FG Forrest, a.s.
Jan Novotný

Automatické testování v praxi 2

Dependency injection (IOC)

- motivace
  - tight coupling vede ke složitým integračním testům (cz.novoj.business.UserManagerNoDi)
- přínos IOC
  - zvyšuje kvalitu aplikace díky vysoké dekompozici
  - výrazně zjednodušuje testování
  - umožňuje jednodušší změny v aplikaci v budoucnosti
  - při použití použitím frameworku se vyhneme psaní velkého množství gluecode (ve srovnání s factory patternem)
- obtíže
  - vyžaduje změnu přístupu
- implementace
  - Spring Framework
  - Google Guice
  - PicoContainer a další

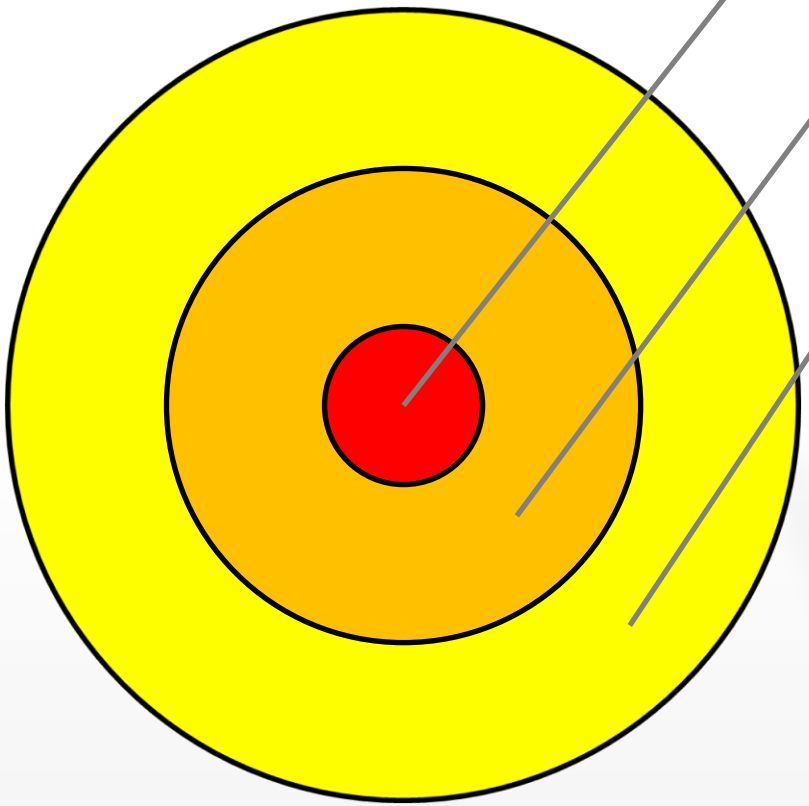
Spring Framework – základní podpora pro testy

- test framework agnostic (@RunWith)
- automatická inicializace a cachování aplikačních kontextů (@ContextConfiguration, @DirtiesContext)
- dependency injection v testech
  - autowire by type (@Autowired)
  - autowire by type and name (@Qualifier)
- podpora profilů i pro testovací frameworky, které toto neposkytují (@ProfileValueSourceConfiguration, @IfProfileValue)

- viz. cz.novoj.dao.mysql.AbstractDaoTest

Techniky testování

Testování různých vrstev systémů přináší odlišné problémy a vyžaduje odlišný přístup



- Datová vrstva
- Aplikační vrstva
- Prezentační vrstva

Datová vrstva (databáze) – patterny #1

- Database sandbox
  - motivace:
    - test / developer race, version mixing
  - řešení
    - každý vývojář má vlastní databázi
    - každé další „použití“ aplikace má vlastní databázi
  - obtíže
    - složitý proces pro vytváření nových databází
    - problematické určování cílové databáze pro běh testů (podrobněji rozabráno dále)

- Automatic database model setup / update
  - motivace
    - zjednodušení setup / update databázového modelu na více prostředích
    - použité technologie nepodporují automatickou tvorbu / aktualizaci datového modelu
  - řešení
    - Hibernate (hibernate.hbm2ddl.auto=create/update)
    - vlastní řešení (spring/spring-autoupdate.xml)
  - obtíže
    - řešení problémů v průběhu aktualizace (u DDL nelze použít rollback)

