ROČNÍKOVÝ PROJEKT 1, 2

Učební text

JAROSLAV PROCHÁZKA

Ostrava 2010

Verze 3.0

Recenzenti:

Doc. Ing. Cyril Klimeš, CSc.
Mgr. Marek Vajgl

Název: Ročníkový projekt 1, 2
Autor: RNDr. Jaroslav Procházka, Ph.D.
Vydání: třetí, 2010
Počet stran: 71

1 Obsah

2 Úvod

Předmět Ročníkový projekt (jdoucí přes 2 semestry) je zaměřen hlavně na praktickou část vývoje podle RUP a jako takový se ji snaží podpořit, nezbytné pro úspěšné absolvování předmětu jsou teoretické základy dosažené hlavně v předmětu SWENG. Nebudeme se k nim již vracet!!! Jen některé důležité stručně zopakujeme v kontextu.

Některé teoretičtější aspekty budeme paralelně probírat na tutoriálech a přednáškách předmětu *Informační systémy 1*. Rozsah a znalost tohoto textu by vám však k absolvování praktické části (tj. předmětu *Ročníkový projekt*) měla stačit.

Tento studijní text je podkladem pro vaši praktickou práci, abyste chápali možnou formu jednotlivých artefaktů a také co kdy a proč v průběhu životního cyklu vývoje SW vytvářet. V textu se nezaobíráme technologickými specifiky, ale kde je možné ukazujeme aplikace principů OpenUP na konkrétních příkladech, včetně použití technologií.

3 Principy iterativního vývoje

V této kapitole se dozvíte:

- Co jsou to principy RUPu?
- Jaké jsou základní principy RUPu?
- Co tyto principy zdůrazňují?
- Před čím varují?

Po jejím prostudování byste měli být schopni:

- Porozumět základním principům iterativního vývoje řízeného riziky.

Klíčová slova této kapitoly:

Principy RUP, komunikace, motivace, iterativní vývoj, řízení rizik, kvalita.

Doba potřebná ke studiu: 4 hodiny

Průvodce studiem

Kapitola zmiňuje principy iterativního, riziky řízeného vývoje a podrobně se věnuje jejich popisu, včetně představení výhod, vzoru, jak princip aplikovat a anti-vzoru, který říká, čemu se vyvarovat.
Na studium této části si vyhraďte 4 hodiny.

Iterativní vývoj je vystavěn na několika základních principech, které si nyní představíme. Některé z nich již byly nastíněny také v předchozí kapitole, konkrétně se tedy jedná o:

- Snahu atakovat rizika projektu co nejdříve a neustále.
- Ujištění se, že dodáváme zákazníkovi přidanou hodnotu.
- Zaměření na spustitelný software.
- Zapracovat změny v časných fázích projektu.
- Brzké nastínění spustitelné architektury.
- Znovupoužití existujících komponent.
- Úzká spolupráce, všichni jsou jeden tým.
- Kvalita je způsob provádění celého projektu, nejen část (testování).

Tyto body jsou obsaženy také v klíčových principech (key principles) RUPu stejně jako v OpenUP, jedná se o:

a) Přizpůsobte proces potřebám projektu (Adapt the process),
b) Vyvažujte vzájemně si konkurující požadavky všech zúčastněných na projektu (Balance competing stakeholders priorities),
c) Spolupracujte napříč týmy (Collaborate across teams),
d) Demonstrujte hodnotu v několika iteracích (Demonstrate value iteratively),
e) Pracujte s úrovní abstrakce (Elevate the level of abstraction),
f) Zaměřte se na kvalitu (Focus on quality).

Nyní si jednotlivé principy A až F představíme jeden po druhém. Jejich struktura je definována přínosy (benefit) použití daného principu ve vývoji software, dále vzorem (pattern), který zachycuje zkušenosti z úspěšných

projektů, tzv. Best practices a anti-vzorem (anti-pattern), který popisuje nevhodné praktiky, které vedou k výše zmíněným problémům při vývoji SW, a tedy říká, jak věci nedělat. Bližší popis těchto principů lze nalézt v [Kr05], [RE] či přímo ve webové aplikaci RUP.

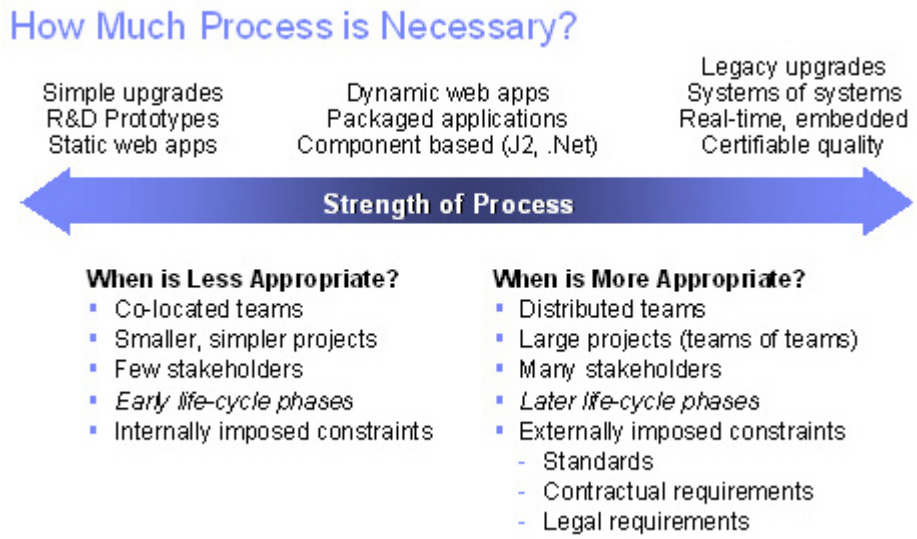
3.1 Adapt the process

Přínos (benefit): Větší efektivnost životního cyklu, lepší komunikace rizik.

Vzor (pattern): Preciznost a formálnost se vyvíjí od nízké po vysokou v průběhu životního cyklu tak, jak jsou odstraněny nejasnosti. Přizpůsobte proces velikosti a distribuci projektového týmu, složitosti aplikace. Neustále vylepšujte Váš proces.

Anti-vzor (anti-pattern): Precizní plány, odhady a postupování podle nich. Mít více procesu (artefaktů, aktivit, rolí) je vždy lepší. Vždy v průběhu životního cyklu udržujte stejnou úroveň formálnosti a preciznosti.

Více procesu ve smyslu tvorby spousty artefaktů, detailní dokumentace není to nejdůležitější, důležitější než toto množství artefaktů či revizí, je správně nastavit proces potřebám projektu. Proces by měl být více formální, disciplinovaný v případě distribuovaného vývoje, použití komplexní technologie či při jeho větší složitosti. V opačném případě, kdy je známá technologie, tým je malý a sedí v jedné místnosti, by měl být proces jednodušší, lehčí (s menším množstvím formalit, artefaktů a detailních specifikací) – situaci znázorňuje následující obrázek.



Obr. 3-1: Formálnost procesu podle potřeb projektu (zdroj: [Kr05])

Dále je třeba držet v každé fázi projektu jiný stupeň formálnosti. V úvodních fázích existuje spousta nejasností a rizik a potřebujeme velké množství kreativity, formální proces by nás svazoval. Naopak v pozdějších fázích, kdy jsou významná rizika odstraněna a vývoj je poměrně předvídatelný, je nutné proces více svázat, aby se vývojáři nezabývali nepodstatnými věcmi. Také je

třeba mít pod kontrolou změny (je třeba proces či disciplína který bude sloužit pro potřeby změnového řízení).

Nepřetržité zlepšování procesu je zaručeno zhodnocením na konci každé iterace, formulováním ponaučení (tzv. lessons learnt) a jejich promítnutí do následných prací a plánů. Z toho také vychází poslední bod tohoto principu jímž je tvorba a úprava plánů. Na začátku, kdy je míra rizika a neznalosti vysoká, vytváříme pouze jakousi roadmap, velmi hrubý plán, podrobně plánujeme kratší časové úseky – iterace, kdy známe více detailů a jsme schopni kratší úseky přesněji odhadovat.

3.2 Balance competing stakeholder priorities

Přínos (benefit): Vývoj aplikace podle potřeb uživatelů, redukce vývoje na zakázku, optimalizace přínosů pro byznys.

Vzor (pattern): Definuj a porozuměj podnikovým procesům a potřebám uživatelů; snaž se porozumět, které komponenty můžeš znovupoužít.

Anti-vzor (anti-pattern): Zachyť precizně veškeré požadavky předtím, než začnou veškeré práce na projektu. Definuj požadavky tak, aby byl celý vývoj zákaznický (bez znovupoužitých komponent).

Jádrem tohoto principu jsou dvě roviny:

- Identifikace klíčových potřeb/požadavků uživatelů.
- Zakázkový vývoj versus znovupoužití existujících komponent.

Většina uživatelů chce od budoucí aplikace přesně takové chování, jaké chtějí oni. Použitím komponent jsme však schopni radikálně snížit cenu vývoje a zkrátit jeho dobu, i když může být chování komponenty mírně odlišné od očekávání uživatelů. Pokud uživatelé či zákazník trvají na všech rysech a vývoji aplikace na zelené louce, pak musí počítat s delší dobou vývoje a také s vyšší cenou.

Jelikož uživatel řídí vývoj a našim cílem je vyvinout aplikaci, která mu přinese užitek, je nutné pochopit problémy a potřeby, které chce zákazník aplikací řešit. Díky pochopení jeho potřeb jsme schopni požadavky prioritizovat a také na základě těch prioritních definovat architekturu.

Anti-vzor říká, že máme precizně a detailně specifikovat požadavky předem, nechat je schválit zadavatelem a pak vyjednávat jejich každou změnu. Další anti-vzor říká, že máme brát v úvahu pouze požadavky nejhlasitějších uživatelů.

3.3 Collaborate across teams

Přínos (benefit): Produktivita týmu, lepší spojení mezi byznysem, vývojem a provozem SW.

Vzor (pattern): Motivuj lidi, aby pracovali nejlépe, jak umějí. Spolupráce mezi jednotlivými funkčními celky, analytik, vývojář a tester pracují dohromady. Ujisti se zda byznys, vývojové a provozní týmy pracují efektivně jako integrovaný celek.

Anti-vzor (anti-pattern): Vychovávej heroické jednotlivce a vybav je mocnými nástroji.

Software je produkován talentovanými a motivovanými lidmi, kteří úzce spolupracují. Při vývoji složitých projektů je třeba spousty lidí s různými dovednostmi, proto je komunikace nezbytným a kritickým faktorem a proto také mluvíme o tzv. „soft skills“. Tyto dovednosti jsou také základem agilního vývoje a proto jsou silně zdůrazněny v Agilním manifestu [AgM].

