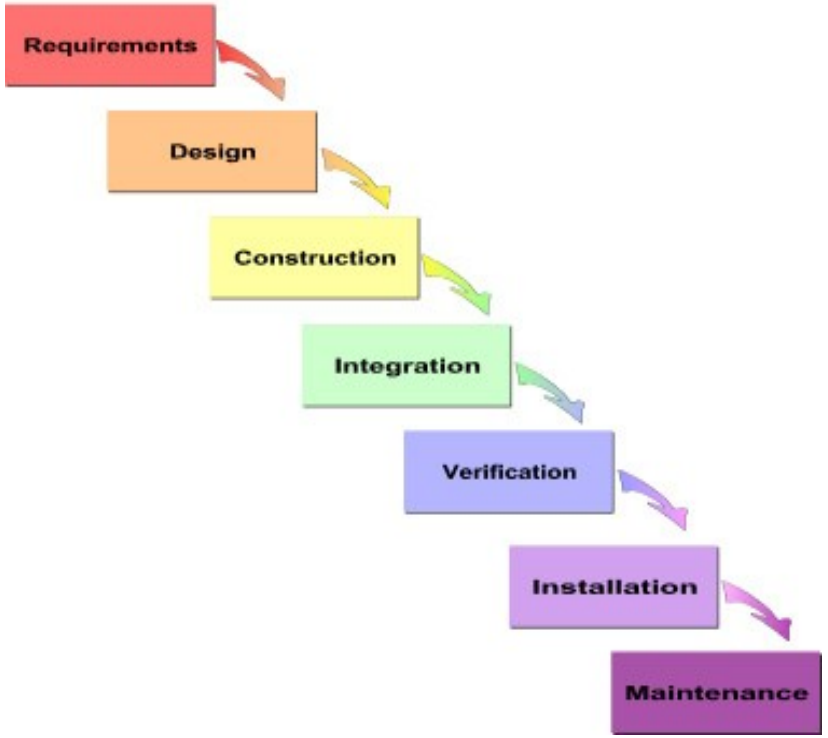
Nástroje pro průběžnou integraci a testování

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 (návrat do předchozí etapy)
  - Rodina po roce bydlení zjistí, že z krbu opadává omítka
  - Návrat na předchozí etapu => firma opraví krb
  - Relativně levná oprava
- Problém 2 (návrat 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
  - ...

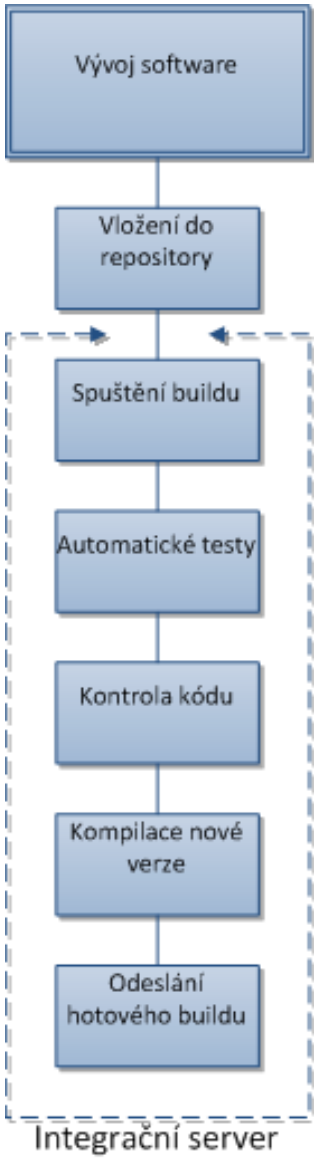
Iterativní a inkrementální vývoj

- 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:
  - CSV – Concurent 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)*
  - Integrační testování – *(Integration Testing)*
  - Testování systému – *(System Testing)*

Automatické testování jednotek

- *Unit testing*
- Testování izolovaných jednotek (komponent, tříd, metod, ...)
- Test-Driven Development: Nejdříve napíši kód, který umí otestovat nějakou funkčnost, pak teprve danou funkčnost naprogramuji a nakonec ji otestuji.
- 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

