Testování Java EE aplikací

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 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 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]

Motivace

Pozice testování v soudobém vývoji SW

Role testování narůstá, testování podstatnou standardní součástí vývoje:

- Agilní metodiky (např. Extreme Programming), Test-driven development…

Testování Java EE aplikací

- Mocné, ale složité aplikační prostředí
- Aplikace typicky vícevláknové, klient-server
- Soudobé metodiky jsou přitom často založeny na testování…

Motivace

- Jasná motivace: SW chyby mohou mít a mají katastrofální následky…
- a přinejmenším zdražují vývoj – čím později je chyba odhalena, tím jsou náklady větší
- Testování patří k „dobrému tónu“ i u open-source vývoje (nebo hlavně tam?)

Specifikace vs. implementace – zdroj chyb

- Nesoulad mezi specifikací a implementací

- Nedostatečná/neúplná/chybná specifikace
- …systém pak specifikaci splňuje a přesto nevyhovuje…

Zásady (Scott Ambler)

- testovat během celého životního cyklu projektu,
- vyvíjet testy ještě před samotným kódem,
- (kontinuálně) testovat všechny artefakty programu,
- testováním odhalovat příčiny chyb a nepřekrývat je,
- při testování používat přitom jednoduché a efektivní nástroje a
- testování zahrnout po všech stránkách do vývoje

Java Enterprise Edition

Architektura

- Platforma pro rozsáhlé, komponentně orientované programové systémy schopné běhu aplikačních serverů.
- Prostředí = middlewarové zázemí pro vývoj, nasazení, konfiguraci, běh, vzájemnou komunikaci a sledování.
- Systémy mají řadu vrstev a jsou často distribuované mezi více fyzických počítačů.

Typická Java EE aplikace

Typická Java EE aplikace s Enterprise JavaBeans (EJB):

- klientské webové rozhraní,
- to komunikuje se servlety a stránkami JSP, JSF...
- aplikační logika realizovaná Session Beans (mezi tím příp. fasáda)
- komponenty Entity Beans s perzistencí zajištěnou relační databází.
- Namísto EJB se dnes často používá objektově-relační mapování běžných (POJO) objektů,
- pro řízení na vyšších vrstvách se využívají webové rámce.

Jak a co testovat?

Type 1: code logic unit testing	Probably the best strategy for these tests is to use a Mock Objects type framework.
Type 2: integration unit testing	These tests will exercise the interactions with the container. (Cactus)
Type 3: functional unit testing	These unit tests will let you test the returned values from your server code. This is for example HttpUnit (Note that HttpUnit also performs standard functional testing - as opposed to functional unit testing -, which let you test full use cases - a login use case for example, which is comprised of several requests/responses).

Techniky a nástroje testování

Obecně použitelné - junit

Nástroj JUnit, http://junit.org

Příklad testu:

```
public class StackTest extends JUnit.framework.TestCase {

    private Stack st;
    public StackTest(String testCaseName) {
        super(testCaseName);
    }
    // vytvoří přípravek, nastaví prostředí každého testu
    public void setUp() {
        st = new Stack(10);
    }
    // testuje prázdnost právě vytvořeného zásobníku
    public void testEmptyAfterCreation() {
      assertTrue("Stack should be empty after creation.",
                 st.isEmpty());
    }
    // testuje neprázdnost zásobníku po vložení prvku
    public void testPushPopEquals() {
      st.push("something");
      assertEquals("What was pushed, must be popped...",
                 "something", st.pop());
    }
    ...
```

```
    // uklidí po testu, je-li třeba
    public void tearDown() {  }
}
```

Integrace testování do vývoje

Integrovaná do všech reálně používaných vývojových nástrojů:

- Maven, Ant,
- IDE (NetBeans, Eclipse, IDEA)
- a dokonce i výuková prostředí – BlueJ.

Dnes se vždy předpokládá, že ke KAŽDÉMU netriviálnímu projektu existují testy.

Testování a správa (verzí) zdrojů

- Systémy správy zdrojových kódů zajišťují centralizovanou evt. distribuovanou správu s podporou verzování (CVS, Subversion)
- Při neúspěšnosti integračních testů se tak můžeme v repozitáři vrátit ke starší verzi určitého modulu a pokusit se chybu najít.
- Systémy evidence chyb (např. BugZilla)

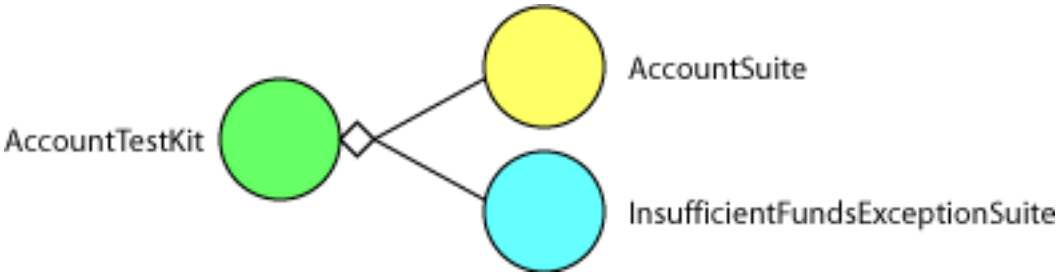
Řízení testů

V netriviálních projektech, tam, kde se integruje např. jednou za den, není možné pokaždé spouštět všechny testy.