Prvním krokem pro efektivní spolupráci je vytvořit tzv. samo-řízené týmy (self-managed teams). Takový tým seznámíme s našim cílem, co chceme zákazníkovi doručit a poté mu dáme také odpovědnost, aby mohl rozhodovat o věcech, které konkrétně ovlivňují celkový výsledek. Pokud lidé cítí opravdovou odpovědnost za výsledek, cítí se více motivováni, než v případě, kdy jim úkoly přiděluje nadřízený, navíc bez celkového obrazu.

Jak již bylo řečeno, vývoj software je týmový sport. Iterativní vývoj zdůrazňuje nutnost těsné a neustálé spolupráce jednotlivých rolí / členů týmu. Je třeba zbořit zdi, které jsou po vzoru klasických metodik vystavěny mezi analytiky, vývojáři a testery. Tito lidé spolu pracují v těsném kontaktu v průběhu celého projektu (v průběhu všech fází a jejich iterací). Navíc každý z členů týmu musí znát a rozumět cílům a vizi projektu.

Pro úspěšnou komunikaci je třeba mít adekvátní prostředí, což znamená také automatizované buildy a zavedení kvalitního Configuration Managementu (správy konfigurací) a změnového řízení. Dále také mít týmovou Wiki, kde lze nalézt důležité dokumenty, artefakty, zápisy z porad, vývojový proces apod. Neméně podstatné je také integrované vývojové prostředí (IDE, např. Elipse, Visual Studio, NetBeans) provázané s repozitury (SVN, CVS, SourceSafe).

Anti-vzor říká, že máme vychovávat heroické jedince schopné pracovat extrémně dlouho každý den, včetně víkendů. Snažit se mít specialisty pro každou oblast, vybavené mocnými nástroji, kteří pracují odděleně a jejichž nástroje nejsou vzájemně integrované. Předpokladem tohoto anti-vzoru je, že pokud každý dělá svou práci, výsledek se dostaví a bude dobrý.

3.4 Demonstrate value iteratively

Přínos (benefit): Brzké zmírnění rizik, vyšší předvídatelnost vývoje, důvěra mezi všemi účastníky projektu.

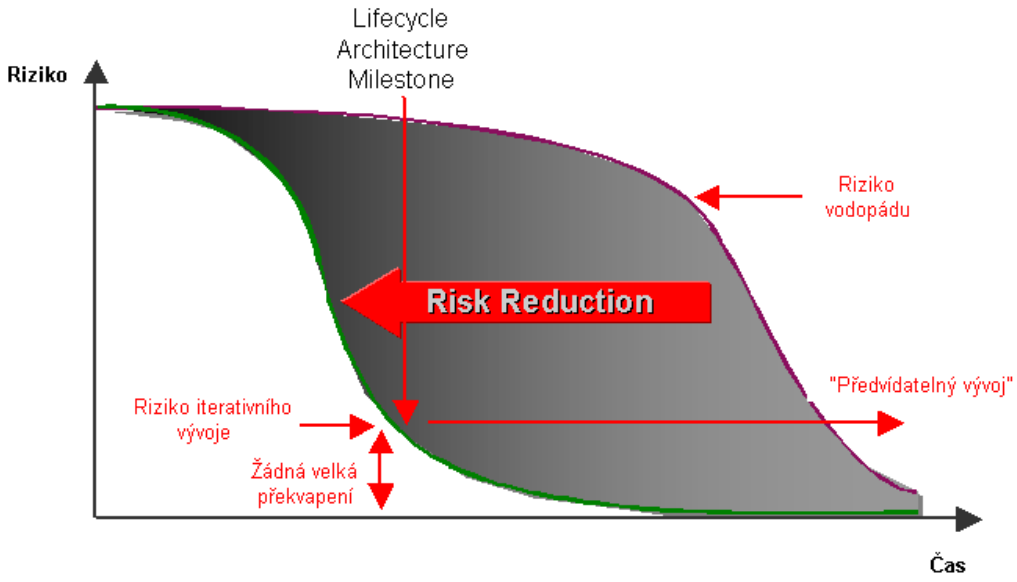
Vzor (pattern): Adaptivní management s použitím iterativního vývoje. Atakuj významná technická, byznys a programátorská rizika co nejdříve. Získej zpětnou vazbu od uživatele tím, že v každé iteraci doručíš nějakou hodnotu.

Anti-vzor (anti-pattern): Naplánuj detailně celý životní cyklus, sleduj změny proti tomuto plánu. Detailní plány jsou lepší plány. Posuzuj status projektu revizí specifikací.

Pod tímto principem se skrývá několik bodů. Prvním z nich je, proč chceme doručovat inkrementálně nějakou hodnotu zákazníkovi – proto, abychom od něj brzy dostali zpětnou vazbu. Toho dosáhneme rozdělením projektu do několika iterací (vždy ale proběhne společně analýza, návrh, implementace, integrace, testování), kterými se přibližujeme výsledné podobě produktu. Díky této zpětné vazbě od zákazníka a uživatelů jsme schopni zjistit, zda se ubíráme správným směrem, zda je třeba změnit některé rysy nebo si zákazník uvědomí zapomenutý rys a chce ho přednostně zařadit místo jiného. Tímto nejen doručujeme zákazníkovi hodnotu, ale také budujeme vzájemnou důvěru tím, že zákazník vidí, zda aplikace opravdu řeší jeho problém, vyhovuje jeho potřebám. Nepřidáváme spoustu rysů podle našeho mínění, které uživatel nikdy nepoužije a přitom nám přidávají spoustu práce.

Druhým bodem tohoto principu je úprava plánů podle zpětné vazby od uživatele. Zaměřujeme se na spustitelný a testovatelný kód, který ukazujeme zákazníkovi, místo nepodstatného hodnocení specifikací. Tímto jsme schopni zjistit, jak moc či málo naplňuje aplikace představy zákazníka a podle toho aktualizovat celkové plány projektu a definovat detailní plán pro příští iteraci.

Třetím bodem principu je zahrnutí a řízení změny. Jelikož je změna v požadavcích v dnešních podmínkách trhu nevyhnutelná, je třeba s ní v projektech (procesu) počítat. Iterativní přístup nám poskytuje způsob, jak inkrementálně implementovat změny. Pro efektivní správu, řízení a implementaci změn je nutné mít definovaný proces a využívat podpůrný nástroj (např. JIRA)



Obr. 3-2: Úroveň rizik pro různé přístupy vývoje (zdroj: RUP)

Posledním, čtvrtým bodem principu je brzké snížení, odstranění rizik projektu (viz Obr. 3-2). Je nutné se věnovat snížení významných technických, byznys a

programátorských rizik tak brzy, jak je možné. Tento přístup je výhodnější, než jejich odsunutí na konec projektu. Jak víme, oprava chyby či implementace změny je 100-1000krát dražší v případě provozované aplikace, než v ranných fázích vývoje. Právě nutnost opravy či implementace nového požadavku může být způsobena špatným odhadem rizik. Proto se snažíme neustále vyhodnocovat různá rizika a snažíme se je v následující iteraci snížit.

Anti-vzorem, dříve běžně používanou softwarovou praktikou, která přispívala k selhání projektů, je detailně plánovat celý životní cyklus předem, kdy je spousta nejasností a rizik v projektu, které určitě nastanou. Dalším anti-vzorem je posuzovat stav projektu na základě revizí a hodnocení specifikací, než na základě demonstrace fungujícího software (implementovaného a otestovaného v několika iteracích).

Příklad:
Vývojový tým má začít pracovat na projektu, jenž bude napsán v technologii J2EE. Vývojový tým však nemá s technologií J2EE žádnou zkušenost.

Proto první iterace v Elaboration (popis fází RUP viz dále v textu) v trvání maximálně 1 měsíce bude mít za cíl vytvořit jednoduchou aplikaci napsanou v J2EE, která řeší podobný, ale jednodušší problém. Na tomto pracuje menší tým (tzv. Tiger team) schopných – senior – vývojářů. Cílem je ověřit si, zda jsme schopni tuto technologii opravdu na projektu úspěšně použít – komunikace uvnitř J2EE, propojení s databází, integrace s jinými systémy. Výsledkem úspěšné iterace bude opět spustitelný a otestovaný build.

Závěry z příkladu: Nečekáme až na fázi implementace, abychom se dozvěděli, zda jsme schopni použít úspěšně J2EE. Předtím bychom vytvořili spoustu dokumentů, modelů a dalších artefaktů, které by však v konečném důsledku byly zbytečné a projekt by musel být ukončen, pokud bychom nebyli schopni J2EE použít.

Výsledkem by kromě nespokojeného zákazníka bylo více utracených peněz a více času stráveného na projektu. Reakce na tuto situaci by byla velmi opožděná.

3.5 Elevate the level of abstraction

Přínos (benefit): Produktivita, nižší komplexnost.

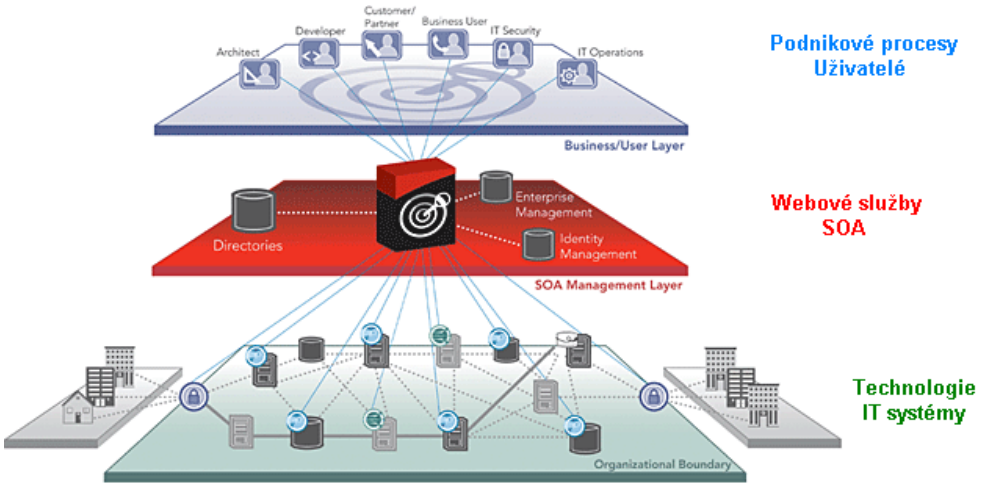
Vzor (pattern): Znovupoužij existující komponenty, redukuj objem ruční práce použitím nástrojů a jazyků vyšší úrovně, navrhuj pružnou, kvalitní a srozumitelnou architekturu.

Anti-vzor (anti-pattern): Jdi přímo od vágních, vysokoúrovňových požadavků uživatelů k psaní kódu celé aplikace.

Možná jedním z největších problémů dnešního vývoje software, je jeho přílišná složitost. Snížení této složitosti má velký dopad na produktivitu. Práce na vyšší úrovni abstrakce snižuje složitost a usnadňuje komunikaci.

Jedním z efektivních přístupů redukujících složitost je znovupoužitelnost existujících „komponent“, jedná se o existující komponenty, existující systémy, podnikové procesy, vzory či Open source software. Příkladem takového znovupoužití, které mělo velký vliv na celou oblast IT je middleware, konkrétně databáze, web servery, portály, v posledních letech pak rozličný Open source.