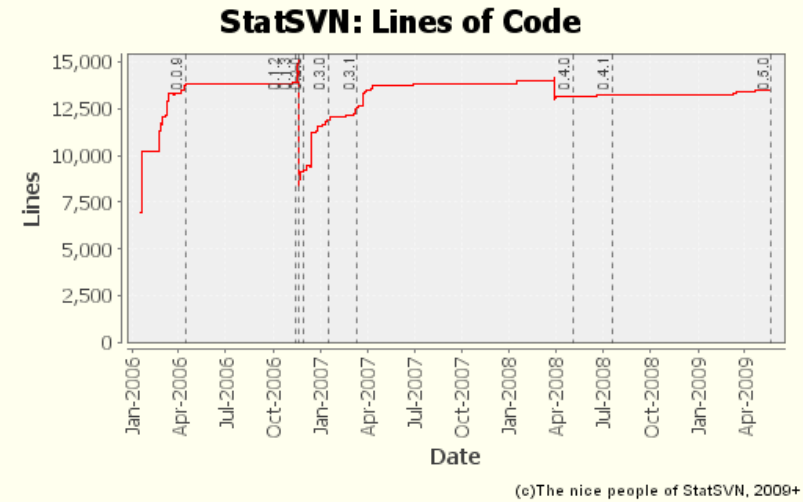
- *Integration testing*
- Testy funkčnosti, výkonu a spolehlivosti větších programových celků (např. propojených komponent).
- Zaměřuje se na chyby způsobené interakcí a spoluprací s hardwarem
- Souvisí s testováním komponentových architektur
- Nástroje pro automatické testování:
  - TestNG (?)
  - Formální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



