovém světě jsou minimálně tyto trendy a možnosti odpovídající KP:

- pro výstavbu velkých systémů (aplikací) se používají Enterprise JavaBeans,

- existují tzv. aplikační servery, které pro ně zajišťují middleware,

## Komponentní dokumenty

Komponentní principy nejsou užívány jen při tvorbě SW, ale i návrh struktur dokumentů:

- Object linking and embedding (OLE) [[http://en.wikipedia.org/wiki/Object\\_linking\\_and\\_embedding](http://en.wikipedia.org/wiki/Object_linking_and_embedding)]
- OpenDoc [<http://en.wikipedia.org/wiki/OpenDoc>]

## Architektury orientované na služby (Service-oriented Architectures - SOA)

### Motivace k SOA

- Obecný příklon k chápání IT jako poskytovatele služby - servisu - nikoli jen technologie.
- Stejně se začíná přistupovat i k vazbě mezi SW komponentami.
- Orientace na služby se ve vývoji SW stává vůdčím směrem.

### Definice SOA

Architektury orientované na služby (Service-oriented Architectures - SOA) představují princip budování softwarových architektur založený na:

- komponentách umístěných v uzlech sítě
- nabízející své služby ostatním prostřednictvím dobře definovaného protokolu
- služby jsou bezstavové
- vnitřní realizace klienta obvykle nezajímá - standardy zajišťují interoperabilitu služeb

### Technologie SOA

SOA v zásadě neváže na žádnou SW platformu, OS, pg. jazyk ani konkrétní standard.

V úzkém slova smyslu se za SOA dnes někdy považují architektury budované na tzv. *Web Services* (webových službách, WS) kolem množiny standardů:

- XML - data reprezentovaná značkovými dokumenty

- HTTP jako základní webový protokol
- SOAP - jako std. protokol WS
- WSDL - popisovač webových aplikací
- UDDI - služby registrace a vyhledávání webových služeb

## Charakteristiky SOA

Orientace na služby přináší:

1. Nahraditelnost služeb
2. Interoperabilitu
3. Možnost nezávislého ladění služeb
4. Usnadňuje integraci systémů třetích stran.

## Vztah komponent a služeb

Oboje jsou moderní až módní záležitosti, vztah lze definovat takto:

- Komponenta je často entitou realizující určitou službu.
- Koncový uživatel nahlíží více na služby, zatímco vývojáře zajímají rovněž komponenty, které je realizují.

## Odkazy

Kromě základního odkazu na Wikipedii [[http://en.wikipedia.org/wiki/Service-oriented\\_architecture](http://en.wikipedia.org/wiki/Service-oriented_architecture)] lze navštívit/pročíst:

- What is Service-Oriented Architecture? [<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>]
- SOA Center [<http://www.soacenter.com/>]

---

# Kapitola 2. Metodiky vývoje - Agilní a extrémní programování. Nástroje správy vývoje a nasazení SW.

## Agilní programování

### Co je Agilní programování

Agilní programování (Agile Programming, AP) je relativně novým přístupem k vývoji software.

- Není to jedna metodika, přesný návod, jak systém navrhnout a realizovat.
- Je to spíše přístup, "filozofie", do které je třeba dodat konkrétní postupy pro jednotlivé úkony vývoje.
- Označení Agilní programování bylo poprvé zavedeno skupinou autoritativních odborníků (praktiků) na SW metodiky.
- Zastánci AP se sdružili při sestavení manifestu AP, formulující jeho hlavní zásady.

### Manifesto for Agile Software Development

„We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: “

„*Individuals and interactions* over processes and tools“

„*Working software* over comprehensive documentation“

„*Customer collaboration* over contract negotiation“

„*Responding to change* over following a plan “

„That is, while there is value in the items on the right, we value the items on the left more.“

### Motivace

Motivací pro AP byla především řada neúspěchů při budování rozsáhlých softwarových děl a řízení realizačních projektů.

- Projekty trvaly déle, než se předpokládalo.
- Stály více peněz.

- Objevovala se negativa typu "hraní na úspěch/výsledek" a nikoli výsledek sám.
- K cíli se nešlo přímočaře, produkovaly se dále nevyužité (nevyužitelné) artefakty, věci do zásoby.
- Mezi zadavatelem (zákazníkem) a tvůrcem byla propast (formalizmus místo spolupráce).
- Vytvořené produkty pak často neodpovídaly aktuální potřebě...