Příklad:
Dalším příkladem z dneška jsou webové služby (WS – Web Services) a architektura SOA. S použitím WS jsme schopni znovupoužít existující aplikace, které vystavíme jako webovou službu nebo pouze rozličné komponenty, které již jako webová služba existují. Potom je možné pomocí katalogu služeb UDDI skládat výslednou aplikaci jako z kostek (jednotlivých funkčností), bez znalosti implementace komponenty/aplikace.



Obr. 3-3: Příklad vyšší abstrakce pomocí webových služeb

Tento princip mluví také o snížení složitosti a zlepšené komunikaci díky použití nástrojů a jazyků vyšší úrovně. Standardní jazyky jako např. UML jsou schopny vyjádřit pojmy vyšší úrovně, jako jsou např. podnikové procesy, komponenty a jejich komunikaci a naopak skrýt v tu chvíli nepodstatné detaily. Návrhové a implementační nástroje (CASE) mohou pomoci přejít z vyšší úrovně ke kódu nejen modelováním různých modelů, ale také automatickým generováním kódu či testů z těchto modelů.

Třetí bod tohoto principu hovoří o zaměření na architekturu. Jak si řekneme později, je našim cílem při vývoji software navrhnout, implementovat a otestovat architekturu produktu v brzkých fázích projektu. To znamená brzy definovat vysokoúrovňové bloky a nejdůležitější komponenty, jejich odpovědnosti a rozhraní, způsob komunikace, ukládání dat. Definicí architektury vytvoříme kostru systému, která definuje základní mechanismy. Tímto usnadníme zvládnutí složitosti v projektu, když se přidá více vývojářů, vznikne více komponent, napíší se tisíce řádků kódu. Samozřejmě ne pro všechno můžeme znovupoužít existující, je třeba také uvažovat nad tím, které části a jak budeme vyvíjet zákaznickým vývojem.

Anti-vzor říká, že máme jít přímo od vágních, vysokoúrovňových požadavků uživatelů k psaní kódu celé aplikace zákaznickým vývojem, bez tvorby abstraktních modelů, které nám umožní identifikovat místa, kde můžeme znovupoužít existující komponenty. Neformálně zachycené požadavky vyžadují spoustu rozhodnutí a ověření pochopení v několika iteracích. Díky tomu, že se minimálně zaměřujeme na architekturu, jsme nuceni v budoucnu k rozsáhlému a neustálému přepracovávání aplikace.

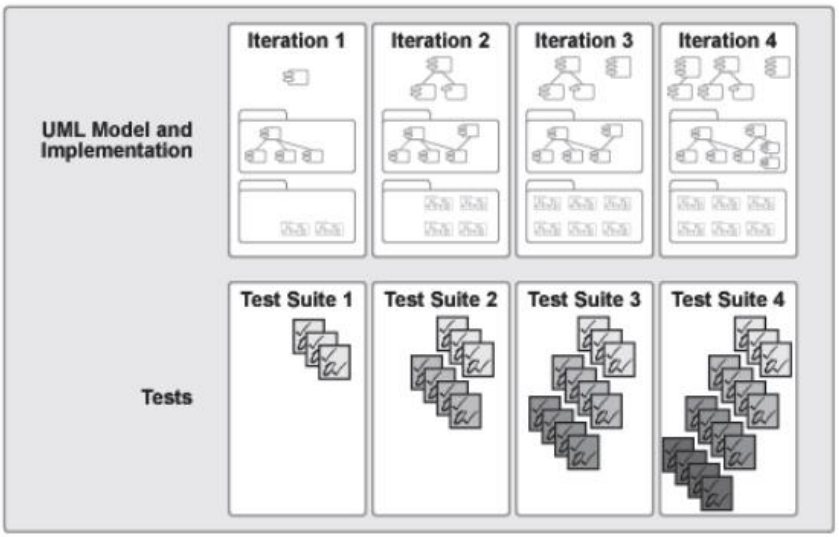
3.6 Focus on quality

Přínos (benefit): Vyšší kvalita, rychlejší pokrok.

Vzor (pattern): Odpovědnost celého týmu za výsledný produkt. Testování a průběžná integrace se stávají prioritními. Inkrementálně zlepšuj testy a automatické testování.

Anti-vzor (anti-pattern): Posuň integrační testování až do fáze, kdy je celý produkt naprogramovaný a otestovaný unit testy. Prováděj revizi všech artefaktů raději než ověřování a testování částečně implementovaného řešení za účelem nalezení problémů.

Zajistit kvalitu produktu znamená mnohem více, než jen účast testovacího týmu na projektu. Zajistit kvalitu znamená, že každý člen týmu vlastní kvalitu, je za ni zodpovědný v průběhu celého životního cyklu. Analytik je zodpovědný za zajištění testovatelnosti požadavků, za jasnou specifikaci požadavků pro potřeby testů, které je třeba vykonat. Vývojáři by měli při návrhu aplikace mít na mysli její testovatelnost a jsou (musí být) odpovědní za testování vlastního kódu unit testy. Management zajišťuje vhodné testovací plány a zdroje pro tvorbu, spouštění a analyzování testů. Testeři jsou experti na kvalitu. Provádí zbytek týmu a vysvětlují hlavní body týkající se kvality software a jsou také odpovědní za funkční, systémové a výkonnostní testování.



Obr. 3-4: Testování, jako vlastní produkt, je v každé iteraci rozšiřováno (zdroj: RUP)

Jedním z největších přínosů iterativního vývoje je brzká a neustálá testovatelnost produktu. Jelikož nejdříve implementujeme důležité vlastnosti produktu a postupně přidáváme další méně důležité, je aplikace provozována s nejdůležitějšími rysy několik měsíců a co je důležitější, je také několik měsíců průběžně testována (viz Obr. 3-4)! Proto je nejviditelnějším přínosem iterativního přístupu vyšší kvalita výsledného produktu.

Stejně jako inkrementálně budujeme výslednou aplikaci, tak bychom měli zvyšovat automatizaci testování. Cílem je nalézt chyby co nejdříve a minimalizovat námahu a náklady do toho vložené. V průběhu návrhu (design) aplikace je třeba uvažovat i nad její testovatelností, některá rozhodnutí v průběhu návrhu totiž mohou výrazně zvýšit možnost automatického testování. Navíc opět připomeneme automatické generování kódu z modelů, jenž může výrazně snížit počet chyb a defektů v kódu. Přístupem, který je populární v Agile komunitě je tzv. Test-first approach, kdy nejdříve píšeme testy a až poté vlastní řešení. Díky dřívějšímu psaní testů vlastně už rozmýšlíme vlastní návrh, který je pak kvalitnější a hlavně píšeme řešení, které je lépe testovatelné.

Anti-vzor tohoto principu zdůrazňuje detailní revize artefaktů (modely, specifikace), což je kontraproduktivní, jelikož nás zdržuje od testování spustitelné aplikace, kde jedině můžeme identifikovat závažné problémy. Druhý anti-vzor zdůrazňuje provést veškeré unit testy před integračním testováním, což opět způsobuje zpoždění v identifikaci závažných problémů.

Kontrolní otázky:

1. Co říká, co zahrnuje princip A?
2. Co říká, co zahrnuje princip B?
3. Co říká, co zahrnuje princip C?
4. Co říká, co zahrnuje princip D?
5. Co říká, co zahrnuje princip E?
6. Co říká, co zahrnuje princip F?
7. Jaká je struktura principů (3 části)?

Úkoly k zamyšlení:
Zmínili jsme 6 základních principů RUPu. Zamyslete se nad tím, jak byste jednotlivé principy implementovali ve vaší dennodenní práce programátora, analytika či projektového manažera. Myslíte si, že je možné tyto principy přijmout všechny najednou? Pokud ne, jaký časový rozsah by byl vhodný?

Korespondenční úkol:
Napište několik důvodů a tyto zdůvodněte, proč není vhodné vytvořit detailní plán pro celý životní cyklus projektu na úvod.

Shrnutí obsahu kapitoly
V této kapitole jste se seznámili se základními principy iterativního vývoje podle RUP. Principy byly pro lepší pochopení doplněny příklady. Principy říkají, jaká je výhoda jejich implementace a představují vzor a anti-vzor. Vzor říká, jakým způsobem princip implementovat, anti-vzor říká, čemu se naopak vyvarovat.

4 Fáze RUP

V této kapitole se dozvíte:

- Jaké jsou RUP fáze?
- Co je to iterace?
- Rozdíl mezi vodopádovou fází a fází v RUPu.

Po jejím prostudování byste měli být schopni:

- Pochopit struktuře RUP projektu.
- Pochopit životní cyklus RUP projektu.
- Porozumět náplni RUP fází.
- Znát milníky RUP fází.

Klíčová slova této kapitoly:

Fáze RUP, iterace, milníky.

Doba potřebná ke studiu: 6 hodin

Průvodce studiem

Kapitola vysvětluje rozdíly mezi fází a iterací, stejně jako mezi fází vodopádu a fází iterativního vývoje. Detailně je pak vysvětlena náplň každé fáze RUP a její milníky.
Na studium této části si vyhraďte 6 hodin.

V této kapitole se budeme věnovat náplni jednotlivých fází projektu podle RUP. Na úvod ukážeme rozdíl mezi klasickými fázemi vodopádu a fázemi v iterativním vývoji. V tradičním modelu jsou fáze definovány/rozděleny podle rolí, ve fázi specifikace požadavků prostě definujeme veškeré požadavky na systém, v analýze toto všechno zanalyzujeme dopodrobna, abychom dále mohli navrhnout řešení atd. Tento model však přináší několik problémů (vodopád viz Obr. 4-1). Předně díky chybějící dennodenní spolupráci mezi členy týmu, jsou často požadavky chápány vývojáři jinak, než to ve skutečnosti zamýšlel analytik a přál si uživatel. Je to způsobeno tím, že analytik definuje požadavky a tyto sepíše do nějakého rozsáhlého dokumentu následujícím způsobem:

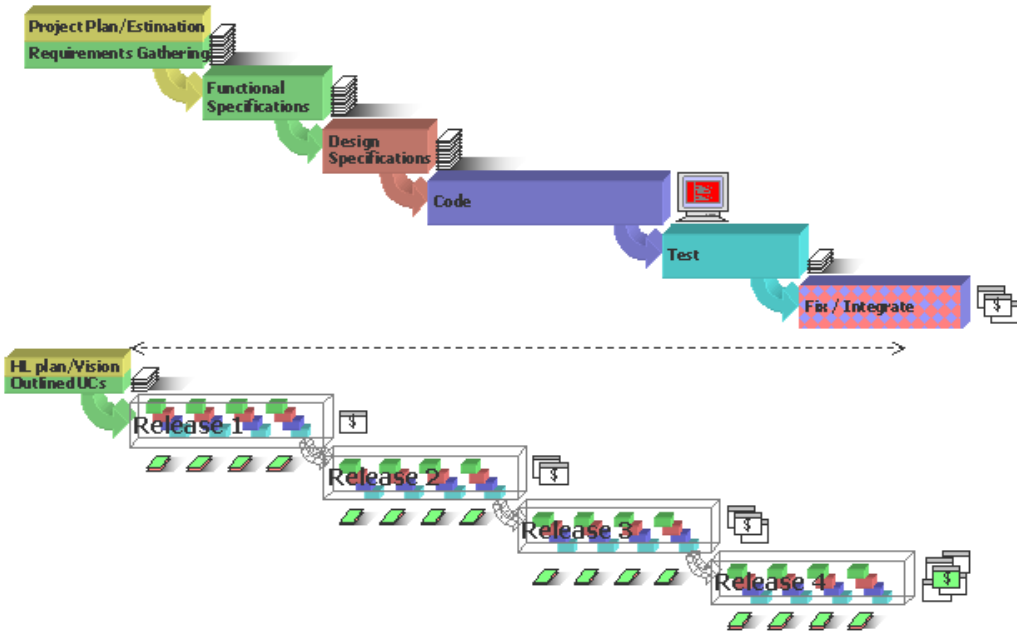
- Systém by měl mít toto ...
- Systém by měl mít tamto ...
- Systém by měl dělat toto ...
- Systém by měl splňovat standard ....
- Systém by měl umožnit ...

