

---

# Nástroje pro průběžnou integraci a testování

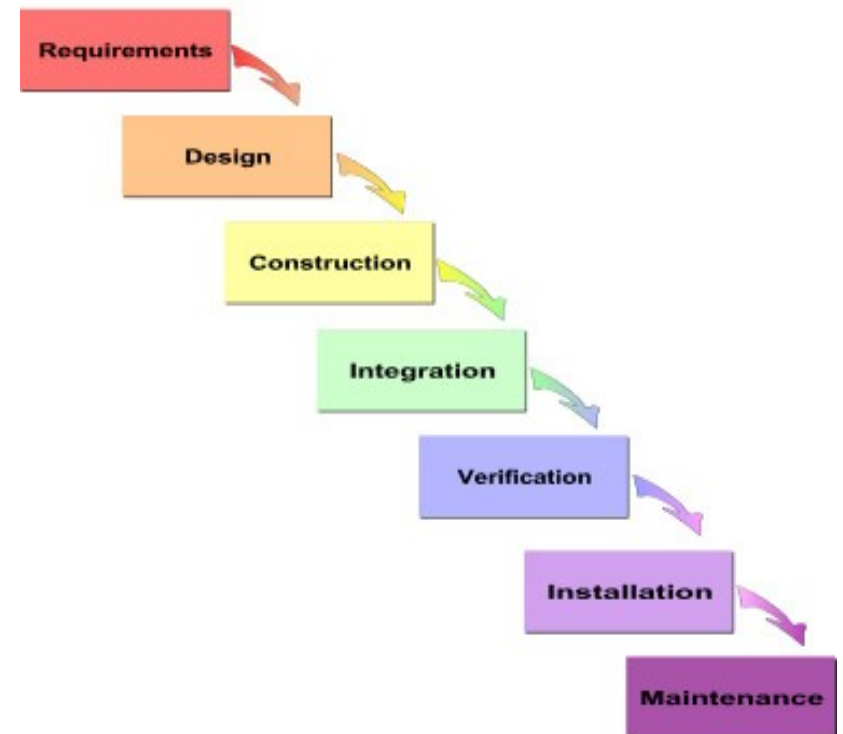
Osnova:

- Úvod do problematiky
  - Životní cyklus softwaru
  - Iterativní a inkrementální vývoj
- Průběžná integrace
  - Nástroje nutné k tomu, aby průběžná integrace fungovala
  - Aplikace pro průběžnou integraci

# Životní cyklus softwaru

---

- Životní cyklus SW popisuje život SW od jeho návrhu, přes implementaci, až po jeho předání a údržbu.
- Existuje řada modelů:
  - Vodopád, letadlo, výzkumník, spirála, ...
- V objektovém světě se masivně v praxi používají dva modely:
  - Vodopád
  - Iterativní a inkrementální vývoj



# Vodopád – analogie s rodinným domem

---

- Kompletní výstavba rodinného domu se zahradou a garáží „na klíč“
- Etapy vývoje:
  - Architekt kompletně navrhne dům, garáž, zahradu, ...
  - Projektant zhotoví všechny technické nákresy
  - Stavební firma vše postaví
  - Rodina se nastěhuje a bydlí
- Problém 1 (návrát do předchozí etapy)
  - Rodina po roce bydlení zjistí, že z krbu opadá omítka
  - Návrát na předchozí etapu => firma opraví krb
  - Relativně levná oprava
- Problém 2 (návrát o několik etap zpět)
  - Rodina po nastěhování zjistí, že krb je sice pěkný, ale nehřeje a kouří dovnitř místnosti. Chyba je v samotném typu a umístění krbu a kouřovodů v domě.
  - Architekt navrhne nové řešení, projektant vyprojektuje, ...
  - Hodně drahá oprava.



# Iterativní vývoj – příklad s domem

---

- Postupné přetváření celé stavební parcely až do finální podoby
- První iterace:
  - Architekt rozvrhne kompletní infrastrukturu (rozvody elektřiny do domu, do garáže i k bazénu, přívod vody do domu a k bazénu apod.). Podrobně rozpracuje samotný dům.
  - Stavební firma postaví samotný dům, ale zatím bez vnějších omítek a dalších „detailů“.
  - Rodina se nastěhuje a (provizorně) bydlí.
- Druhá iterace:
  - Architekt rozpracuje garáž a bazén.
  - Stavební firma dodělá vnější omítky a postaví garáž.
- Třetí iterace:
  - Stavební firma vybuduje bazén a upraví okolí.