## Principy

- Realizovat věci nejjednodušším možným způsobem.
- Tak se předejde nadměrné složitosti vytvářeného artefaktu.
- Je větší šance, že produkt bude bez chyb.
- Vynakládání prostředků lze při dodržení této zásady lépe kontrolovat.

## Zásada - stálý kontakt se zákazníkem - zadavatelem

Namísto jednorázového (obvykle složitého) jednání o smlouvách s přesnou (a v podstatě statickou) specifikací zadání:

- Zákazník (budoucí uživatel) je ve stálém, nejlépe denním a osobním kontaktu s vývojářem.
- Takové dynamické týmy musejí být dobře motivované, nehrát si na úspěch, ale musejí o něj stát.

## Zásada - časté uvolňování funkčního SW

- Zákazník často dostává nové verze (měsíčně, ještě lépe týdně), ihned konfrontuje stav s požadavky, hledá chyby...

## Zásada - vysoká kvalita

- Technická kvalita je základem. Neváhá se kvůli tomu podstoupit i radikální refaktoring kódu.

## Zásada - netvořit do zásoby

- Nemá se tvořit (programovat) "dopředu", "do zásoby".
- Ze zkušeností totiž vyplývá, že takové artefakty obvykle už nepoužijeme.

# Extremní programování (Extreme Programming, XP)

## Motivace pro Extreme Programming (XP)

Na vývoj software má vliv mnoho faktorů, které se neustále mění (zadání, návrh, technologie, trh, složení týmu apod.). Obecně lze říci, že změna je jedinou konstantou vývoje software. Problémem vývoje softwaru je schopnost úspěšného zvládnutí těchto změn za přijatelné náklady.

- Toto je východisko pro řadu moderních metodik programování (řízení SW projektů), které se souhrnně označují jako *agile programming*.
- Mezi ně patří i *Extreme Programming (XP)*.

## Co je XP

Viz J. Ministr, IT pro praxi, 2004:

- Extrémní programování (XP) je metodika vývoje softwaru, která je postavena na tvorbě zdrojového textu v týmech, jež tvoří dva až deset programátorů. XP používá obecně známé principy a postupy vývoje softwaru, tyto však dotahuje do extrémů.

## Charakteristika XP

Od ostatních metodik se XP odlišuje v následujících vlastnostech:

Automatizované testování	testy jsou tvořeny před samotnou tvorbou zdrojového textu za účelem následného ověření skutečného pokroku ve vývoji softwaru z hlediska jeho požadované funkcionality (testy navrhuje zákazník) a architektury programového modulu (testy navrhuje programátor).
Verbální komunikace	společně s testy a zdrojovým textem slouží ke sdělování systémové struktury a záměru projektu.
Intuice	Postupy, které podporují intuici programátorů a dlouhodobé zájmy projektu (testování, refaktORIZACE, integrace apod.).

## Východiska řízení týmu podle XP

Komunikace	Udržení řádných komunikačních toků ovlivňuje kvalitu jednotlivých postupů XP (testování modulů, párové programování, stanovení metrik apod.).
Jednoduchost	Vývoj software je řízen zásadou, že je lepší udělat jednoduchou věc dnes, s vědomím, že zítra bude možná nutné provést další změnu, spíše než udělat složitější změnu dnes, která nemusí být uživatelem využita. Jednoduchost a komunikace jsou spolu komplementární. Čím více tým komunikuje, tím je mu jasnější co přesně má dělat. Naopak čím je jednodušší systém, tím méně potřebujeme komunikovat.
Odvaha	Členové týmu jsou ochotni experimentovat s vyšším rizikem a ziskem.

## Hlavní zásady XP

Kvalitní práce	představuje fixní proměnnou ze čtyř proměnných pro posouzení projektu (šíře zadání, náklady, čas a kvalita) s hodnotou vynikající, při horší hodnotě členy týmu práce nebude bavit a projekt může skončit neúspěchem.
----------------	---

## Vedlejší zásady XP

Hraní na výhru	představuje soustředění práce týmu na kvalitu vyvíjeného produktu, nikoli na zbytečné alibistické činnosti, kdy tým pracuje „podle předpisů“ (mnoho papírů a porad), aby se tzv. „neprohrálo“.
Konkrétní experimenty	kdy všechna abstraktní rozhodnutí by měla být převedena do řady experimentů, které jsou následně otestovány.
Práce v souladu s lidskými instinkty	a nikoli proti nim představuje práci s krátkodobými zájmy lidí, kteří se rádi učí, vyhrávají, komunikují s ostatními apod.
Cestování nalehko	představuje hodnotné a účinné nástroje vývoje softwaru, které tvoří především testy a zdrojový text.