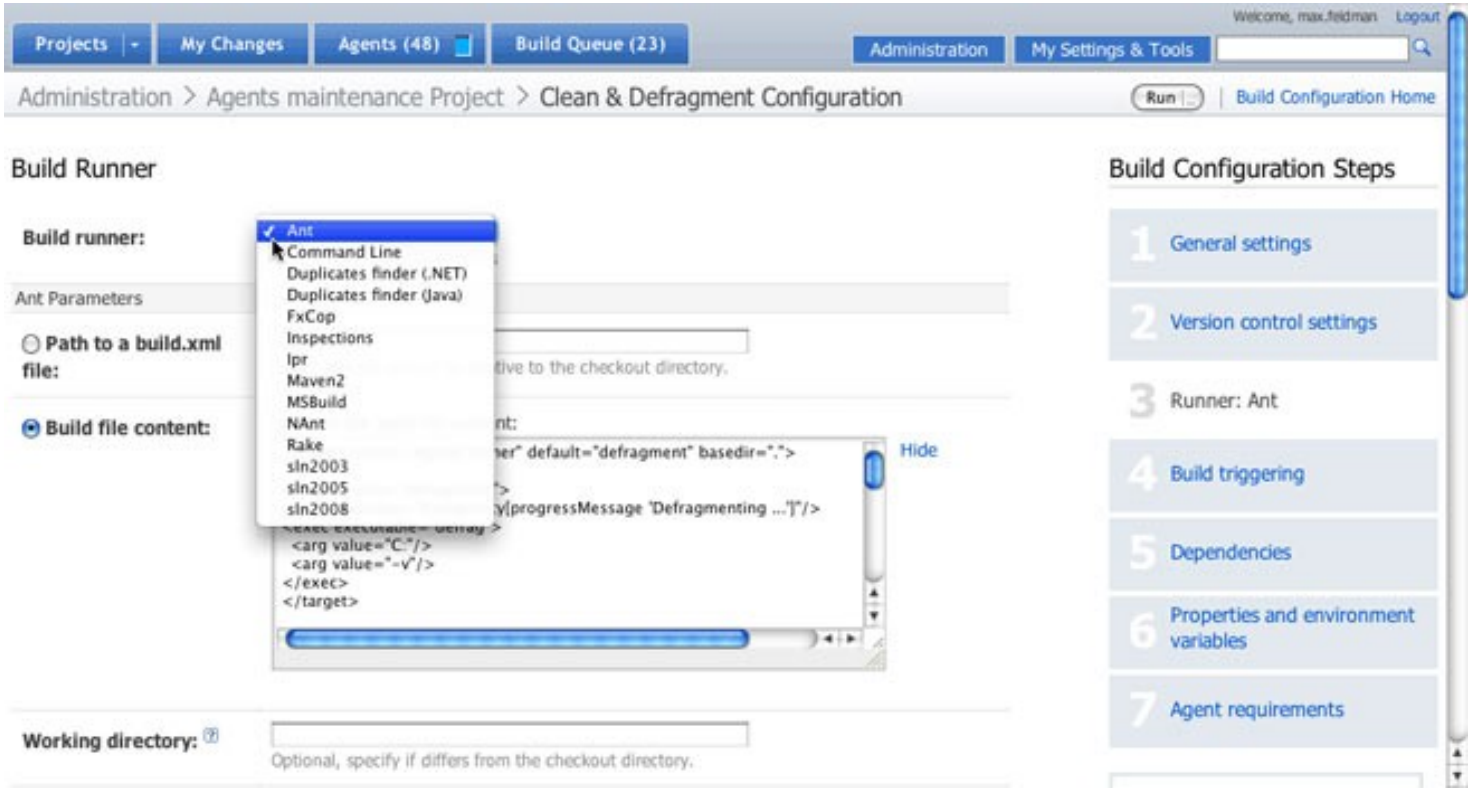
Aplikace pro průběžnou integraci

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

TeamCity

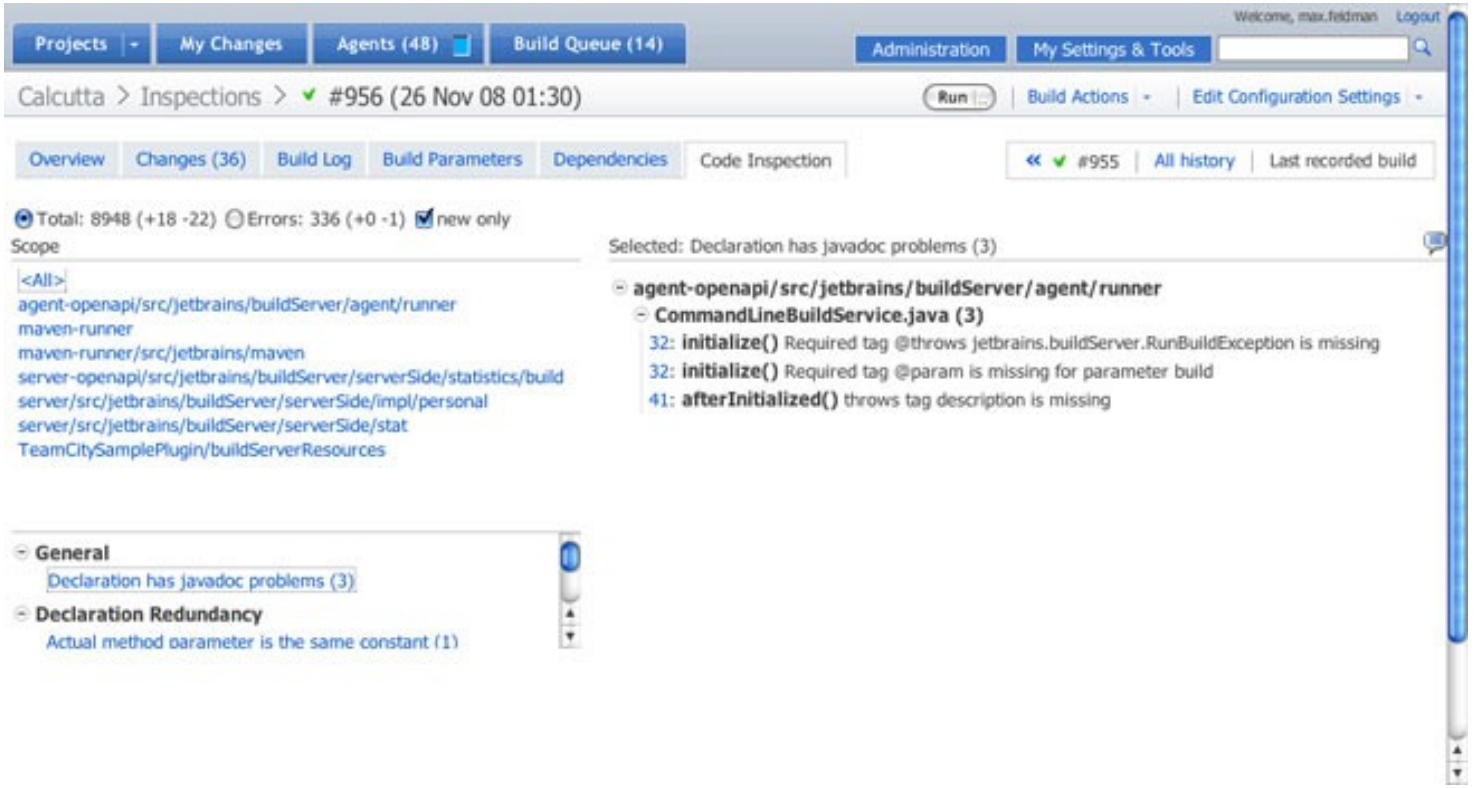
- Pro Javu a .NET
- Centrální ukládání zdrojových kódů a správu verzí:
  - Subversion, CSV, StarTeam, Perforce, ...
- Automatický build:
  - Ant, Maven, MSBuild, ...
- Automatické testování:
  - JUnit, TestNG, NUnit, MSTest
  - Neobsahuje testování systému
- Sledování a vyhodnocování kvality kódu:
 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