Takový dokument poté předá návrháři a je většinou převelen na jiný projekt. Bohužel si s sebou pak odnese i znalost zákazníka a jeho prostředí, pochopení jeho problémové domény a další důležité věci či vazby, které jsou psány mezi řádky. Tudíž o nich nemůže mít návrhář či programátor ani tušení, jelikož se neúčastnili sezení se zákazníkem a analytik již není k dispozici, aby toto vysvětlil. Navíc takový dokument může být těžce použitelný, představme si 20 stran takových požadavků, jak zajistíme, že na straně 12 není požadavek protichůdný proti požadavku na straně 19? Jak zajistíme či naznačíme

návaznost a závislosti mezi jednoduchými požadavky, když použijeme pouhý text?

Dalším problémem je tvorba detailních plánů hned na úvod, kdy ještě nevíme spoustu věcí, nepočítá se v plánech s riziky a nečekanými událostmi (problémy s technologií, složitost problémové domény, ...), proto je tak jednoduché plány přestřelit nebo naopak (což je o moc více obvyklé) podhodnotil.

Díky způsobu vývoje, kdy je třeba mít na začátku definované všechny požadavky, se setkáváme s dalším problémem. Uživatel nám nadiktuje opravdu vše, co by kdy mohl potřebovat. Tím se dostaneme do stavu na **Chyba! Nenalezen zdroj odkazů.** (Standish Group výzkum), kdy máme v aplikaci 80% rysů, které uživatel nikdy nevyužije nebo je využije velice zřídka. Jako vývojáři však všechny tyto zbytečné rysy musíme vytvořit. Potom pravděpodobně není problém s produktivitou vývoje.



Obr. 4-1: Tradiční vs. iterativní model vývoje

Posledním významným problémem, který ve spojitosti s tradičním modelem zmíníme je integrace komponent a testování. To probíhá až na konci projektu, kdy projdeme všemi fázemi. V této části projektu je však nejen díky nepřesným plánům již málo času na řešení odhalených chyb a také je na toto odhalování již příliš pozdě, stojí více, než kdyby např. sami vývojáři spouštěli Unit testy hned po napsaní kódu, kdyby byla architektura integrována a ověřena dříve apod.

4.1 Iterace

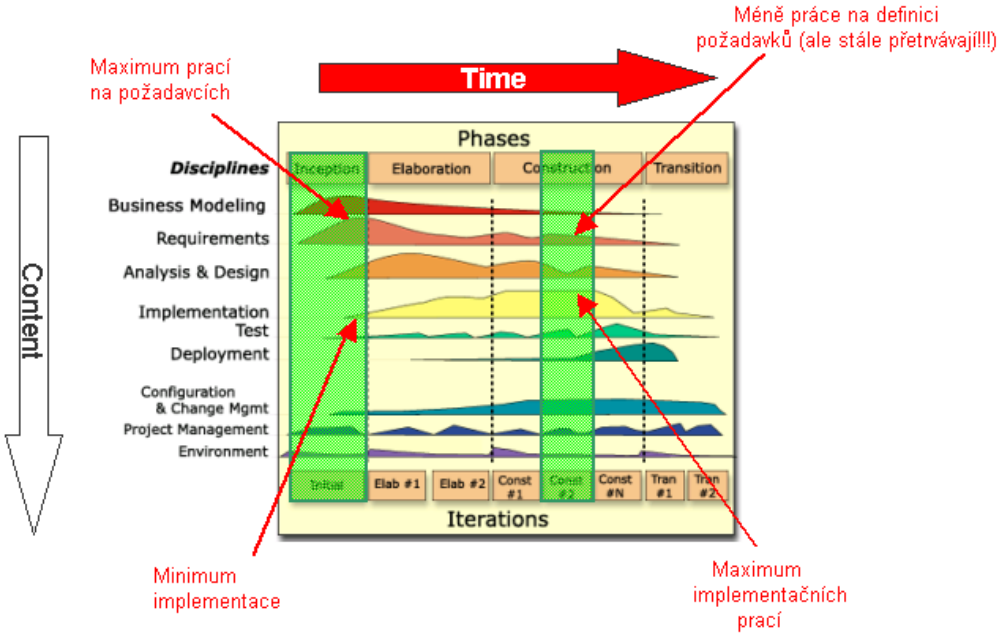
Jak je patrné z Obr. 4-1, mezi fázemi RUPu, resp. iterativního vývoje a vodopádu je zásadní rozdíl. Iterativní vývoj probíhá v několika tzv. iteracích (opakováních). Takových zásadních rozdílů je hned několik:

- Každá iterace produkuje spustitelný a otestovaný build obsahující nově implementované funkčnosti (scénáře) – proto abychom ho mohli dát k dispozici uživateli a dostali od něj zpětnou vazbu, jdeme správným směrem, pochopili jsme jeho potřeby dobře?
- Každá iterace má definovaný přesný cíl, který se snažíme naplnit (paralelní analýza, návrh, implementace a testování vybrané nové funkčnosti – jejich scénářů).
- Iterace je miniprojekt, což znamená, že má svůj začátek, konec a pevně definovaný časový rozsah (většinou 2-4 týdny) – což se již předvídá a detailně plánuje lépe, než například 2 roky.
- V průběhu jedné iterace provádíme všechny disciplíny! Tj. definice požadavků, analýza a návrh, implementace, integrace a testování!!!
- Zřetězením iterací nabalujeme jednotlivé funkčnosti až do výsledného produktu.

Hlavní výhodou je, že již po relativně krátké době má zákazník k dispozici nějakou verzi výsledného produktu, i když jde třeba o nestabilní produkt, tak nám již může říct, co se mu líbí, nelíbí (poskytne velmi cenou zpětnou vazbu) a může s ním dokonce začít pracovat, čili aplikace již může vydělávat, i když neobsahuje zdaleka tolik rysů jako výsledný produkt. V průběhu každé fáze je možno mít 1 až n iterací v závislosti na typu projektu, blíže viz následující kapitoly. Na Obr. 4-2 jsou 2 z iterací naznačeny zeleně.

4.2 RUP fáze

Jak již bylo naznačeno výše, RUP fáze jsou zcela odlišné od fází vodopádu, nejde tedy jen o jejich „přejmenování“. Fáze v RUP jsou spíše jednotlivé statusy projektu, jeho evoluce v čase. Obecně můžeme říci, že výstupem každé fáze iterativního projektu je spustitelný kód. Výsledkem fází vodopádového projektu jsou dokumenty, modely a další artefakty týkající se podobných činností, které je třeba při vývoji SW provést.

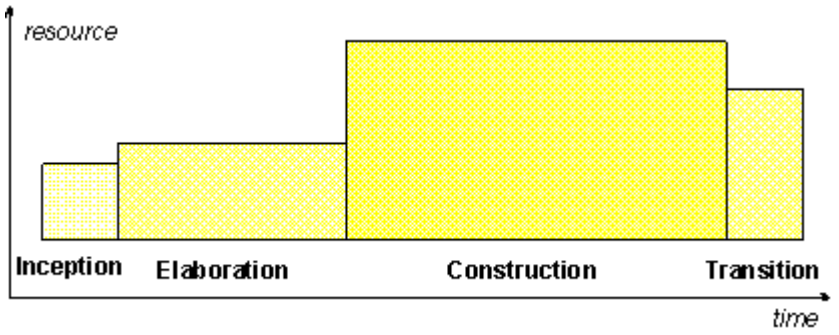


Obr. 4-2: Dvojrozměrný model RUP - fáze a disciplíny

Výsledkem Inception je pochopení problematiky, vize projektu, identifikovaná rizika, shoda na obsahu projektu a jeho financování. Výstupem Elaboration je spustitelná, otestovaná architektura (= fungující část aplikace) a implementované rizikové scénáře, na kterých architekturu demonstrujeme. Výstupem Construction je beta-release aplikace, relativně stabilní, opět spustitelná, téměř kompletní aplikace. V této fázi také doimplementujeme chybové a alternativní scénáře rizikových scénářů/use case, které jsme neimplementovali ve fázi Elaboration. Výstupem Transition je pak již produkt připravený k finálnímu nasazení včetně veškeré dokumentace a hardware. Zásadní rozdíl je také v tom, že každá fáze může obsahovat několik iterací, v nichž se však vždy provádí všechny disciplíny s různým objemem prací – což je reprezentováno plochou pod danou křivkou (viz Obr. 4-2)

Je nutné zdůraznit, že každé fáze se většinou účastní různý počet vývojářů. V úvodu, kdy je třeba identifikovat požadavky, často komunikovat se zákazníkem, navrhnout architekturu, ověřit komunikaci s jinými systémy apod. jsou zainteresováni často jen projektový manažer (Project Manager), systémový analytik (System Analyst), zákazník, architekt, návrhář testů (Test Designer). V pozdějších fázích, kdy je stabilní architektura přibude větší množství vývojářů i testerů. Tito lidé (různé role), ale stále pracují spolu v jednom či více týmech a jsou k dispozici pro vysvětlení nejasností!!!

Poslední zásadní věcí týkající se fází, je rozložení prací na projektu v jednotlivých fázích. Následující obrázek a tabulka toto ukazují. Je vidět, že cílem Inception je opravdu rychle definovat vizi a rizika projektu a základní projektové věci a co nejrychleji přejít do Elaboration, abychom mohli zákazníkovi co nejdříve poskytnout spustitelný SW. Účast zdrojů na této fázi je omezená, většinou se jedná o projektového manažera, systémového analytika, architekta, test manažera a test designera + zástupci zákazníka a uživatelů. Celkově by Inception měla zabírat 10% celkového času, spíše méně.