Datová vrstva (databáze) – patterny #2

- InMemory database
  - motivace
    - opětovné vyváření dat v DB z nuly před každým testem výrazně navyšuje čas testů
    - udržování konzistence testovacích dat ručně programátorem je velmi náročné a podléhá chybám, které se projeví až při běhu více testů
  - řešení
    - použití některé z memory databází (např. http://hsqldb.org/) – na začátku testu se vždy vytvoří kompletně nová databáze, tabulky a vytvoří se data, na konci testu se databáze dropne
    - http://www.theserverside.com/tt/articles/article.tss?l=UnitTesting
  - obtíže
    - spolehlivá implementace není jednoduchá
    - nutné podporovat další typ databázového stroje (může být nutné upravit datovou vrstvu)

Datová vrstva (databáze) – patterny #3

- Transaction rollback teardown (cz.novoj.dao.mysql.UserDaoImplTest)
  - motivace
    - opětovné vyváření dat v DB z nuly před každým testem výrazně navyšuje čas testů
    - udržování konzistence testovacích dat ručně programátorem je velmi náročné a podléhá chybám, které se projeví až při běhu více testů
  - řešení
    - na začátku testu otevřeme transakci a zajistíme, aby veškerá logika testu a logika z testu volaná se na této transakci podílela, na konci testu se volá rollback
    - podpora ve Spring Frameworku - @Transactional, @NotTransactional
  - obtíže
    - spolehlivá implementace není jednoduchá (vyžaduje zapojení TransactionManagera)
    - modifikace dat uvnitř testu není zvenčí vidět (záleží na izolaci transakce), pokud test selže, na datech nelze poznat proč (změny jsou rollbacknuty)
    - nelze použít na kód, který nemůže běžet v transakci (DDL – a to i např. temporary table)

Aplikační vrstva – techniky #1– test double

- motivace
  - testy jsou pomalé
  - nemáme dosud cílovou implementaci (např. dodává třetí strana)
  - příprava prostředí pro test je příliš složitá
  - ověření správnosti aplikační logiky je příliš složité (je třeba testovat tzv. „side effects“)
- řešení
  - použití Fake objektů
    - spočívá ve výměně původní implementace za jednodušší určenou pouze pro testy
    - tip: http://josql.sourceforge.net
  - použití stub objektů (cz.novoj.emt.stub.BusinessObjectStubTest)
    - princip „kukaččího vejce“
    - výměna původní implementace za implementaci:
      - která simuluje vstupy do testovacího objektu tak, aby mohlo dojít k otestování požadované situace
      - která přijímá a zaznamenává výstupy testovaného objektu, aby mohlo dojít k ověření požadované situace
  - použití mock objektů (cz.novoj.emt.mock.BusinessObjectMockTest)
    - princip „natáčení filmu“
    - podobají se stub objektům, ale pracuje se s nimi odlišně
    - mock objekt je možné „naprogramovat“ na očekávané chování objektu, který se testuje, nakonec umějí toto chování ověřit
- obtíže
  - duplikuje se funkcionalita – „dvojí práce“
  - nahrazená aplikační logika není testována, není testována ani správná spolupráce mezi testovaným objektem a okolím které je nahrazováno
- pro zajímavost:
  - http://martinfowler.com/articles/mocksArentStubs.html

Aplikační vrstva – techniky #2

- testování private metod (cz.novoj.emt.pmt.BusinessObjectPrivateMethodTest)
  - motivace
    - potřeba testovat logiku v private metodách
  - obtíže
    - private metody nejsou z jiné (tedy ani testovací) třídy dostupné
  - řešení (hezky popsáno zde: http://www.artima.com/suiterunner/private2.html)
    - netestovat – nebo zveřejnit
    - rozšířit viditelnost na friendly / protected
    - inner class v produkční class určená pro testování
    - reflexe
      - ve Springu např. ReflectionTestUtils
        - void setField(Object target, String name, Object value, Class type)
        - void invokeSetterMethod(Object target, String name, Object value, Class type)