TeamCity – screenshots (I)



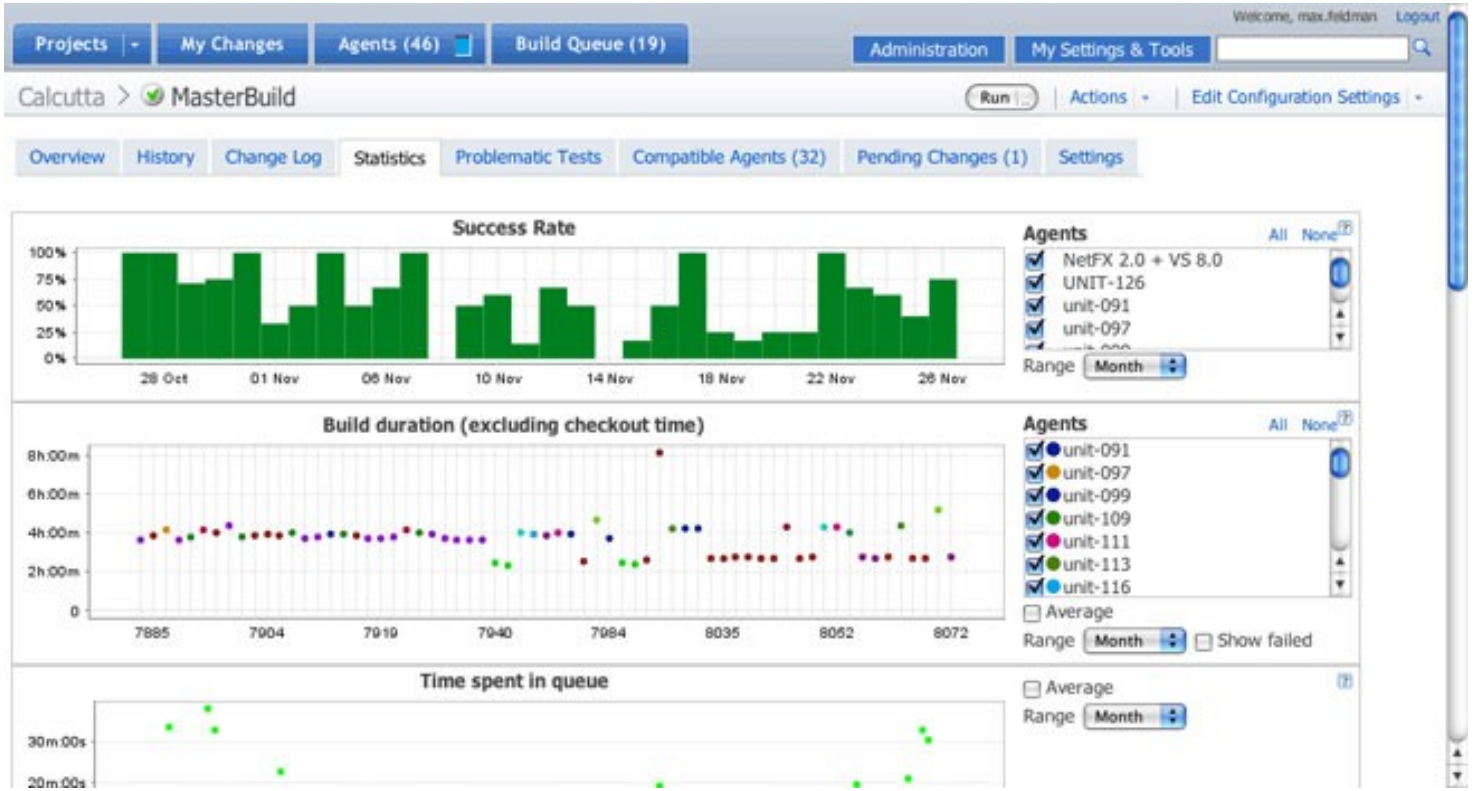
Broad range of supported IDEs, build tools, testing frameworks and version control systems