Obr. 4-3: Objem prací a časové trvání jednotlivých fází RUP (zdroj: RUP)

	Inception	Elaboration	Construction	Transition
Pracnost	~5 %	20 %	65 %	10%
Plán	10 %	30 %	50 %	10%

Tabulka 4-1: Průměrné časy trvání jednotlivých fází RUP (zdroj: RUP)

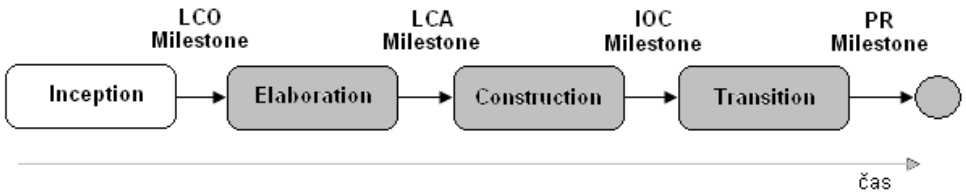
Elaboration obsahuje o málo více zdrojů a časově je její trvání zhruba 30% celkového času. Z obrázku i tabulky je patrné, že nejvíce času a nejvíce zdrojů vyčerpá Construction, kdy je v několika iteracích implementován výsledný produkt. Časově to znamená přibližně 50% celkového času projektu. Transition již pak zabírá 10% času, ale zdrojů je využíváno pořád hodně, jelikož dodělváme zbývající jednodušší rysy, řešíme finální vyladění produktu, jeho výkonnost, kompletujeme dokumentaci a provádíme závěrečné testy.

Následující text se věnuje popisu jednotlivých fází, jejich náplně a milníků detailněji. Více se také zabývá počtem potřebných iterací v jednotlivých fázích.

4.3 Inception phase

Cílem úvodní fáze je pochopení cílů projektu, požadovaných rysů aplikace, výběr vhodné technologie, definice vývojového procesu a výběr a nastavení nástrojů. Přesněji má Inception fáze těchto 5 cílů:

1. Porozumění tomu, co vytvořit – vytvoření vize, definice rozsahu systému, jeho hranic; definice toho, kdo chce vytvářený systém a co mu to přinese.
2. Identifikace klíčových funkcionalit systému – identifikace nejkritičtějších Use Casů.
3. Návrh alespoň jednoho možného řešení (architektury).
4. Srozumění s náklady, plánem projektu, riziky.
5. Definice/úprava procesu, výběr a nastavení nástrojů.



Obr. 4-4: Fáze Inception

V průběhu první fáze proběhne ve většině projektů pouze jedna jediná iterace. Proto, abychom dosáhli cílů této fáze, je však možné provést více iterací. Mezi důvody, které k tomuto přispívají můžeme zařadit následující:

- Rozsáhlý projekt, kde je těžké pochopit zaměření a rozsah systému.
- Jedná se o nový systém v neznámé problémové doméně, je obtížné definovat, co by měl systém dělat.
- Vyskytují se velká technická rizika, která je třeba snížit implementací prototypu nebo konceptem architektury před vlastním představením / schválením projektu.

4.3.1 Cíl 1 – porozumění tomu, co vytvořit

Pochopení potřeb uživatele je často největším problémem softwarových projektů (viz graf Standish – **Chyba! Nenalezen zdroj odkazů.**). Každý ze zúčastněných na projektu pod nějakým cílem může chápat něco jiného. Něco jiného chápe manažer, něco jiného člověk, který danou práci vykonává denně,

něco jiného pak analytik, který nemusí detailně znát podnikové procesy dané organizace. Správné pochopení potřeb uživatele, zákazníka je náplní analytika.

Výsledkem této práce by měla být vize. Vize nám říká, kterým směrem se bude projekt ubírat, je však definována z pohledu uživatelů aplikace (definuje jeho klíčové potřeby a rysy aplikace), ne technickou řečí! Obsahem vize jsou nastíněné klíčové požadavky na systém, vize tedy poskytuje jakýsi základ pro detailnější technické požadavky. Na vizi by se měli všichni účastníci projektu shodnout, pokud shoda nenastane, není možné jít do další fáze, jíž je Elaboration.

Příklad jednoduché vize pro jednoduchý projekt časovače, který pracuje na stanice vývojáře a reportuje určitý čas strávený prací na daném projektu (komplexnější šablona vize z OpenUP viz Příloha A – Vision):

Osobní časovač: Vize
Problém (co řešíme, co nefunguje, co nás trápí) Gary není schopný sbírat konsistentní časové údaje od vývojářů reprezentující čas strávený na různých projektech. Není tedy možné monitorovat a porovnat postup oproti plánům, fakturovat řádné časy, platit externí spolupracovníky a samozřejmě také na základě těchto dat dělat věrné odhady dalších iterací.
Řešení (jak tento problém popsaný v předchozím kroku budeme řešit) Osobní časovač (OČ) měří čas strávený na projektech, shromažďuje a ukládá tato data pro pozdější zobrazení (stylem Post-it poznámek), aby mohl Gary systematicky organizovat a hodnotit projekty, sledovat aktuální postup prací a ty porovnávat s plánovanými odhady pro jednotlivé projekty
Zainteresované strany (anglicky *stakeholders*) - jednotlivý vývojáři - pracovníci administrativy - projektový manažer
Use Cases (základní obecné funkčnosti) - Změř čas aktivity - Sesbírej týdenní data - Sluč / konsoliduj data pro každý projekt - Nastav nástroj a databázi pro projekt

Tabulka 4-2: Příklad vize pro osobní časovač

Cíl 1 tedy zahrnuje ***identifikaci aktorů*** budoucího systému. S těmito aktory poté identifikujeme základní Use Casy systému a stručný nástin scénářů (většinou pouze základní scénář, ne alternativní). Tyto popisy jsou samozřejmě v průběhu několika sezení iterativně upřesňovány. V dalších fázích pak dále doplňovány o alternativní toky (scénáře) a propracovány, realizovány kódem.

4.3.2 Cíl 2 – klíčová funkcionalita systému

Dalším bodem Inception fáze je identifikace kritických Use Casů (většinou se jedná o 20 – 30% všech Use Casů). Kritické jsou z důvodu technologického (tvoří kostru aplikace – vrstvy, rozhraní, způsob komunikace či ukládání dat) nebo z důvodu nutných funkčností, které musí aplikace obsahovat. Kritické UC mají významný dopad na architekturu systému, jinými slovy ji tvoří (spolu s nefunkčními požadavky). Analytiky je vytvořen jejich detailnější popis, je možné posunout zpodrobnění některých alternativních toků těchto UC do dalších iterací dalších fází (hlavně Elaboration, částečně i do Construction),

pokud nemají významný dopad na architekturu. Poté co jsou identifikovány, jsou kritické UC představeny zbytku týmu a vysvětleny, proč jsou kritické. Zbytek týmu by s nimi potom měl souhlasit.

Kritické UC systému jsou vyjmenovány v SAD – Software Architecture Document.

Součástí definice požadavků jsou také ***nefunkční požadavky***, které následně ***určují architekturu řešení***. Nefunkční požadavky jsou požadavky, které ovlivňují chod apikace, ale nejsou její viditelnou, očekávanou funkčností. Patří mezi ně například výkonnost (počet zpracovaných transakcí), doba odezvy, formát uživatelského rozhraní, bezpečnostní požadavky apod. V našem případě osobního časovače popsaného ve vizi se jedná o následující:

- NF1: současný přístup více uživatelů,
- NF2: přístup i mimo firemní síť,
- NF3: přístup pomocí různých typů klientů (PC, notebook, smart phone),
- NF4: doba odezvy do 300ms uvnitř firemní sítě,
- NF5: doba odezvy do 2 vteřin vně firemní síť.

Tyto požadavky posléze namapujeme na funkční požadavky (Use Casy), jelikož všechny nemusí platit pro všechny Use Casy.

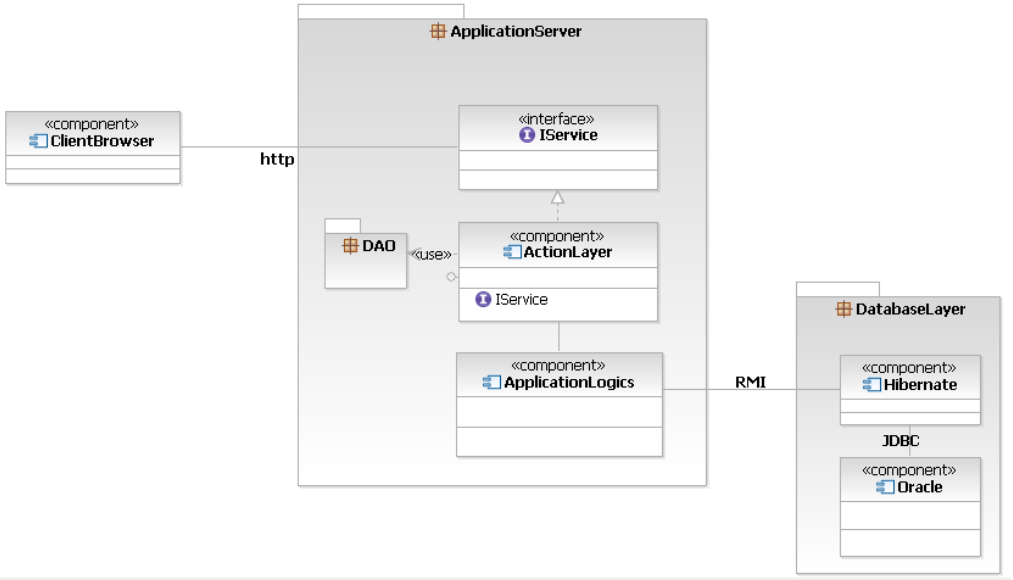
4.3.3 Cíl 3 – Návrh možného řešení

Cílem Inception fáze je určit, zda má smysl pokračovat dále v projektu, proto si musíme být jisti, že ***existuje alespoň jedna*** technologická, respektive softwarová ***architektura***, která umožní implementovat systém s rozumným podílem rizik a s rozumnými náklady. Technologickou architekturou rozumíme nastínění různých typů zařízení (HW, klienti), jejich spojení a způsob komunikace (TCP, RMI, Web Services). Softwarovou potom rozumíme detailnější pohled na strukturu, vrstvy a komunikaci mezi jednotlivými SW komponentami uvnitř těchto zařízení.