=> rychlejší (dílčí) výsledky – rodina dříve bydlí, byť provizorně

=> rychlejší odhalení chyb – rodina zjistí mnohem dříve, že krb je špatný

# Inkrementální vývoj – příklad s domem

- Postupné přidávání nových věcí k již hotovým
- První přírůstek – rodinný dům:
  - V rámci prvního přírůstku vzniká nový rodinný dům např. metodou iterativního vývoje
  - Výstupem je kompletně hotový a zabydlený dům
- Druhý přírůstek – garáž:
  - V rámci druhého přírůstku vzniká garáž např. pomocí vodopádu (architekt navrhne garáž a její umístění na pozemku, projektant zhotoví technické nákresy garáže, stavební firma garáž postaví)
  - Pro garáž je nutné probourat jednu zeď rodinného domu a upravit elektrorozvodnou skříň, aby šlo přivést elektřinu do garáže => integrace
  - Výstupem je kompletní funkční celek „dům s garáží“
- Třetí přírůstek – bazén:
  - Majitel domu si sám navrhne a vybuduje na zahradě bazén
  - Je třeba přivést z domu vodu a elektřinu pro osvětlení bazénu => integrace



=> výstupem je vždy hotový funkční systém

# Agilní metody vývoje

---

- Jsou přímo založeny na velkém počtu malých přírůstků (i několik přírůstků denně)
- Např:
  - XP – Extreme Programming
  - SCRUM
  - FDD – Feature-Driven Development
  - ...

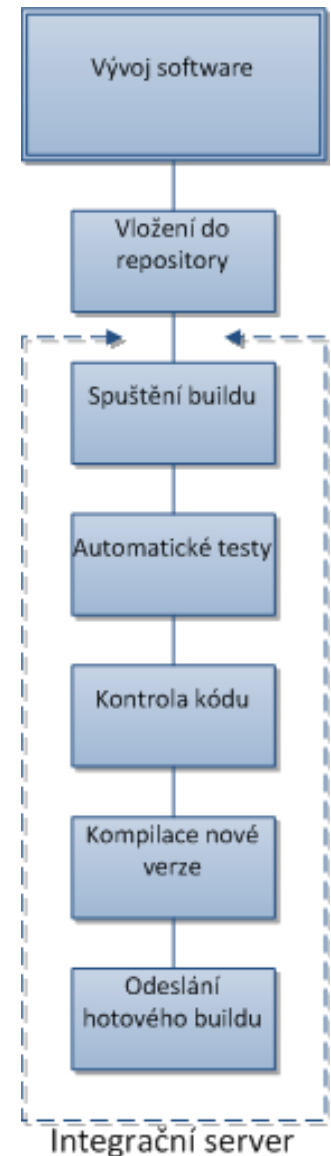
# Iterativní a inkrementální vývoj

---

- Průběžná implementace a testování => včasné varování o chybách
- Rychlé předání (neúplného systému) zákazníkovi
- Nutnost předělávat kód
  - Předpokládá se, že pozdější iterace nebo přírůstky budou upravovat nebo mazat existující kód
  - Ve vývoji SW to nemusí být až tak velká nevýhoda – je lepší špatný kód přepsat, než ho nějak obcházet

# Průběžná integrace

- Nástroje a postupy umožňující přidávat změny (přírůstky) do existujícího kódu bez toho, aby:
  - se zavedly do existujícího kódu nové chyby
  - integrace trvala neúměrně dlouho (déle než samotný vývoj přírůstku) a byla neúměrně pracná
- Jsou proto nutné nástroje pro:
  - centrální ukládání zdrojových kódů a správu verzí
  - automatický „build“
  - automatické testování
  - sledování a vyhodnocování kvality kódu





# Úložiště zdrojových kódů

---

- *Code Repository*
- Požaduje se:
  - Uchování nejen zdrojových kódů, ale všech artefaktů nutných k buildu (konfigurační soubory, XML data, apod.).
  - Řešení kolizí při současné úpravě stejného kódu více lidmi.
  - Správa verzí včetně návratu k předchozím verzím.
  - Statistiky (které části jsou nejčastěji měněny, kdo nejčastěji mění danou část systému apod.) -- viz Kontrola kvality kódu.
- Příklady nástrojů:
  - CVS – Concurrent Version System
  - SVN – Subversion
  - Perforce
  - ...