Aplikační vrstva – techniky #3

- práce se systémovým časem (cz.novoj.business.PollManagerTest)
  - motivace
    - v aplikační logice pracujeme s aktuálním časem, který se mění
    - je problematické napsat test tak, aby fungoval dnes i po nějakém čase
  - řešení
    - využití knihovny JodaTime (http://joda-time.sourceforge.net/)

- testování odeslání emailů (cz.novoj.emt.smtp.SmtpTest)
  - motivace
    - v aplikační logice odesíláme emaily přes SMTP server - jak ověřit, že email složíme správným způsobem, který dokáže SMTP server zpracovat?
    - při využití mock objektů ověříme pouze, že došlo k pokusu o odeslání – nikoliv, že zpráva byla složena tak, aby byla SMTP serverem zpracovatelná
  - řešení
    - využití knihovny SubEthaSmtp (http://subethasmtp.tigris.org/) – jednoduchá implementace SMTP serveru

Aplikační vrstva – techniky #4

- použití specifické konfigurace pro konkrétní prostředí (cz.novoj.spring.utils.HostConfigurableContextLoader)
  - Motivace
    - testy spouštíme pravidelně na více prostředích (lokální vývojová prostředí, CI)
    - trávíme spoustu času zprovozněním testů na konkrétním prostředí (odlišné umístění databází, hesla, přístupová jména, url apod.)
  - řešení
    - automatická detekce prostředí a úprava konfigurace
    - v prostředí Springu - využití PropertyPlaceholderConfigurer, nebo dokonce použití odlišných XML konfigurací
  - obtíže
    - fixace všech konfigurací pro všechna možná prostředí na classpath (včetně hesel)

- testování aspektů (cz.novoj.business.UserManagerIntegrationTest)
  - motivace
    - v kódu používáme aspekty např. pro řízení transakcí, security apod. a chceme otestovat, zda se při běhu správně aplikují
    - obtížně se zjišťuje zda se při konkrétním volání metody třídy aplikovala advice, kterou očekáváme – obvykle se jedná jen o konfigurační testování, které se ve výsledku ale obtížně testuje
      - buď jsme nuceni k tomu testovat side-effect naší advice
      - nebo musíme vyměnit runtime advice za testovací, což zesložiťuje setup a navíc již ztrácíme integrační hodnotu takového testu
  - řešení
    - využití logování (např. log4j) v advice k ověření její aplikace a průběhu
    - blíže rozebráno zde:
      - http://blog.novoj.net/2008/09/20/testing-aspect-pointcuts-is-there-an-easy-way/
      - http://blog.novoj.net/2008/08/31/jopenspace-2008-audio-1/
  - obtíže
    - jsme odkázáni na množství a kvalitu logovacích informací dané advice

Prezentační vrstva - techniky

- Desktop aplikace založené na Swingu
  - http://jemmy.netbeans.org
  - http://www.uispec4j.org
  - http://abbot.sourceforge.net/doc/overview.shtml

- Server aplikace orientované na web
  - Využití mock objektů
    - mohou být součástí použitého frameworku – např. Spring poskytuje podporu pro
      - Servlet API
        - org.springframework.mock.web.* (MockFilterChain, MockFilterConfig, MockHttpServletRequest, MockHttpServletResponse, MockHttpSession, MockServletConfig, MockServletContext a další)
      - Portlet API
        - org.springframework.mock.web.portlet.*
      - Spring MVC
        - org.springframework.test.web.AbstractModelAndViewTests
        - org.springframework.test.web.ModelAndViewAssert
  - HtmlUnit – embeded testy
    - podpora JavaScriptu, ale brzy narazíte na limity
  - Selenium (neomezuje se pouze na Javu)
    - testy běží v reálném prohlížeči
    - je možné testovat funkcionalitu v různých prohlížečích
    - lze provozovat na různých počítačích (distribuovaně), ale všechny musí být schopni spustit prohlížeč
    - lze testovat i Ajax aplikace

Code smells & antipatterns #1

- kód je obtížně testovatelný
  - možné příčiny
    - testy se píší až ve chvíli, kdy je kód hotový
    - testovaná třída:
      - má natvrdo definované závislosti (objekt si vytváří instance)
      - obsahuje příliš velké množství logiky
      - logika pracující s thready
  - možná řešení
    - decoupling – rozbití logiky do menších částí (TDD, IOC)
    - použití fake, stub, mock objektů pro nahrazení části logiky, která není pro test zásadní
    - vyhnutí se asynchronicitě v kódu

