OSTRAVSKÁ UNIVERZITA V OSTRAVĚ
FAKULTA PŘÍRODOVĚDECKÁ
KATEDRA INFORMATIKY A POČÍTAČŮ

Využití IBM Rational Team Concert v softwarovém projektu

DIPLOMOVÁ PRÁCE

Autor práce: Bc. Martin Cieslar
Vedoucí práce: RNDr. Jaroslav Procházka, Ph.D.

2012

UNIVERSITY OF OSTRAVA
FACULTY OF SCIENCE
DEPARTMENT OF INFORMATICS AND COMPUTERS

Use of IBM Rational Team Concert in software project

THESIS

Author: Bc. Martin Cieslar
Supervisor: RNDr. Jaroslav Procházka, Ph.D.

2012

ČESTNÉ PROHLÁŠENÍ

Já, níže podepsaný/á student/ka tímto čestně prohlašuji, že text mnou odevzdané závěrečné práce v písemné podobě, nebo na CD nosiči, je totožný s textem závěrečné práce vloženým v databázi DIPL2.

V Ostravě dne

……
podpis studenta/ky

(Zadání vysokoškolské kvalifikační práce)

ABSTRAKT

Hlavním cílem práce je ukázat a změřit, jak integrovaný nástroj pomůže zefektivnit vývoj softwaru, potažmo vyhodnotit přínosy a negativa z toho vyplývající. Práce je rozdělená do dvou částí. V první, teoretické části, se zabývám agilními metodami vývoje softwaru a nástroji, které je podporují. V druhé, praktické části, popisuji, jakým způsobem byl zautomatizován vývoj *Systému pro koordinaci humanitární pomoci*. Uvádím také případy užití a scénáře, které jsme v rámci projektu definovali a následně implementovali za pomocí IBM Rational Team Concert. Zamýšlím se nad dopady použití integrovaného nástroje při vývoji softwaru a v neposlední řadě analyzuji, jaké to mělo přínosy či negativa na efektivitu práce, kvalitu vývoje a jiné relevantní aspekty.

Klíčová slova: Rational Team Concert, integrované nástroje, agilní vývoj

ABSTRACT

The main aim of this thesis is to show and to measure, how the integrated tool can make software development more efficient. This thesis is divided into two parts. The first part deals with software development agile processes and supporting tools. In the second practical part is described how development of the system humanitarian help coordination was automated. I describe use cases and scenarios which we defined and implemented via IBM Rational Team Concert during the project. I evaluate the usage of Rational Team Concert, and last but not least, I analyse impacts on work efficiency, the quality of the development process and some other related aspects.

Keywords: Rational Team Concert, Integrated Tools, Agile Development

Poděkování

Na tomto místě bych rád poděkoval RNDr. Jaroslavu Procházkovi, Ph.D. za jeho odborné vedení, za jeho připomínky, trpělivost, nápady, náměty a energii, čímž mi dodával víru a sílu k dokončení této diplomové práce. Děkuji také svým nejbližším za jejich podporu a nesčetné ústupky, kterými se mi snažili vytvořit co nejlepší pracovní prostředí k dokončení práce.

Prohlašuji, že předložená práce je mým původním autorským dílem, které jsem vypracoval/a samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal/a, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.

V Ostravě dne . . . . . .

. . . . . .

(podpis)

OBSAH

1 ÚVOD

V rámci ročníkového projektu byl vyvinut informační systém, jehož úkolem je pomáhat v době přírodních katastrof s monitorováním a přerozdělováním humanitární pomoci. Využil jsem tedy této možnosti k tomu, abych zjistil a demonstroval, jakým způsobem lze automatizovat a integrovat agilní principy při vývoji informačního systému.

1.1 Metodologie práce

V úvodu práce popisuji vlastními slovy, na základě informací získaných z odborných pramenů, principy iterativně inkrementálních metodik. V základních rysech rozebírám moderní nástroje, které jsou používány při vývoji a údržbě softwaru. V praktické části prezentuji, jakým způsobem lze automatizovat vývoj informačního systému s použitím jednoho integrovaného nástroje.

V praktické části je dále popsáno, jakým způsobem byl customizován daný integrační nástroj včetně důvodů, proč tak bylo učiněno. Zmiňuji také důsledky, které to mělo na vývoj *Systému pro koordinaci humanitární pomoci*. Po kapitole týkající se nastavení Rational Team Concert následuje kapitola, která přibližuje samotný vývoj, tedy jaké případy užití a rizika byly definovány a jakým způsobem se automatizace vývoje promítla do jednotlivých fází projektu. Jedna podkapitola se také detailně zabývá ukázkovou iterací tak, aby bylo zřejmé, jak doopravdy Rational Team Concert zasahuje do běžných procesů.

1.2 Cíle

- Představení dvou nástrojů IBM Rational Team Concert a Microsoft Visual Studio Team Foundation Server podporujících agilní vývoj softwaru.
- Vytvoření vývojového procesu a demonstrace použití Rational Team Concert na softwarovém projektu.
- Instalace a customizace IBM Rational Team Concert.

  Nastavení serveru, vytvoření uživatelů, naplánování fází a iterací.
- Měření efektivity vývoje software.

  Rozbor Burn-up grafu, pracovních položek či změření rychlosti vývoje softwaru s použitím integrovaného nástroje nebo bez něj.
- Dopady IBM Rational Team Concert na vývoj informačního systému.

  Shrnutí toho, jak integrovaný nástroj pomáhá zefektivnit vývoj softwaru a vyhodnocení přínosů a negativ z toho vyplývajících.

2 METODY VÝVOJE SOFTWARE

Tato teoretická kapitola vychází ze skript RNDr. Jaroslava Procházky, Ph.D., Mgr. Marka Vajgla, Ph.D. a Doc. Ing. Cyrila Klimeše, Csc.[PK08],[PKV10], dále vychází z knihy *Metodiky vývoje a údržby informačních systémů* Ing. Aleny Buchalcevové, Ph.D. [BU04]. V této kapitole se seznámíme s metodikami pro vývoj softwaru a s vybranými podpůrnými nástroji. Bylo důležité nastudovat metodiky a seznámit se s různými nástroji, abychom pak mohli maximálně využít IBM Rational Team Concert.

Než přistoupím k samotným principům iterativně inkrementální metodiky, alespoň ve stručnosti bych se zmínil o vodopádovém přístupu, který těmto metodikám předcházel. Vodopádový přístup je totiž i dnes do značné míry používán. Obecně se uvádí, že 40 % veškerého vývoje probíhá vodopádovým přístupem. Důvodů, proč tomu tak je, existuje několik. Iterativně inkrementální metodiky jsou stále poměrně nové a jejich aplikace může představovat, vzhledem k minimální reálné zkušenosti vývojových týmů s jejich použitím, četná rizika. Dalším hlediskem je tlak na cenu projektu. Například při stanovení fixní ceny za dodání produktu většinou není dostatek vůle prosadit inovátorské způsoby, jejichž nasazení může vést k tomu, že kvůli neznalosti a praktické nezkušenosti s agilními metodikami, se projekt může dostat do problémů typu – zpoždění dodávky či nedodržení rozpočtu. Z těchto a jistě i několika dalších důvodů bude vodopádový přístup do značné míry uplatňován i v budoucnu. Dle mého názoru minimálně v oblasti údržby a také v oblasti menšího vývoje.

2.1 Principy iterativně inkrementální metodiky

Všechny metodiky, které uvádím v této kapitole, mají několik společných rysů. Obecně můžeme říci, že iterativně inkrementální metodiky jsou založené na postupném zodpovědném vývoji, v němž se všechny kroky a funkcionality průběžně vyhodnocují a testují. Všechny agilní metodiky počítají s aktivní účastí zákazníka, kterému je průběžně doručován produkt k testování a připomínkám, které jsou vítané. Základním rysem je i spolupráce na různých úrovních a celková „otevřenost“ k novým myšlenkám. Značné množství metodik má definované milníky, které pomáhají odpovědět na otázku, jestli jde

vývoj správnou cestou. Agilní metodiky se snaží poučit se z historických chyb jiných metodik, jako například vodopádového přístupu.

2.1.1 Scrum

Metodika *Scrum* byla vyvinuta Kenem Schwaberem a Jeffem Sutherlandem. Používá se především pro řízení projektu. Vývoj softwaru probíhá v rámci 30denních (nebo jinak dlouhých) iteracích, kterým se říká *Sprint* neboli iterace. *Sprint* se skládá z **plánování, vývoje, review meetingu a retrospektivy.** Většinou každý den probíhají porady (*Scrum meetings)*, které bývají dlouhé čtvrt hodiny a jejich cílem je koordinovat a integrovat práci mezi jednotlivými členy.[BU04]

Tato metodika má tři pilíře – **transparentnost** výsledků, **kontrolu** různých aspektů, které by mohly způsobit potíže a **adaptaci**. Jsou definovány tři role.

- Scrum Master

  Osoba odpovědná za dodržování postupů a pravidel. Komunikuje s klientem i managementem. *Scrum Master* je jakýmsi koučem týmu. Snahou je vytvořit kvalitnější produkt v kratším čase.

- Vlastník produktu

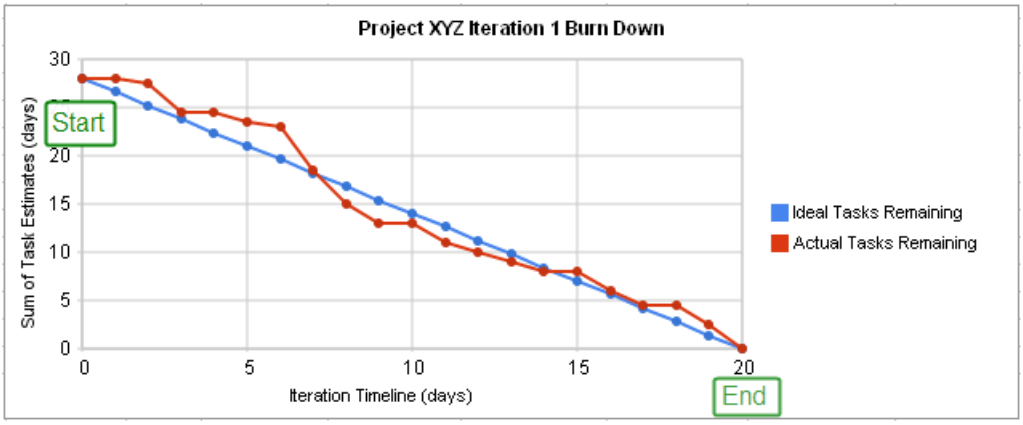
  Odpovídá za správu produktového *backlogu* (tzn. požadavky na produkt, které se vyvíjejí a jsou seřazeny podle priority)

- Tým

  Týmy vývojářů zpracovávají produktový *backlog*. Tým se skládá z členů, kteří disponují určitými dovednostmi jako programování, analýza, architektura apod.

Velice užitečnou pomůckou je *Sprint backlog burndown graf*. Lze z něj vyčíst, jaké množství práce zbývá vykonat v průběhu času vzhledem k dokončení dané iterace.[1] Toto tvrzené je ilustrováno na obrázku *Burn Down Graph,* který byl publikován na Wikipedii.[2] Modrá přímka představuje ideální průběh iterace. Červenou čáru lze považovat za velmi dobrý výsledek iterace.

[1] Schwaber K. a Sutherland J., Scrum Guide
[2] Staženo z http://upload.wikimedia.org/wikipedia/commons/8/8c/Burn_down_chart.png [online] 2012-02-26.

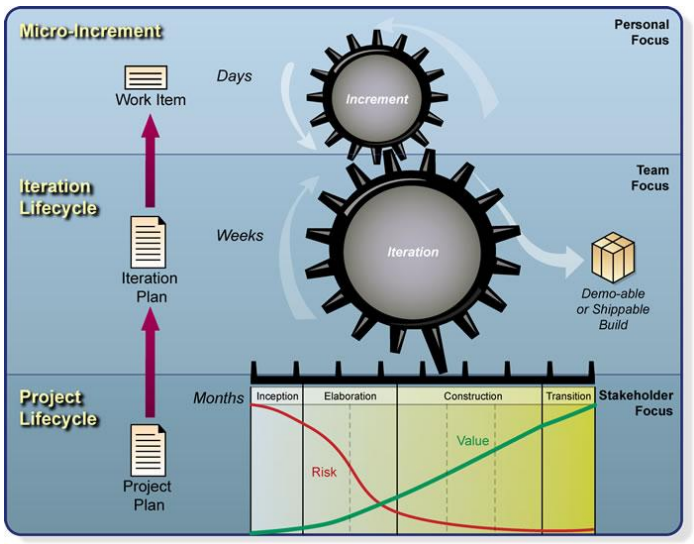


Obr. 1 Burn Down Graph (zdroj: Wikipedie)

Tento graf tedy říká, že s přibývajícím časem by se měl lineárně snižovat počet úkolů k vyřešení.

2.1.2 Open Unified Process

Open Unified Process, častěji označován jako OpenUP, je součást procesního frameworku vyvinutého Eclipse foundation. Obrázek, který jsem pojmenoval *OpenUP vrstvy* nám pomáhá porozumět principu této metodiky. Tzv. *Micro-increment* představuje malý, měřitelný krok, který má dopomoci k dosažení cíle dané iterace. Časové ohraničení pro jeden *Micro-increment* se volí v rozpětí hodin, maximálně dnů. Na jednom *Micro-incrementu* pracuje většinou jeden člověk nebo dva lidé.



Obr. 2 OpenUP vrstvy[3] (zdroj: Eclipse foundation)

Iteration Lifecycle, v překladu „životní cyklus iterace“, vychází z iteračního plánu. Iterační plán je vytvářen na základě prioritně identifikovaných pracovních položek (angl. Work Items). Jednotlivé iterace napomáhají k tomu, aby tým pravidelně zvyšoval přidanou hodnotu produktu. Průběžně jej testuje a pravidelně představuje zákazníkovi. Iterace začíná meetingem týmů, který může být i několik hodin dlouhý. Ze začátku se tým zabývá plánováním a architekturou, přičemž klade důraz na pochopení závislostí a logického pořadí pracovních položek. Většinu času životního cyklu iterace zabere provádění *mikro inkrementů.* V každé iteraci se tým snaží zvyšovat kvalitu produktu. Iterace končí zhodnocením, které proběhne společně se zákazníkem a retrospektivně se snaží poučit tým tak, aby další iterace byla lepší.

Životní cyklus projektu má čtyři fáze – *Inception, Elaboration, Construction, Transition.* Tyto fáze jsou stejné jako u RUP procesu a popisuji je v následující kapitole.

Pro úplnost doplňuji, že tato kapitola byla napsána na základě informací, které jsem získal z oficiálních internetových stránek Eclipse foundation.

OpenUP metodiku budeme využívat při vývoji *Systému pro koordinaci humanitární pomoci*.

2.1.3 Rational Unified Process

Rational Unified Process je iterativní přístup vývoje softwaru, který je řízen případy užití. RUP mohou používat malé i velké týmy. RUP se snaží identifikovat všechna rizika a aktivně je řešit. Cílem RUP je dodat zákazníkovi kvalitní, zdokumentovaný software.[KK03]

Základní principy RUP:

- Identifikovat a odstraňovat hlavní rizika co nejdříve a průběžně

  Rizika jsou značená do seznamu rizik včetně jejich priorit.

- Zaměření na spustitelný (funkční) software

[3] Obrázek stažen z http://epf.eclipse.org/wikis/openup/ [online] 2012-01-17.

Pokud se zaměříme na spustitelný software, eliminujeme tím rizika v podobě „přeanalyzování“ a přílišné teorie. Je preferováno zaměření na praktické využití.

- Dodávat zákazníkovi kvalitní software

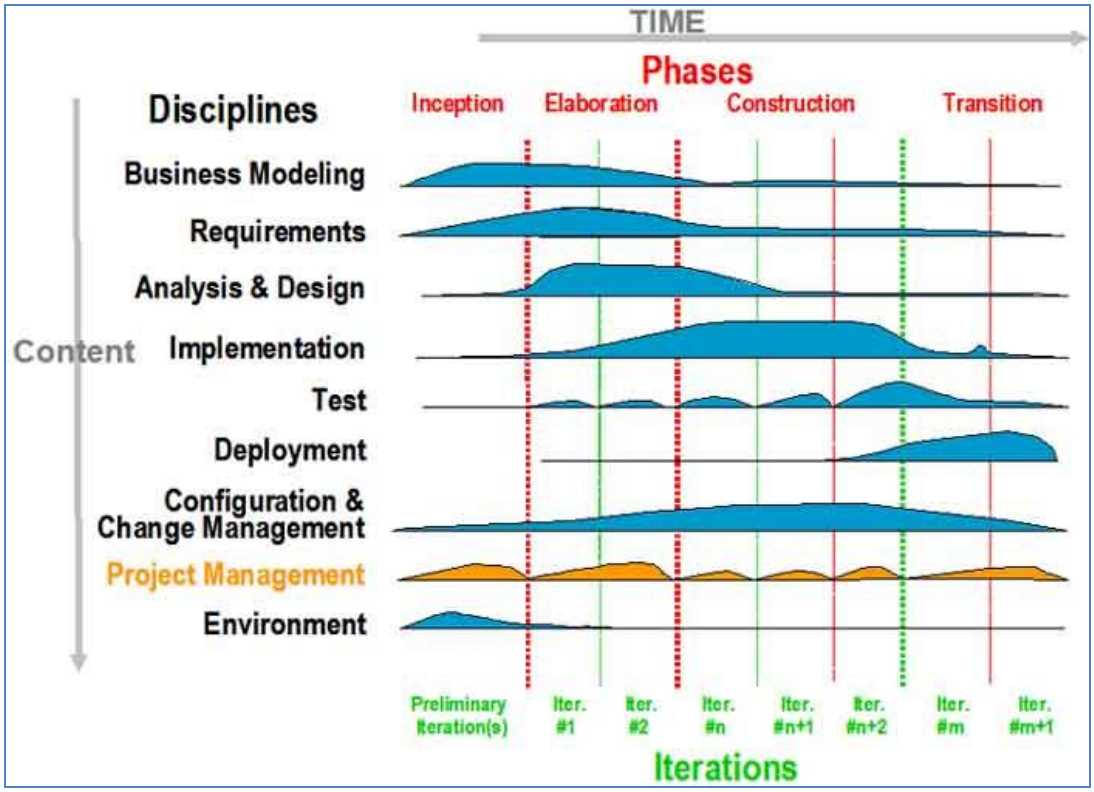
  Hodnotný je pouze software, který je řádně otestovaný.

- Implementovat změny co nejdříve je to možné

  Včasnou implementací docílíme toho, že náklady na zabudování nových či změnových požadavků budou co nejmenší.

- Pracovat týmově

  Produkt, který vznikne v pozitivní a přátelské atmosféře, bývá kvalitnější.



Obr. 3 RUP disciplíny (zdroj: IBM)

Z obrázku *RUP disciplíny* publikovaného IBM[4] vyplývá, že tato metodika používá čtyři fáze – *inception, elaboration, construction, transition.* Každá z těchto fází protíná

[4] Obrázek stažen z http://www.ibm.com/developerworks/rational/library/4763.html [online] 2004-05-26

několik disciplín v pravidelných cyklech, jak je vidět ze zmíněného obrázku. Počet iterací závisí na náročnosti projektu. Délka iterace záleží na týmu, obecně se ale doporučuje volit iterace jedno nebo dvoutýdenní.

Inception fáze

Hlavním cílem úvodní fáze je pochopit problematiku projektu, identifikovat rizika, use casy a sepsat vizi projektu. Vizí projektu se rozumí definice produktu z pohledu uživatele, obsahuje klíčové požadavky. V *Inception* fázi se také volí technologie a vybírají se nástroje (integrované vývojové prostředí, nástroje pro správu, vizualizaci a další). Na začátku projektu je také nutné vytvořit odhady nákladů a hlavně dojít ke společnému konsenzu – to znamená získat souhlas zákazníka, že jsme správně definovali a pochopili cíle projektu a jeho souhlas s odhadovanými náklady. Možným výsledkem *Inception* fáze může být ukončení projektu.

Elaboration fáze

Pokud jsme došli se zákazníkem ke shodě, jaký bude rozsah projektu, jaké jsou jeho požadavky a rizika, můžeme přejít do *Elaboration* fáze. Tato fáze si klade za úkol identifikovat a sepsat klíčové funkčnosti a především zjistit, zda jsme schopni odstranit všechna rizika, která se definovala v počáteční fázi. V této fázi rovněž definujeme kritické případy užití. K dosažení cílů dopomůže vytvoření spustitelného prototypu, který bude obsahovat základní funkcionalitu. Zavedením automatizovaných *buildů* dosáhneme toho, že komponenty budou včasně integrovány. Jestliže si odpovíme kladně na otázky typu: „Je architektura stabilní? Je navržené řešení spustitelné? A podařilo se identifikovat a správně odhadnout rizika?“, můžeme přejít do další fáze, kterou je *Construction*.