# Automatizace „buildu“

---

- *Build automation*
- Build = složitý proces zahrnující kompilaci kódu, přesuny souborů, vytváření instalačních balíčků, registraci knihoven, deployment, atd.
- Cílem je provést celý build jedním kliknutím, jedním příkazem
- Příklady nástrojů:
  - Make
  - Apache Ant
  - Apache Maven
  - MS Build
- Distributed Builds
  - Buildy jsou časově velmi náročné a přitom se dělají velice často.
  - Distribuce buildu na více strojů.
  - Nestačí vzít např. Ant skripty, rozdělit je a poslat na několik strojů. Je třeba zjistit závislosti mezi zdrojovými kódy a umět řídit jednotlivé kroky buildu.
  - Automaticky se to moc neumí, často je třeba ruční konfigurace.

# Automatické testování

---

- Úspěšný build neznamená, že je program funkční.
- *Automated Testing*
  - Testování jednotek (*Unit Testing*)
  - Integrovaní testování – (*Integration Testing*)
  - Testování systému – (*System Testing*)

# Automatické testování jednotek

---

- *Unit testing*
- Testování izolovaných jednotek (komponent, tříd, metod, ...)
- Test-Driven Development: Nejdříve napíšeš kód, který umí otestovat nějakou funkčnost, pak teprve danou funkčnost naprogramuješ a nakonec ji otestuješ.
- Psaní testů zabere čas.
- Unit testy umožňují „bezpečný“ refaktoring (předělání) kódu.
- Unit testy nikdy nezachytí všechny chyby, zejména ne chyby vznikající na vyšší úrovni (interakcí objektů, komponent atd.).
- Nástroje pro automatické testování :
  - CUnit (pro C#), JUnit (pro Javu) a mnoho dalších
- **Možný úkol: porovnat specifika různých testovacích nástrojů pro jeden vybraný programovací jazyk**

# Automatické integrační testování

---

- *Integration testing*
- Testy funkčnosti, výkonu a spolehlivosti větších programových celků (např. propojených komponent).
- Zaměřuje se na chyby způsobené interakcí a spoluprací s hardwarem
- Souvisí s testováním komponentových architektur
- Nástroje pro automatické testování:
  - TestNG (?)
  - Formální testování – viz PARADISE
  - viz Bára Bührenová a problematika testování komponent

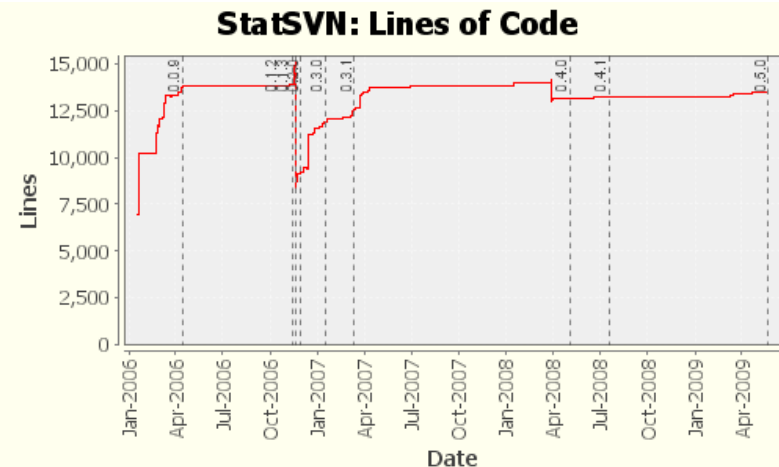
# Automatické testování celého systému

---

- *System testing*
- Jedná se o tzv. black-box testing
- Testuje se funkčnost celého systému
- Zahrnuje testování GUI (např. web accessibility), použitelnosti, výkonu, bezpečnosti a další.
- Nástroje pro automatické testování:
  - Pro testování webových aplikací: Selenium, Sahi
  - Ostatní oblasti ???
- **Možný úkol: udělat přehled existujících nástrojů pro automatické testování celého systému**