Code smells & antipatterns #2

- křehké testy
  - možné příčiny
    - cut & paste programování
    - citlivost na změnu rozhraní
    - citlivost na změnu chování
      - příliš komplikované testy
      - více testů testuje (je závislých) na stejné logice
      - nahrávané testy (např. Selenium) mohou být obzvláště náchylné ke křehkosti
    - citlivost na změnu dat
      - způsob práce s daty používanými v testech
  - možná řešení
    - změna rozhraní
      - většinou se neřeší – jediné možnosti jsou zapouzdření práce s rozhraním v testech – např. vytvořením jazyka vyšší úrovně, který se používá v testech, použití custom creation / verification methods
    - změna chování
      - použití tzv. creation methods, custom assertions / verification methods
    - změna dat
      - zajištění stálosti dat v úložišti (fresh fixture, database sandbox, teardown rollback)
      - používání hlavy při psaní assertů – nedefinovat asserty na ověření něčeho, co se může s přidáním dat měnit

Code smells & antipatterns #3

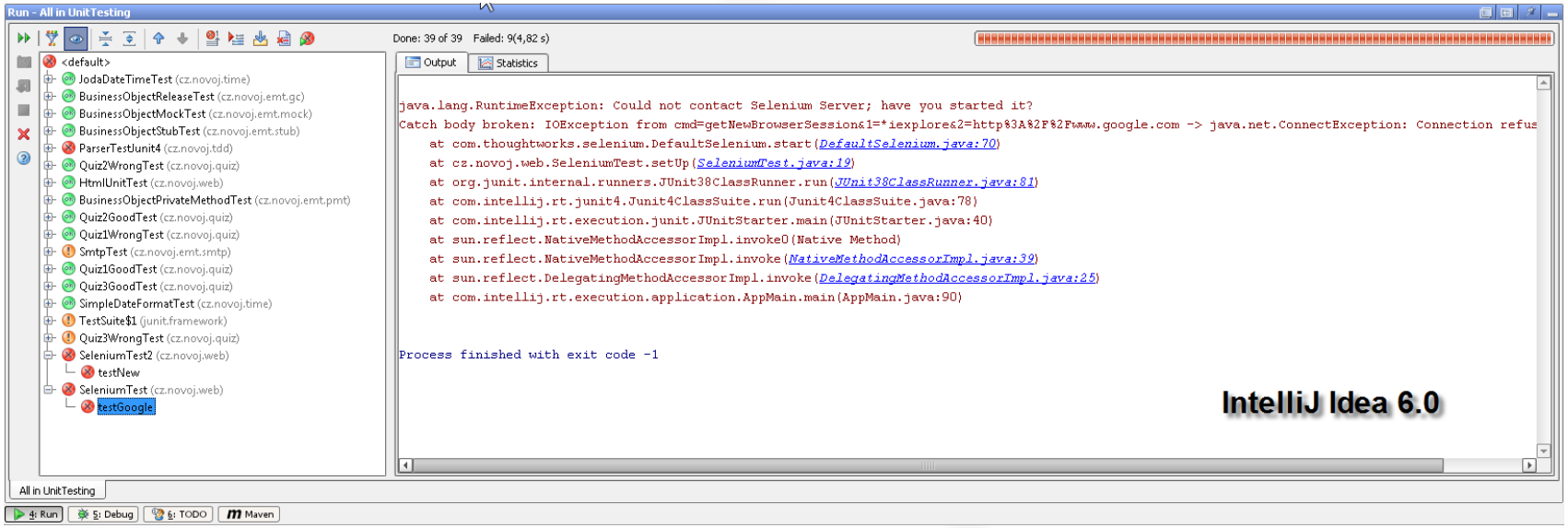
- problémy při analýze chyby testu
  - možné příčiny
    - malá znalost testovacího frameworku v týmu – používání nevhodných assert metod, ověřování logiky mimo test
    - přílišná komplikovanost testů (snažíme se testovat co největší množství logiky v testech)
    - příliš složité / nečitelné testy
    - sdílená setUp připravující globální data, ze které není vidět průnik do testované logiky
    - součástí testu jsou deklarace nesouvisející s testovací logikou, zatemňující podstatu věci
  - možná řešení
    - rozbití komplexních testů do více jednodušších testů
    - edukace v týmu
    - psaní error hlášení v assert metodách (v IDE si vystačíme se stacktrace, ale problémy mohou nastat např. při spouštění testů v command-line např. v CI apod., kde nemusí být cesta k stacktrace tak jednoduchá)