## Vývojové činnosti XP

1. Psaní zdrojového textu
2. Testování
3. Poslouchání
4. Navrhování logiky systému

## Hlavní techniky XP

- Plánovací hra stanoví šíři zadání následující verze software pomocí kombinace obchodních priorit a technických odhadů.
- Malá verze představuje rychlé uvedení jednoduchého systému do provozu. Následně jsou uvolňovány malé přírůstky systému ve velmi krátkých cyklech.
- Metafora pomáhá všem v projektu pochopit základní prvky systému a vztahy mezi nimi na základě jednoduchého přirovnání.
- Jednoduchý návrh u něhož je nadbytečná složitost ihned odstraněna v okamžiku jejího zjištění z návrhu.
- Testování představuje činnost programátorů a zákazníků, kdy programátoři testují zdrojový text z hlediska jeho programových vlastností, aby mohli pokračovat v jeho dalším psaní, a kdy uživatelé otestují funkcionality modulu, která je úspěšným provedením testu dokončena.
- RefaktORIZACE představuje restrukturalizaci systému s cílem zdokonalení jeho nefunkčních kvalit (pružnost, zjednodušení) bez vlivu na jeho chování.
- Párové programování představuje vývoj zdrojového textu dvěma programátory na jednom počítači.
- Společné vlastnictví
- Nepřetržitá integrace - okamžitá integrace dokončeného otestovaného přírůstku do systému.
- 40 hodinový týden plus se nepracuje nikdy přesčas dva týdny za sebou.
- Zákazník na pracovišti - odpovídá na otázky programátorů při vývoji software.
- Standardy pro psaní zdrojového textu

## Fáze XP projektu

Plánování	představuje stanovení termínu programátory společně se zákazníky na základě postupu plánovací hry. Do uvolnění první verze by mělo trvat mezi dvěma až šesti měsíci.
Smrt	představuje stav systému, kdy je software neschopen své existence z důvodu neekonomického rozšíření jeho funkcionality, entropie (tendence k většímu počtu chyb). Zákazníci i manažeři by měli souhlasit s ukončením údržby systému s tím, že se snaží identifikovat příčiny zániku systému.

## KISS (Keep It Simple, Stupid)

### Princip KISS

Netýká se jen programování, je to obecný princip vývoje či realizace čehokoli.

Snahou je produkovat co nejjednodušeji, bez zbytečností, bez nadbytečností, s minimem chyb.



KISS přístup patří do základního arzenálu *Agilního programování*.

Ideovými předchůdci principu jsou např.:

- tzv. Occamova břitva (Occam's Razor [[http://en.wikipedia.org/wiki/Occam%27s\\_Razor](http://en.wikipedia.org/wiki/Occam%27s_Razor)])

## Zásada - navrhovat jednoduše

- Realizovat věci nejjednodušším možným způsobem.
- Tak se předejde nadměrné složitosti vytvářeného artefaktu.
- Je větší šance, že produkt bude bez chyb.
- Vynakládání prostředků lze při dodržení této zásady lépe kontrolovat.

## Zásada - netvořit do zásoby

- Nemá se tvořit (programovat) "dopředu", "do zásoby".
- Ze zkušeností totiž vyplývá, že takové artefakty obvykle už nepoužijeme.

# Refaktoring

## Refaktoring - proč

Aplikace se vyvíjejí často překotně, pak je třeba "předělávat".

Refaktoring je právě takové "předělávání":

- přepis beze změny (vnějšího) chování/rozhraní
- směřuje ke zpřehlednění,
- optimalizaci,
- lepší rozšiřitelnosti
- robustnosti atd.

## Refaktoring - metody

Hlavní metody:

- manipulace na úrovni návrhu: přesuny tříd mezi balíky, přejmenování tříd
- přesuny metod mezi třídami
- vysunutí metody do nadtřídy
- "vytknutí" (do) rozhraní
- zapouzření datového prvku (proměnné) přístupovými metodami
- ...