# Kontrola kvality kódu

- *Code Quality Analysis*
- Analýza kódu
  - Chyby (ne syntaktické, ty odhalí překladač).
  - Bad practice.
  - „Riskantní“ kód (např. průchod switchem bez zachycení).
  - Příklad aplikace: FindBugs pro Javu
- Analýza úložiště (repository) a jeho historie
  - Vyžaduje podporu statistik v aktuálním repository (CSV, SVN, ...) a také uchovávání statistik historie buildů
  - Příklady statistik:
    - Počet chyb v buildech a jejich trend.
    - Které části systému jsou nejčastěji měněny.
    - Průměrná velikost zdrojových souborů.
    - Čas integrace přírůstků, trend.
  - Příklad aplikace: StatSVN



(c)The nice people of StatSVN, 2009+

# Aplikace pro průběžnou integraci

---

- *Continuous Integration Applications*
- Systémy zahrnující výše uvedené principy a aplikace
- Např:
  - Hudson
  - TeamCity
  - CruiseControl
  - Bamboo



# TeamCity

---

- Pro Javu a .NET
- Centrální ukládání zdrojových kódů a správu verzí:
  - Subversion, CSV, StarTeam, Perforce, ...
- Automatický build:
  - Ant, Maven, MSBuild, ...
- Automatické testování:
  - JUnit, TestNG, NUnit, MSTest
  - Neobsahuje testování systému
- Sledování a vyhodnocování kvality kódu:
  - Vlastní statistiky ve vlastním vývojovém prostředí IntelliJ IDEA

# TeamCity – screenshots (I)

The screenshot shows the TeamCity web interface for configuring a build runner. The top navigation bar includes 'Projects', 'My Changes', 'Agents (48)', 'Build Queue (23)', 'Administration', and 'My Settings & Tools'. The current page is 'Administration > Agents maintenance Project > Clean & Defragment Configuration'. The 'Build Runner' section is active, showing a dropdown menu with options: Ant (selected), Command Line, Duplicates finder (.NET), Duplicates finder (Java), FxCop, Inspections, Ipr, Maven2, MSBuild, NAnt, Rake, sln2003, sln2005, and sln2008. Below the dropdown, there are fields for 'Path to a build.xml file' and 'Build file content'. The 'Build Configuration Steps' sidebar on the right lists seven steps: 1 General settings, 2 Version control settings, 3 Runner: Ant, 4 Build triggering, 5 Dependencies, 6 Properties and environment variables, and 7 Agent requirements.

Broad range of supported IDEs, build tools, testing frameworks and version control systems

# TeamCity – screenshots (II)

Projects - My Changes Agents (48) Build Queue (14) Administration My Settings & Tools

Welcome, max.feldman Logout

Calcutta > Inspections > #956 (26 Nov 08 01:30) Run Build Actions Edit Configuration Settings

Overview Changes (36) Build Log Build Parameters Dependencies Code Inspection << #956 All history Last recorded build

Total: 8948 (+18 -22) Errors: 336 (+0 -1) new only

Scope

- <All>
- agent-openapi/src/jetbrains/buildServer/agent/runner
- maven-runner
- maven-runner/src/jetbrains/maven
- server-openapi/src/jetbrains/buildServer/serverSide/statistics/build
- server/src/jetbrains/buildServer/serverSide/impl/personal
- server/src/jetbrains/buildServer/serverSide/stat
- TeamCitySamplePlugin/buildServerResources

Selected: Declaration has javadoc problems (3)