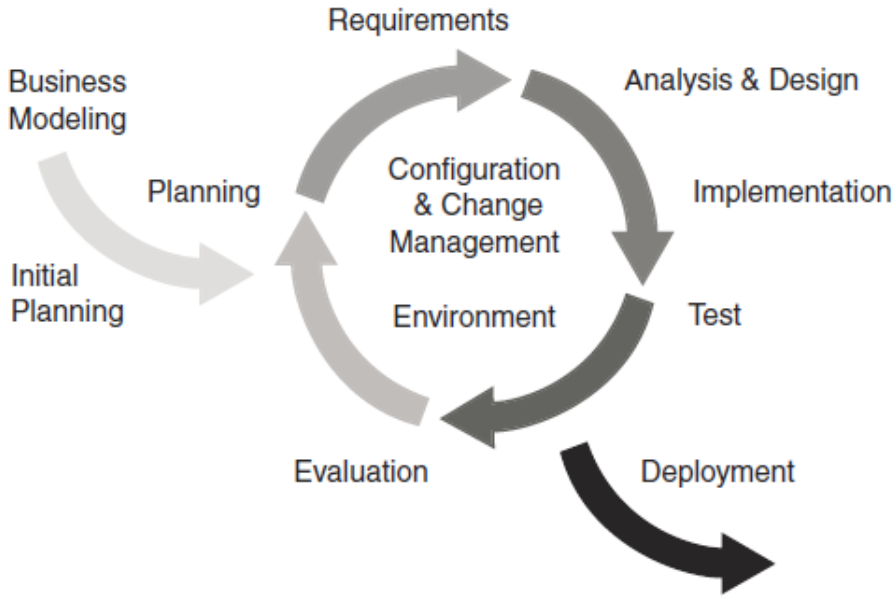
Construction fáze

Jak název této fáze napovídá, jejím cílem je detailní zkonstruování produktu. Snahou je implementovat zbývajících 80 % *use casů*. Cílem je vytvořit produkt téměř finální, připravený k zákaznickému testování. Základem je neustálá integrace, která je podpořena softwarem pro správu konfigurací, což umožňuje sestavit *build* ze správných verzí. Produkt se vyvíjí iterativně a průběžně se testuje chování všech komponent pomocí definovaných testovacích případů. Tato fáze je úspěšně zvládnutá, jestliže je produkt dostatečně stabilní, aby mohl být testován komunitou uživatelů. Nesmíme zapomenout na kontrolu výdajů, tedy zda se projekt drží ve stanovených limitech.

Transition fáze

Cílem poslední fáze je vyladit finální produkt po všech stránkách. To znamená maximalizovat výkonnost, uživatelskou použitelnost a celkovou kvalitu produktu. V této fázi již neprovádíme žádné zásadní změny, pouze dolaďujeme chybějící minoritní funkčnosti. Na řadu přichází řádné proškolení koncových uživatelů a správců produktu. Rozhodujícím cílem je dosažení souhlasu zákazníka, že aplikace splňuje všechny aspekty, které byly uvedeny ve vizi projektu.

Pro lepší pochopení pomůže obrázek *Iterativní vývoj podle RUP*, který názorně ilustruje vývoj v cyklech. Po identifikaci (a hlavně zákazníkovým souhlasem) byznys požadavků může projektový vedoucí začít s plánováním, analytici s analýzou, vývojoví pracovníci s implementací a testeři s testováním. Tyto základní kroky se opakují v každé iteraci.



Obr. 4 Iterativní vývoj podle RUP (zdroj: [KK03])

Jak je patrné z předchozího obrázku *RUP disciplíny*, mění se především poměr nákladů na jednotlivé disciplíny. Sběr požadavků bude probíhat hlavně v *Inception* a *Elaboration* fázi, kdežto nasazení (Deployment) bude probíhat u zákazníka ve fázi předávání.

Rational Unified Process obsahuje stovky rolí a artefaktů se kterými je možné pracovat. Na rozdíl od OpenUP procesu a jedná se o placený produkt.

3 PODPORA VÝVOJE NÁSTROJI

V dnešní době existuje celá řada podpůrných nástrojů. Jejich zaměření je různé – některé podporují management, jiné správu úkolů, konfigurací nebo jiné disciplíny. Abych byl schopen posoudit dopady použití Rational Team Concert, tak bylo nutné se seznámit i s jinými nástroji.

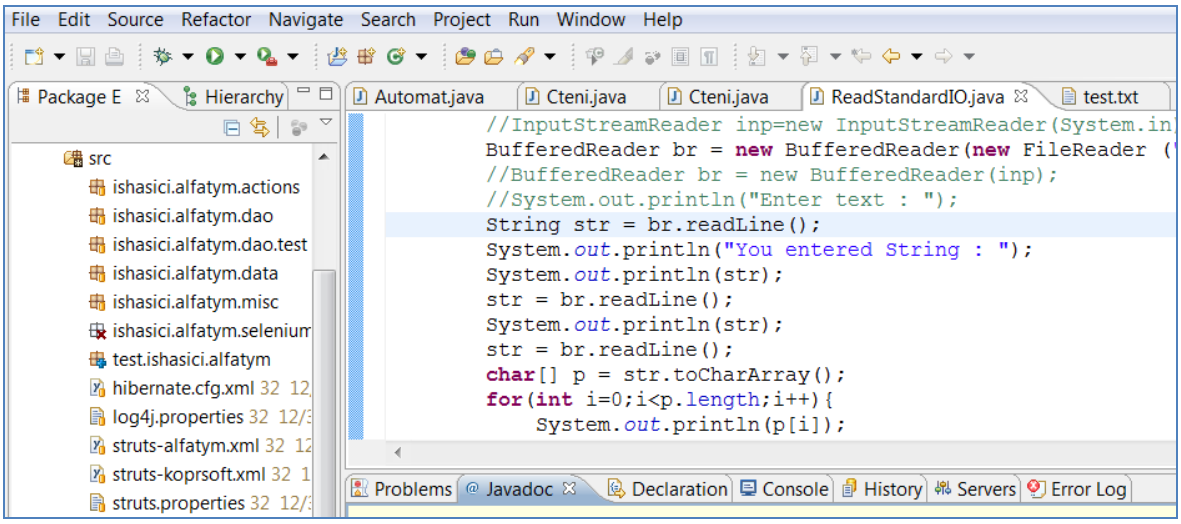
3.1 IDE nástroje

IDE (Integrated Development Environment), je do češtiny překládané jako integrované vývojové prostředí. Cílem těchto nástrojů je usnadnit vývojářům vývoj a údržbu kódu. IDE nativně obsahují nápovědu, webové prohlížeče nebo třeba FTP klienty. Moderní vývojové prostředí mají v sobě zakomponovanou podporu issue trackingu a pomáhají například s automatizací některých kroků při refaktoringu kódu.

Mezi nejznámější vývojové prostředí patří Eclipse, NetBeans či Visual Studio. Tato prostředí umožňují vyvíjet v několika programovacích jazycích desktopové, mobilní či webové aplikace. Jak Eclipse, tak i NetBeans jsou zdarma k použití. Veledůležitou vlastností vývojových prostředí je, že umožňují použití plug-in modulů. Například pro Eclipse existuje již přes milión nejrůznějších doplňků, které je možné stáhnout z oficiálních stránek společnosti.[5]

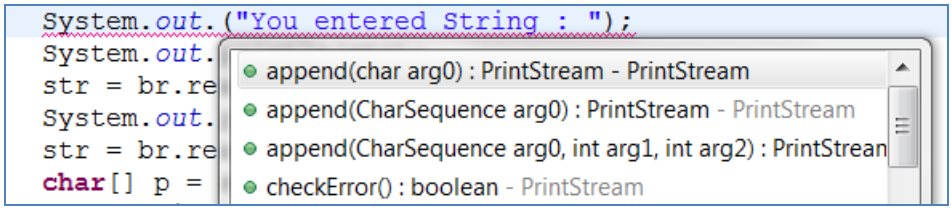
Následující obrázek číslo pět je print screen z prostředí Eclipse verze Galileo. Lze vidět, že prostředí je vybaveno několika záložkami, které vytváří komfortní prostředí. Eclipse Galileo používá Javadoc, což je dokumentace, kterou vytváříme za pomocí speciálních tagů přímo v kódu. Po vygenerování je pak dokumentace dostupná v podobě html souborů. Záložka *Problems* zobrazuje chyby v kódu, které odhalil kompilátor. V záložce *Servers* se pak zobrazuje seznam nainstalovaných serverů (např. Tomcat) včetně informace o stavu, v jakém se server nachází.

[5] http://marketplace.eclipse.org/



Obr. 5 Eclipse Galileo (zdroj: vlastní)

Z horního menu obrázku zaujme menu *Refactor*, který napomáhá vývojáři se zvýšením kvality kódu. Poslední obrázek této podkapitoly, který jsem pojmenoval *Našeptávač*, ukazuje, že nechybí ani inteligentní nápověda – Eclipse prostředí nám po stisknutí kombinace kláves Ctrl a mezerníku nabízí v závislosti na pozici v kódu, kde se nacházíme, které metody nebo proměnné můžeme využít.



Obr. 6 Našeptávač (zdroj: vlastní)

3.2 Správa úkolů

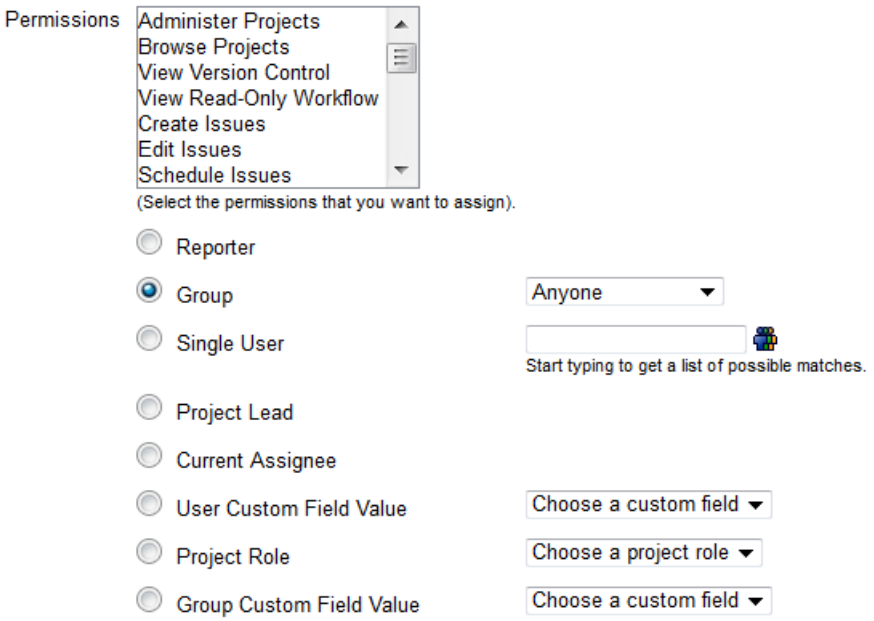
Správu úkolů (angl. issue tracking) lze provozovat například s pomocí nástroje Jira[6] nebo Bugzilla[7]. Tyto nástroje umožňují sledovat změny v kódu a monitorovat různé defekty či požadavky. Jira může být propojena i s Eclipse, Visual Studiem a dokonce i s nástrojem pro verzování kódu.

[6] http://www.atlassian.com/software/jira/overview
[7] http://www.bugzilla.org/

Jednotlivé položky je možné vyhledávat, aktualizovat a samozřejmě i reportovat. Issue tracking pomáhá zvyšovat produktivitu práce a v konečném součtu zlepšovat kvalitu výsledného produktu.

JIRA

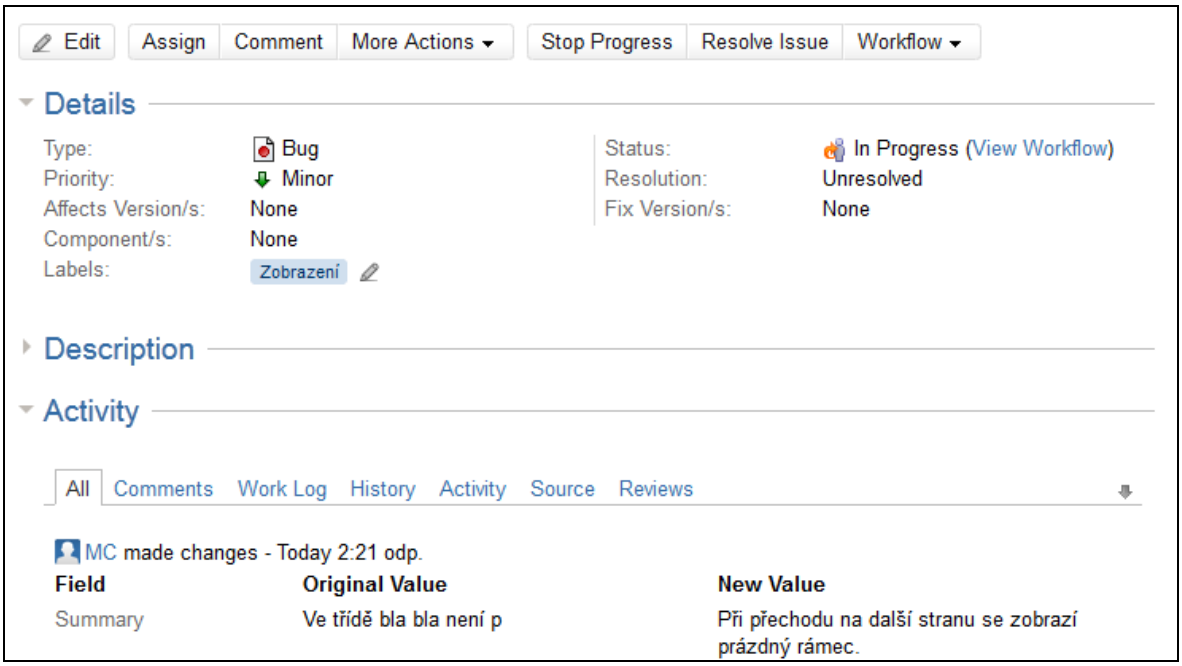
Pro vyzkoušení Jiry je nutné si nainstalovat lokální server a zaregistrovat se na oficiálních stránkách. Po zadání ID serveru je vygenerován devadesátidenní licenční klíč. Práce s Jirou začíná vytvořením alespoň jednoho uživatelského účtu a projektu. Pro incidenty (angl. issues) můžeme definovat celou řadu oprávnění. Je zde možnost nastavit, kteří uživatelé, skupiny či projekty mohou vytvářet, editovat, plánovat, přiřazovat, přesouvat, odstraňovat či propojovat incidenty. Incidentům také můžeme přiřadit určitý stupeň bezpečnosti, čímž docílíme toho, že pouze lidé s určitým oprávněním s nimi mohou manipulovat, což můžeme vidět z následujícího obrázku *Oprávnění incidentů*.



Obr. 7 Oprávnění incidentů. (zdroj: vlastní)

Incidenty jsou v prostředí Jiry defaultně rozděleny do pěti typů (chyba, nová funkcionalita, úloha, zlepšení, dílčí úloha), existuje však i možnost definovat si vlastní typy, stejně jako definovat mimo základní statusy (otevřeno, zpracovává se, znovuotevřeno, vyřešeno, uzavřeno) své vlastní statusy a podstatusy.

Obrázek *Incident* ukazuje, jaké informace týkající se incidentů můžeme spravovat.



Obr. 8 Incident (zdroj: vlastní)

Jira umožňuje vytvoření a nahrání vlastních pluginů nebo stažení z oficiálních stránek (z těch zajímavých jmenujme JIRA Importers Plugin, což je plugin, který umožňuje migrovat data ze systémů Bugzilla, Mantis, FogBugz, Trac a Pitoval Tracker). Oficiálně podporované pluginy, stejně jako samotná Jira jsou zdarma k vyzkoušení pouze na určitou dobu.

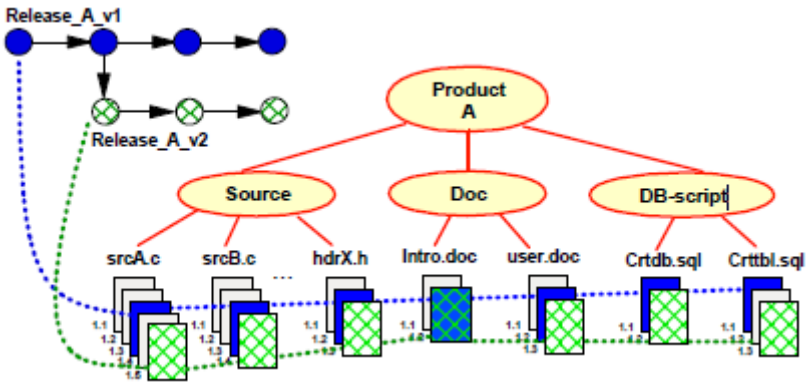
Podobně jako jiné nástroje i Jira nabízí monitorování toho, co nás zajímá, a to skrze panel Dashboard. K dispozici je již přes dvacet gadgetů, které umožňují zobrazovat informace o aktivitě týmu, přiřazení pracovních položek, času k jejich řešení a podobně. Stejně jako můžeme vytvářet vlastní pluginy, je možné vytvořit i vlastní gadgety. Firma Atlassian nabízí zdarma tutoriály a návody, jak správně postupovat při vytváření vlatních funkcionalit.[8]

3.3 Správa konfigurací a změn

Jedna z univerzálních definic říká, že Správa konfigurací (angl. Configuration Management) je unikátní identifikace, řízené skladování a řízení změn včetně hlášení statusu vybraných komponent a produktu během života systému.[CC03] Správu konfigurací definujeme ve fázi *Inception*. Poslední změny se pak dějí ve fázi *Elaboration.*

[8] https://developer.atlassian.com/display/GADGETS/Writing+an+Atlassian+Gadget.[online] 2012-02-02

Základním prvkem správy konfigurací je konfigurační databáze, která obsahuje informace o všech komponentech. Databáze detailně monitoruje stav všech konfiguračních položek, včetně jejich vzájemných vztahů a artefaktů (dokumenty, zdrojové kódy, buildy či konfigurační soubory). Správa konfigurací napomáhá k tomu, aby se minimalizovaly dopady různých změn jejich „verzováním“. Změny mohou provádět pouze osoby s příslušným oprávněním.



Obr. 9 Konfigurace produktu (zdroj: IRB1)

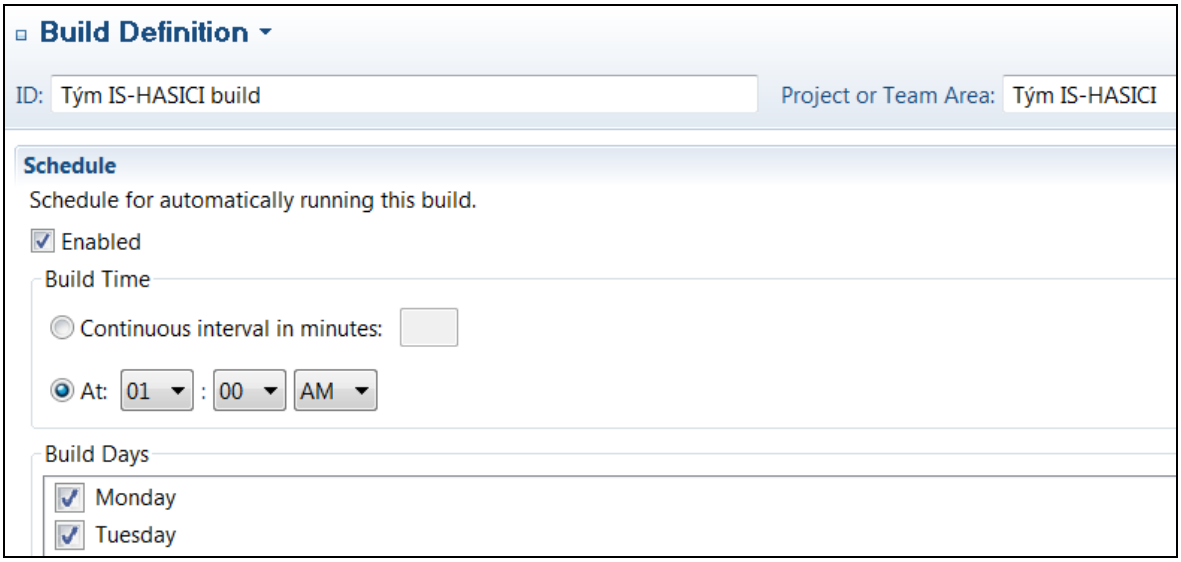
Obrázek konfigurace produktu je dobrým příkladem k názornému vysvětlení toho, co obnáší správa konfigurací a změn. Imaginární produkt A se skládá z několika složek (Source, Doc, DB-script), přičemž každá z těchto složek obsahuje několik verzí zmíněných souborů. Pod pojmem release si tedy můžeme představit mapování či logické spojení všech relevantních artefaktů, které spolu souvisí vůči produktu a jsou testovány a sestaveny společně. Platí, že jeden artefakt může být součástí několika releasů.[PK08]

3.4 Nástroje pro správu sestavení

Jedním ze základních principů iterativního vývoje je pravidelné doručování demo aplikací zákazníkovi. Abychom toho docílili, musíme prvně doručit funkcionalitu nebo kód na server. Nástroje pro správu sestavení se starají o automatický, plánovaný nebo ruční přesun kódu tak, aby nedocházelo k nekonsistencím. Tento proces je velice důležitý, neboť včasná a správná integrace jednotlivých komponent zaručí, že aplikace bude spustitelná. Mnoho integrovaných vývojových prostředí má vlastní nástroje pro sestavení. Z externích nástrojů uvádím Buidbot[9]. Pro dokreslení představy o správě sestavení cituji ze skript

[9] http://trac.buildbot.net/.

