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

Obsah

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 produk-

	tu, 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 funkcionalitu 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])

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 způsobe vývoje SW předpokládá testování jako následnou fázi - testy se píší 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]:

- 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

RCS Revision Control System

CVS Concurrent Version System

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
ckeckout	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 webových 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` WIKIPEDIA The Free Encyclopedia [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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 popsaný 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` WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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` WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=build.xml], návrhem adresářové struktury...
- nicméně, Maven je založen na Ant, jeho `build.xml` WIKIPEDIA
  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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 umisťování 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`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=.exe].

Pak se nainstaluje do `Program Files`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Program Files]

Po instalace je třeba nastavit proměnnou prostředí `MAVEN_HOME`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=MAVEN_HOME] na adresář, kam se nainstaloval.

Kromě toho ještě přidat adresář `%MAVEN_HOME%\bin`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=%MAVEN_HOME%\bin] do `PATH`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=PATH].

Příklad POM (project.xml)

Příklad minimálního popisovače `project.xml`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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` WIKIPEDIA The Free Encyclopedia

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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 zdro
      <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`WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=project.xml].

- Vyskytne-li se zavedení určité vlastnosti (property) vícekrát, uplatní se *poslední*.

- Vlastnosti jsou vyhledávány v pořadí:

  1. `project.properties`WIKIPEDIA

     [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=project.properties]

  2. `build.properties`WIKIPEDIA

     [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=build.properties]

  3. `${user.home}/build.properties`WIKIPEDIA

     [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=${user.home}/build.properties]

  4. vlastnosti specifikované na příkazové řádce -D*`key=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`*]**WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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` WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=target])
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** WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=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` WIKIPEDIA
   [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=maven.xml] ve stejném adresáři jako `project.xml` WIKIPEDIA
   [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=project.xml] nebo
2. v zásuvném modulu (pluginu)

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