TeamCity – screenshots (II)



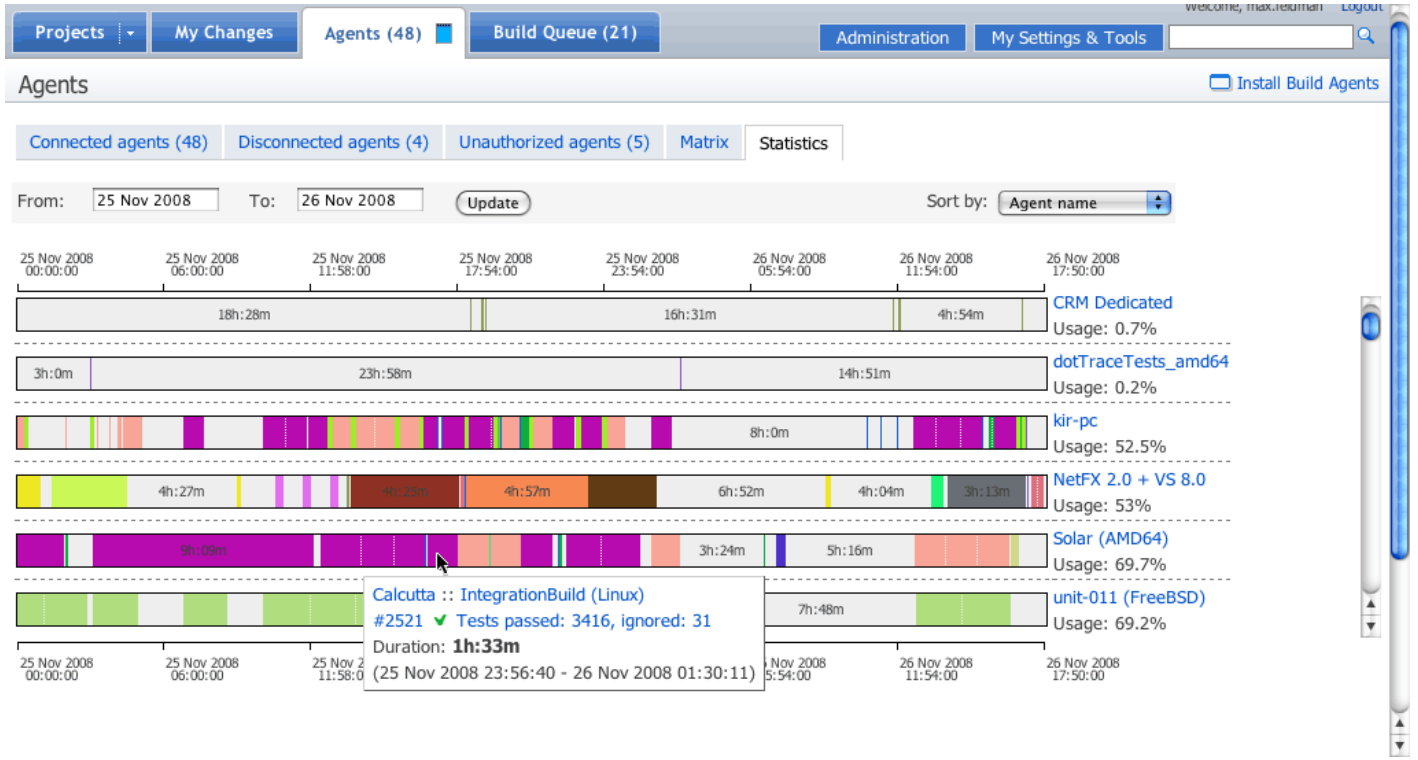
Over 600 Java code inspections, code coverage and duplicates search - out of the box