Pokročilé nástroje je umějí

- uskupovat
- parametrizovat
- rozběhnout jen testy, které předtím neprošly
- pohodlně spustit např. všechny testy v jednom balíku

Např. *Artima SuiteRunner*:



Protokolování testů

Např. Maven pracuje hojně s protokoly (reports) vč. testovacích.

Většina testovacích nástrojů umí kromě "transientního" zobrazení průběhu výsledky testu zaznamenat.

Elementárně např. ve dvojici nástrojů - Ant tasks - `junit` (otestuje), `junitreport`WIKIPEDIA
The Free Encyclopedia
[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=] (zaformátuje protokoly):

Otestování:

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <pathelement location="${build.tests}"/>
    <pathelement path="${java.class.path}"/>
  </classpath>

  <formatter type="plain"/>

  <test name="my.test.TestCase" haltonfailure="no" outfile="result">
    <formatter type="xml"/>
  </test>

  <batchtest fork="yes" todir="${reports.tests}">
    <fileset dir="${src.tests}">
      <include name="**/*Test*.java"/>
      <exclude name="**/AllTests.java"/>
    </fileset>
  </batchtest>
</junit>
```

Formátování reportu:

```
<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="./report/html"/>
</junitreport>
```

Testování vrstev aplikace

Testy datové vrstvy

Co je u DB aplikací třeba?

- uvedení prostředí před testem do vhodného stavu
- po testu “uklidit”

DBUnit – rozšíření JUnit (http://dbunit.sf.net)

- dokáže podle definičního souboru vytvořit tabulky,
- z XML či jiných zdrojů do nich načíst data
- a spustit testy.

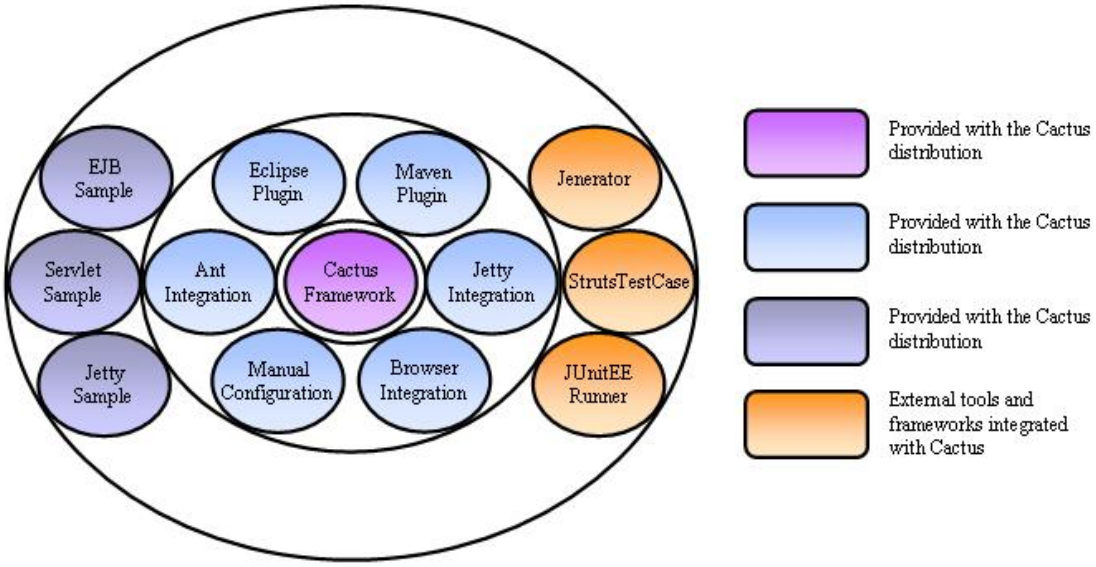
Testy aplikační vrstvy

Používá se JUnit a nadstavby, resp. obdobné nástroje:

TestNG	http://testng.org
	moderní, na anotacích založený systém, lepší než junit díky jemnější organizovatelnosti testů (skupiny testů, parametry, protokolování...)
djunit	http://www.fi.muni.cz/~tomp/djunit
	rozšíření junit s možností "krmit" testy daty ze souborů, proudů, zdrojů (resources)
junit v4	nová verze JUnit s podporou anotacemi
Artima SuiteRunner	http://www.artima.com/suiterunner/index.html
	Reporter: Collect and present test results in a highly customizable way to the user. Examples are text, graphics, web pages, database output, CSV files, XML, and email alerts. Runpath: Load classes for your tests from anywhere with this easy to configure list filenames, directory paths, and/or URLs. Recipe File: Capture and save the properties of a run of a particular suite of tests in a file for easy reuse.

Integrační testy

Systém Apache Cactus, http://jakarta.apache.org/cactus/



The Cactus Ecosystem

Architektura Cactus:

The Cactus Framework	This is the heart of Cactus. It is the engine that provides the API to write Cactus tests.
The Cactus Integration Modules	They are front ends and frameworks that provide easy ways of using the Cactus Framework (Ant scripts, Eclipse plugin, Maven plugin, ...).

Testy prezentační vrstvy – webové rozhraní

Nástroj JWebUnit umožní virtuálně "klikat" po webové aplikaci a srovnávat výstupy od serveru s očekávanými.