V našem případě přichází v potaz softwarová architektura založená na architektuře klient-server (C/S), jelikož víme, že máme vytvořit systém pro evidenci práce na projektech, která musí být přístupná všem lidem zůčastněným na projektu. Navíc víme, že máme také několik lidí v terénu (prodejci a analytici), kteří potřebují flexibilní přístup k aplikaci odkudkoliv v kteroukoliv dobu, aby nemuseli reportovat až po příjezdu do kanceláře. Proto je důležité webové řešení s tenkým klientem (prohlížečem či jednoduchou aplikací), pro případ použití mobilního telefonu či kapesního počítače. Dané řešení můžeme realizovat pomocí několika vrstev a také s využitím několika technologií:

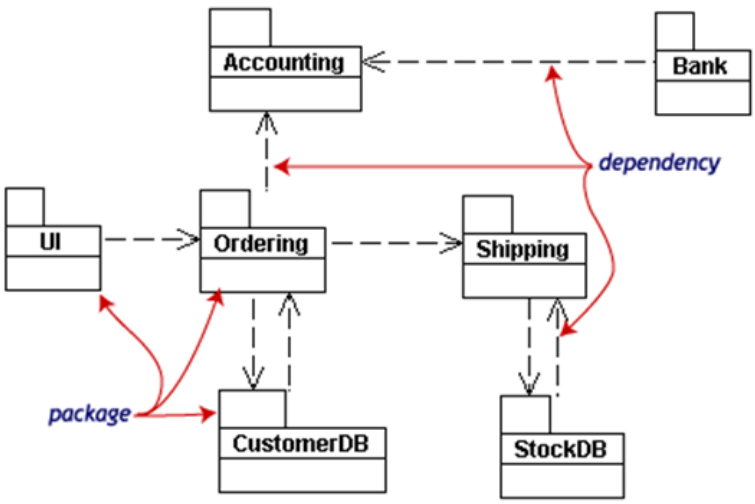
- První varianta může obsahovat 2 vrstvy – databázovou a tlustého klienta (prezenční + aplikační), tu jsme však zamítli z výše zmíněných důvodů, jímž je především potřeba malých výpočetních nároků na klienta.
- Druhá varianta může být složena ze tří vrstev (viz následující obrázek) – prezentační vrstva reprezentována standardním webovým prohlížečem, aplikační realizována webovým serverem se servisní

vrstvou, vrstvou akcí a aplikační vrstvou, které je možné realizovat buď pomocí JEE či .NET a datová reprezentována frameworkem Hibernate napojeného na databázi Oracle. Podle potřeby je možné využít například post-relační databázi Caché či XML databázi.

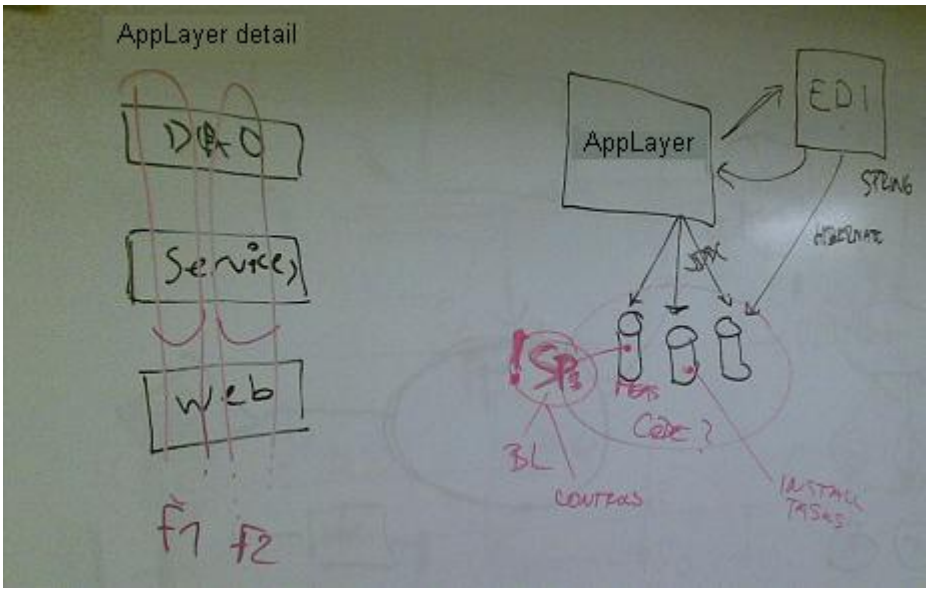


Obr. 4-5: Varianta klient-server (C/S) softwarové architektury popsané pomocí UML modelu komponent a balíčků.

Pro modelování architektury je možné použít několik forem modelů. Důležité je popsat použité technologie, vrstvy, objekty (rozuměj zatím pouze základní komponenty) a spojení mezi nimi, případně zmínit jaký model je použit. Následující obrázek ukazuje diagram balíčků, které je také možné použít pro popis vrstev a komunikace budoucí architektury.



Obr. 4-6: Návrh softwarové architektury pomocí UML diagramu balíčků. Pro doplnění je ještě třeba popsat jaké technologie jsou použity na jednotlivých vrstvách a jak spolu komunikují (http, TCP, RMI, web services, …).



Obr. 4-7: Další, méně formální varianta modelu softwarové architektury. Vrstvený model vlevo je detailem aplikační vrstvy vysokoúrovňového modelu.

Pro popis technologické a softwarové architektury je tedy možné použít následujících variant diagramů UML:

- diagram tříd a balíčků,
- diagram nasazení,
- diagram komponent,
- či jejich vzájemné kombinace.

Je možné, že v Inception fázi bude nezbytné prototyp či způsob komunikace některých komponent architektury ověřit, implementovat, abychom snížili na přijatelnou úroveň či odstranili možná technologická rizika související s použitou technologií, kompatibilitu, komunikaci s ostatními systémy či nákladovou stránku údržby nebo nutnost potřeby využití nějaké komponenty.

Model architektury (jak technologické, tak softwarové) musí obsahovat popis jednotlivých komponent, vrstev, balíčků a použitých frameworků: jaká technologie je použita a také, jak spolu jednotlivé vrstvy komunikují. Model tedy musí obsahovat buď legendu, co který objekt a spoj znamená nebo odvolávku na použitý standard, např. UML diagram nasazení.

Tým se může pro naplnění cíle 3 ptát také na architekturu a technologie použité v předchozích projektech podobného rozsahu a zaměření, na jejich údržbu a cenu. Dále by se měl zabývat otázkami potřeby softwarových komponent a možnosti znovupoužití či nákupu již existujících. S tím také souvisí náklady na jejich pořízení a spojená rizika. ***Každá z variant bude mít samozřejmě své výhody a nevýhody, které je důležité při rozhodnutí zmínit:***

- známe danou technologii a umíme ji efektivně využít (např. JEE knihovny, vzory, frameworky třetích stran)?

- umíme danou technologii efektivně provozovat (nastavení, provoz a správa web kontaineru)?
- má smysl použití komponent a sběrnice webových služeb, když by to vyžadovalo rozsáhlý zásah do architektury celého podnikového řešení?
- je použití dané technologie pro daný případ vhodné? Nejdeme takzvaně *„s kanónem na komára?“* Nebude nám stačit zabezpečené řešení na bázi PHP? Toto vše vychází z potřeb projektu a je třeba uvážit a ***zahrnout do seznamu rizik***.

Na konci Inception bychom měli být obeznámeni s riziky, kterým budeme vystaveni dále v projektu, hlavně se jedná o rizika spojená s vybranou technologií či znovupoužitými komponentami.

4.3.4 Cíl 4 – porozumění nákladům, plánu, rizikům

Pro celý projekt je kritické pochopení toho, co chceme vytvořit, stejně tak kritické je ale také vědět, jak toho dosáhnout a s jakými náklady. Hodně nákladů je například spojeno se zdroji a také se snižováním/odstraňováním rizik. Podle zdrojů, které máme k dispozici, jsme schopni odhadnout nejen dobu, ale také přibližnou cenu projektu.

Výsledkem tohoto bodu je dokument zvaný Business Case. Dokument popisuje ekonomické přínosy produktu z pohledu kvantifikovatelných veličin. Dobrým příkladem může být použití návratnosti investic, tzv. ROI (Return of Investments), jenž vypočítáme ze vzorce (*100 abychom dostali procenta):

$$ROI = \frac{zisk - náklady}{náklady} \qquad \text{nebo} \qquad ROI = \frac{úspory}{náklady}$$

Příklad:
Pokud nás tedy softwarový projekt stojí 1.000.000 Kč a díky němu (automatizace práce zaměstnanců) ušetříme za rok při 25 zaměstnancích s náklady za jednoho 500 Kč za hodinu a při 200 pracovních dnech práci 5ti zaměstnanců, bude zisk následující:

8 hodin * 500 Kč/h * 5 zaměstnanců * 200 pracovních dnů = 4.000.000 Kč

pak výsledná návratnost investic bude (úspory/zisk * 100):

$$ROI = \frac{4.000\ .000}{1.000\ .000} = 400\ \%$$

Je zřejmé, že v tomto případě se projekt vyplatí, jelikož návratnost investic je 400%, což znamená, že vložená investice se nám vrátí 4krát.

V malém projektu může mít Business Case formu e-mailu nebo poznámky, v rozsáhlejším pak bude mít klidně několik stránek.

Pokud je rozpočet dán shora (např. v interních IT organizacích), je předmětem tohoto kroku odhad rozsahu projektu a časových plánů, tj. co jsme schopni s tímto rozpočtem doručit a za jakou dobu.

Velmi důležitou aktivitou a výstupem Inception fáze je také definice rizik a návrh akcí na jejich odstranění. Rizika mohou pocházet z několika oblastí, nejběžněji to může být oblast:

- Organizační,
- Finanční,
- Lidská,
- Technologická,
- Prostředí.

Rizika jsou důležitá pro další fázi Elaboration, která je zaměřena na jejich snížení či odstranění, hlavně rizik technologických. Iterace a jejich náplň v Elaboration jsou vlastně provedené navržené akce na odstranění rizik navrhnuté v Inception. Proto je důležité před přechodem do další fáze rizika identifikovat. Všechna možná rizika by měla být popsána podle určité struktury:

- Riziko je třeba nějak pojmenovat a stručně popsat,
- Je důležité znát dopad rizika, jak moc ovlivní jeho výskyt výstup projektu.
- Pravděpodobnost výskytu rizika.
- Důležitost – většinou počítána jako dopad * pravděpodobnost výskytu. Hodnota je použita k třídění rizik, aby bylo zaručeno že ty s nejvyšší důležitostí jsou řešena nejdříve.
- Vlastník rizika – osoba odpovědná za odstranění či snížení rizika.
- Strategie či plán na odstranění rizika.

Následující příklad ukazuje možná rizika na projektu a akce na jejich snížení/odstranění. Jako ve všech případech, je možné u jednodušších projektů následující tabulku zjednodušit, formu upravit podle potřeb, stejně tak jako u složitějších doplnit o další potřebné údaje.

Pokud mluvíme o rizicích, je třeba také zmínit, co to vlastně riziko je:

Riziko je událost, jejíž výskyt může zabránit úspěšnému dokončení projektu či mít vliv na jeho kvalitu, včasné doručení.