Informačních systémů: „Buildování je integrace a sestavení spustitelné aplikace nebo některé její části.“[PVK10]



Obr. 10 Definice sestavení (zdroj: vlastní)

Z obrázku *Definice sestavení* můžeme vidět, že vedle zadání ID buildu vybíráme projekt nebo tým, kterého se sestavení týká. Co se plánování týče, můžeme určit, zda build budeme generovat v pravidelném časovém intervalu (udaném v minutách) nebo v určitý čas ve zvolené dny.

3.5 Integrované nástroje

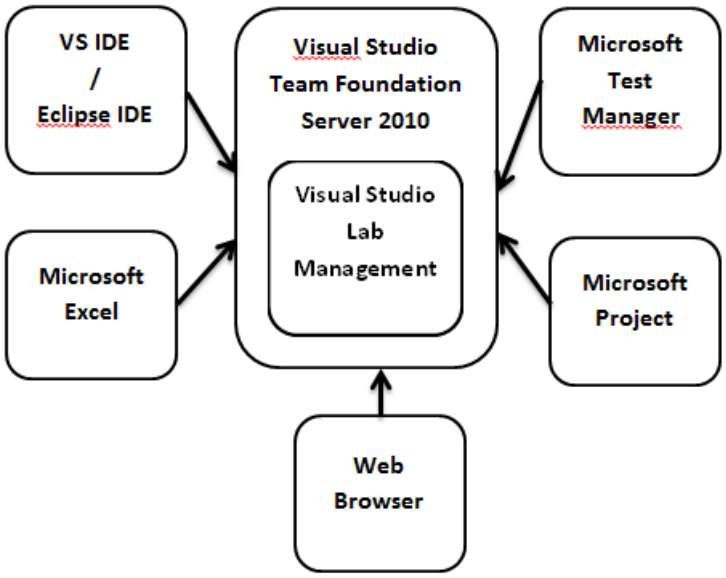
Na trhu také existují komplexní nástroje, které obsahují integrované vývojové prostředí, nástroje pro správu konfigurací a změn, správu sestavení a samozřejmě i podporu monitorování incidentů.

Tyto integrované nástroje lze používat s dalšími nástroji (od téhož nebo jiného, podporovaného výrobce), které obsahují jiné funkcionality či nějakým způsobem rozšiřují stávající.

Většinou se jedná se o kvalitní, ucelené řešení, které je plně nebo částečně zpoplatněno (podle počtu uživatelů) a podléhající licenčním podmínkám. Nejrozšířenějšími nástroji jsou Team Foundation Server od Microsoftu a Rational Team Concert od společnosti IBM.

3.5.1 Microsoft Team Foundation Server

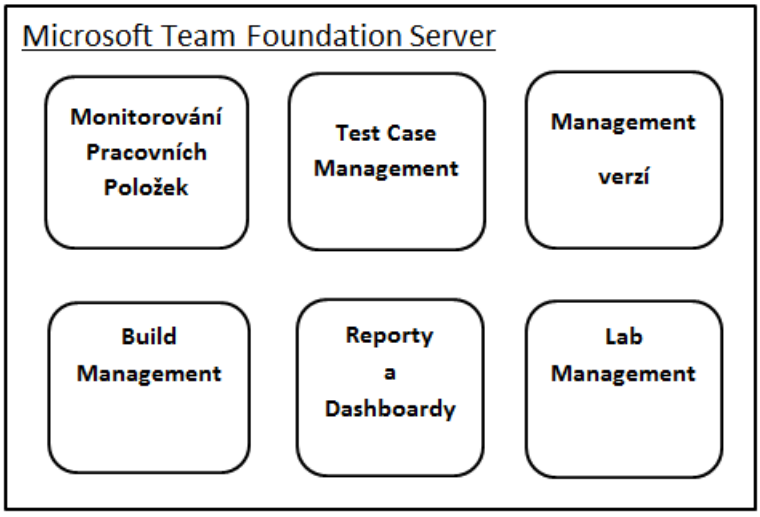
Microsoft Team Foundation Server je jedna ze základních komponent integrovaného vývojového prostředí nazvaného Visual Studio, která je následovníkem produktu Visual SourceSave. Podobně jako produkty z platformy Jazz od společnosti IBM i zde je možné jednotlivé produkty různě komponovat, jak lze vidět na obrázku *Začlenění TFS v rámci Visual Studio 2010.*



Obr. 11 Začlenění TFS v rámci Visual Studio 2010 (zdroj: vlastní)

Team Foundation Server je centrální úložiště, které lze propojit s dalšími nástroji od firmy Microsoft. Vývoj je možné provádět v některém z podporovaných programovacích jazyků. Mezi nejznámější jazyky, které jsou podporovány, patří C#, Visual Basic, C++ nebo například F#.

Velký přínos pro vývojový tým spočívá v podpoře Scrum metodiky. Team Foundation server totiž podporuje agilní procesy. Díky stažení a nainstalování procesní šablony z oficiálních stránek bude k dispozici plánování iterací či používání Backlogu ke správě pracovních položek. Plná trasovatelnost požadavků definuje vztahy mezi požadavky, položkami ke zpracování a také relevantními testovacími případy.

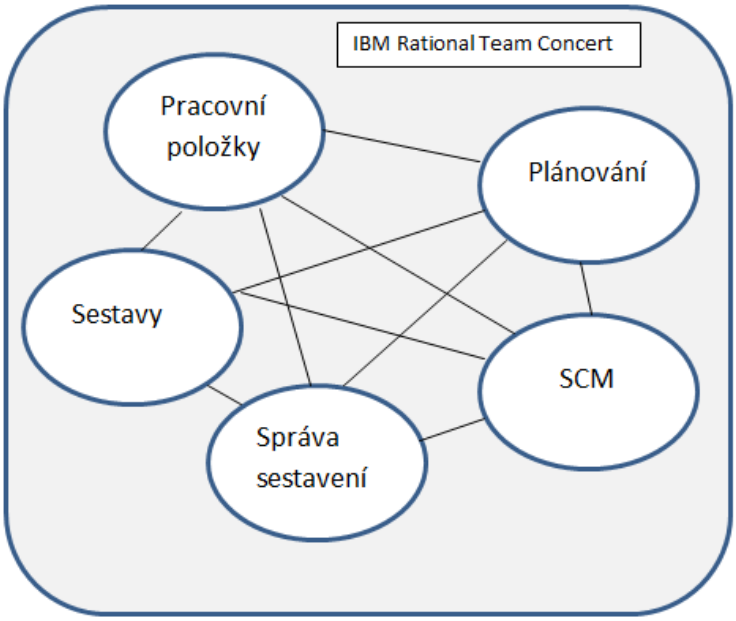


Obr. 12 Microsoft Team Foundation Server (zdroj: vlastní)

Na oficiálních stránkách produktu se můžeme dočíst, že Team Foundation server je platforma pro spolupráci v rámci řešení pro správu životního cyklu aplikace. V datovém skladu se uchovávají informace o pracovních položkách, správy revizí, testovacích nástrojů i buildů. V reálném čase jsou pomocí sestav či panelu Dashboard prezentovány užitečné informace o stavu komponent či historických trendech týkajících se například pracovních položek. Pro menší týmy je k dispozici zjednodušená instalace – není tedy nutné detailně všechno nastavovat, ale na základně výchozího nastavení se rovnou pustit do vývoje. Díky integraci s Microsoft Project a Microsoft Office Project server je možné získat informace o stavu projektu nebo je poskytnout zákazníkovi.[MS12]

3.5.2 IBM Rational Team Concert

Rational Team Concert (zkráceně RTC) je produkt vyvinutý firmou IBM. Na trh byl uveden v roce 2008. Rational Team Concert patří do skupiny softwaru Rational software, který obsahuje několik desítek produktů, které lze vzájemně kombinovat. Tento produkt je založen na platformě Jazz, což je technologická platforma, která podporuje týmovou spolupráci v globálním pojetí. IBM uvádí, že RTC lze přizpůsobit malým i velkým týmům. Obrázek *RTC nástroje* ukazuje propojení jednotlivých okruhů. Můžeme tedy vidět, že sestavu lze generovat pro pracovní položky, buildy i plánování.



Obr. 13 RTC nástroje (zdroj: vlastní)

IBM Rational Team Concert byl použit při vývoji *Systému pro koordinaci humanitární pomoci*. Detailní seznámení s Rational Team Concert a vytvořením vývojového procesu, který jsem v rámci RTC aplikoval, prezentuji ve čtvrté kapitole.

3.6 Souhrn kapitoly

Cílem této kapitoly bylo ukázat, jaké nástroje máme v dnešní době k dispozici pro vývoj a správu informačních systémů. Postupně byly popsány základní rysy integrovaných vývojových prostředí, správy položek, správy konfigurací i nástrojů pro práci s buildy. Velmi komplexními nástroji jsou Microsoft Team Foundation Server a IBM Rational Team Concert. Rational Team Concert byl vybrán pro použití při vývoji informačního systému, pro omezený počet uživatelů je totiž zdarma. Jeho customizaci, přínosy a nevýhody v rámci vývojového procesu a jiné vlastnosti popisuji v následující kapitole.

4 SYSTÉM PRO KOORDINACI HUMANITÁRNÍ POMOCI

Podstatnou část diplomové práce jsem propojil s letním semestrem prvního ročníku navazujícího magisterského studia, kdy jsme v rámci předmětu *Ročníkový projekt 2* vyvíjeli *Systém pro koordinaci humanitární pomoci*.

V následující kapitole zmiňuji, které případy užití a scénáře jsme realizovali. Tato kapitola také popisuje vytvoření vývojového procesu a jeho nastavení v Rational Team Concert.

Než však přistoupím k samotné kapitole, zmíním, s jakými úskalími se můžeme setkat při vývoji informačních systémů podle Mary a Toma Poppendickovi. V knize Leading Lean Software Development: Results are not the point [PP09] dvojice uvádí pět hlavních příčin, které obvykle nastávají při vývoji informačních systémů. Stručně je popíši, protože při hodnocení projektu z nich budu vycházet.

- Komplexnost

Systémy obsahují mnoho funkčností typu „nice to have“, které však nenajdou reálného a smysluplného využití. Tyto „vymoženosti“ mají za důsledek zvýšení komplexnosti kódu, což znamená i nelineární nárůst nákladů.

- Ekonomické hledisko

Ideálním řešením je, když se nám za pětinu celkových nákladů podaří vyvinout systém, který by byl reálně použitelný. Teprve poté má smysl sesbírat od uživatelů jejich požadavky a postupně, podle důležitosti, je implementovat. Od uživatelů dostaneme totiž mnohem přesnější a tedy i cennější informace o jejich potřebách a tým se může zaměřit na to, co je skutečně podstatné.

- Ukazatele produktivity

Produktivita by se nikdy neměla posuzovat podle toho, kolik řádků kódu, metod či tříd se vytvořilo. Tyto ukazatele mohou být zajímavé, ale neměly by být použité pro zhodnocení výkonnosti. Laické porovnání dvou rybářských vest pomůže s objasněním, proč je kvalitnější kód lepší než kvantitativní. Nelze přece říci, že vesta s velkým množstvím kapes je lepší, než vesta s několika kapsami, které jsou však dobře rozmístěny a kvalitně přišity.

- Udržitelnost kódu

Architektura řešení by vždy měla být zvolena s ohledem na to, aby kód mohl být v budoucnu snadno rozšiřován. Je vhodné nastavit takovou politiku, která myslí na refaktoring kódu.

- Technická složitost

Zákazníci chtějí, aby software zautomatizoval jejich složité procesy, které jsou často velmi nejasné. To je však chybná úvaha. Mnohem efektivnější a je jeví zamyslet se nad tím, jak lze jejich byznys procesy zjednodušit a až potom se je snažit implementovat.

Jak jsem již avizoval – z výše zmíněných informací budu vycházet při hodnocení projektu.

4.1 Inception fáze

V rámci ročníkového projektu jsme měli vytvořit systém pro koordinaci humanitární pomoci, který by pomáhal v dobách nouze (např. povodně či jiné katastrofy).

Důvodem pro zahájení projektu byla skutečnost, že současné přidělování humanitární pomoci postiženým občanům probíhalo neefektivně a nekoordinovaně. Chyběla přehledná evidence nabízených potřeb či pomoci. Nebylo možné systematicky párovat nabídky s poptávkami neboli určit co, kdy, kam a v jakém množství má být rozděleno.

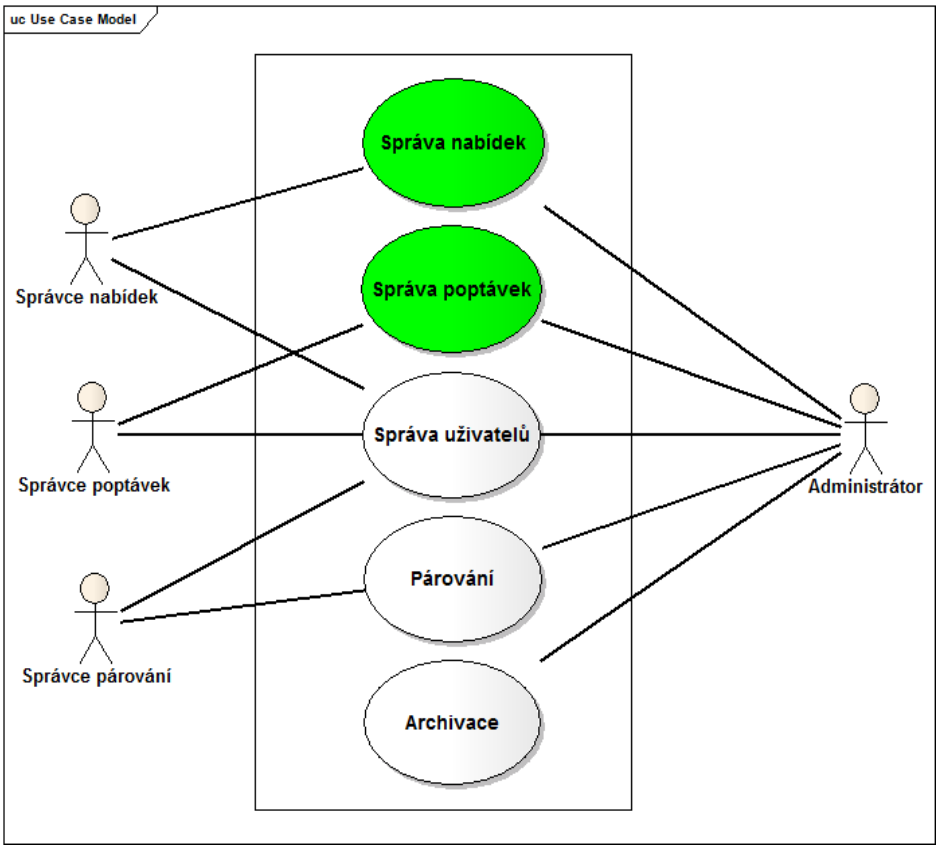
Problém	Přidělování humanitární pomoci je neefektivní. Tento nedostatek odstraní informační systém, který bude umožňovat párování nabídek a poptávek.
Řešení	Vytvoření informačního systému, který zautomatizuje evidenci a párování nabídek a poptávek.
Zainteresované strany	Terénní pracovníci
	Dárci
	Operátor call centra
	Realizátor párování
Případy užití	Správa nabídek
	Správa poptávek

	Správa uživatelů
	Párování
	Archivace

Tab. 1 Vize (zdroj: vlastní)

Úkolem Alfatýmu bylo vyřešit dva případy užití, a to správu nabídek a správu poptávek. Pro úplnost bych dodal, že jsme byli rozděleni do 2 velkých týmů, které měly zpracovat řešení v rozdílných technologiích. Já jsem byl součástí týmu, který vyvíjel v Javě. Tyto dva týmy byly následně rozděleny na podtýmy, z nichž každý měl na starost určité případy užití.

Z obrázku *Use Case Model* můžeme názorně vidět, které případy užití byly přiřazeny uživatelským rolím.



Obr. 14 Use Case Model (zdroj: vlastní)

Identifikovali jsme, že správce nabídek má být spjat s případem užití správa nabídek a správcem uživatelů. Analogicky byl pak spojen správce poptávek s use case správce poptávek a správcem uživatelů.

4.1.1 I0: Scénáře pro případ užití správa poptávek

Pro účely této práce je postačující, abych zde popsal scénáře pouze pro jeden use case, jelikož use casy přidělené *Alfatýmu* (správa nabídek a správa poptávek) si byly velice podobné.

Scénář má ve strukturované formě zachycovat, jakým způsobem hodlá uživatel systém používat a v podrobné míře také popsat, jaké funkcionality od něj uživatel očekává.[PK08]

Definovali jsme si čtyři scénáře pro případ užití Správa poptávek.

- Vložení nabídky
- Prohlížení nabídky
- Editace nabídky
- Smazání nabídky

Počáteční podmínkou pro zmíněné scénáře bylo, že uživateli se zobrazí úvodní obrazovka, kde vyplní své přihlašovací údaje. Systém jej pak přihlásí s příslušnými oprávněními.

Use Case	Správa poptávek
Scénář	**Vložení poptávek**
Use Case Description	Vložení poptávek do IS je nutné z důvodu párování nabídky a poptávky, které chceme evidovat.
Pre-condition	1. Uživatel je přihlášen s oprávněním pro vložení poptávky.
Basic flow	1. Uživatel vyplní formulář. 2. Systém zkontroluje vyplněné údaje a zobrazí uživateli výsledek. 3. Uživatel opraví údaje a požádá systém o validaci. a) opakuje se krok 2. b) data jsou v pořádku a pokračuje se krokem 4. 4. Uživatel potvrdí správnost dat 5. Systém uloží zadané informace.
Alternative flows	1. Uživatel může kdykoliv opustit obrazovku.
Post-condition:	Uživateli je zobrazen čistý formulář pro další vkládání.
References	UC správa poptávek, skripta ROPR1

Tab. 2 Scénář vložení poptávek (zdroj: vlastní)

Use Case	Správa poptávek
Scénář	**Prohlížení poptávek**
Use Case Description	Prohlížení poptávek bude realizováno ve formě výpisu. Cílem je graficky zobrazit poptávky.
Pre-condition	1. Uživatel je přihlášen s oprávněním pro prohlížení poptávek
Basic flow	1. Uživatel požádá systém o zobrazení poptávek. 2. Systém zobrazí seznam poptávek. 3. Uživatel si vybere některou z poptávek. 4. Systém zobrazí všechny informace dané poptávky.
Alternative flows	1. Uživatel kdykoliv může opustit obrazovku.
Special requirements	1. Filtr, který bude umožňovat vybírat poptávky podle zadaných kritérií. 2. Tlačítko pro zobrazení formy.
Post-codition:	1. Uživatel si může prohlédnout jinou poptávku.
references	Scénáře Editace nabídky a Smazání nabídky

Tab. 3 Scénář prohlížení poptávek (zdroj: vlastní)

Use Case	Správa poptávek
Scénář	**Editace poptávek**
Use Case Description	Editace poptávky slouží k úpravám již uložených údajů.
Pre-condition	1. Uživatel je přihlášen s oprávněním pro editaci poptávek. 2. Uživatel může editovat pouze již vytvořené poptávky.
Basic flow	1. Uživatel vyhledá konkrétní poptávku, kterou chce zobrazit. 2. Systém zobrazí danou poptávku se všemi údaji. 3. Uživatel provede změnu vybraných údajů. 4. Systém zkontroluje změněné údaje. 5. Uživatel potvrdí provedení změn. 6. Systém uloží danou poptávku.
Alternative flows	1. Uživatel může kdykoliv opustit obrazovku editace poptávky.
Special requirements	1. Zámek, který zabrání tomu, aby jedna nabídka mohla být editována zároveň dvěma uživateli.
Post-codition:	1. Uživatel má možnost zvolit si ze seznamu jinou poptávku k editaci.
References	Scénář Prohlížení

Tab. 4 Scénář editace poptávek (zdroj: vlastní)

4.1.2 I0: Risk list

Každý tým si definoval svá rizika, která můžou projekt nějakým způsobem ohrozit či dokonce zhatit. Naše rizika jsou uvedena v tabulce *Risk list*. Rizikům jsme přiřazovali prioritu ze stupnice 1 – 5, přičemž priorita 5 znamenala nejvyšší ohrožení. Z našeho

pohledu jsme ale takové riziko neměli. V každé iteraci jsme se snažili některé riziko odstranit, přičemž prvořadě byly zpracovávány položky s nejvyšší prioritou.