```
public class JWebUnitSearchExample extends WebTestCase {
   ...
   public void setUp() {
      getTestContext().setBaseUrl("http://www.google.com");
   }
   public void testSearch() {
      beginAt("/"); // na kterém URL začít
      setFormElement("q", "httpunit"); // co na stránce vybrat
      submit("btnG"); // jaké tlačítko stisknout
      clickLinkWithText("HttpUnit"); // kam kliknout
      assertTitleEquals("HttpUnit"); // co má od serveru přijít
      assertLinkPresentWithText("User's Manual");
```

```
    }
}
```

Dalšími příklady jsou:

- HttpUnit
- příp. komplexnější, které webovou vrstvu otestují také - Cactus nebo MockRunner

Testy prezentační vrstvy – desktopové aplikace/GUI

Nástroj Marathon umožní virtuálně "klikat" po GUI aplikaci a srovnávat výstupy od serveru s očekávanými.

- obecně se testuje těžko…
- většinou na principu zachycení či naskriptování uživatelských akcí nad GUI
- na jednodušší v Javě lze použít "robota" pro javové GUI

Příklad v JWebUnit:

```
public class JWebUnitSearchExample extends WebTestCase {
   ...
   public void setUp() {
      getTestContext().setBaseUrl("http://www.google.com");
   }
   public void testSearch() {
      beginAt("/"); // na kterém URL začít
      setFormElement("q", "httpunit"); // co na stránce vybrat
      submit("btnG"); // jaké tlačítko stisknout
      clickLinkWithText("HttpUnit"); // kam kliknout
      assertTitleEquals("HttpUnit"); // co má od serveru přijít
      assertLinkPresentWithText("User's Manual");
   }
}
```

Mock-objekty

Motivace

Testování Java EE aplikací je náročné, protože:

- reálné prostředí běhu JavaEE aplikací je příliš složité, dlouho startuje…
- Často stačí testovat nad prostředím nasimulovaným pomocí mock-objektů (nepravých, zástupných objektů)
- Je-li vše OK, následně se testuje v "ostrém" prostředí jinými nástroji (např. Cactus, JWebUnit, DBUnit...)

Mockrunner

Mockrunner umí pomocí mock-objektů testovat JDBC, webové (servletové) aplikace i EJB.

Příklad:

```
public class RedirectServletTest extends BasicServletTestCaseAdapter {
    protected void setUp() throws Exception {
        super.setUp();
        createServlet(RedirectServlet.class);
    }
    public void testServletOutputAsXML() throws Exception {
        addRequestParameter("redirecturl", "http://www.mockrunner.com");
        doPost();
        Element root = getOutputAsJDOMDocument().getRootElement();
        assertEquals("html", root.getName());
        Element head = root.getChild("head");
        Element meta = head.getChild("meta");
        assertEquals("refresh", meta.getAttributeValue("http-equiv"));
        assertEquals("0;URL=http://www.mockrunner.com",
                     meta.getAttributeValue("content"));
    }
}
```

Zátěžové testování

Apache JMeter

Apache JMeter [http://jakarta.apache.org/jmeter/] 100% pure Java desktop application designed to load test functional behavior and measure performance.

Nejen pro webové aplikace, ale i např. databázové (SQL) dotazy, JMS komunikaci, FTP přenosy...

- Can load and performance test HTTP and FTP servers as well as arbitrary database queries (via JDBC)
- Complete portability and 100% Java purity

- Full multithreading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by seperate thread groups.
- Caching and offline analysis/replaying of test results.

Highly Extensible: o Pluggable Samplers allow unlimited testing capabilities. o Several load statistics may be choosen with pluggable timers . o Data analysis and visualization plugins allow great extendibility as well as personalization. o Functions (which include JavaScript) can be used to provide dynamic input to a test o Scriptable Samplers (BeanShell is supported in version 1.9.2 and above)

JMeter - prostředí