Název rizika	Popis	Dopad	Pravděpodobnost	Důležitost	Vlastník
Nedostatečné zapojení všech stakeholderů	Předchozí zkušenost nám říká že lidé z oddělení X nemusí pochopit či souhlasit s požadavky, výsledkem budou požadavky na zásadní změny po Beta-releasu.	3 vysoký	90%	2.7	Analytik Honza
Integrace se systémem X	Není zřejmé, jak integrovat naši aplikaci s historickým systémem X.	3 vysoký	80%	2.4	Architekt Petr
Tréninkové materiály	Nemáme oprávnění vytvořit kvalitní tréninkové materiály, což může vést k nekvalitnímu tréninku.	2 střední	100%	2.0	Manažerka Anička
Nedostatečná zkušenost s JEE	Je riziko, že vytvoříme druhořadé, méně technicky kvalitní řešení v důvodu nezkušenosti s platformou JEE.	2 střední	60%	1.2	Vývojář Tom

Tabulka 4-3: Příklad seznamu rizik (risk list), zdroj [Kr06]

Je běžné, že seznam rizik v praxi často obsahuje spíše seznam faktů, věcí, které na projektu již nastaly a nejsou již tedy rizikem, nemáme u nich již co odtraňovat a jen můžeme počítat škody. K tomu, abychom tento nešvar odstranili, je vhodné přidat do tabulky či seznamu rizik ještě jedno políčko, které *bude popisovat fakt* a v popisu rizika bude z něj vycházející riziko:

Fakt: Nezkušený tým složený z absolventů (nízká praktická znalost technologie, návrhových vzorů a „opravdového programování“).

Riziko: Možná nízká kvalita produktu z důvodu nevhodného použití technologie .NET a úplného opomenutí použití návrhových vzorů.

Také je běžné, že seznam rizik je vytvořen na začátku projektu, ale již k němu není dále v projektu přihlíženo (seznam nežije), natož aby podle něj byly plánovány následné iterace v Elaboration.

Díky seznamu rizik jsme schopní identifikovat možné příčiny rizik a účinně je odstranit dříve, než se mohou projevit. Proto monitorujeme situaci i v průběhu ostatních fází projektu a v případě identifikace dalšího rizika toto okamžitě vyhodnotíme a přijmeme nutné akce na jeho odstranění. Díky tomuto riziky řízenému přístupu řešíme problémy na úvod projektu a v nejhorším případě můžeme projekt ukončit bez výraznějších ztrát. Nečekáme až nastanou a tudíž

nejsme překvapeni například na konci projektu při integraci s dalšími systémy organizace.

Nyní tedy již k vlastním akcím navrženým na snížení identifikovaných rizik. Seznam rizik doplněný o akce:

Název rizika	Popis	Strategie
Nedostatečné zapojení všech stakeholderů	Předchozí zkušenost nám říká že lidé z oddělení X nemusí pochopit či souhlasit s požadavky, výsledkem budou požadavky na zásadní změny po Beta-releasu.	**Strategie: Snížení rizika** Po vytvoření use casů požadovaných tímto oddělením, vytvoříme jednoduchý UI prototyp a částečnou implementaci. Uspořádáme meeting se zástupci tohoto oddělení, projdeme definované use casy, prototyp a budeme kreslit storyboardy. Trvejte na smysluplné zpětné vazbě od všech zúčastněných. V průběhu projektu budeme zástupce tohoto oddělení detailněji informovat o postupu a poskytovat demo verze a alfareleasy.
Integrace se systémem X	Není zřejmé, jak integrovat naši aplikaci s historickým systémem X.	**Strategie A: Snížení rizika** Vytvoříme „tiger team“ sestávající z několika málo zkušených vývojářů, jejichž cílem je ověřit na integraci prototypu naší aplikace se systémem X. Integrace a její způsobmůže být velmi primitivní, cílem je však ověřit schopnost propojení se systémem X. Navrhujte, implementujte a testujte dané use casy v průběhu projektu, aby bylo toto propojení neustále validováno. **Strategie B: Předejítí rizika** Změna náplně projektu tak, že daná integrace nebude obsahem, předmětem.
Tréninkové materiály	Nemáme oprávnění vytvořit kvalitní tréninkové materiály, což může vést k nekvalitnímu tréninku.	**Strategie: Přesun rizika** Outsourcing vývoje tréninkových materiálů v externí organizaci (je třeba si uvědomit, že tento krok může přinést, generovat jiná rizika).
Nedostatečná zkušenost s JEE	Je riziko, že vytvoříme druhořadé, méně technicky kvalitní řešení v důvodu nezkušenosti s platformou JEE.	**Strategie A: Snížení rizika** Poslat několik vývojářů na školení Microsoft .NET a nalezení rezervy v rozpočtu na mentora 2 dny v týdnu po dobu prvních dvou měsíců trvání projektu. Nábor nového člena týmu se znalostí .NET platformy. Vytvoření prototypu pro ověření použití .NET platformy.

Tabulka 4-4: Příklad seznamu rizik (risk list) doplněný o akce na jejich snížení či odstranění, zdroj [Kr06]

Všimněte si, že dané akce jsou navrženy s ohledem na opravdové odstranění rizika, ne jen na nějaké konstatování že riziko budeme monitorovat (Strategie: „Opravdu to udělej, zkus to, ať máš jistotu že to jde či nejde!“). Potom tedy v případě neznalosti technologie pouze nepošleme vývojáře na školení, ale necháme je udělat jednoduchý prototyp, budeme volit kratší iterace, refaktorovat daný prototyp, přizveme zkušeného architekta z jiného projektu či externího na konzultace atd.

Jako poslední zmíníme, že k řešení rizik se používají tři strategie, které již byly uvedeny v tabulce, jedná se o:

- Odstranění, snížení rizika – snížení závažnosti rizika.
- Předejití riziku – modifikace projektu za účelem zamezení výskytu rizika.
- Přesun rizika na jiný subjekt – reorganizace projektu za účelem přesunu odpovědnosti za riziko na jinou organizaci (jiná organizace vlastní dané riziko).

Pro každou z těchto strategií pak můžeme u daného rizika definovat konkrétní akce (viz předchozí Tabulka 4-4).

4.3.5 Cíl 5 – proces a nástroje

V úvodní iteraci/iteracích je také místo pro výběr a definici procesu, který bude vývojový tým dále v projektu následovat, stejně jako pro výběr a nastavení prostředí a nástrojů. Proces i nástroje by měly být vybírány podle specifických potřeb projektu. U procesu jde o množství činností a kroků, které jsou prováděny, o množství a formu artefaktů, které jsou vytvářeny (formální / neformální, rozsáhlý, pouze screenshot, ...). V případě nástrojů uvažujeme vždy jaký je přínos v porovnání s investovanou snahou a také musíme počítat s nutnosti údržby detailního modelu vytvořeného pomocí mocného nástroje, což stojí čas a tedy peníze.

V první iteraci bychom měli nastínit proces a prostředí, v druhé iteraci nasadit a upravit podle okamžité zpětné vazby ode všech zúčastněných a poté samozřejmě stále iterativně vylepšovat a upravovat (aplikace principu A – Adapt the process).

Pokud jsme vybrali a nastavili proces, můžeme se věnovat výběru nástrojů. Dané nástroje by měly podporovat zvolený proces, metodu! Příkladem je například sada nástrojů Rational a integrace s RUP, existence tzv. tool mentorů (popisy krok za krokem, jak dané aktivity procesu vykonávat pomocí učitého nástroje). Pokud pro tuto oblast existují firemní standardy, měli bychom je samozřejmě vzít také v potaz. Pokud ne, uvažujeme následující oblasti:

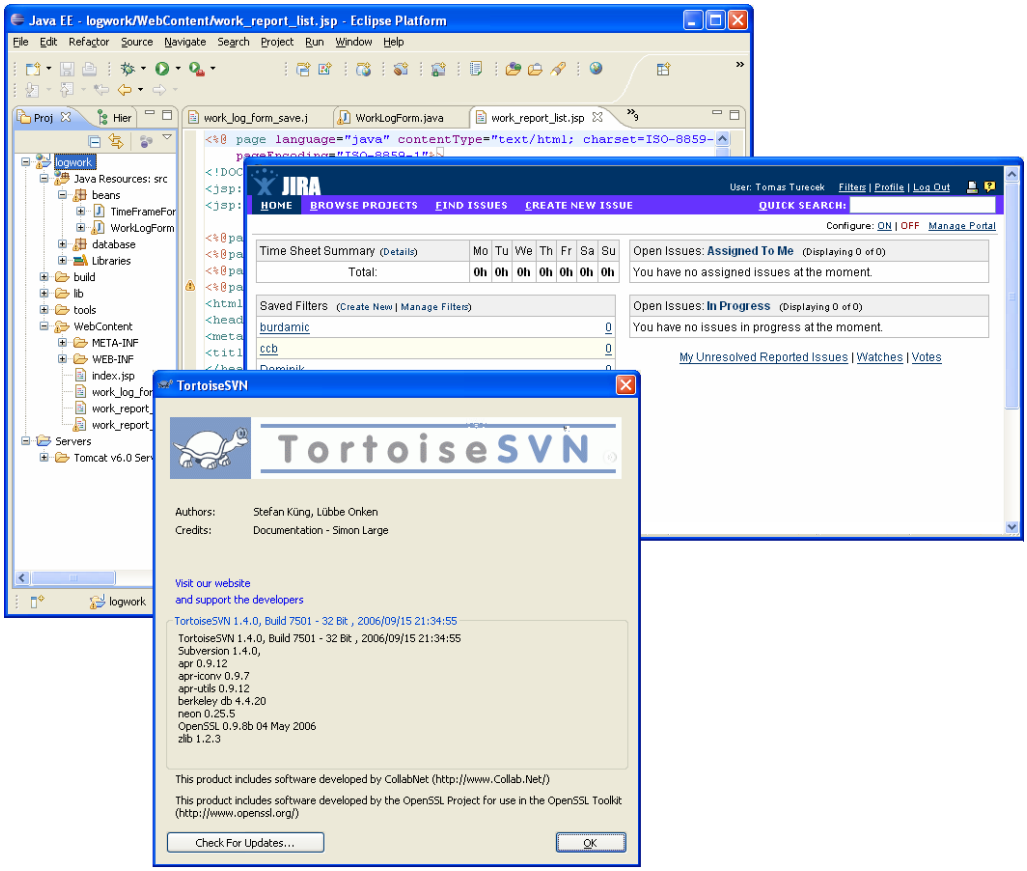
- IDE – integrované vývojové prostředí (př. Eclipse, Visual Studio, NetBeans, …).
- Nástroj pro správu požadavků, chyb (př. Rational Requisite Pro, Jira, Rational Team Concert).
- Vizuální modelovací nástroje (př. Magic Draw, Borland Together, Eclipse pluginy, …)
- Nástroje pro správu konfigurací a změn – Configuration and Change Management Tools (CVS, SVN, Jira).

Jelikož je vývoj software týmovým sportem, kde několik vývojářů, analytiků, testerů neustále spolupracuje, je třeba podpořit paralelní práci a sdílení kódu také nástroji. Jedná se hlavně o následující oblasti:

- Úložiště zdrojových kódů a dokumentace (repozitory) – cílem by mělo být dosáhnout možnosti sestavení kompletní aplikace z repozitory.