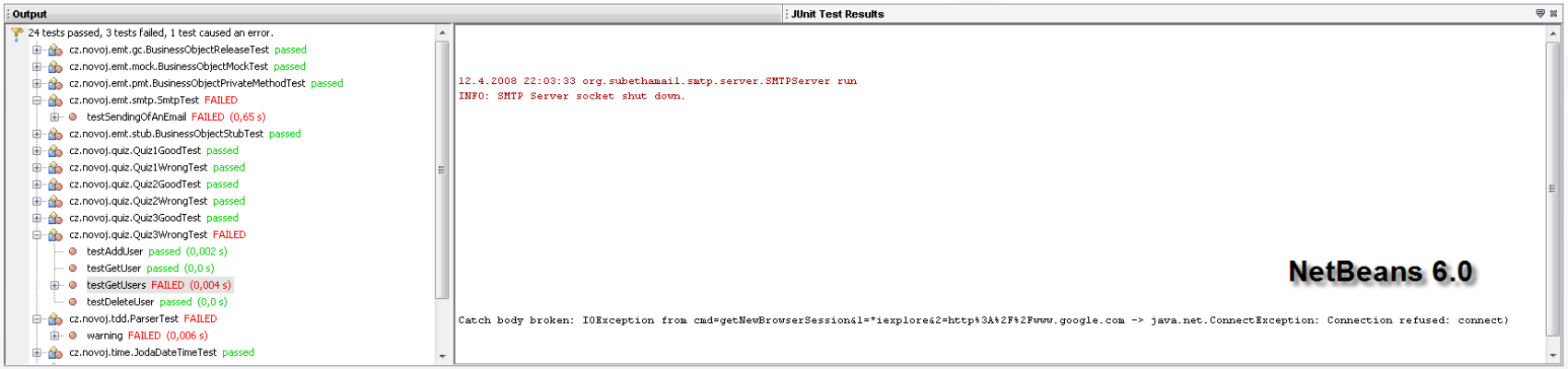
Code smells & antipatterns #4

- testovací kód v produkčním kódu & duplikace produkčního kódu v testech
  - možné příčiny
    - testování private metod
    - potřeba odlišit chování produkčního kódu při běhu v testech
    - špatný dependency management – testovací knihovna se dostává do standardního buildu
  - možná řešení
    - refaktorizace : použití strategy patternu (pro test upravíme strategii a nasetujeme do produkčního kódu)
    - použití AOP
    - použití Maven pro buildování, nebo zavedení vlastní strategie pro odlišení test a production dependencí
- nevypočitatelné testy
  - možné příčiny
    - výsledek testu je ovlivněn během ostatních testů
    - může znamenat resource leak
    - může být příliš optimistický při práci s externími zdroji (ve smyslu jejich existence, kódování, zámky)
    - obsahují neopakovatelnou logiku (závislou např. na systemdate)
    - kolize testů v případě, že je stejný test spuštěn paralelně (sdílená db)
  - možná řešení
    - není jednoduché rady – v závislosti na typu problému může být lékem použití některého z probíraných patternů
- pomalé testy
  - možné příčiny
    - závislost na pomalých komponentách systému
    - příliš mnoho testů
    - používání timeoutů např. pro testování asynchroních procesů
  - možná řešení
    - není jednoduché rady – v závislosti na typu problému může být lékem použití některého z probíraných patternů