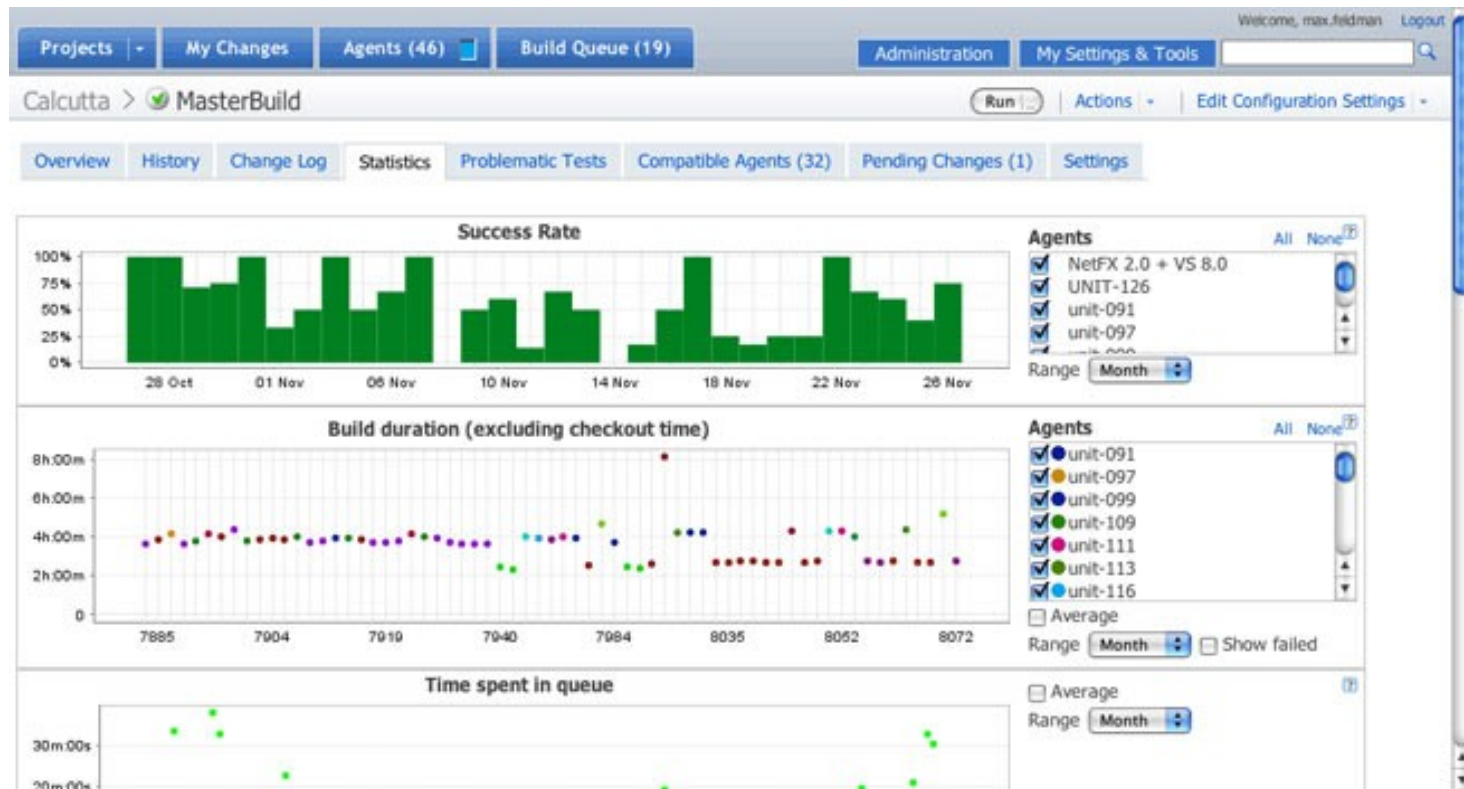
- agent-openapi/src/jetbrains/buildServer/agent/runner
  - CommandLineBuildService.java (3)
    - 32: initialize() Required tag @throws jetbrains.buildServer.RunBuildException is missing
    - 32: initialize() Required tag @param is missing for parameter build
    - 41: afterInitialized() throws tag description is missing

General Declaration has javadoc problems (3)

Declaration Redundancy Actual method parameter is the same constant (1)