## Refaktoring - nástroje kom.

Kvalitní komerční:

- RefactorIT - vč. free trial

## Refaktoring - nástroje o-s

Základní i v open-source:

- Eclipse 3.0
- NetBeans 4.0

# Programování řízené testy (Test-drive Development, TDD)

## Motivace

- Klasický způsob vývoje SW předpokládá testování jako následnou fázi - testy se píšou a spouštějí až po návrhu software
- To je však paradoxní, protože návrh testů vychází - stejně jako návrh vlastního kódu - ze specifikace a to dokonce přímočařeji. Test je přímý odraz specifikace, protože programovým kódem hlídá chování, které je specifikací očekáváno.
- Programování řízené testy proto předřazuje návrh a implementaci testů před vývoj vlastního SW.
- Jde o metodiku vývoje samotného SW, ne testů.

## Principy

- Nejprve sestavit test, pak psát kód.
- Stejný cíl jako *Návrh dle kontraktu* (Design by Contract), ale separuje testy od kódu, testuje "zvenčí".
- Výhody: včasná (okamžitá) detekce chyb v kódu
- Nevýhody: nelze použít tam, kde je obtížné automatizovaně testovat

## Kde nelze testovat?

... resp. teoreticky lze, prakticky však chybí přijatelné nástroje a metodiky:

- distribuovaná prostředí (částečně již lze: např. Cactus pro webové aplikace)
- grafická uživatelská rozhraní (i tam částečně lze: pomocí "robotů" na obsluhu GUI - na simulaci klikání a ovládání klávesnice)

## Generický postup TDD

1. Napsat test - na základě specifikace (požadavků) reprezentovaných např. případy užití (usecases).  
Test bude používat samotný kód prostřednictvím rozhraní.
2. Napsat kód - aby splnil specifikaci a komunikoval se světem pomocí rozhraní (API).
3. Spustit automatizované testy.
4. V případě chyb kód přeprogramovat (refactoring) a
5. spustit testy znovu.

## Odkazy

některé převzaty z Wikipedie, hesla Test-driven Development [[http://en.wikipedia.org/wiki/Test\\_driven\\_development](http://en.wikipedia.org/wiki/Test_driven_development)]:

- S.W.Ambler: Introduction to Test Driven Development (TDD) [<http://www.agiledata.org/essays/tdd.html>]
- TestDriven.com [<http://www.testdriven.com/>]
- Test Driven Development (jiná Wiki) [<http://c2.com/cgi/wiki/TestDrivenDevelopment>]

# Systémy správy verzí

## Motivace

Systémy pro správu verzí jsou nezbytností pro

- údržbu větších SW projektů
- projektů s více účastníky
- projektů s vývojem z více míst

Pouhý sdílený, vzdáleně přístupný souborový systém nestačí.

## Principy

- efektivně ukládat více verzí souborů i adresářů (často s malými změnami)
- rychle získávat aktuální verzi, ale
- mít možnost vrátit se ke starší verzi
- zjistit rozdíly mezi verzemi
- zajišťovat proti souběžné editaci z více míst
- umožnit i lokální práci s následným potvrzením do systému (commit)

## Klasická řešení - RCS a CVS

RC    Revision Control System  
S  
CV    Concurrent Version System  
S

RCS je klasický, původně na UNIXech existující systém navržený pro sledování více verzí souborů.

Prvotním cílem bylo zefektivnit ukládání více verzí

- s novou verzí se nemusí ukládat celý soubor
- rychle se dají zjistit rozdíly (změny) mezi verzemi
- historie změn (changelog, history) se lehce udržuje
- změny lze identifikovat i názvy - pojmenovat symbolickými klíčovými slovy (\$Author\$, \$Date\$...)

## Typické příkazy správy verzí

(podobné jsou v CVS, SVN)

commit	publikuje (odešle, potvrdí) změny z lokálního pracovního prostoru do skladu (repository)
remove	maže soubory z lokálního pracovního adresáře, ze skladu se smažou až po "commit"
add	přidá nový soubor do pracovního adresáře, do skladu se přidají až po "commit"
update	aktualizuje lokální kopii pomocí změn zaregistrovaných ostatními ve skladu
checkout	vytvoří soukromou lokální kopii požadovaných souborů ze skladu, tyto můžeme lokálně editovat a posléze potvrdit (commit) zpět

## Klienti

Syst. správy verzí nabízejí

- nativní klienty integrované do hostujícího prostředí
- webové rozhraní spíše pro prohlížení
- API pro přímý/programový přístup

## Pro co se systémy nehodí?

- pro ukládání (stabilních) artefaktů
- jako úložiště (read-only) souborů pro stahování
- ... kdyby se to hodilo na všechno, nabízel by to každý filesystém...