Integrace do IDE

- V současnosti naprosto integrální součástí

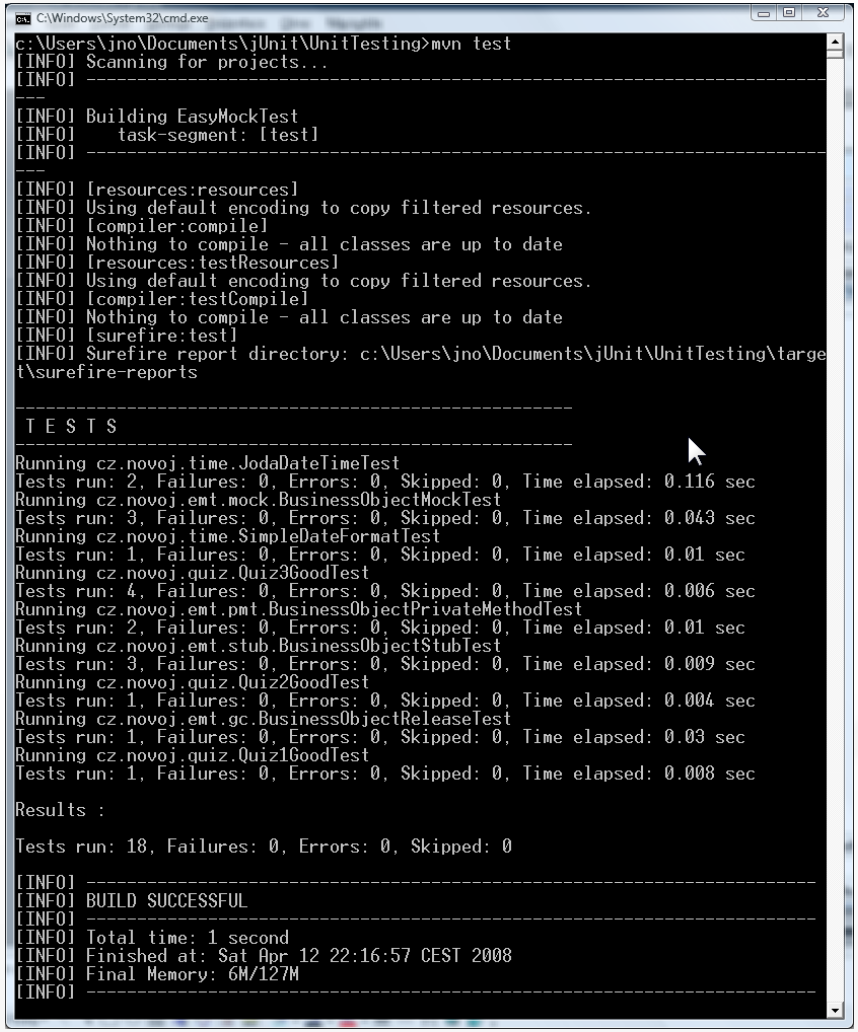




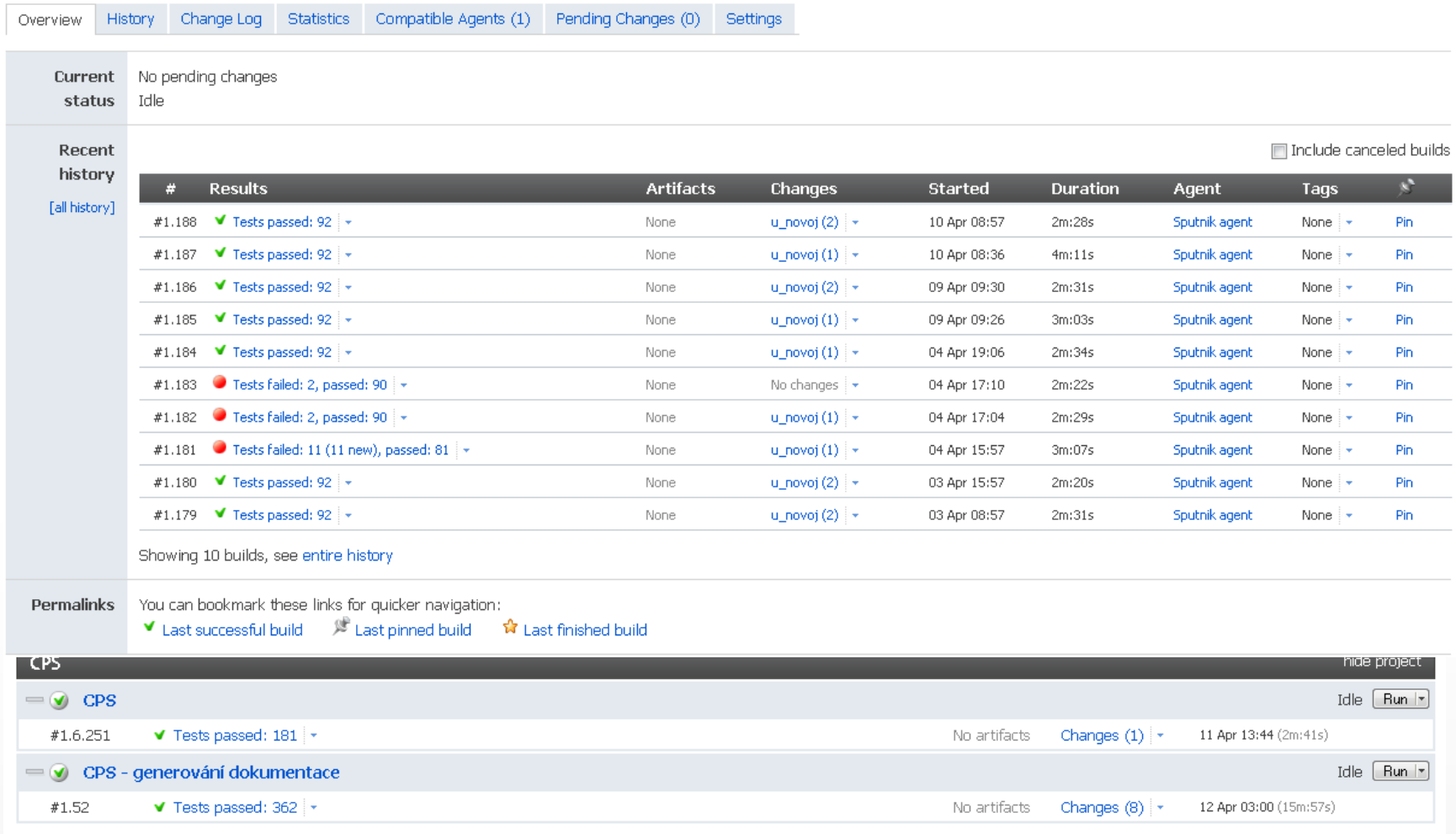
Integrace do build nástrojů

- Maven 2
  - součástí standardního buildu
  - „mvn test“
  - Maven-surefire-plugin
- Ant
  - JUnit Task

```
<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>
```



Integrační server



Overview | History | Change Log | Statistics | Compatible Agents (1) | Pending Changes (0) | Settings

Current status
No pending changes
Idle

Recent history
[all history]

Include canceled builds

#	Results	Artifacts	Changes	Started	Duration	Agent	Tags	
#1.188	Tests passed: 92	None	u_novoj (2)	10 Apr 08:57	2m:28s	Sputnik agent	None	Pin
#1.187	Tests passed: 92	None	u_novoj (1)	10 Apr 08:36	4m:11s	Sputnik agent	None	Pin
#1.186	Tests passed: 92	None	u_novoj (2)	09 Apr 09:30	2m:31s	Sputnik agent	None	Pin
#1.185	Tests passed: 92	None	u_novoj (1)	09 Apr 09:26	3m:03s	Sputnik agent	None	Pin
#1.184	Tests passed: 92	None	u_novoj (1)	04 Apr 19:06	2m:34s	Sputnik agent	None	Pin
#1.183	Tests failed: 2, passed: 90	None	No changes	04 Apr 17:10	2m:22s	Sputnik agent	None	Pin
#1.182	Tests failed: 2, passed: 90	None	u_novoj (1)	04 Apr 17:04	2m:29s	Sputnik agent	None	Pin
#1.181	Tests failed: 11 (11 new), passed: 81	None	u_novoj (1)	04 Apr 15:57	3m:07s	Sputnik agent	None	Pin
#1.180	Tests passed: 92	None	u_novoj (2)	03 Apr 15:57	2m:20s	Sputnik agent	None	Pin
#1.179	Tests passed: 92	None	u_novoj (2)	03 Apr 08:57	2m:31s	Sputnik agent	None	Pin