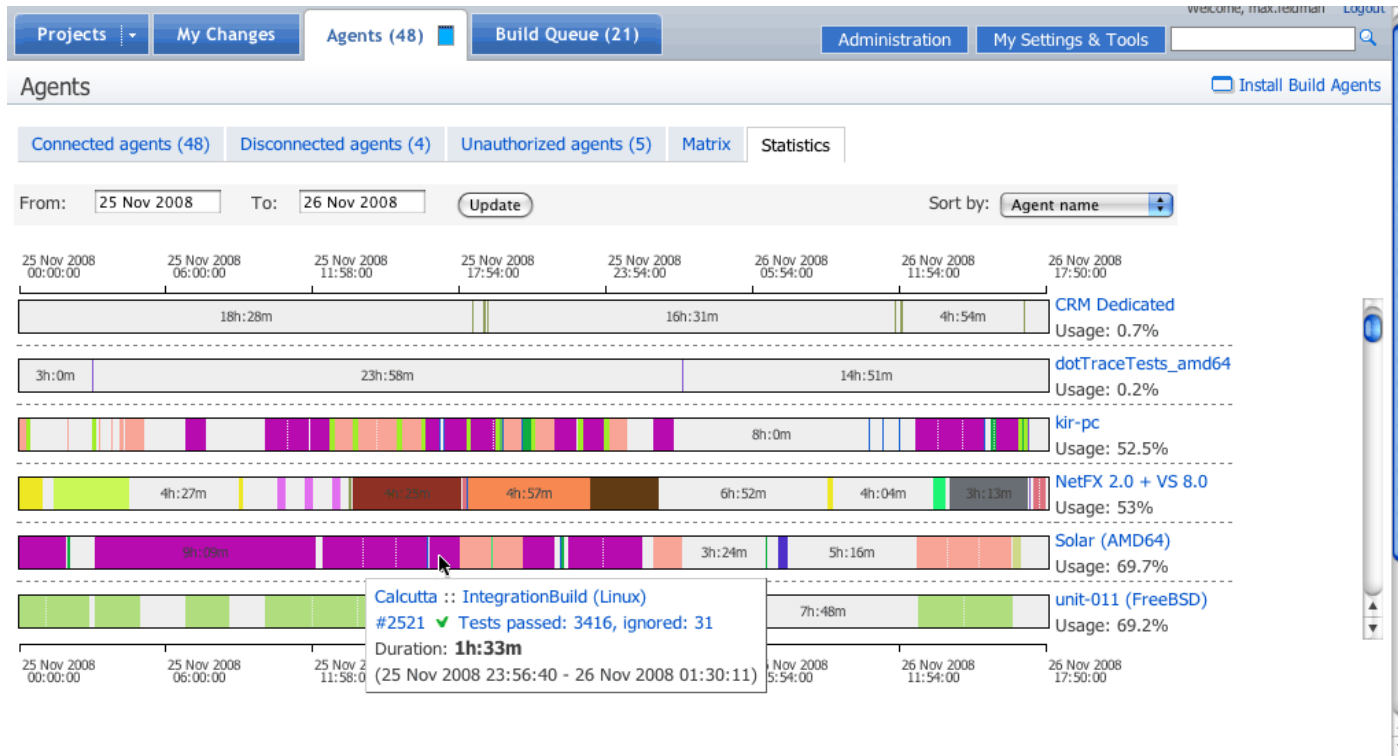
Over 600 Java code inspections, code coverage and duplicates search - out of the box

# TeamCity – screenshots (III)



Detailed statistics reports

# TeamCity – screenshots (IV)



Agents statistics and load matrix help better optimize hardware resources usage in the large-scale infrastructure

# TeamCity – screenshots (V)

Projects - My Changes Agents (47) Build Queue (15) Administration My Settings & Tools

Calcutta > Integration-IPR Run Actions Edit Configuration Settings

Overview History Change Log Statistics Problematic Tests Compatible Agents (32) Pending Changes (1) Settings

**Current status**  
1 change - pending for the next build  
Running 4 builds

Build ID	Test Results	Artifacts	Changes	Time	Actions
#7242	Tests failed: 2 (2 new), passed: 139, ignored: 133	No artifacts	Changes (1)	27:51m left	Stop
#7241	Tests passed: 1064, ignored: 19	No artifacts	Changes (2)	30m: 79% left	Stop
#7239	Tests failed: 2 (2 new), passed: 3550, ignored: 53	No artifacts	Changes (1)	overtime: 27m:52s	Stop
#7236	Tests failed: 2 (2 new), passed: 4139, ignored: 41	No artifacts	Changes (1)	overtime: 1h:30m	Stop

**Responsible**  
[Take responsibility](#) for current problems in this build

**Recent history**  
[all history]

#	Results	Artifacts	Changes	Time	Agent	Build	Actions
#7240	Tests failed: 1 (1 new), passed: 4941, ignored: 43						
#7238	Tests passed: 4942, ignored: 43	None	pavel.sher (2)	26 Nov 08 13:303h:02m	unit-148	None	Pin
#7237	Tests failed: 5 (5 new), passed: 4816, ignored: 165	None	yegor.yarko (1)	26 Nov 08 12:583h:04m	unit-124	None	Pin

**Build shortcuts**  
2 tests failed (2 new)  
NotificationRulesManagerTest.testRulesOrder (Server Suite: jetbrains.buildServer.notification)  
NotificationRulesManagerTest.testDefaultRulesOrder (Server Suite: jetbrains.buildServer.notification)

6.0

Detailed overview of all currently running project's builds with unit tests results

---

Děkuji za pozornost