## Subversion

- implementace systému řízení verzí
- moderní alternativa CVS
- jako server dostupný na všechny běžné platf. (zejm. Linux, Win)
- server existuje jako samostatná aplikace, standardní použití je však s webovým serverem Apache
- klienti taktéž, např. na Win

## Subversion - klient Tortoise pro Windows

Klient pro Win 2k, XP i 98... vč. integrace do GUI

- kontextové nabídky
- nativní nástroje

## Tortoise -- vzdálený přístup

- Tortoise umožňuje bezpečný přístup k vzdálenému úložišti i prostřednictvím šifrovaných protokolů
- (server potom ale musí běžet přes Apache...)
- používá se upravený klient PuTTY

## Správa sestavování - Ant

### Charakteristika

- Ant je platformově přenositelnou alternativou nástroje typu Make
- Ant je rozšiřitelný - lze psát nejen nové "cíle", ale i definovat dílčí kroky - "úlohy"
- Popisovače sestavení používají XML syntaxi, psaní není tak náročné jako makefile

### Motivace

- Proč Ant, když je tu dlouho a dobře fungující Make?
- Make byl koncipován jako doplněk shellu, psaly se skripty a nativní (platformové) nástroje
- Syntaxe byla složitá a neflexibilní
- Make jako takový nebyl rozšiřitelný
- Make byl zaměřen převážně na potřeby původního použití - jazyk C, unixový shell

### Struktura projektu

Řízení sestavování Antem postupuje podle popisu v souboru build.xml.

Ten obsahuje následující prvky:

project	celý projekt, obsahuje sadu cílů, jeden z nich je hlavní/implicitní
target	cíl, nejmenší zvenčí spustitelná jednotka algoritmu sestavování
task	úloha, atomický krok sestavení, lze použít task vestavěný nebo uživatelský

## Příklad 1

## Závislosti

- Mezi cíly mohou být definovány závislosti.
- Závisí-li volaný cíl na jiném, musí být nejprve splněn cíl výchozí a pak teprve cíl závisející.
- Cíl může záviset i na více jiných.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C, B, A"/>
```



### Varování

Pozor na cyklické závislosti!

## Příklad 2

## Maven

## Motivace

Pro sestavování, správu a údržbu SW projektů menšího a středního rozsahu se delší dobu úspěšně využíval systém Ant [<http://ant.apache.org>].

Oproti klasickým nástrojům typu unixového make poskytoval Ant platformově nezávislou možnost popsat i složité postupy sestavení, testování a nasazení výsledků SW projektu.

Ant měl však i nevýhody:

- pro každý projekt (i když už jsme podobný řešili) musíme znovu sestavit - poměrně velmi technický

- popisovač (build.xml **InstantWeb**)  
[<http://www.instantweb.com/foldoc/foldoc.cgi?build.xml>)]

- popisovač je vždy téměř stejný a tudíž
- neříká nic o *obsahu* vlastního projektu, je jen o procesu sestavení, nasazení...
- neumožňoval zachytit *metadata* nezbytná pro zařazení projektu do širšího kontextu, mezi související projekty, atd.

## Maven - charakteristika

- nástroj řízení SW projektů
- open-source, součást skupiny nástrojů kolem Apache
- dostupný a popsáný na <http://maven.apache.org>
- momentálně (září 2004) již jako použitelná, ostrá, verze 1.0

## Project Object Model (POM)

- projekt řízený Mavenem je popsán tzv. *POM* (Project Object Model), obvykle `project.xml` **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?project.xml>]
- POM nepopisuje postup sestavení, ale *obsah* projektu, jeho název, autora, umístění, licenci...
- postup sestavení je "zadrátován" v Mavenu, protože je pro většinu projektů stejný
- programátor není frustrován opakováním psaní popisovačů `build.xml` **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?build.xml>], návrhem adresářové struktury...
- nicméně, Maven je založen na Ant, jeho `build.xml` **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?build.xml>] popisovače lze znovupoužít

## Projekt v Mavenu

Základní filozofie projektu v Mavenu:

- jeden projekt => jeden tzv. *artefakt*

Artefaktem může být typicky:



- .jar - obyčejná aplikace nebo knihovna (javové třídy, soubory .properties, obrázky...)
- .war - webová aplikace (servlety, JSP, HTML, další zdroje, popisovače)
- .ear - enterprise (EJB) aplikace (vše výše uvedené pro EJB, popisovače)

## Maven repository

- základním organizačním nástrojem pro správu vytvořených (nebo používaných) artefaktů je *repository*
- artefakt, tj. výstup projektu, se může v repository vyskytovat ve více verzích
- repository je:
  - vzdálená (remote) slouží k centralizovanému umístění jak vytvořených, tak používaných artefaktů
  - dosažitelná pro čtení pomocí HTTP: je to de-facto běžné webové místo
  - lokální (local) slouží k ozrcadlení používaných artefaktů ze vzdálené repository
  - typicky zvlášť každému uživateli - v jeho domovském adresáři
  - slouží též k vystavení vytvořených artefaktů "pro vlastní potřebu"
- Maven má nástroje (pluginy) pro vystavování artefaktů do repository

## Instalace a nastavení

Maven lze stáhnout z <http://maven.apache.org> v binární i zdrojové distribuci.

Binární distribuce je buďto čistě "java-based" nebo ve formě windowsového .exe **InstantWeb**  
[<http://www.instantweb.com/foldoc/foldoc.cgi?.exe>].

Pak se nainstaluje do Program Files **InstantWeb**  
[<http://www.instantweb.com/foldoc/foldoc.cgi?Program Files>]

Po instalaci je třeba nastavit proměnnou prostředí MAVEN\_HOME **InstantWeb**  
[[http://www.instantweb.com/foldoc/foldoc.cgi?MAVEN\\_HOME](http://www.instantweb.com/foldoc/foldoc.cgi?MAVEN_HOME)] na adresář, kam se nainstaloval.

Kromě toho ještě přidat adresář %MAVEN\_HOME%\bin **InstantWeb**  
[[http://www.instantweb.com/foldoc/foldoc.cgi?%MAVEN\\_HOME%\bin](http://www.instantweb.com/foldoc/foldoc.cgi?%MAVEN_HOME%\bin)] do PATH **InstantWeb**  
[<http://www.instantweb.com/foldoc/foldoc.cgi?PATH>].

## Příklad POM (project.xml)

Příklad                      minimálního                      popisovače                      project.xml      **InstantWeb**

[<http://www.instantweb.com/foldoc/foldoc.cgi?project.xml>]:

```
<project>
  <pomVersion>3</pomVersion><!-- verze POM - zatím vždy 3 -->
  <id>RunningCalculator</id><!-- jednoznačné id projektu -->
  <name>RunningCalculator</name><!-- (krátké) jméno/nemusí být jednoznačné -->
  <currentVersion>0.1</currentVersion><!-- momentální verze -->
  <organization><!-- organizace vytvářející projekt -->
    <name>Object Computing, Inc.</name>
  </organization>
  <inceptionYear>2004</inceptionYear><!-- rok zahájení projektu -->
  <shortDescription>calculates running pace</shortDescription><!-- stručný popis
<developers/>
```

## Příklad POM - pokračování

Příklad                      minimálního                      popisovače                      project.xml      **InstantWeb**

[<http://www.instantweb.com/foldoc/foldoc.cgi?project.xml>]:

```
<dependencies><!-- závislosti -->
  <dependency><!-- závislost -->
    <groupId>junit</groupId><!-- skupina artefaktu -->
    <artifactId>junit</artifactId><!-- označení artefaktu -->
    <version>3.8.1</version><!-- verze artefaktu -->
  </dependency>
</dependencies>
<build><!-- odkud se co a jak sestavuje... -->
  <sourceDirectory>src/java</sourceDirectory><!-- adresář zdrojů -->
  <unitTestSourceDirectory>src/test</unitTestSourceDirectory><!-- adresář zdrojů
  <unitTest><!-- které soubory jsou třídy testů -->
    <includes>
      <include>/**/*.Test.java</include>
    </includes>
  </unitTest>
</build>
</project>
```

## Struktura POM obecně

```
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="MAVEN_HOME/maven-project.xsd">
  <pomVersion>3</pomVersion>
```