Číslo	Fakt	Riziko	Prio	Řešení
R1	Málo zkušeností s programováním	prodleva dokončení	3	tvorba prototypu
R2	Absence týmové praxe	váznoucí spolupráce	2	audio konference, pravidelné meetingy
R3	Špatné pochopení zákazníka	časové prodlevy, nedodržení termínů	3	konzultace se zákazníkem, ukázka aplikace na konci iterace
R4	Žádné zkušenosti s testováním softwaru	špatná kvalita softwaru - neodhalené chyby	3	zkoušení testovacích frameworku, ukázkových testů
R5	Neznalost ORM Hibernate	časové prodlevy, horší integrace	3	Naučit se, otestovat, vytvořit prototyp s ORM mapováním
R6	Neznalost MVC frameworku Struts2	časové prodlevy	3	Naučit se, otestovat, vytvořit prototyp Struts2 spolu s Hibernate

Tab. 5 Risk list (zdroj: vlastní)

Na tomto místě bych také rád zmínil, že tabulka rizik mohla být kdykoliv upravena podle potřeby. Pokud se například objevilo nové riziko, nepředstavovalo překážku, ale pouze záležitost k vyřešení, což je ostatně krédem všech agilních metodik. Tuto tabulku jsme spravovali mimo prostředí RTC, a to pomocí služby Google Docs.

4.1.3 I0: Projektový plán

Tabulka *Projektový plán* ukazuje, kolik iterací bylo celkem naplánováno. Fáze *Inception* a *Elaboration* byly uskutečněny během kurzu *Ročníkový projekt 1*, fáze *Construction* a *Elaboration* během kurzu *Ročníkový projekt 2*.

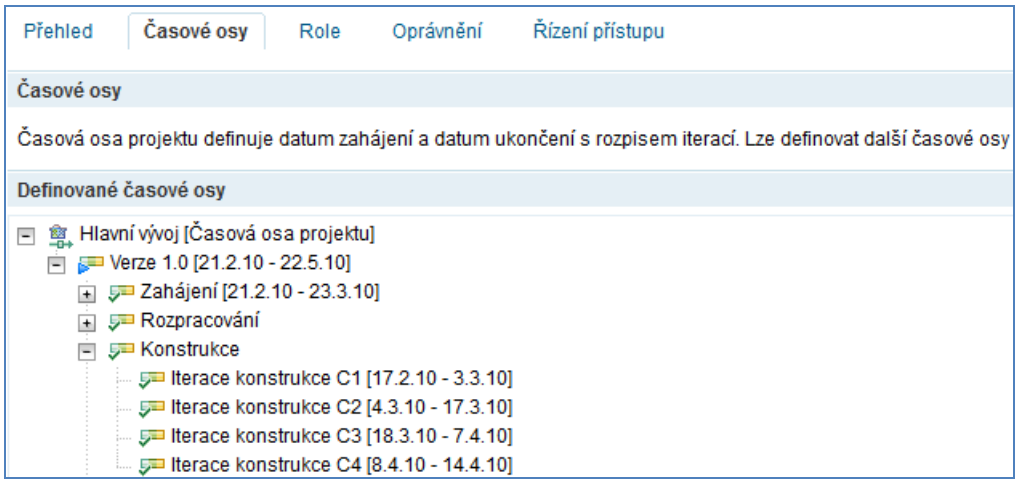
Celkem tedy bylo naplánováno 12 iterací, jak je vidno z tabulky *Projektový plán*. Podrobnější projektový plán, který obsahuje všechny případy užití s příslušnými úlohami, je přiložen v příloze.

Fáze	Počet iterací	Hlavní cíl fáze	Časový harmonogram
Inception	3	Identifikace problémů a vymezení rozsahu projektu, vytvoření prototypu	23.9.2009 – 14.10.2009
Elaboration	5	Sjednocení UC a scénářů, vytvoření a otestování architektury, implementace scénářů Zadaní nabídky/poptávky do systému	15.10.2009 – 16.12.2009
Construction	4	Implementace scénářů editace / smazání nabídky a poptávky, prohlížení nabídek a poptávek, filtrování, grafická tvorba	17.12.2009 – 14.4.2010
Transition	2	Příprava produktu k nasazení	15.4.2010 – 5.5.2010

Tab. 6 Projektový plán (zdroj: vlastní)

4.1.4 I1: Časová osa

Plány se definují v procesní šabloně (angl. process template). V záložce *Časová osa* můžeme vytvářet jednotlivé iterace pro dané fáze, jak ilustruje obrázek *Časové osy plánů*, který zobrazuje reálný plán projektu.



Obr. 15 Časové osy plánů (zdroj: vlastní)

Iterace mohou být v případě potřeby různě dlouhé. Samozřejmostí je i specifikace rolí a oprávnění jednotlivých uživatelů a týmů pro danou fázi nebo iteraci. V záložce řízení

přístupu je pak možné nastavit, komu bude umožněn přístup pro čtení (všem, nikomu, členům či specifikovaným uživatelům). *Alfatým* byl jediný tým, který Rational Team Concert využíval, takže přístup pro čtení jsem povolil každému.

4.1.5 Shrnutí

V počáteční fázi projektu byly identifikovány klíčové případy užití a scénáře. Pojmenovali jsme možná rizika pro dokončení nebo zpoždění projektu a přiřadili jim prioritu. V prostředí RTC jsem vytvořil projektový plán a časovou osu. Milník Lifecycle Objective Milestone (LCO) říká, že k tomu, aby se mohlo pokračovat ve vývoji, je nutné, aby se účastníci projektu shodli na jeho rozsahu, plánu, požadavcích i rizicích a plánu na jejich snížení. Tohoto milníku bylo dosaženo po fázi I2.

4.2 Elaboration fáze

Během druhé fáze jsme identifikovali kritické případy užití a stanovili architekturu řešení. V prostředí RTC jsem vytvořil projekt, definoval role a oprávnění jednotlivých uživatelů.

4.2.1 E1: Instalace a první spuštění RTC

V květnu roku 2010, kdy jsem tento produkt z oficiálních stránek stáhnul, byla aktuální verze s označením 2.0.0.2 iFix 2. S touto verzí jsem pracoval během *Elaboration, Construction* a *Transition* fází projektu. V následujícím textu podrobně popisuji vytvoření vývojového procesu a jednotlivé funkčnosti RTC, stejně jako customizace, které jsem učinil.

Rational Team Concert je možné provozovat na platformách Windows, Linux i Solaris. Veškerý software z rodiny produktů Rational je možné stáhnout až po zaregistrování se na komunitním webu.[10] Na výběr máme z několika možností.

[10] http://jazz.net.

verze	*Express-C*	*Express*	*Standard*	*Enterprise*
typ licence	permanentní	trial	trial	trial
počet uživatelů	10	50	250	neomezený
aplikační server	Tomcat	Tomcat/ WebSphere	Tomcat/ WebSphere	Tomcat/ WebSphere
HTTP(S) proxy suport	ne	ne	ano	ano

Tab. 7 Typy licencí (zdroj: vlastní)

V tabulce *Typy licencí*, která vychází z tabulky na oficiálních internetových stránkách[11], můžeme vidět, že hlavní rozdíl mezi jednotlivými verzemi je v počtu uživatelů, kteří mohou systém používat. Já jsem zvolil ke stažení verzi Express-C, která podporuje maximálně desetičlenné týmy, což bylo pro náš projekt dostačující. Také omezení v podobě aplikačního serveru Tomcat nepředstavovalo žádný problém.

Licence pro používání Express-C verze je permanentní a za její užívání není potřeba hradit žádný poplatek. IBM umožňuje v zásadě instalovat server dvěma způsoby.

Každá varianta nabízí možnost stáhnout server a klienta zvlášť. Server je možné provozovat na systémech Solaris-64, AIX PPC, Linux a samozřejmě Windows. Kompletní seznam toho, co je možné stáhnout je dostupný na oficiálních stránkách.[IBM09]

První způsob instaluje RTC, skrze Instalation Managera. Tuto variantu je vhodné využít v případě, že plánujeme používat více produktů Rational. Tato cesta je velice komfortní a přehledná. Z obrázku *Instalation Manager* můžeme vidět, že na výběr máme několik voleb (např. **aktualizovat** a **odvolání**). Odvolání vrátí produkt k předchozí verzi instalovaných softwarových balíků, aniž by došlo k nekonsistenci. Aktualizace naopak znamená přechod na vyšší verze. Je však nutné počítat i s tím, že mohou být vyžadovány manuální kroky.

[11] http://www-01.ibm.com/support/docview.wss?rs=3488&uid=swg27015716 [online] 2010-09-04.

Obr. 16 Instalation Manager (zdroj: vlastní)

Po kliknutí na možnost instalovat vyzve instalační manager uživatele, aby vybral balíky (tedy úložiště), ve kterých se nachází instalační soubory. V našem případě instalujeme pouze Rational Team Concert. Tato varianta je již obsažena, takže není třeba vyhledávat ani stahovat dodatečné balíky. Uživatel dále vybere úložiště, kde se má instalace provést a tímto jsou úkony skrze manažera dokončeny.

Druhou možností, jak postupovat, je, že si Rational Team Concert stáhneme v archivu. Po stažení přibližně 700 MB dat z oficiálních stránek můžeme začít se samotnou inicializací serveru. V tomto případě není potřeba nic instalovat – stačí *rozbalit* archiv a spustit dávkový příkaz *server.startup,* který nalezneme v *instalační_adresář/server/.* Následně dostaneme výpis o tom, které služby se spustily a jak dlouho to trvalo.

Nastavení, popsané v této kapitole, jsem učinil hlavně v průběhu *Elaboration* fáze, protože cílem bylo mít připravené prostředí do *Construction* fáze.

Prvotní customizaci je možné provést skrze webový prohlížeč po zadání této adresy *https://<IP adresa serveru>:9443/jazz/setup*. Výchozí administrátorský účet se jmenuje *Admin*, defaultní heslo je také *Admin*. Pro nastavení se nabízí dvě varianty, a to **rychlá** nebo **vlastní** instalace.

Rychlou instalaci je možné shrnout do dvou bodů:

- Nastavení registru uživatelů

  Zde volíme databázi. Vybral jsem databázi uživatelů Tomcat, protože nemám k dispozici žádný externí registr.

- Vytvoření administrátorského účtu

  Pro větší bezpečnost je doporučeno zakázat původní administrátorský účet a vytvořit nový administrátorský účet, takže jej vytvořím.

  Dále je rychlá a vlastní instalace společná. Popisuji ji v následujících podkapitolách.

4.2.2 E1: Vytvoření projektu

Nyní je možné přistoupit k vytvoření projektu. Jako první informace se vyplňuje název projektu s jeho popisem. Dalším krokem, který už není tak triviální, je volba procesu, který budeme používat.

Volba procesu

Při vytvoření nového projektu máme v RTC na výběr ze tří procesních šablon:

- Jednoduchý týmový proces
  Osoby přiřazené v týmu mohou provádět všechny dostupné akce. Osoby mimo tým nikoli.

- Scrum proces
  Jeden z agilních přístupů. Scrum metodika je popsána v kapitole 2.1.1.

- Proces OpenUp

  Tuto metodiku jsem popsal v kapitole 2.1.2.

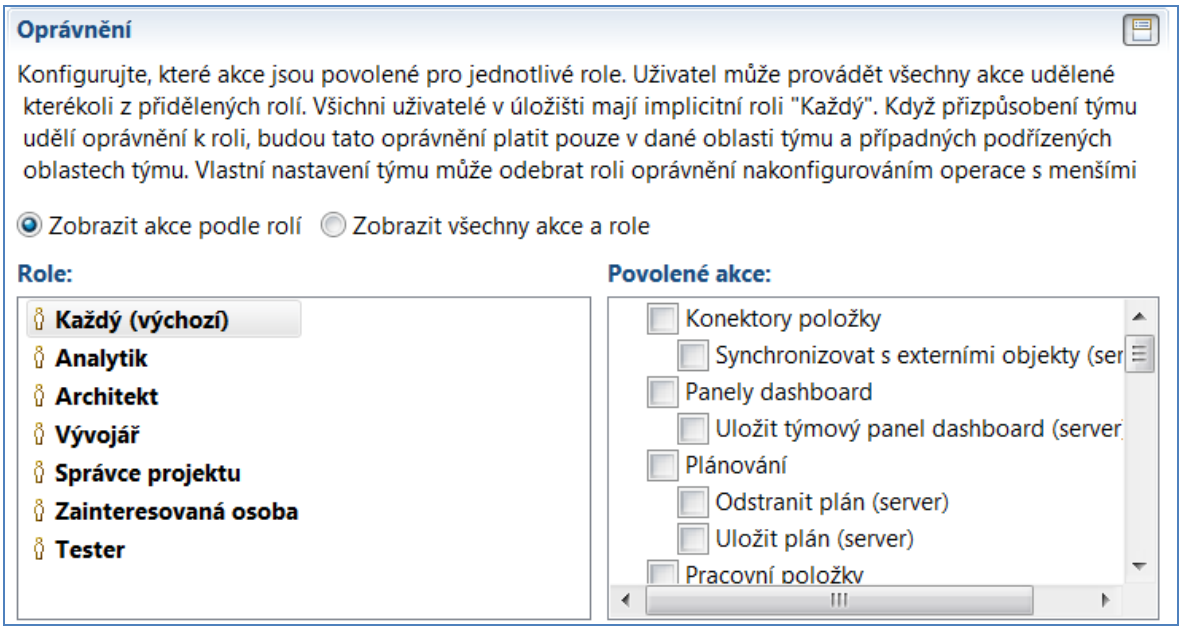
Z nabízených možností jsem vybral OpenUp proces, protože pro náš pětičlenný tým se jevil jako naprosto dostačující s ohledem na to, že členové týmu již měli teoretické znalosti RUP.

4.2.3 E1: Role a oprávnění

Proces OpenUp, podle kterého jsme postupovali, umožňuje použít následující role:

- Project Manager
- Team Lead
- Developer
- Tester

Jak je vidno z obrázku *Oprávnění*, pro každou roli můžeme nastavit, které akce budou povoleny či zakázány. K jednomu účtu může být přiřazeno více rolí.

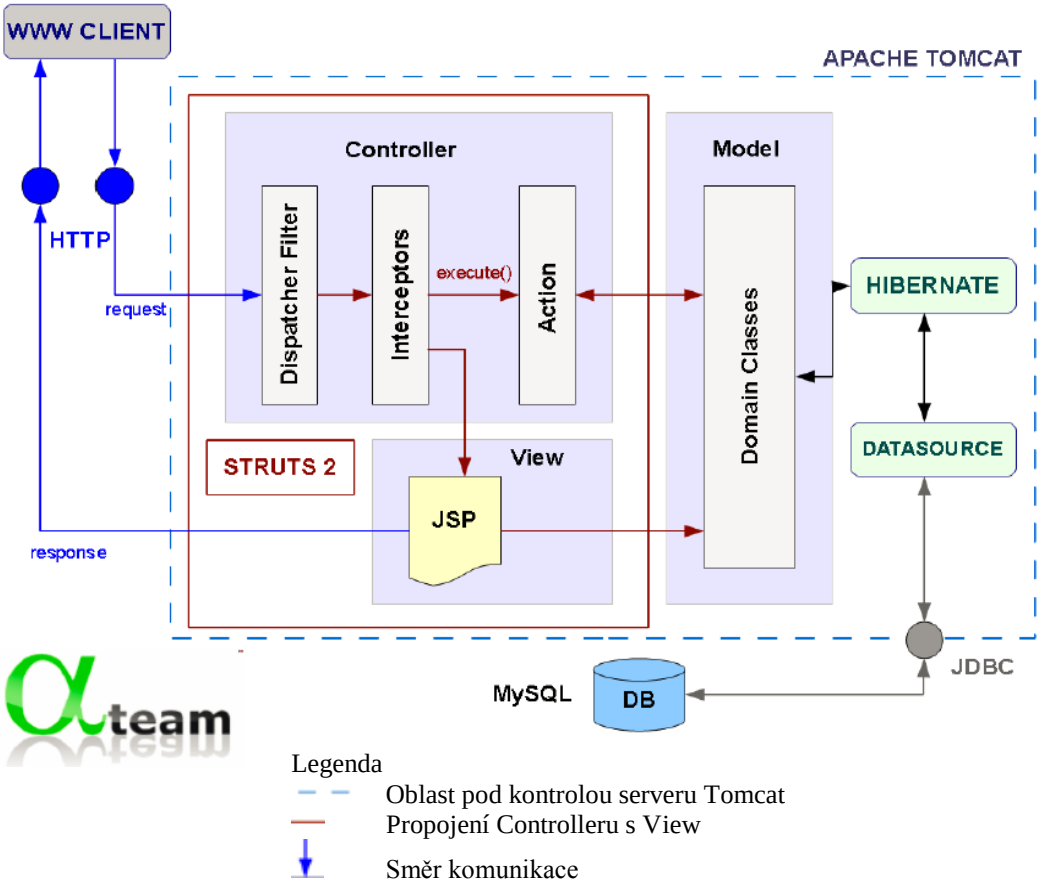


Obr. 17 Oprávnění (zdroj: vlastní)

Jednotlivé role jsem prošel a zkontroloval, že mají povoleny ty akce, které jsem uznal za vhodné. Role je možné definovat a modifikovat kdykoliv. Dokonce je možné definovat role i pro určité fáze projektu.

4.2.4 E2: Architektura řešení

Řešení bylo postaveno na frameworku Struts 2 s použitím MySQL databáze. Jako server byl vybrán Apache Tomcat, který přistupoval k databázi prostřednictvím frameworku Hibernate. Uživatelské prostředí bylo přístupné skrze webové rozhraní, jak ukazuje obrázek *Architektura řešení.*

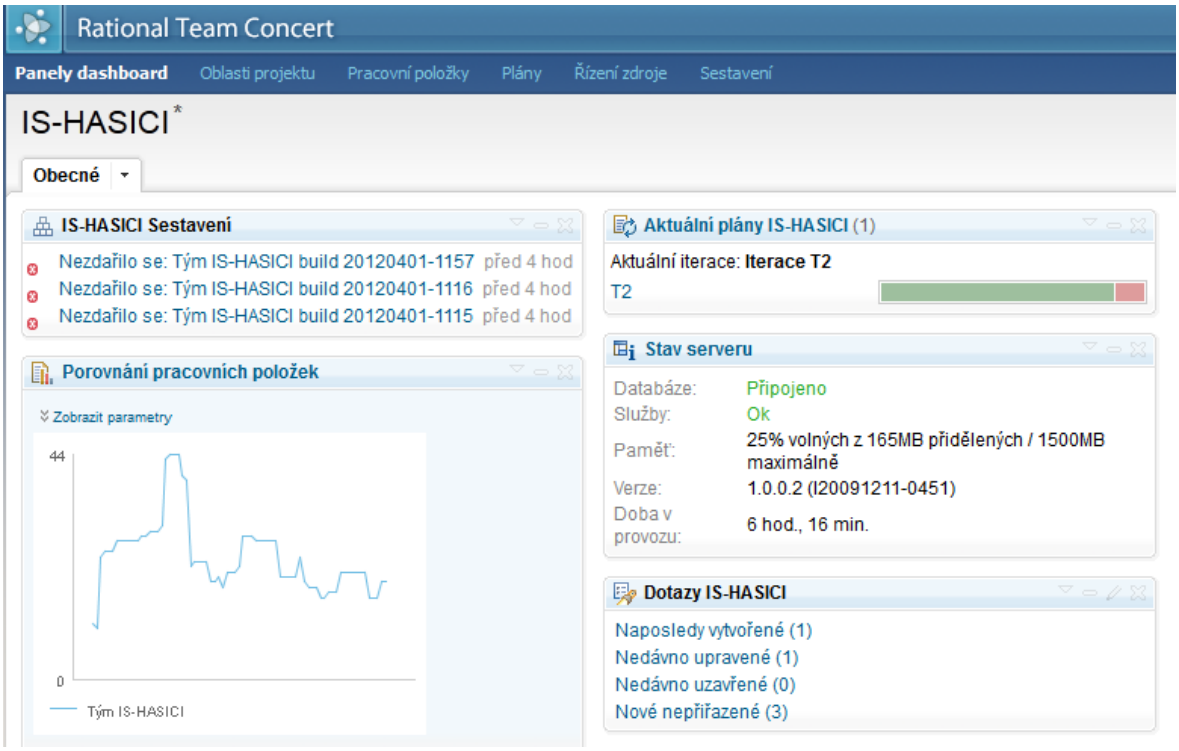


Obr. 18 Architektura řešení (zdroj: Alfatým)

V této fázi se také vytvořilo první spustitelné demo a zavedly se automatizované buildy, kdy se Eclipse IDE propojilo s Google repository, což bylo úložiště kódu, které bylo dostupné všem členům „velkého“ týmu. RTC repository nemohla být využita z důvodu omezení licence pro maximálně deset členů. Architektura řešení byla v rámci *Elaboration* fáze průběžně upřesňována a to na základě průběžně získaných zkušeností.

4.2.5 Tenký klient

Webový klient neposkytuje tolik funkčností, jako tlustý klient nainstalovaný na počítači uživatele. Skrze tenkého klienta není možné posílat kód do repository, vytvářet vlastní reporty nebo třeba provádět testy. Naopak, pokud potřebujeme informace z Dashboardu, informace týkající se plánování či pracovních položek, je tenký klient vyhovující. Jeho největší výhodou je, že na lokální stanici nepotřebujeme nic instalovat.