TeamCity – screenshots (III)



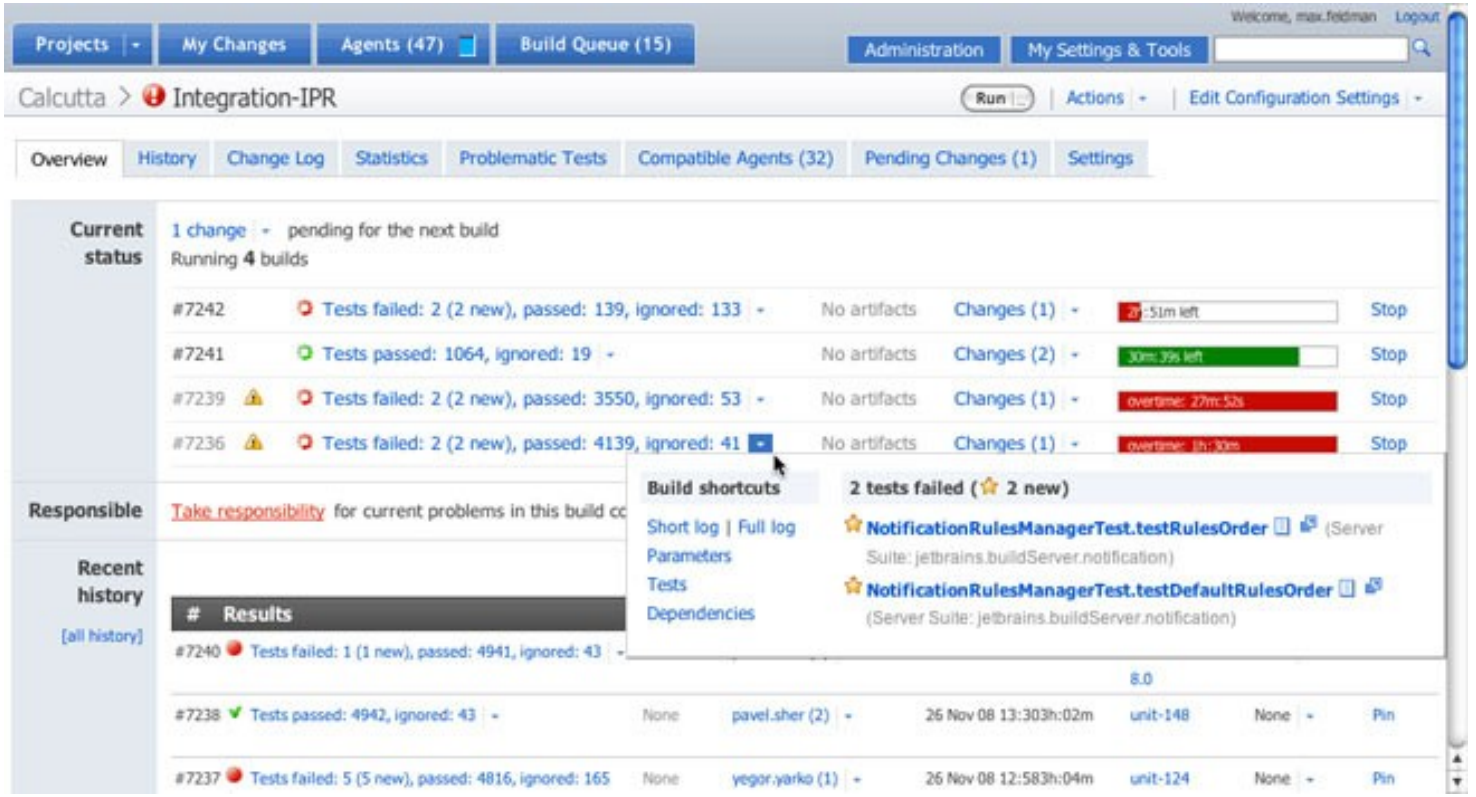
Detailed statistics reports

TeamCity – screenshots (IV)



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

TeamCity – screenshots (V)



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

Děkuji za pozornost