Showing 10 builds, see entire history

Permalinks
You can bookmark these links for quicker navigation:
Last successful build | Last pinned build | Last finished build

CPS — hide project

CPS — Idle — Run

#1.6.251	Tests passed: 181	No artifacts	Changes (1)	11 Apr 13:44 (2m:41s)

CPS - generování dokumentace — Idle — Run

#1.52	Tests passed: 362	No artifacts	Changes (8)	12 Apr 03:00 (15m:57s)

Reporting & statistiky

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	145	50%	57%	1.702
com.fg.form.business	1	88%	100%	2
com.fg.form.core.context	3	72%	100%	1.03
com.fg.form.core.data	3	60%	47%	1.318
com.fg.form.core.data.message	5	38%	N/A	1
com.fg.form.core.data.message.container	3	51%	65%	2.04
com.fg.form.core.data.message.localization	4	18%	33%	1.562
com.fg.form.core.data.state	5	69%	0%	1
com.fg.form.core.definition	6	96%	100%	2.308
com.fg.form.core.definition.form	3	92%	100%	1.056
com.fg.form.core.definition.page	4	86%	100%	1
com.fg.form.core.definition.widget	4	77%	100%	1.032
com.fg.form.core.definition.widget.impl	11	44%	0%	1.214
com.fg.form.core.environment	4	N/A	N/A	1
com.fg.form.core.environment.http	2	53%	62%	2.348
com.fg.form.core.environment.http.parameterMap	4	48%	67%	1.791
com.fg.form.core.environment.http.resolution	5	43%	N/A	1.056
com.fg.form.core.environment.http.scope	1	25%	0%	2
com.fg.form.core.environment.scope	4	24%	0%	1.5
com.fg.form.core.exception	2	0%	N/A	1
com.fg.form.core.lifecycle	3	43%	55%	4.048
com.fg.form.core.lifecycle.collect	5	62%	75%	2.556
com.fg.form.core.lifecycle.command	2	0%	N/A	1
com.fg.form.core.lifecycle.command.generic	1	25%	N/A	1.5
com.fg.form.core.lifecycle.convert	2	92%	85%	2.857
com.fg.form.core.lifecycle.convert.impl	6	75%	81%	2.172
com.fg.form.core.lifecycle.event	9	11%	N/A	1
com.fg.form.core.lifecycle.event.listener	2	72%	50%	2.5
com.fg.form.core.lifecycle.populate	2	94%	89%	2.8
com.fg.form.core.lifecycle.populate.impl	3	67%	100%	1.222
com.fg.form.core.lifecycle.validate	2	100%	100%	2.5
com.fg.form.core.lifecycle.validate.impl	12	16%	16%	1.5
com.fg.form.core.lifecycle.validate.provider	2	100%	N/A	1
com.fg.form.core.resolution	3	97%	100%	5
com.fg.form.cps	1	81%	100%	1.5
com.fg.form.cps.config	2	77%	89%	3.269
com.fg.form.web	1	33%	25%	2
com.fg.form.web.tag	3	0%	0%	2.25
com.fg.form.wizard.data	1	21%	0%	1.167
com.fg.form.wizard.definition.form	1	33%	21%	3.429
com.fg.form.wizard.definition.page	1	36%	N/A	1
com.fg.form.wizard.lifecycle	1	0%	0%	1.5
com.fg.form.wizard.lifecycle.analyze	4	19%	0%	1.895
com.fg.form.wizard.lifecycle.command	2	0%	0%	2

Report generated by Cobertura 1.7 on 4/12/08 3:14 AM.

- Run time statistics
- Test coverage report

Užitečné odkazy

- http://junit.org (jUnit homepage)
- http://www.devx.com/Java/Article/31983 (jUnit 4.x)
- http://xunitpatterns.com (patterny & antipatterny)
- http://www.mockobjects.com (techniky testování)
- http://www.easymock.org (mock testing tool)
- http://selenium.openqa.org (testování web UI)
- http://www.martinfowler.com/ (blog Martina Fowlera)
- http://blog.novoj.net (Myšlenky dne Otce Fura)
- http://www.fg.cz (Web společnosti FG Forrest - články)