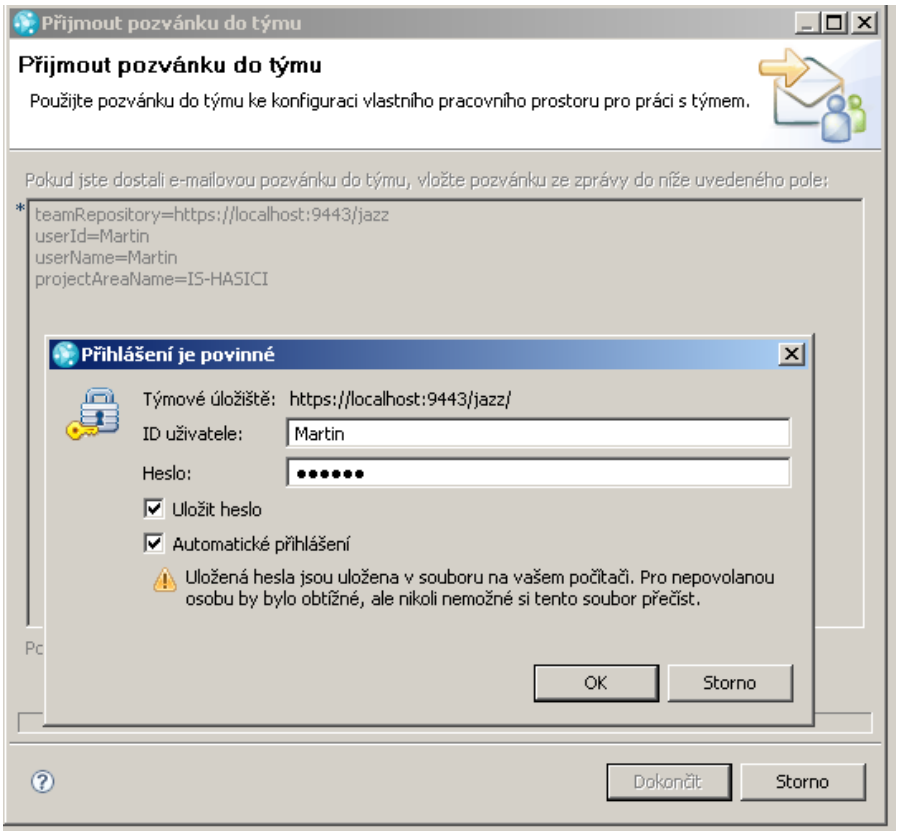
Obr. 19 Tenký klient (zdroj: vlastní)

Tenký klient je také dostačující k tomu, abychom měli kontrolu nad stavem jednotlivých komponentů a služeb, stejně jako informace o licenčních klíčích. Ukázka tenkého klienta je patrná z obrázku *Tenký klient*.

4.2.6 Tlustý klient

Eclipse klient po rozbalení z archivu zabírá na disku více než 800 MB. Samotná instalace tohoto obsáhlého klienta zabere několik minut. Po prvním spuštění jsme vyzváni k vložení pozvánky, kterou obdržíme e-mailem.

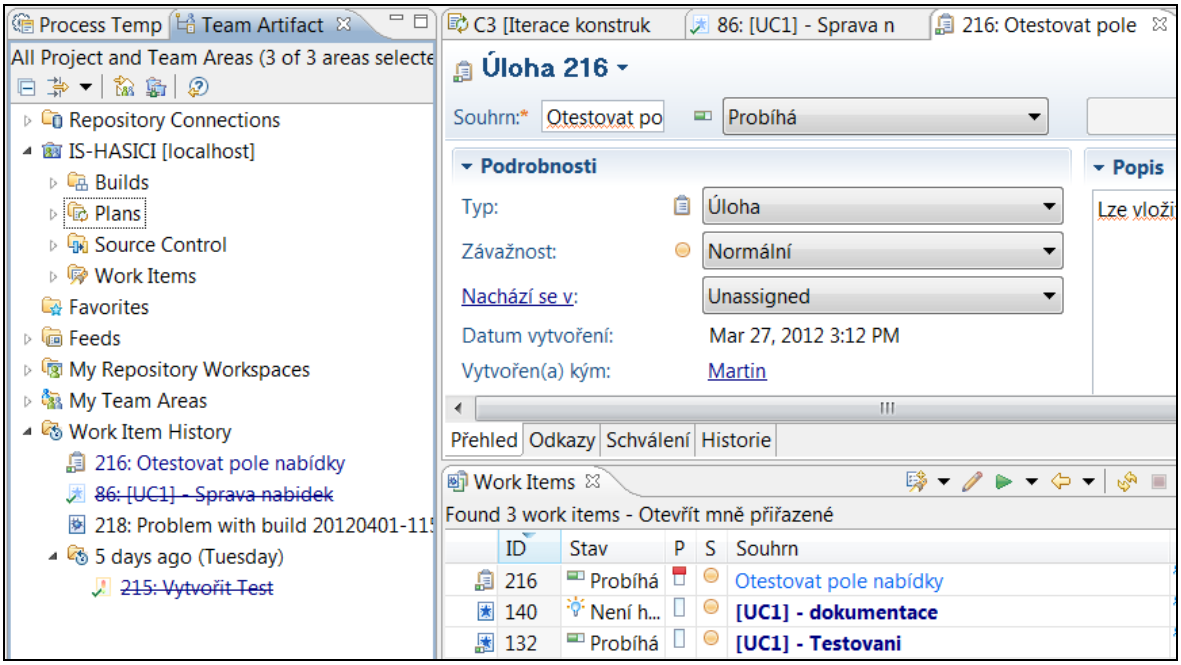
Pozvánka obsahuje několik údajů – adresu, na které je nainstalovaný server, ID uživatele a heslo. Pokud tuto pozvánku e-mailem neobdržíme, například z toho důvodu, že není nakonfigurovaný SMTP server, jak tomu bylo v našem případě, můžeme si ji sami vygenerovat přes webové rozhraní a poté ručně vložit do klienta.



Obr. 20 Pozvánka do týmu (zdroj: vlastní)

Po zadání hesla se klient připojí k úložišti a uživatel tak už může začít pracovat (viz. obrázek *Pozvánka do týmu*). Pokud se stane, že například kvůli chybějícímu SMTP serveru neobdržíme pozvánku, můžeme se připojit k týmu ručně.

Eclipse klient nám nabízí několik pohledů (Team artefacts, Java browsing, Jazz administration, Work items a jiné v závislosti na tom, jaké pluginy máme nainstalovány).



Obr. 21 Eclipse Client pro RTC (zdroj: vlastní)

Obrázek *Eclipse Client pro RTC* ukazuje pracovní položku (Úloha 216) společně s nabídkou *týmové artefakty* a seznamem pracovních položek přiřazených danému vývojáři. Zjistil jsem, že RTC „nádstavbu“ pro Eclipse lze doinstalovat i ručně. Podrobný návod s instrukcemi, co je třeba udělat, lze nalézt na komunitních stránkách.[SD09]

4.2.7 E3: Vytvoření uživatelů

Každému uživateli můžeme dát na výběr, ke kterému úložišti bude mít oprávnění.

- JazzAdmins – úplný přístup
- jazzDWAdmins – úplný přístup k datovému skladu
- JazzGuests – pouze čtení úložiště
- JazzUsers – čtení a zápis do úložiště
- JazzProjectAdmins – administrátoři úložiště se specifickými oprávnění pro manipulaci s oblastmi projektů a se šablonami

Pro administrátorský účet jsem zvolil všechna oprávnění kromě JazzGuests. Kolegům z týmu jsem pak nastavil oprávnění JazzUsers a JazzProjectAdmins, což pro

jejich potřeby bylo dostačující. Tato úroveň definování přístupu ještě nic nevypovídala o tom, jaké role bude mít uživatel přiděleny v rámci daného projektu.

RTC má definované i *licence pro klientský přístup.* Tyto licence jsou k dispozici podle toho, jakou verzi Rational Team Concert máme k dispozici. Verze Express-C, kterou jsme používali, má k dispozici licence:

- Contributor
- Developer
- Build System
- Clear Case Connector
- Clear Quest Connector (není obsažena ve verzi Express-C)

Clear Case Connector zajišťuje spolupráci mezi komponentou Řízení zdrojů produktu Rational Team Concert a produktem Rational ClearCase. Vzhledem k tomu, že jsme nepoužívali produkt Rational ClearCase, tuto licenci jsme nevyužili, stejně jako licenci Clear Quest Connector, která ve verzi Express-C není dostupná.

Každý uživatel si individuálně nastavil své pracovní prostředí, což znamená kolik hodin denně a kdy bude pracovat. Toto nastavení je důležité k tomu, aby se zjistilo, kolik času uživatel skutečně stráví prací.

Aby bylo možné využívat funkcionalitu monitorující vytížení týmů/členů, je nutné, aby každý člen vyplnil svůj pracovní týden, jak lze vidět na obrázku *Týdenní plán*.

Den	Délka pracovního dne	Čas ukončení práce
Pondělí	4 hod	17:00
Úterý	4 hod	17:00
Středa	3 hod	17:00
Čtvrtek	4 hod	17:00
Pátek	2 hod	17:00
Sobota	(Není)	(Není)
Neděle	(Není)	(Není)

Obr. 22 Týdenní plán (zdroj: vlastní)

Pro globální týmy má jistě smysl vyplnění časového pásma a regionálního prostředí, což však v našem případě nemělo smysl. Uživatel má také možnost vyplnit svou absenci, tak, aby to projektový nebo liniový manažer mohl zhodnotit při plánování lidských zdrojů do dalších fází a iterací.

4.2.8 Shrnutí

Milník Elaboration fáze Lifecycle Architecture Milestone (LCA) byl splněn tím, že systém byl spustitelný, architektura byla zvolena a otestována a klíčové rizika byla vyřešena. RTC jsem nastavil, vytvořil projekt i uživatele a přidělil jim potřebná oprávnění. Produkt byl zákazníkovi předveden a díky jeho souhlasu jsme mohli pokračovat fází konstrukce.

4.3 Construction fáze

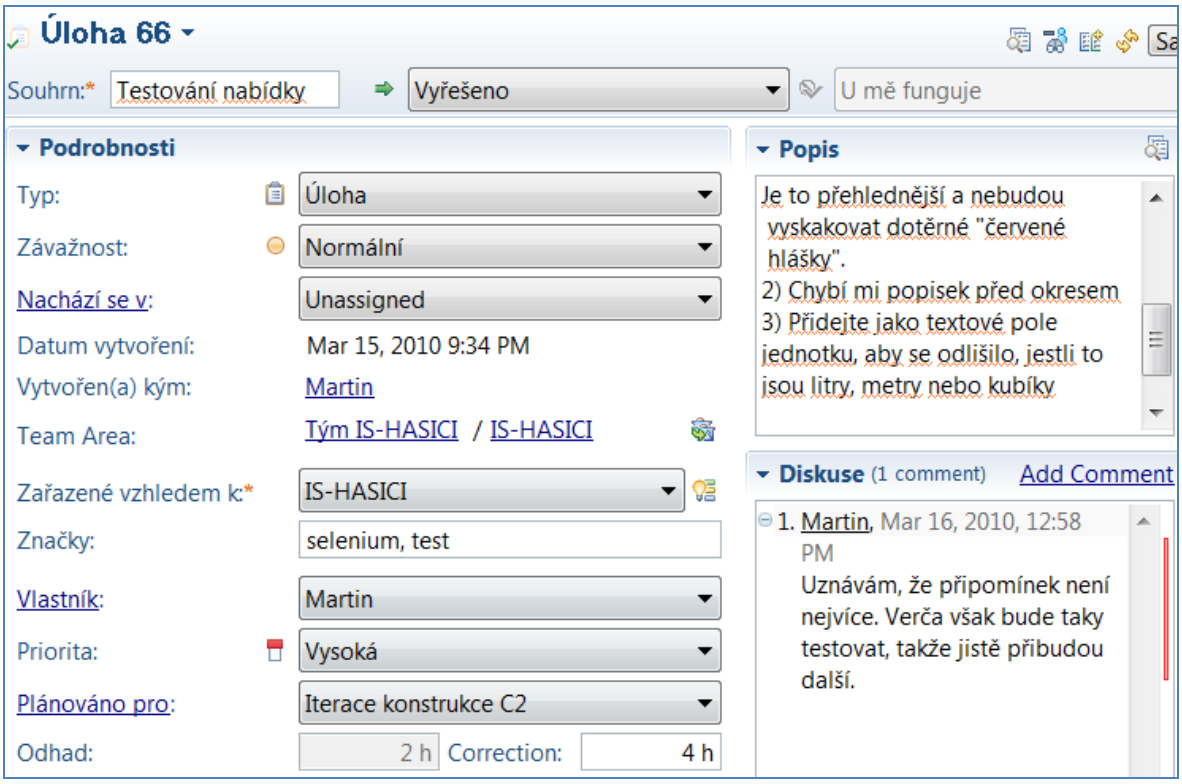
Třetí fáze projektu představuje detailní implementaci a testování zvolených scénářů. Tabulka *Scénáře v construction fázi* ukazuje, ve kterých iteracích byly zpracovány jednotlivé scénáře.

Iterace	Use Case	Scénář
C1	Správa nabídek	Editace nabídky
		Smazání nabídky
	Správa poptávek	Editace poptávky
		Smazání poptávky
C2	Správa nabídek	Prohlížení nabídek
	Správa poptávek	Prohlížení poptávek
C3	Správa nabídek	Prohlížení nabídek
		Filtrování nabídek
	Správa poptávek	Prohlížení poptávek
		Filtrování poptávek
C4	Správa poptávek	Filtrování poptávek

Tab. 8 Scénáře v construction fázi (zdroj: vlastní)

Během třetí fáze jsme hodně využívali pracovní položky. Pracovní položka (Work Item) je záznam, který může být několika typů – závada, úloha, vylepšení, riziko, případ

užití. Pracovní položka má jednoho vlastníka, který je zároveň jejím řešitelem. Můžeme jim také přiřadit závažnost.



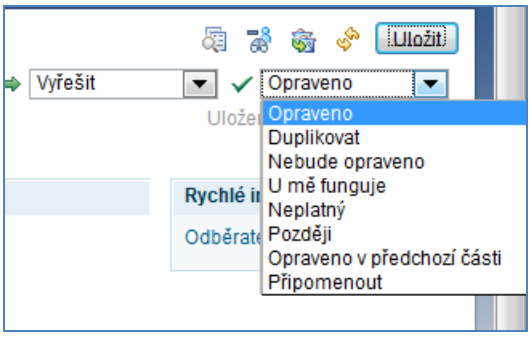
Obr. 23 Detail pracovní položky (zdroj: vlastní)

Obrázek *Detail pracovní položky* ukazuje, co všechno je možné nastavit. Jak vidíme, pracovní položku lze přiřadit pouze k jednomu projektu a ke konkrétní iteraci. O daném problému je možné diskutovat v části *Diskuse*. Velice důležitý je rámeček *Popis*, ve kterém je slovně popsán problém. Je na každém uživateli, do jaké míry jej popíše – žádná šablona neexistuje. My jsme se v průběhu projektu přesvědčili, že do popisu je vhodné co nejpřesněji a nejdetailněji vyplnit, co má daná pracovní položka řešit. Pokud tak neučíme, velice snadno se může stát, že některé položky budou řešit tu samou funkčnost nebo chybu, protože z popisu nebude jasné, co konkrétně má řešit. Položce můžeme přiřadit status (stav), který nám sděluje, v jakém stádiu se pracovní položka nachází. Podobně jako v jiných solution managerech[12] má pracovní položka podstatus, kterým se upřesňuje, jaká akce má následovat nebo v jaké se nachází. Variant, jak můžeme pracovní položky kombinovat, je několik.

[12] http://help.sap.com/saphelp_sm71_sp01/helpdata/en/45/51fbdbd4941803e10000000a1553f7/frameset.htm [online] 2012-02-13

Po vytvoření má položka status *nový*. Položka se následně objeví v backlogu, kde se osoba, která rozumí dané problematice, přiřadí jako vlastník nebo jí někdo tuto položku jednoduše přiřadí. Samotné zpracování položky začíná tím, že se nastaví status *zahájit práci*. Po uložení se status změní na „probíhá“.

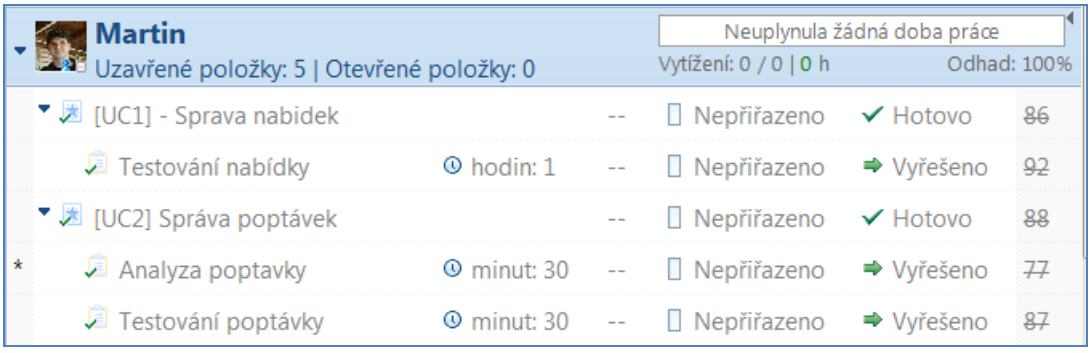
Na rozdíl například od SAP Solution managera zde chybí status, kterým by se dalo vlastníkovi najevo, že řešitel potřebuje upřesnit chybu nebo se doptat na nejasnosti (v Solution manageru značeno jako Customer Action), jak ilustruje obrázek *Statusy pracovní položky*.



Obr. 24 Statusy pracovní položky (zdroj: vlastní)

Musíme tedy spoléhat na to, že tvůrce položky má nastavený odběr změn pracovní položky a sám si aktivně přečte vložené informace nebo že si kontroluje ručně sobě přiřazené nové pracovní položky.

Velice přehledně se naopak jeví **seznam pracovních položek**, přiřazených uživateli. Obrázek *Pracovní položky* znázorňuje, že případy užití (v tomto případě správa nabídek a správa poptávek) mají definované pracovní položky *testování nabídky*, respektive *testování poptávky* a *analýza poptávky*. Vidíme, že v tomto případě byly všechny položky vyřešeny. Nepřiřazeno znamená, že dané položky neměly nastaveny žádnou prioritu.



Obr. 25 Pracovní položky (zdroj: vlastní)

Pracovní položky se mohou hierarchicky řadit, můžeme graficky rozlišit, která položka je jakoby nadřazená (například *[UC1] – Správa nabídek* je nadřazeno *Testování nabídky*).

Z důvodu lepší orientace v pracovních položkách jsem zavedl používání prefixů. Tedy pro každou iteraci a každý use case jsme vytvořili jednu pracovní položku, která začínala prefixem „[UCX] Název případu užití“, kde X znamenalo číslo daného případu užití.

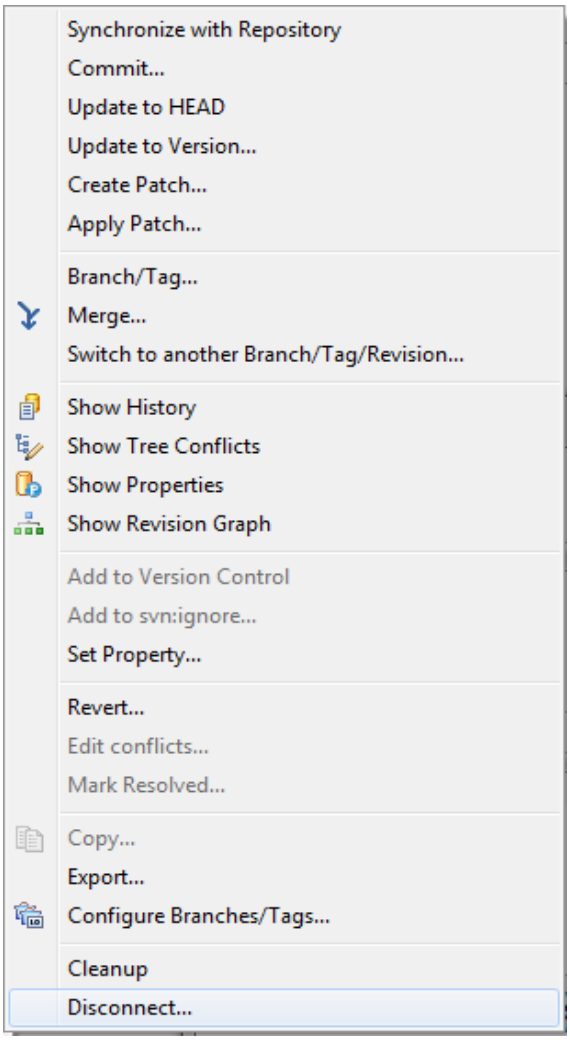
4.3.1 C1: Zpracování pracovních položek

Zpracování pracovní položky samozřejmě záviselo na povaze dané položky. Některé akce se týkaly vytvoření dokumentace nebo třeba scénářů, jiné (a to ve většině) modifikace nebo vytvoření zdrojového kódu. V takovém případě se kód odeslal do repository, jak vidíme z obrázku *Revize kódu.*

Re...	Date	Author	Comment
***55**	**2.3.10 17:56**	**MartyCaesar**	**Přidány nové testy na nabídku a poptávku**
54	1.3.10 21:13	p.klimanek	Commit změn - editace/mazání nabídky/poptávky
52	13.12.09 21:39	p.klimanek	
51	13.12.09 21:37	p.klimanek	
50	13.12.09 20:47	lordlight@se...	Integrace na urovni strutstu, pokus 1. Prepsani context.xml, ...
49	13.12.09 19:12	embrouz2	
48	8.12.09 20:41	lordlight@se...	Comit martyho selenium testu na spravne misto.
46	7.12.09 20:17	lordlight@se...	Chyba v testech, getAll() prepokladalo ze je db prazdna. FIX...