```
<id>unique-project-id</id>
<name>project-name</name>
<groupId>repository-directory-name</groupId>
<currentVersion>version</currentVersion>
<!-- Management Section -->
<!-- Dependency Section -->
<!-- Build Section -->
<!-- Reports Section -->
</project>
```

## Proměnné (properties) v POM

- Jsou podobně jako u Antu definovatelné a využitelné (odkazovatelné) v popisovači, zde `project.xml` [InstantWeb](http://www.instantweb.com/foldoc/foldoc.cgi?project.xml) [http://www.instantweb.com/foldoc/foldoc.cgi?project.xml].
- Vyskytne-li se zavedení určité vlastnosti (property) vícekrát, uplatní se *poslední*.
- Vlastnosti jsou vyhledávány v pořadí:
  1. `project.properties` [InstantWeb](http://www.instantweb.com/foldoc/foldoc.cgi?project.properties)  
[http://www.instantweb.com/foldoc/foldoc.cgi?project.properties]
  2. `build.properties` [InstantWeb](http://www.instantweb.com/foldoc/foldoc.cgi?build.properties)  
[http://www.instantweb.com/foldoc/foldoc.cgi?build.properties]
  3. `${user.home}/build.properties` [InstantWeb](http://www.instantweb.com/foldoc/foldoc.cgi?${user.home}/build.properties)  
[http://www.instantweb.com/foldoc/foldoc.cgi?\${user.home}/build.properties]
  4. vlastnosti specifikované na příkazové řádce `-Dkey=value`
- Na vlastnost se lze odvolat pomocí `${property-name}`

## Struktura repository

Týká se jak vzdálené, tak lokální repository.

Obecně je relativní cesta v rámci repository k hledanému artefaktu: `repository/resource-directory/jars/jar-file`

konkrétní např.: `repository/junit/jars/junit-3.8.1.jar`

## Cíle v Mavenu

Cíle (goals) v Mavenu odpovídají zhruba antovým cílům (target).

Spouštění Mavenu vlastně odpovídá příkazu k dosažení cíle ("attaining the goal"):

**maven** *plugin-name[:goal-name]* **InstantWeb**  
[<http://www.instantweb.com/foldoc/foldoc.cgi?maven> ]

## Maven Plugins

Zásuvné moduly (plugins) obsahují předdefinované cíle (goals) a jsou taktéž uloženy v repository.

Většinou jsou jednoúčelové, slouží/směřují k jednomu typu artefaktu, např.:

- checkstyle, clean, clover, cruisecontrol, dist, ear, eclipse, ejb, fo, genapp, jalopy, jar, java, javadoc, jboss, jcoverage, maven-junit-report-plugin, pom, site, test, war, xdoc

## Často používané cíle

clean	smaže vygenerované soubory (podstrom target <b>InstantWeb</b> ) [ <a href="http://www.instantweb.com/foldoc/foldoc.cgi?target">http://www.instantweb.com/foldoc/foldoc.cgi?target</a> ]
java:compile	přeloží všechny javové zdroje
test	spustí všechny testy
site	vygeneruje webové sídlo projektu
dist	vygeneruje kompletní distribuci

## Vytvoření projektu

1. vytvořit prázdný adresář pro vytvářený projekt
2. spustit **maven genapp** **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?maven> genapp] (Zeptá se na id projektu, jeho jméno a hlavní balík. V něm předgeneruje jednu třídu.)
3. tím se vytvoří následující soubory:
  - project.xml, project.properties
  - src/conf/app.properties
  - src/java/package-dirs/App.java
  - src/test/package-dirs/AbstractTestCase.java
  - src/test/package-dirs/AppTest.java

- `src/test/package-dirs/NaughtyTest.java`

## Reporting

Generování reportů (zpráv) je jednou se základních funkcí Mavenu.

Které reporty se generují, je regulováno v `project.xml` v sekci *reports*:

```
<reports>
  <report>maven-checkstyle-plugin</report>
  <report>maven-javadoc-plugin</report>
  <report>maven-junit-report-plugin</report>
</reports>
```

## Rozšíření možností Mavenu

Cílů (goals) je sice v Mavenu řada, ale přesto nemusejí stačit anebo je třeba měnit jejich implicitní chování.

Potom lze před nebo po určitý cíl připojit další cíl pomocí *preGoal* a *postGoal*.

Ty se specifikují buďto v nebo.

1. `maven.xml` **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?maven.xml>] ve stejném adresáři jako `project.xml` **InstantWeb** [<http://www.instantweb.com/foldoc/foldoc.cgi?project.xml>] nebo
2. v zásuvném modulu (pluginu)

Zcela nové cíle je možné napsat ve skriptovacím jazyku *jelly* (s XML syntaxí).