Obr. 26 Revize kódu (zdroj: vlastní)

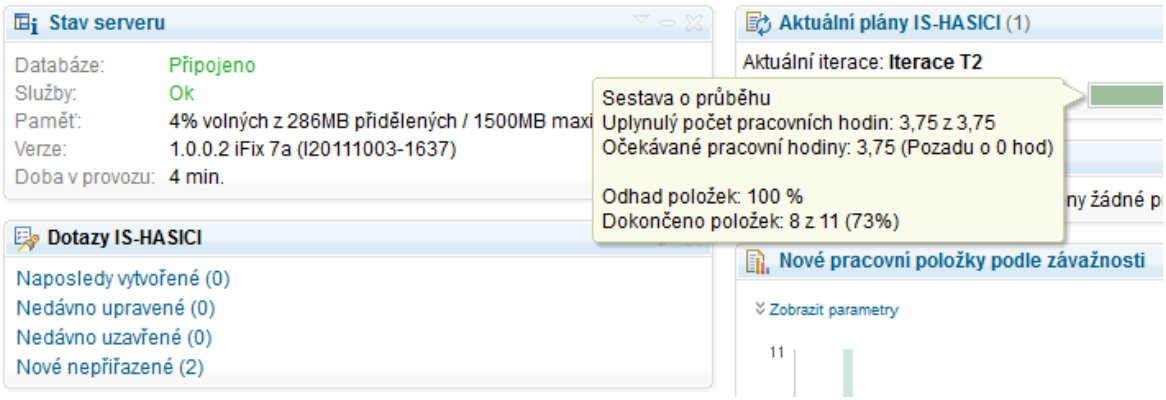
V případě, že pracovní položka zahrnovala úpravu kódu, po samotné úpravě následovala synchronizace s kódem v repository v podobě přidání revize s popiskem. Možností, jakým způsobem kód poslat nebo získat, je několik a jsou patrné z obrázku *Synchronizace kódu.*



Obr. 27 Synchronizace kódu (zdroj: vlastní)

4.3.2 C1: Dashboard panel

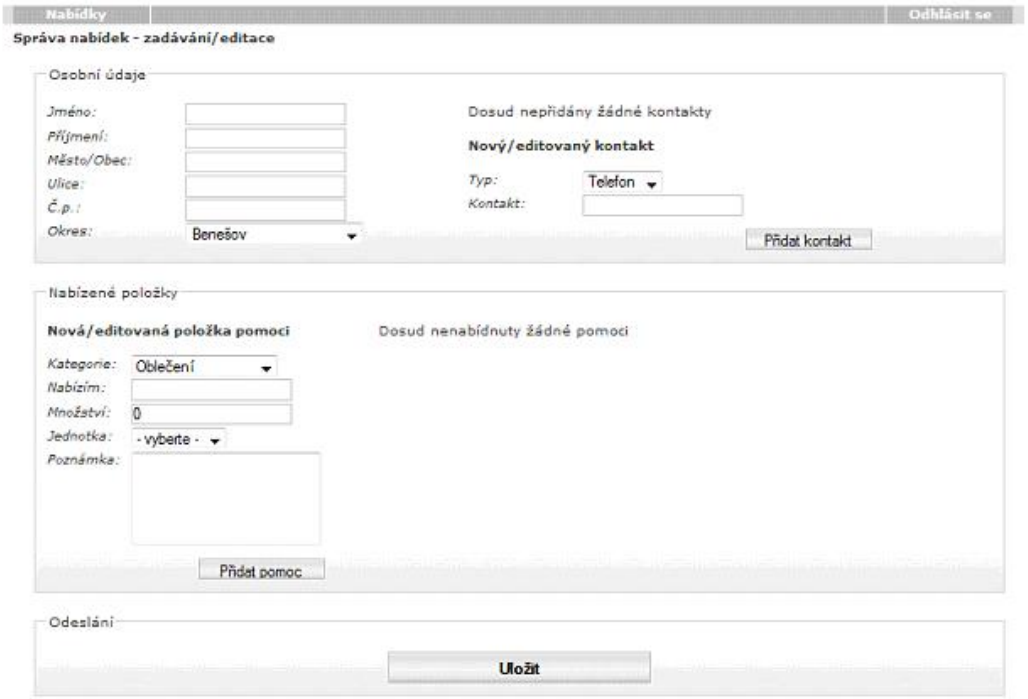
Dashboard panel je komponenta webového uživatelského rozhraní, která poskytuje přehledně informace o stavu projektu, stavu serveru nebo třeba databáze. D*ashboard* je sestaven z takzvaných *viewletů*. *Viewlet* je většinou sestava, která ukazuje stav konkrétní komponenty. Může to být například report nad pracovními položkami, využití přidělené paměti, atd. *Dashobard* panel v RTC umožňuje zobrazovat i *viewlety* z jiných serverů. Předpokladem pro zobrazení těchto *viewletů* je řádně nakonfigurovaná meziserverová komunikace. V závislosti na typu RTC můžeme zakládat a modifikovat karty na panelu. Verze RTC Express-C umožňuje použití pouze jedné karty.



Obr. 28 Dashboard (zdroj: vlastní)

Obrázek *Dashboard* ukazuje, jakým způsobem můžeme zobrazit jednotlivé sestavy. *Viewlety* je možné minimalizovat, zavírat a dokonce i měnit pořadí a pozice. Záleží pouze na uživateli, které informace chce vidět. V neplacené verzi Express-C není možné změny provedené v *Dashboard* uložit.

Hlavní vývoj informačního systému proběhl v *Construction* fázi, v níž byly implementovány zbývající případy užití a definováno grafické rozhraní, jak je zobrazeno na obrázku níže.



Obr. 29 GUI - Správa nabídek (zdroj: vlastní)

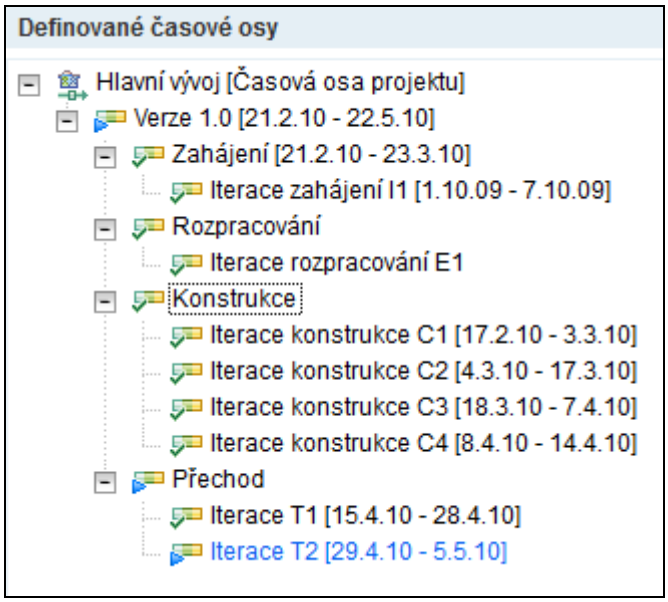
Pravidelně probíhalo i testování klíčových funkcionalit, a to prostřednictvím JUnit testů i uživatelského testování pomocí pluginu Selenium určeného pro prohlížeč Mozilla.

4.3.3 Plánování iterace

Jednotlivé iterace vycházely z projektového plánu, který je možné nalézt v příloze této práce. Na doporučení lektorů jsme si zvolili iterace dvoutýdenní. Tím jsme docílili toho, že naše iterace měly jasně ohraničený termín, což nás nutilo volit dosažitelné cíle a zároveň tyto cíle plnit.

Bylo dohodnuto, že přidávání iterací bude probíhat pouze v rámci fáze a to za předpokladu, že se nepodaří splnit stanovené cíle, takže případný přechod do další fáze by nebyl možný.

Informace o tom, kdy probíhala daná iterace, byla v RTC uložena na dvou místech. Jednak to bylo definované v časové ose, jak lze vidět na obrázku *Hlavní časová osa*,



Obr. 30 Hlavní časová osa (zdroj: vlastní)

a duplicitně jsme si tuto informaci ještě vedli na záložce přehled, aby tato informace byla viditelná na první pohled. Výběr aktuální iterace bylo možné provést skrze menu pravým tlačítkem. Pokud nevyvstaly časové konflikty (tzn. všechny pracovní položky byly uzavřené), tak se následující iterace začala zobrazovat jako aktuální na panelu *Dashboard*.

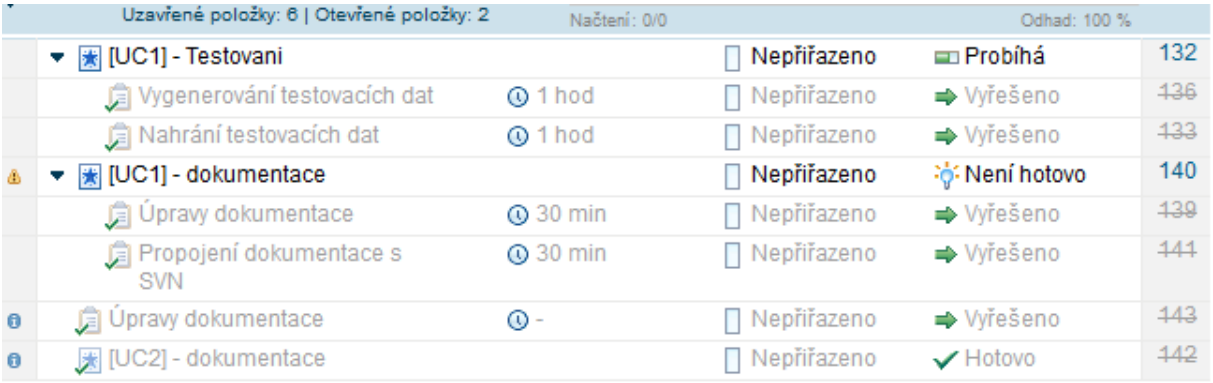
4.3.4 Cíle iterace

Cíle iterace byly voleny vždy s ohledem na výstup z předchozí iterace. Pokud některý cíl nebyl splněn v předchozí iteraci, byl převeden do následující iterace s tím, že bylo vyhodnoceno, z jakého důvodu nebyl splněn a co je potřeba udělat pro to, aby se situace příště neopakovala (angl. lessons learnt). Cíle byly voleny s ohledem na aktuální rizika, takže vypadaly například takto:

- Ladění a testování
- Atribut – přidání poznámky k adrese nabídky
- Nový typ kontaktu (textový atribut)
- Zrušit povinnost zadávat číslo popisné
- Tvorba uživatelské dokumentace (Nabídky, Poptávky, Administrace)
- Tvorba Java dokumentace.

4.3.5 Úkoly

Úkoly byly zaznačeny v RTC jako pracovní položky. V případě, že někdo objevil chybu či nějakou věc, kterou by bylo třeba vyřešit, vytvořil sobě nebo kolegovi novou pracovní položku, ve které zmínil symptomy problému a další detaily, které by mohly pomoci k jeho vyřešení. Řešitel úkolu pak nejprve vyhodnotil, jestli daný problém spadá do jeho kompetencí a až následně se pustil do jeho řešení, přičemž vyplnil odhad práce, aby kolegové a vedoucí mohli vědět, na kolik jej daný úkol vytíží. Jak takový seznam pracovních položek vypadal, vidíme na obrázku *Pracovní položky uživatele.*



Obr. 31 Pracovní položky uživatele (zdroj: vlastní)

V řádcích můžeme vidět aktuální status položek včetně odhadu práce a priority (na obrázku „Nepřiřazeno“). Kliknutím na název položky nebo její číslo se dostaneme na detail této položky.

Úkoly jsme si značili krátce, ale výstižně:

- Petr: Optimalizace, testování, tvorba Javadoc dokumentace
- Jirka: Implementovat „Zrušit povinnost zadávat ČP“

  Atribut poznámka, adresa nabídky, nový typ kontaktu, tvorba Javadoc dokumentace
- Martin: Tvorba uživatelské dokumentace, testování
- Veronika: Tvorba uživatelské dokumentace, testování.

4.3.6 Evaluace a zhodnocení iterace

Evaluační kritéria vycházejí z vytyčených cílů dané iterace. Cíle je třeba definovat jasně a srozumitelně, mají-li být následně jednoduše vyhodnoceny:

- Všechny *unitové* testy prošly.
- Všechny funkční testy prošly.
- Kompletní dokumentace.
- Ověření a zhodnocení buildu zákazníkem.

Pokud jsme si odpověděli na všechny evaluační kritéria kladně, přešli jsme do další iterace bez nutnosti změny termínu splnění.

Hodnocení iterací probíhalo vždy na jejich konci. Společně jsme analyzovali, zda byly plány iterace splněny či nesplněny a do jaké míry. Z tabulky *Zhodnocení T2* můžeme vidět, jak byla evaluační kritéria pro tuto iteraci vyhodnocena.

Kritérium	Status
Všechny unitové testy prošly	OK
Všechny funkční testy prošly	OK
Ověření a zhodnocení buildu zákazníkem	OK
Zkompletování dokumentace	OK

Tab. 9 Zhodnocení T2 (zdroj: vlastní)

Součástí hodnocení iterace bylo i *lessons learnt*, tedy zamyšlení se nad tím, co by se mohlo příště provést jinak, jaký proces by se mohl zefektivnit. Tyto informace jsme si značili do přehledu iterace.

4.3.7 Rozšířené nastavení

V závislosti na typu verze RTC máme k dispozici rozšířené nastavení. Možností, které lze nastavit, je opravdu hodně. Ze začátku jsem ponechal nastavení ve výchozím stavu. Později jsem modifikoval tato nastavení:

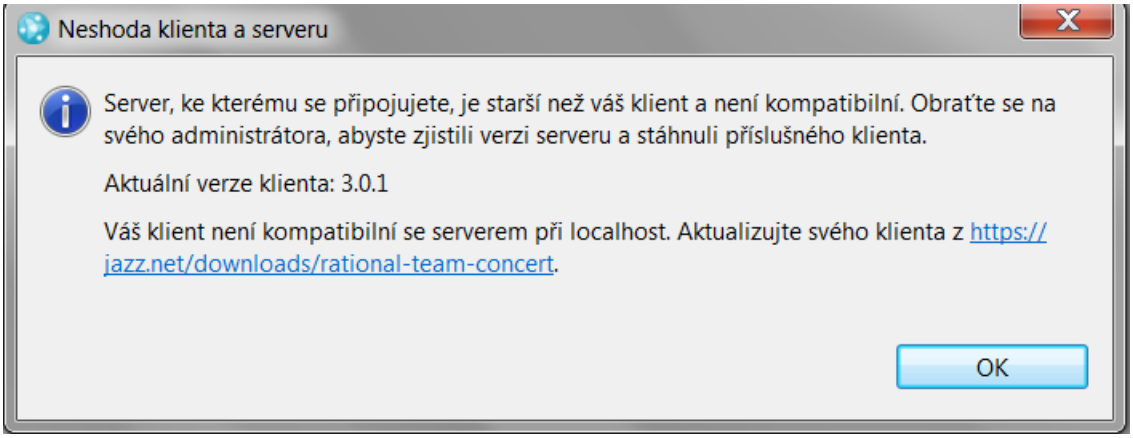
- konfigurace http a https portů
- nastavení SMTP serveru pro zasílání pošty
- povolení e-mailových oznámení
- veřejný identifikátor URI

Nejednalo se o klíčové funkcionality. HTTP port jsem oproti výchozímu portu 80 zvolil, protože tento port měl vyhrazen jeden z doprovodných programů, který jsem používal pro vzdálenou správu serveru.

4.3.8 Upgrade RTC

Jelikož během vývoje informačního systému vyšla nová verze Rational Team Concert 3.0.1.1, rozhodl jsem se, že zkusit propojit RTC klienta verze 3 s RTC serverem verze 2.

Z oficiálních stránek jsem stáhnul klienta verze 3.0.1.1. Ukázalo se, že tato kombinace není možná, protože novější klient není kompatibilní se starší verzí, jak je vidno z obrázku N*eshoda klienta a serveru.*



Obr. 32 Neshoda klienta a serveru (zdroj: vlastní)

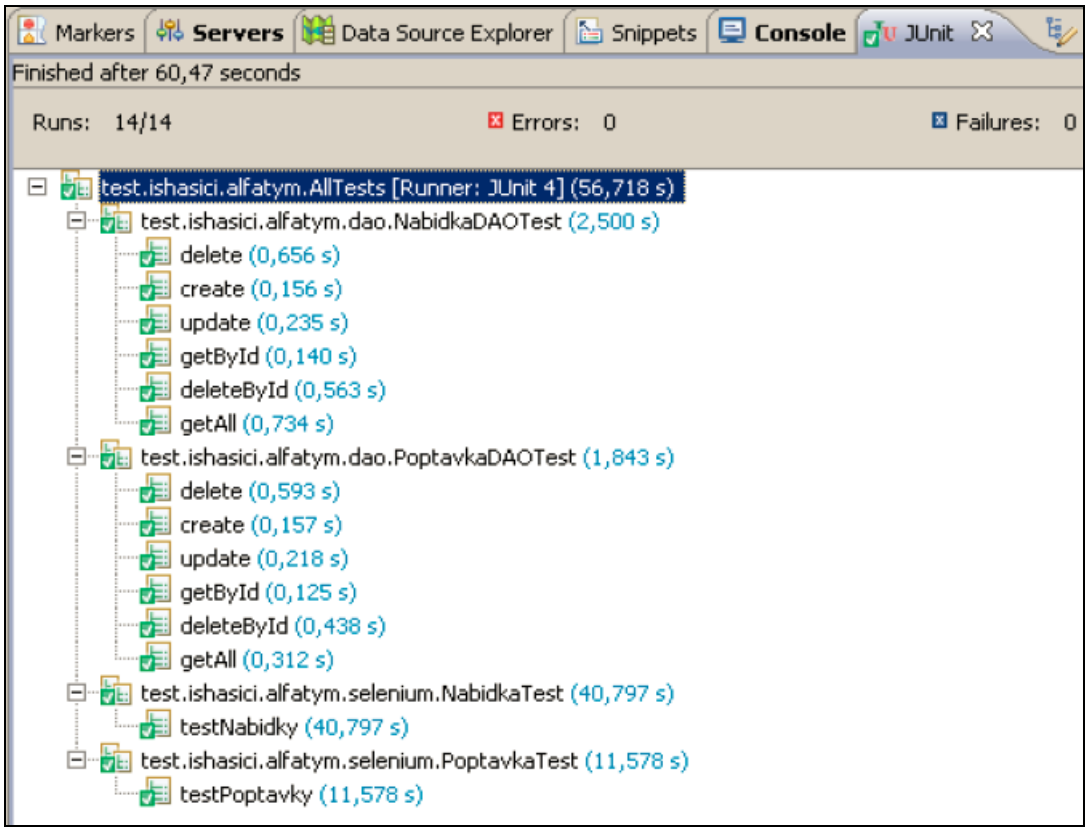
Stále jsme však nemohli používat některé předdefinované reporty. Jednoduše jsme je v Artefaktech týmu neviděli. Obrátil jsem se s žádostí o pomoc i na uživatele a vývojáře z oficiálního fóra, ale ani oni neznali důvod a naváděli mě k řešení, které bylo mylné. Nakonec jsem zkusil použít jinou verzi RTC. Místo verze Express-C pro 10 vývojářů jsem stáhnul 60denní trial ve verzi Standard. Rázem byly jak v tenkém klientovi, tak i v tlustém přístupné všechny reporty. Bohužel tuto skutečnost se na stránkách, kde si vybíráte, kterou verzi chcete stáhnout, nedočtete. Najdete ji až v oficiální nápovědě. Tabulku, která obsahuje informace o omezení jednotlivých verzích, je možné nalézt v příloze této práce.

Nyní jsem mohl používat Reporty k analýze efektivity práce týmu a jiných aspektů, ale na serveru bez dat. Zkoušel jsem migrovat data z nižší verze do vyšší verze podle návodu z oficiálních stránek.[13] K požadovanému cíli to ovšem nevedlo, protože se objevila blíže nespecifikovaná chyba.

Pokusil jsem se tedy najít alternativní způsob řešení. Z oficiálních stránek jsem stáhnul verzi Standard Server 2.0.0.2 iFix7, která obsahovala podporu reportů. Mé pozornosti neunikla skutečnost, že obě instalace obsahují adresář *repositoryDB*. Udělal jsem si zálohu a pak odstranil zmíněný adresář ze Standard verze a nahradil jej adresářem z RTC verze Express-C. Toto nestandardní řešení zafungovalo bezchybně. V novém

13 http://publib.boulder.ibm.com/infocenter/clmhelp/v3r0m1/index.jsp?topic=%2Fcom.ibm.jazz.install.doc%2Ftopics%2Fc_understand_upgrade.html [online] 2010-09-09.

serveru bylo vše, co jsem očekával – projekt, modifikovaný panel dashboard, tak jak jsem jej sestavil, pracovní položky i všichni uživatelé. V produkčním prostředí by takovéto řešení mohlo mít nepříjemné následky, ale pro mé testovací účely jsem si to se zálohovaným původním serverem mohl dovolit a používat tak funkce dostupné v této verzi.



Obr. 33 Výsledky JUnit Testů (zdroj: vlastní)

4.3.9 Shrnutí

Třetí fáze představovala implementaci 80% případů užití a dokončení use case, které se začaly implementovat v předchozí fázi *Elaboration*. Aktivně jsme používali RTC, které bylo propojeno s vývojovým prostředím Eclipse. Burn-up graf (který je popsán v kapitole 5.1.2) nám pomáhal rovnoměrně rozložit během iterací a díky jeho pravidelnému zhodnocení jsem mohl korigovat využití jednotlivých členů týmu či zpřesňovat odhad práce v příští iteraci. Milník Initial Operational Capability Milestone (IOC) byl splněn ve chvíli, kdy produkt byl připraven k nasazení a beta testování.

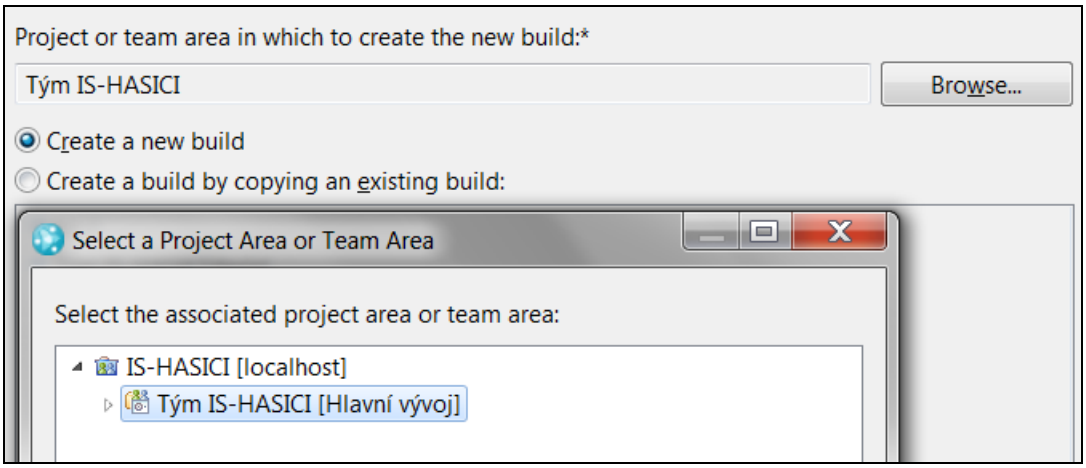
4.4 Transition fáze

V této kapitole zabývající se poslední fází projektu, která obnáší aplikování a doručení výsledného produktu zákazníkovi, je popsáno buildování aplikace a závěrečné testování.

4.4.1 T1: Finální sestavení

Důležitou činností je vytvoření buildu. Než se vytvoří samotný build, je třeba učinit několik nastavení. V nabídce *Team Artifacts* se definují tzv. pracovní proudy, které se používají pro přenos změn. Příchozí proud vytvoří lokální kopii změn, které provedli jiní členové týmu. Analogicky odchozím proudem přenášíme námi provedené změny do společného úložiště.

V tlustém klientovi skrze File->New->Team->Build definition vybereme úložiště, kde se nachází soubory k sestavení. Build lze vytvořit nový nebo na základě již existujícího buildu.

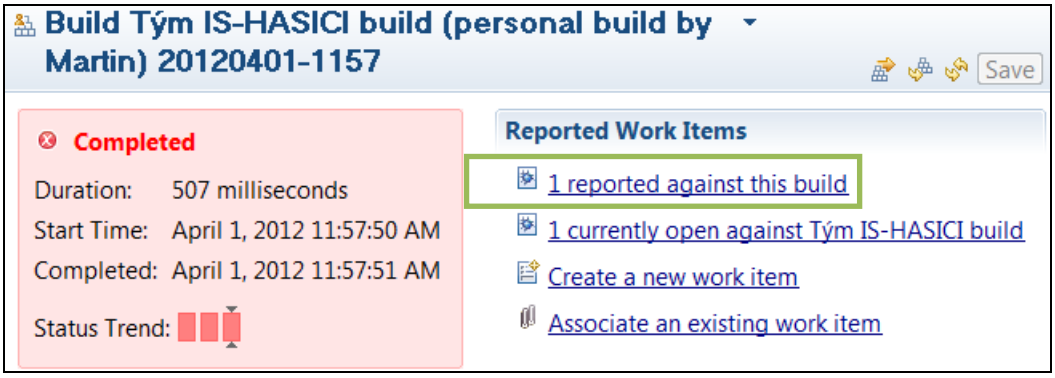


Obr. 34 Vytvoření buildu (zdroj: vlastní)

Následně vybíráme tzv. build template, což je šablona pro některý z buildovacích nástrojů (Ant, Maven ad.).

Předpokladem pro práci s buildy je spuštění enginu s parametrem, který se nachází v adresáři *instalační_adresář/buildsystem/buildengine/eclipse/jbe.* Konkrétně v mém případě jsem spouštěl engine v příkazové řádce v tomto znění:

jbe -repository http://localhost:9080/jazz -userId Martin -pass Martin -engineId MEngine. Pak už stačí v definici buildu naplánovat, kdy se má build zhotovit a výsledkem může být situace podobná obrázku *Build*.



Obr. 35 Build (zdroj: vlastní)

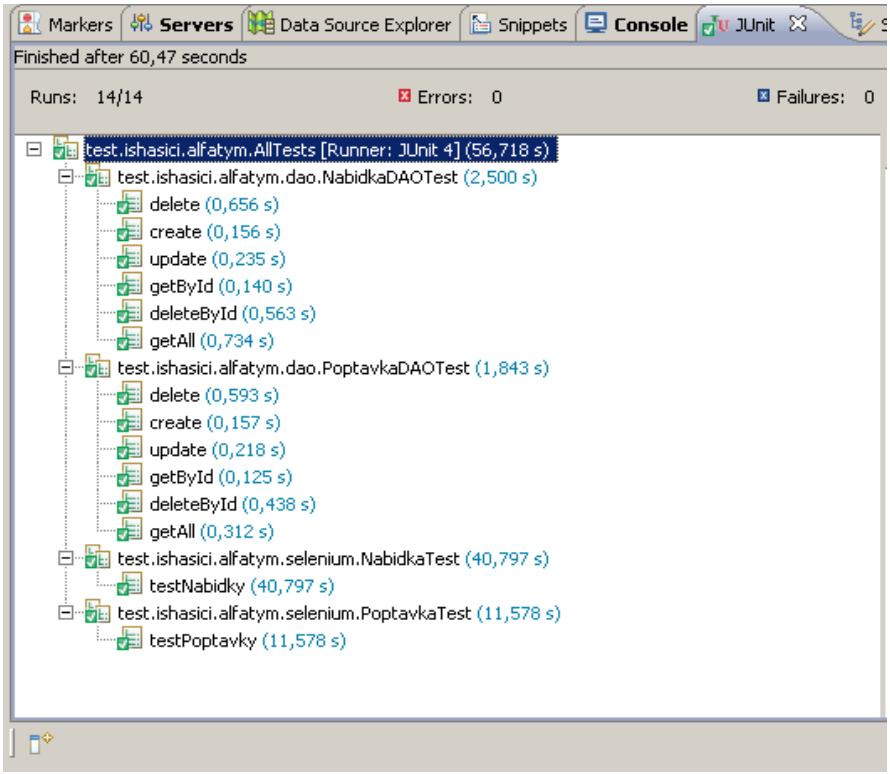
Na tomto konkrétním případě vidíme, že vytvoření buildu selhalo a je k němu vytvořená jedna pracovní položka, což je signalizace, že někdo již tento problém řeší. Ihned máme přehled o aktuálním stavu. Rozkliknutím logu pak dostaneme detailní informace, proč se akce nezdařila.

4.4.2 T2: Testy

Client Eclipse nativně obsahuje testovací framework JUnit. Pravidelně a průběžně jsme testovali klíčové funkcionality v maximální možné míře, abychom odhalili včas co nejvíce chyb, což v praxi znamená snížení nákladů na jejich opravení. Testování jsem zautomatizoval do té míry, že skrze jeden hlavní test se jedním klikem spustily definované testy.

Pro úplnost uvádím, že uživatelské testy jsme zaznamenávali a automatizovali pomocí nástroje Selenium[14].

[14] Selenium je dostupné z http://seleniumhq.org/



Obr. 36 Závěrečné JUnit testy (zdroj: vlastní)

Jak vypadal scénář testování prohlížení nabídky, je patrné z tabulky *Testovací scénář.*

	Prohlížení nabídky
Introduction	Vložení nabídky do systému
Use Case Description	Vložení nabídek do IS je nutné z důvodu možných pozdějších úprav a hlavně párování
Pre-condition	Uživatel musí být přihlášen s právem "správce nabídek"
Flow of event	Tento bod pro zjednodušení považuji za basic flow, proto nevyplňuji
Basic flow	Postupným vkládáním informací uživatel vyplňuje formulář, který je při odeslání zkontrolován bez chyb a jeho obsah je uložen do databáze. V případě špatně zadaných dat (nesprávný formát, nevyplněno) bude uživatel vyzván k opravě údajů)
Alternative flows	Pokud nastane výpadek úložiště či vzniknou problémy s odesláním dat, je uživatel obeznámen s touto situací
Post-codition:	Uživateli je zobrazen čistý formulář pro další vkládání.
References	Google Docs - ucSpravaNabidek, skripta ROPR1 str. 31

Tab. 10 Testovací scénář (zdroj: vlastní)

4.4.3 Shrnutí

Posledním milníkem je tzv. Product Release Milestone (PRM), který završuje vývojový cyklus systému. Docílíme jej předáním systému zákazníkovi, proškolením klíčových uživatelů a tím, že zákazník potvrdí, že všechny klíčové funkcionality a požadavky dané na začátku a v průběhu vývoje jsou řádně implementovány.

5 MĚŘENÍ UKAZATELŮ PROJEKTU

Nejen projektoví manažeři a vedoucí pracovníci potřebují měřitelné výstupy, které jim poskytnou informace o tom, na kolik procent jsou jednotliví pracovníci či vývojové týmy vytížení, kolik procent incidentů má nejvyšší prioritu či třeba kolik pracovních položek se denně v průměru vytvoří nebo uzavře. RTC sbírá informace o pracovních položkách v několika sestavách. Níže uvádím ty nejzajímavější z nich:

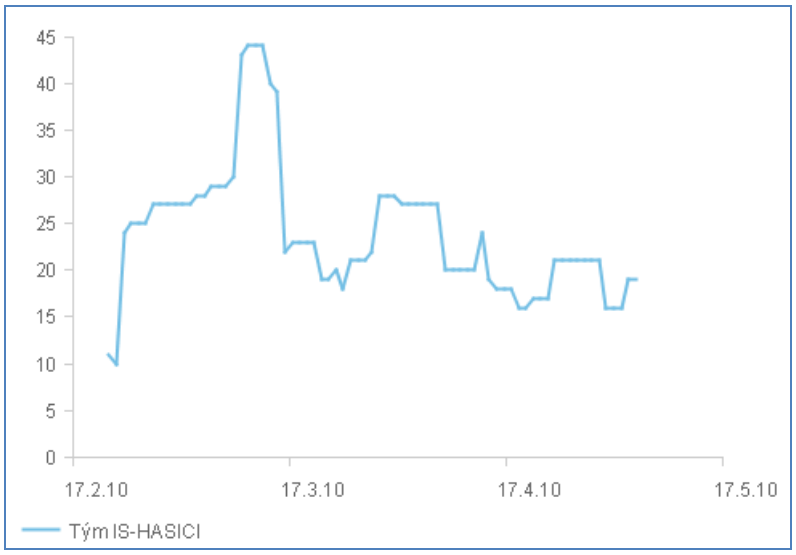
- Uzavřené pracovní položky podle priority
- Denně uzavřené pracovní položky
- Otevřené pracovní položky podle priority
- Otevřené pracovní položky podle typu
- Otevřené versus uzavřené pracovní položky
- Odložené pracovní položky
- Porovnání pracovních položek

Domnívám se, že jejich názvy nám jasně říkají, co mají dané sestavy zobrazovat. Pro ilustraci jsem si do následující kapitoly vybral sestavu *otevřené pracovní položky*.

5.1.1 Otevřené pracovní položky

Podkladem pro následující graf byly pracovní položky ze všech iterací, během kterých bylo RTC používáno.

Z obrázku *Otevřené pracovní položky* lze vyčíst, že nejvíce otevřených pracovních položek bylo během iterace **C2**, která probíhala 4. 3. 2010 až 17. 3. 2010. Bohužel ostatní týmy nepoužívaly prostředí RTC a proto na této sestavě, nevidíme více křivek.



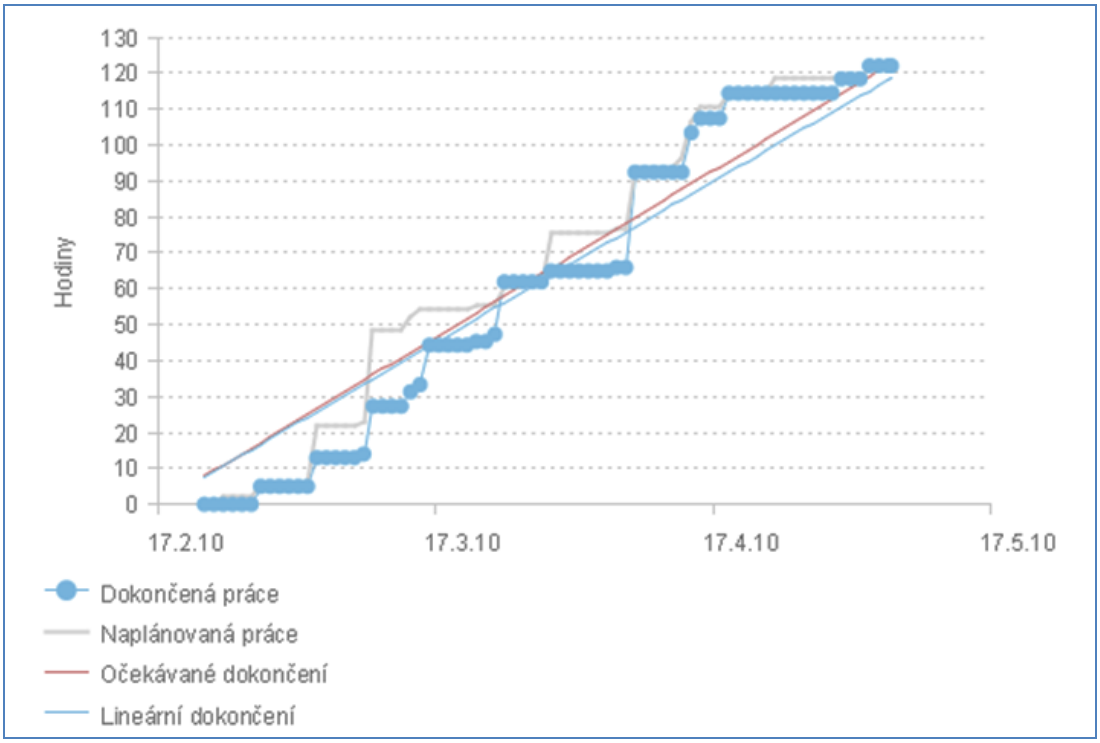
Obr. 37 Otevřené pracovní položky (zdroj: vlastní)

Kdyby zde byly i křivky zbývajících dvou týmů, mohli bychom dostat informaci o tom, který tým má přidělených nejvíce pracovních položek, což by nás navedlo k tomu, která oblast či komponenta je nejvíce problematická nebo má nejvíce úkolů k řešení.

5.1.2 Burn-up graf

Následující sestavu považuji za jeden z nejdůležitějších ukazatelů, který jsem z předdefinovaných sestav získat. *Burn-up graf* zobrazuje dokončenou práce vzhledem k reálnému času, který byl zapotřebí k vyřešení. Pro vygenerování takovéto sestavy je potřeba poctivě vyplňovat především tyto údaje: odhad práce, jak dlouho bude trvat dokončení pracovní položky a čas, tedy jak dlouho skutečně zabralo řešení dané pracovní položky. Elementárním předpokladem je, že vyplňujeme čas zahájení a čas dokončení jednotlivých iterací. Pokud bychom tak neučinili, dostali bychom značně zkreslené výsledky.

V ideálním případě graf zobrazuje s blížícím se koncem iterace či sprintu trend blížící se k plánovanému počtu hodin práce. V našem případě je vidět, že jsme poměrně dobře určovali odhad práce a reálnou práci.



Obr. 38 Burn-up graf (zdroj: vlastní)

Pravidelná kontrola této sestavy nám dávala zpětnou vazbu o tom, jak se dařilo plnit naplánované úkoly a odhadovat práci týmu.

5.1.3 Rychlost

V následujícím případě se snažím potvrdit hypotézu, že vývoj s použitím jednoho integrovaného nástroje je rychlejší než vývoj s použitím více (například tří) nástrojů. Jedná se o nekvalifikovaný statistický odhad, který vychází ze zkušenosti, kterou jsme získali během projektu.

Vycházejme z těchto předpokladů:

- Jedno kliknutí (provázání) trvá dvě sekundy. Mám na mysli akci, kdy například přecházíme (v rámci jednoho nástroje) z monitorování pracovních položek do správy konfigurací. Jelikož tyto části jsou propojené, není třeba se znovu přihlašovat, přepínat se do jiného okna ani explicitně spouštět nějakou službu
- Při použití tří různých nástrojů výše zmíněné aktivity (tedy přihlášení se do jiného toolu, nastavení obrazovky a spuštění) zabere přibližně jednu minutu

Počet použitých nástrojů	1	3
Doba akce [s]	2	30
Výsledná rychlost	30 /2 = 15krát rychlejší	

Tab. 11 Jeden nástroj vs. tři nástroje (zdroj: vlastní)

Tabulka *Jeden nástroj vs. tři nástroje* nám ukazuje, že při použití jednoho nástroje mohou být některé akce až patnáctkrát rychlejší než při kombinaci několika různých nástrojů.

5.1.4 Plýtvání

V následujícím příkladu se budu snažit ukázat, kolik času může ušetřit použití jednoho nástroje místo kombinace několika různých nástrojů. Také v tomto případě se jedná o nekvalifikovaný statistický odhad vycházející z vlastní zkušenosti a tohoto předpokladu:

- Za jednu hodinu se vývojář desetkrát přepne mezi nástroji

Tabulka *Plýtvání* znázorňuje velice zajímavé výsledky. Premisou je, že přepnutí mezi nástroji zabere 30 sekund, a že vývojář se mezi nimi přepne průměrně 10krát za hodinu. Vezmeme-li v úvahu, že v jedné iteraci se kóduje 10 hodin, vyjde mi, že při jednom nástroji je neproduktivní čas dvě stě sekund, kdežto při použití tří různých nástrojů je to již tři tisíce sekund.

Počet použitých nástrojů	1	3
Doba přepnutí [s]	2	30
Doba 10ti přepnutí za 1 hodinu [s]	20	300
Vyplýtvaný čas za 10 hodin [s]	200	3000
Vyplýtvaný čas za 10 iterací [h]	0,5	8,33

Tab. 12 Plýtvání (zdroj: vlastní)

V projektu jsme celkem absolvovali patnáct iterací. Odhaduji, že přibližně v deseti z nich docházelo ke zmíněnému přepínání mezi nástroji, respektive programování a buildování aplikace. Docházím k závěru, že téměř jednu celou iteraci (přes osm hodin času) zabralo přepínání mezi nástroji.

Domnívám se, že toto je důležitý argument, který může pomoci při rozhodování, zda zvolit variantu jednoho nebo více provázaných nástrojů.

5.2 Zhodnocení projektu bez RTC

Projekt prošel fází *Inception* a částí fáze *Elaboration* bez použití Rational Team Concert. Během tohoto období jsme veškeré informace schraňovali v úložišti Google Docs. Google Docs je v prvé řadě úložiště, které umožňuje sdílení složek či souborů a jejich vícenásobné editování v reálném čase. Není však zde nic, co by se podobalo panelu *Dashboard* z RTC. Nebyla možnost vidět na jedné obrazovce více informací či nějaké přehledy. Veškeré informace se přenášely *ručně*, nebyla zde žádná interakce.

Úkoly, lépe řečeno pracovní položky, nebyly nikde udržovány. Zpětně si uvědomuji, že jsme se mohli alespoň pokusit udržovat je v nějaké strukturované formě pomocí tabulky. I to by jistě mělo nějaký přínos. My jsme je ovšem značili do sešitů a na ICQ, takže nikdo pořádně nevěděl, co další člen týmu dělá, neboť neměl šanci si nikde seznam existujících úloh prohlédnout.

Jako nedomyšlené hodnotím naše popisky, když jsme posílali do repository nové revize kódu. Bylo by vhodnější, kdybychom si zavedli nějakou mnemotechnickou pomůcku – např. uvádět do textu číslo pracovní položky a pro kterou verzi je to relevantní. Kdybychom používali repository od RTC, tento problém bychom neměli. Viděli bychom totiž spojitost mezi kódem a pracovní položkou.

5.3 Zhodnocení projektu s RTC

Ze začátku bylo dosti obtížné se sžít s tak komplexním nástrojem, jako je RTC. Upřímně řečeno to, co se mi na začátku projektu zdálo těžkopádné a neohrabané, ukázalo se jako velmi praktické. Některé věci však zůstaly záporem i nadále. Například každé připojení skrze webový prohlížeč znamenalo přihlašování s bezpečnostním certifikátem – bylo nutné opakovaně zadávat své přihlašovací jméno a heslo. (Po skončení projektu jsem na oficiálních stránkách vyčetl, že tento bug je opraven v jedné z později vydaných verzí).

Nedostatky také vidím v záložce *Overview*. Chyběla zde podpora chytrého textu, vkládání souborů či obrázků a rovněž informace o tom, kdo je vlastníkem této záložky či kdo ji naposledy modifikoval.

Bylo až překvapující, že značné množství akcí nelze vrátit zpět. Například vytvoříte-li nový projet a stane se vám, že omylem zvolíte špatnou šablonu, tento krok již nemůžete vrátit. Maximálně můžete daný projekt „archivovat“. Bohužel toto ve verzi, kterou jsem požíval (tzn. maximálně deset developerů), toto nelze provést, takže jsem nemohl zjistit, co přesně archivace znamená.

Kniha Provozujte IT jinak [PK11] mne mimo jiné přivedla na myšlenku, že byť má RUP, potažmo OpenUP v odlehčené verzi, důkladně propracovaný systém rolí a oprávnění, nedává nám odpověď na jednu z nejdůležitějších otázek – za co je každý člen týmu zodpovědný, jaké jsou jeho povinnosti. Jistě, můžeme definovat role či matici odpovědnosti s informacemi, kdo je schopen daný problém vyřešit. Dokážu si ale představit, že by se v RTC nabízela na základě toho, kdo již řešil podobný problém (např. chybu v třídě) či na základě klíčových slov jakási nápověda s vhodnými řešiteli. Tato vlastnost by však měla využití v týmech čítajících mnohem více než pět lidí.

Zamyšlením se také nad výstupy z iterací se dostávám k tomu, že RTC automatizuje vývoj, ale rozhodně ne ve všech aspektech. Procesní frameworky počítají s tím, že máme milníky, ale v prostředí RTC nikde nevidíme, že bychom je mohli definovat či je propojit s konkrétními pracovními položkami, jejichž splněním dosáhneme zvolených milníků. Máme sice k dispozici velké množství reportů týkajících se pracovních položek, ale žádný, který by je dokázal propojit s milníky.

Jisté rezervy vidím v tom, že RTC neumožňoval efektivně monitorovat, jestli jsou milníky mezi jednotlivými fázemi splněny nebo ne. Ze začátku jsem se domníval, že RTC je vhodné pouze pro velké týmy, čítající dvacet a více lidí. V průběhu projektu se však ukázalo, že tomu tak není. Vycházel jsem z předpokladu, že když dva lidé dělají na stejné věci, je efektivnější, aby spolu komunikovali ústně. Tato úvaha však nebyla správná. Stačilo, aby dva lidé dělali na jedné věci a už se začaly objevovat problémy typu „Je ten dokument aktuální? Kterou část zpracováváš, kterou mám zpracovat?“ V situaci, kdy dva a více vývojářů či testerů nesedí ve stejné místnosti a nemusí pracovat ve stejnou dobu, se ukázalo, že podrobné zachycení činností, které se odehrávají a mají odehrávat, je nezbytné. Nikdy totiž nemůžete vědět, kdy vám někdo z týmu vypadne a v takovém případě by se mohlo stát, že to, co jeden člen týmu naprogramoval nebo otestoval, zůstane nepředané, protože k tomu bude chybět snadno dostupná dokumentace.

Jasně se ukázalo, že pokud se informace nespravovaly skrze RTC (respektive jeden CASE nástroj), ale duplikovaly se do jiných dokumentů umístěných na Google Docs, vznikaly z toho zmatky, které znamenaly časové prodlevy a neefektivitu. V období, kdy jsme neměli připravené RTC jsme zkoušeli komunikaci urychlit tím, že jsme se dorozumívali skrze instant messaging (ICQ), ale postrádalo to dlouhodobý efekt. Nebylo totiž možné, respektive pouze s velkou náročností, vysledovat, jaké změny se udály, kdy, pro co byly relevantní. Komunikace přes instant messaging byla útržkovitá a nepřístupná jiným než aktivním účastníkům diskuse.

5.4 Vlastnosti nepoužité při vývoji

Ne všechny funkcionality, které Rational Team Concert umožňuje, byly během projektu použity. Níže zmiňuji, o které konkrétní funkcionality se jednalo a uvádím i důvody, proč tomu tak bylo.

Source Content management

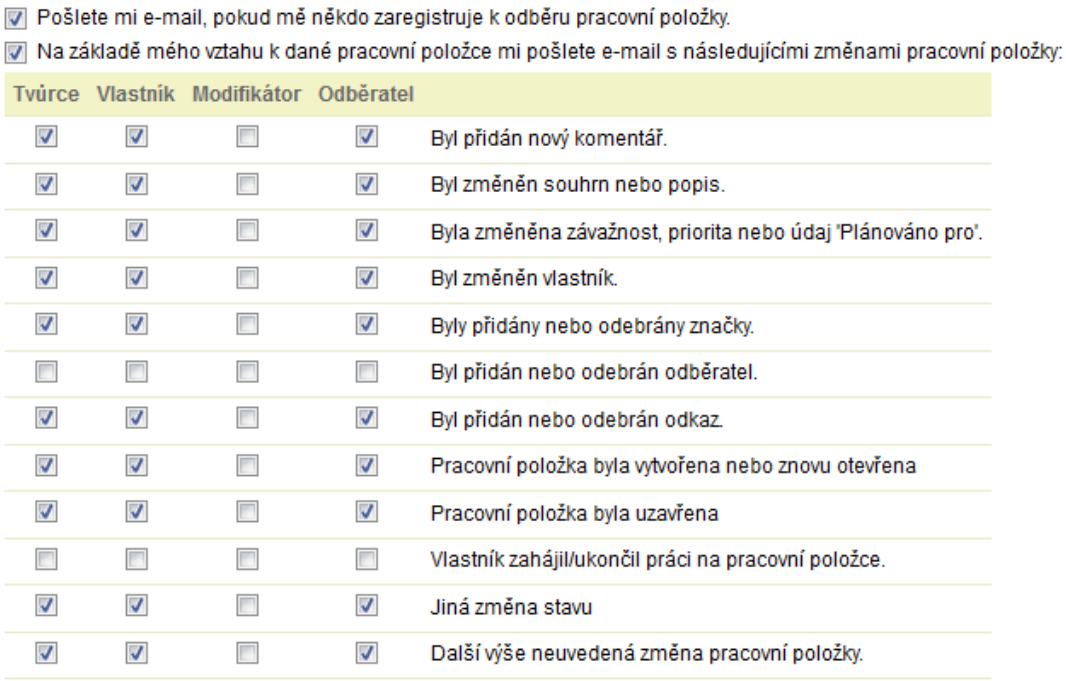
Správa zdrojových kódu (SCM) byla prováděna mimo prostředí RTC. Důsledkem tohoto řešení bylo, že jsme nemohli vidět spárování pracovní položky přímo s částí kódu. Přišli jsme tím o značný kus automatizace, protože jsme museli explicitně vypsat, o jakou třídu či metodu, kde je potřeba učinit změnu, se jednalo. Ještě citelnějším nedostatkem však bylo, že jsme přišli o možnost přiřadit pracovní položku k určité verzi třídy. Jinými slovy bylo nutné ručně vypsat, ve kterém *buildu* nebo kterého *commitu* se daná pracovní položka týkala. V angličtině je tato činnost známá pod pojmem *tracebility*.

Týmové proudy

Proud, angl. stream znamená v tomto významu přenos kódu do úložiště. Klasickým workflow je, že poté, co vytvoříme nebo modifikujeme (a otestujeme) daný zdrojový kód, jej můžeme poslat do repository. Proudy mohou být příchozí (změny, které vývojář přijímá) a odchozí, což jsou ty změny, které jdou od něj do repository.

E-maily

RTC umožňuje zasílat nejrůznější informace na e-mail. Protože jsme však neměli SMTP server k dispozici, tuto možnost jsme nevyužili. Z obrázku *Možnosti e-mailu* vidíme, při jaké akci můžeme dostat notifikaci.



Obr. 39 Možnosti e-mailu (zdroj: vlastní)

Důvodů, proč nebyly využity všechny vlastnosti, bylo několik. Jedním z nich bylo to, že ostatní týmy, pracovaly bez RTC a proto jsme používali všem dostupnou repository třetí strany. Neobjevil jsem způsob, jak bychom mohli pracovat se dvěma různými úložišti naráz, respektive jak docílit toho, že bychom používali dvě repository – jednu v RTC, kde by se v případě incidentů mohl párovat kód s pracovními položkami a druhou, globální, se kterou pracovaly všechny týmy.

Zpětně jsem zjistil, že by bylo možné používat v prostředí RTC jinou, než nativní repository. V rámci zachování stability vývoje a tomu, aby se nezpozdila dodávka systému, jsem od zrealizování tohoto řešení upustil. Každopádně postup, jak docílit použití jiné repository, již byl zdokumentován.[LV09]

Použití Rational Team Concert nám ukázalo, že agilně vyvíjet je pro nezkušené týmy velice náročné. Integrované vývojové prostředí napomáhá uzavíráním iterací, automatizací buildů a jinými mechanismy překonat nejrůznější útrapy, a neodklonit se tak od agilních zásad a principů.

ZÁVĚR

Cílem této diplomové práce bylo ukázat a změřit, jak integrovaný nástroj pomáhá zefektivnit vývoj softwaru. Vytvořil jsem vývojový proces, podle kterého byl vyvinut v prostředí integrovaného nástroje *Systém pro koordinaci humanitární pomoci*. Na základě zkušeností a výstupů z tohoto projektu byly v předchozích kapitolách zhodnoceny dopady použití integrovaného nástroje při vývoji software.

Nekvalifikovaným statistickým odhadem jsem dokázal, že použití jednoho integrovaného nástroje může znamenat až trojnásobnou úsporu času v některých činnostech v porovnání s kombinací tří různých nástrojů. Z hlediska vývoje softwaru to pak znamená, že pokud používáme kombinaci několika nástrojů, tak téměř jedna desetina doby projektu je využita neefektivně.

Integrovaná prostředí, jako je IBM Rational Team Concert, ukazují cestu pro moderní vývoj informačních systémů. I pro malé týmy jsou k dispozici all-in-one produkty pro podporu vývoje informačních systémů. Velkou výhodou je, že není nutné dodatečně customizovat nástroje pro správu konfigurací, správu položek či zdrojového kódu. Mimo komfort, který tato ucelené řešení přinášejí, je jedním z hlavních přínosů automatizace vývoje. Komunikace v týmu se stává velice efektivní díky pracovním položkám, které je možné propojit s příslušným testovacím scénářem či dokonce s kódem, ve kterém se chyba projevuje. Na základě nekvalifikovaných statistických odhadů bylo dokázáno, že v některých ohledech mohou integrované prostředí ušetřit desetinu nákladů.

Automatické kontroly zabraňují poslat ze své lokální stanice do vzdáleného úložiště kód, který by obsahoval chyby či vytvořit nekonsistentní build. Díky automatickým metrikám provedených nad databází můžeme přehledně vidět na panelu Dashboard, jaký pokrok jsme za minulý den udělali a v jakém stavu jsou nejrůznější komponenty. V prostředí Rational Team Concert je k dispozici mnoho předdefinovaných sestav, které nám zobrazují důležité údaje v okně internetového prohlížeče nebo v tlustém klientovi. RTC napomáhá v řízení projektu, zadání a zpracování pracovních položek nebo třeba s hlídáním vytíženosti jednotlivých členů týmu, aby byli vytíženi v rozumné míře a tudíž efektivně pracovali.

Byť můžeme Rational Team Concert považovat za poměrně komplexní moderní nástroj, tak existují ještě oblasti, ve kterých by mohlo dojít k zlepšení, kterým by mohla být

například automatizace uživatelských testů nebo monitorování milníků a jejich propojení s pracovními položkami.

SEZNAM POUŽITÉ LITERATURY

[CC03] **Haas A. M. Jonassen .** *Configurationa Management Principles and Practice.* Boston: Addison-Wesley Professional, 2003. 432 s. ISBN 978-0321117663.

[BU04] **Buchalcevová, Alena**. *Metodiky vývoje a údržby informačních systémů*. Praha: Grada Publishing, 2004. 164 s. ISBN 80-247-1075-7.

[IBM09] **IBM.** *Vydání a licence produktu Rational Team Concert verze 2.0.0.2*.[online]. Září 2009. Dostupný z WWW: <http://publib.boulder.ibm.com/infocenter/rtc/v2r0m0/index.jsp?topic=/com.ibm.team.concert.doc/topics/c_rtc-editions.html>.

[IRB1] **Wahli, U., Brown, J., Teinonen, M., Trulsson, L.** *Configuration Management: Clear Case for IBM Rational ClearCase and ClearQuest UCM. IBM Redbook. IBM Corp. 2004.* [online]. Dostupný z WWW: <http://www.redbooks.ibm.com/redbooks/pdfs/sg246399.pdf>.

[JT10] **Jazz Jumpstart Team.** *The Jazz Tutorial.*[online]. Červenec 2010. Dostupný z WWW: <https://jazz.net/library/article/119>.

[KA04] **Kadle Václav.** *Agilní programování.* Brno: Computer Press, 2004. 278 s. ISBN 978-0-321-16609-8.

[KK03] **Kroll Per a Kruchten, Phillipe.** *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Boston: Addison-Wesley Professional, 2003. 416 s. ISBN 978-0-321-16609-8.

[LV09] **Lafreniere David, Valenta Michael**. *Integrating other SCM Systems with Rational Team Concert 2.0*. Jazz Library.[online]. Červen 2009, [cit. 2012-02-26]. Dostupný z WWW: https://jazz.net/library/article/194/>.

[MS12] **Microsoft.** *Visual Studio Team Foundation Server 2010.* [online]. [cit. 2012-03-19]. Dostupný z WWW: <http://www.microsoft.com/cze/msdn/vstudio/2010/team-foundation-server.aspx>

[PK08] **Procházka, Jaroslav a Klimeš, Cyril.** *Informační systémy 1.*Ostrava, 2008.

[PP09] **Poppendieck Mary a Poppendieck Tom.** *Leading Lean Software Development:*

*Results are not the point.*Indiana: RR Donnelly, 2009. 312 s. ISBN 978-0321620705.

[PVK10] Procházka, Jaroslav a Vajgl, Marek a Klimeš, Cyril. *Informační systémy 2.*Ostrava, 2010.

[SD09] Dimitrov, Sonia. *Tip: Installing the Rational Team Concert client into Eclipse 3.5.x,* Jazz Community Site. [online], Prosinec 2009, Dostupný z WWW: <https://jazz.net/library/article/384/>.

[SR12] Šmiřák, Roman. *RTC versus Jira/SVN/CruiseControl*. Jazz Community Site. [online], Duben 2009, [cit. 9. ledna 2012]. Dostupný z WWW: <https://jazz.net/forums/viewtopic.php?t=4383&sid=a89afa7f72c99cb6ad0aee60b2685900>.

[SS10] Schwaber Ken, Sutherland, Jeff. Scrum.org. *Scrum Guides*. [Online], Únor 2010. [cit. 10. ledna 2012]. Dostupný z WWW: <http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20CS.pdf>

[PK11] Procházka Jaroslav, Klimeš Cyril. *Provozujte IT jinak.* Praha: Grada Publishing, 2011. 288 s. ISBN 978-80-247-4137-6.

SEZNAM POUŽITÝCH SYMBOLŮ

CASE	Computer Aided Software Engineering
FTP	File Transfer Protocol
IDE	Integrated Development Environment
ITIL	Information Technology Infrastructure Library
OpenUP	Open Unified Process
RUP	IBM Rational Unified Process
RTC	Rational Team Concert
SCM	Supply Chain Management
SVN	Subversion
SW	Software

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Přiložený DVD nosič obsahuje:

- Cieslar.pdf – diplomová práce
- Zkompilovaný projekt Systém pro koordinaci humanitární pomoci
- IBM Rational Team Server
- Eclipse Client
- Prezentace Systému pro koordinaci humanitární pomoci

Seznam omezení dostupných verzí RTC

Funkce	Express-C	Express	Standard	Enterprise
Maximální počet vývojářů	10	50	250	neomezeno
Agilní plánování	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Správa zdrojovkého kódu	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Správa sestavení	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Sledování pracovních položek	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Možnost vlastní úpravy procesu	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Integrace nástroje pro správu verzí Subversion	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Oprávněí k procesu na základě rolí	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Process level access control	zahrnuto	zahrnuto	zahrnuto	zahrnuto
LDAP authentifikace	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Panely dashboard	omezeno	omezeno	zahrnuto	zahrnuto
Meziprojektové panely dashboard	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Meziúložišťové panely dashboard	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Možnost vlastní úpravy attributů prac. Položek	zahrnuto	zahrnuto	zahrnuto	zahrnuto
ClearCase Importer	zahrnuto	zahrnuto	zahrnuto	zahrnuto
ClearQuest Importer	zahrnuto	zahrnuto	zahrnuto	zahrnuto
Floating licenses available	zahrnuto	zahrnuto	zahrnuto	zahrnuto

Možnost vlastní úpravy a sledu pracovních položek	nezahrnuto	zahrnuto	zahrnuto
Plán zhodnocení rizika		zahrnuto	zahrnuto
Sestavy		zahrnuto	zahrnuto
Role based process permissions		zahrnuto	zahrnuto
ClearCase Synchronizer		zahrnuto	zahrnuto
ClearCase Bridge		zahrnuto	zahrnuto
ClearQuest® Connector		zahrnuto	zahrnuto
Most mezi záznamem ClearQuest a pracovní položkou Jazz		zahrnuto	zahrnuto
LDAP import		zahrnuto	zahrnuto
Vysoká dostupnost	nezahrnuto		zahrnuto

Projektový plán

Fáze	Iterace	Úloha	Časový harmonogram
Inception	I0	Odhalení problému	23.9. - 30.9.2009
		Vytvoření vize	
		Vytvoření RiskListu	
		Popis scénářů	
		Identifikace use case	
		Identifikace a vymezení zúčastněných stran	
	I1	Odsouhlasení plánu projektu zákazníkem	1.10. 2009 - 7.10.2009
		Vytvoření prototypu pro připojení do db	
		Popis scénářů z I1	
	I2	Vkládání nabídky do databáze přes webový formulář	8.10.2009 - 14.10.2009
		Stanovení architektury	
		LCO	10.14.2009

Elaboration	E1	Stanovení architektury	15.10.2009 - 21.10.2009
		Prezentace (vize,UC model, scénářů, arch.)	
	E2	Stanovení architektury	22.10.2009 - 4.11.2009
		Prezentace (iteračních plánů, architektury)	
		Sjednocení UC a scénářů s ostatními týmy	
		UC 1 [BF] - Zadání nabídky do systému	
		Testovací scénář - Zadání nabídky do systému	
	E3	Stanovení architektury	5.11.2009 - 11.11.2009
		Prezentace (iteračních plánů, architektury)	
		UC 1 [BF] - Zadání nabídky do systému	
		Testovací scénář - Zadání nabídky do systému	

	E4	UC 1 [BF] - Zadání nabídky do systému	12.11.20009 - 5.11.2009
		UC 2 [BF] - Zadání poptávky do systému	
		Testovací scénář - Zadání nabídky do systému	
		Testovací scénář - Zadání poptávky do systému	
	E5	UC 1 [BF] - Zadání nabídky do systému	26.11.2009 - 9.12.2009
		UC 2 [BF] - Zadání poptávky do systému	
		Prezentace (architektury, implementace kritických scénářů)	
	E6	Roadmap na Consruction , Transition	10.12.2009 - 16.12.2009
		Prezentace (architektury, implementace kritických scénářů)	
		LCA (snížena rizika, spustitelná a otestovaná architektura	16.12.2009

Construction	C1	UC1 [AF] - Editace nabídky	17.2.2009 - 3.3.2010
		UC1 [AF] - Smazání nabídky	
		UC2 [AF] - Editace poptávky	
		UC2 [AF] - Smazání poptávky	
	C2	UC1 [BF] - Prohlížení nabídek	4.3.2010 - 17.3.2010
		UC2 [BF] - Prohlížení poptávek	
	C3	UC1 [BF] - Prohlížení nabídek	18.3.2010 - 7.4.2010
		UC2 [BF] - Prohlížení poptávek	
		UC1 [AF] - Filtrování nabídek	
		UC2 [AF] - Filtrování poptávek	
		Grafická tvorba, css, tvorba skriptů	
	C4	UC2 [AF] - Filtrování poptávek	8.4.2010 - 14.4.2010
		IOC (beta-release)	4.14.2010

Transition	T1	Příprava produktu k nasazení	15.4.2010 - 28.4.2010
	T2	Příprava produktu k nasazení	29.4.2010 - 5.5.2010