- Automatizace sestavení aplikace (build) a testování – pomocí nástrojů make, Ant (nUnit testy jako task).
- Ukládání (commit) pouze hotových věcí do repozitory – nehotové věci mohou způsobit problémy a chyby.

Dohodnutá či psaná pravidla jsou důležitá, ale lze je obejít. Pokud chceme jejich dodržování implicitně vynutit, pomohou nám právě nástroje. Neumožní nám commit pro neotestované či chybové třídy, nutí nás nejdříve vytvořit test, kontrolují čitelnost kódu a překrytí metod vůči standardu a spousta dalších.

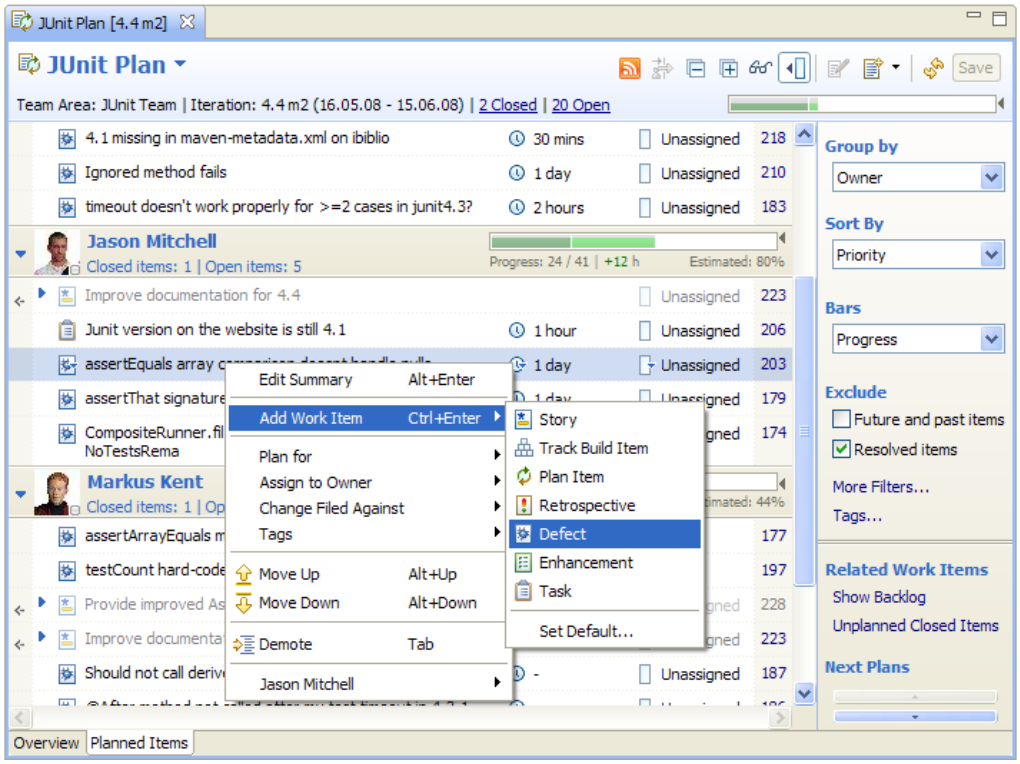


Obr. 4-8: Eclipse IDE, SVN repository, Jira issue tracking tool

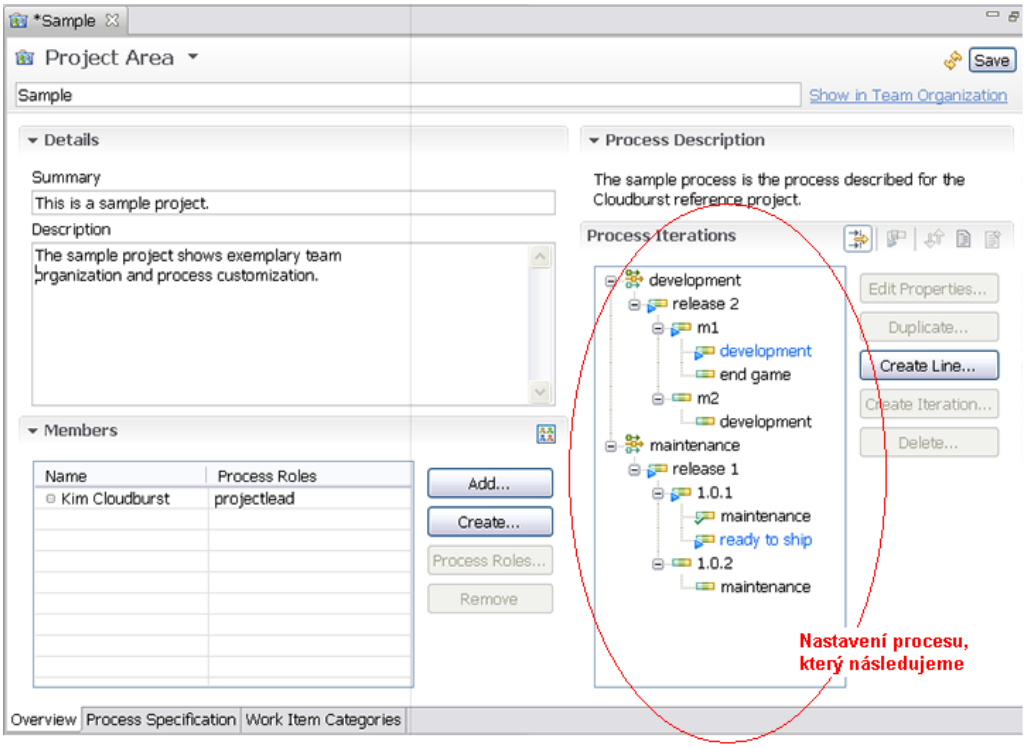
Výborným, i když ne zcela levným počinem na poli nástrojů je nový produkt IBM zvaný Rational Team Concert (RTC) běžící na platformě Jazz. Tento nástroj v sobě integruje definici procesu podle kterého při vývoji postupujeme, dále agilní plánování, wiki, repozitory, buildovací mechanismus, správu členů týmu, prostě veškeré potřebné nástroje pro vývoj. Na rozdíl od výše zmíněné kombinace nástrojů (viz Obr. 4-8) je toto jeden nástroj (není třeba odkazovat a integrovat, vše je provázáno) a hlavně není orientován pouze na jednoho člena týmu, neobsahuje pouze pohledy na jeho práci, nýbrž základním pohledem je tzv. team view.

Následující obrázek ukazuje tento pohled, kde vidíme jednotlivé položky, tzv. work items (přiřazené úkoly, chyby, požadavky, rizika, …) každého člena týmu, jeho odhady pracnosti a stav dané položky (otevřená, uzavřená, právě řešená). Navíc tento pohled obsahuje také týmové informace (kolik

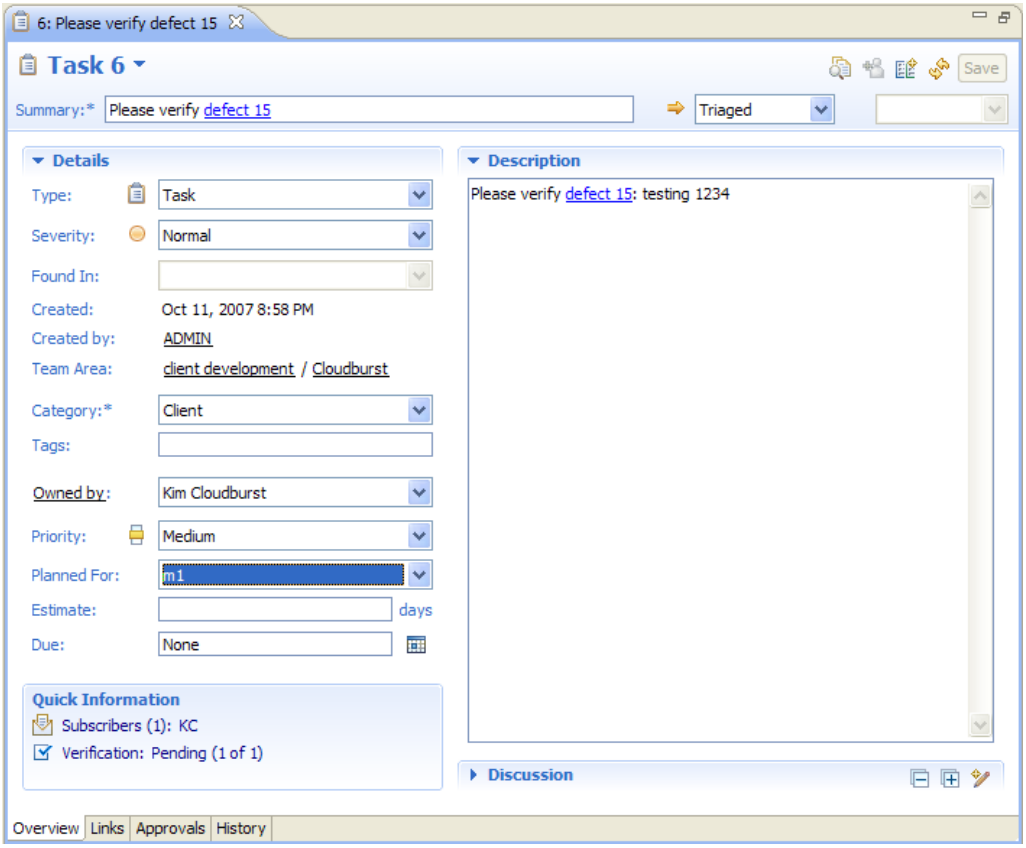
z celkového počtu položek je již uzavřeno, kolik zbývá). Pohled umožňuje jednoduchou manipulaci (drag and drop) a také vytvoření položek.



Obr. 4-9: Team Central View IBM Rational Team Concert, zdroj [Jazz]



Obr. 4-10: Process View - definice procesu, který tým následuje, zdroj [Jazz]



Obr. 4-11: Detail work item typu úkol (task), zdroj [Jazz]

Měl jsem možnost několik měsíců pilotovat beta verzi RTC a často jsem spolupracoval s autory produktu (zpětná vazba, nové požadavky, reportování chyb), nyní pracuji s oficiální verzí 1.0 na denní bázi a už si moc nedovedu představit práci s jiným nástrojem, resp. kombinací jiných. Je možné vidět, kdo na čem dělá a mít tak přehled zda neděláme dva na 1 věci, výborná je podpora agilního plánování a redukce generování dokumentů (wiki, provázanost položek), při commitu do repozitory je automaticky k dané sadě přiřazen otevřený úkol, nástroj přímo si vynucuje pracovat podle námi definovaného procesu, je možné definovat RSS a různé pohledy, grafy.

Více informací o platformě Jazz a souvisejících aplikacích na [Jazz].

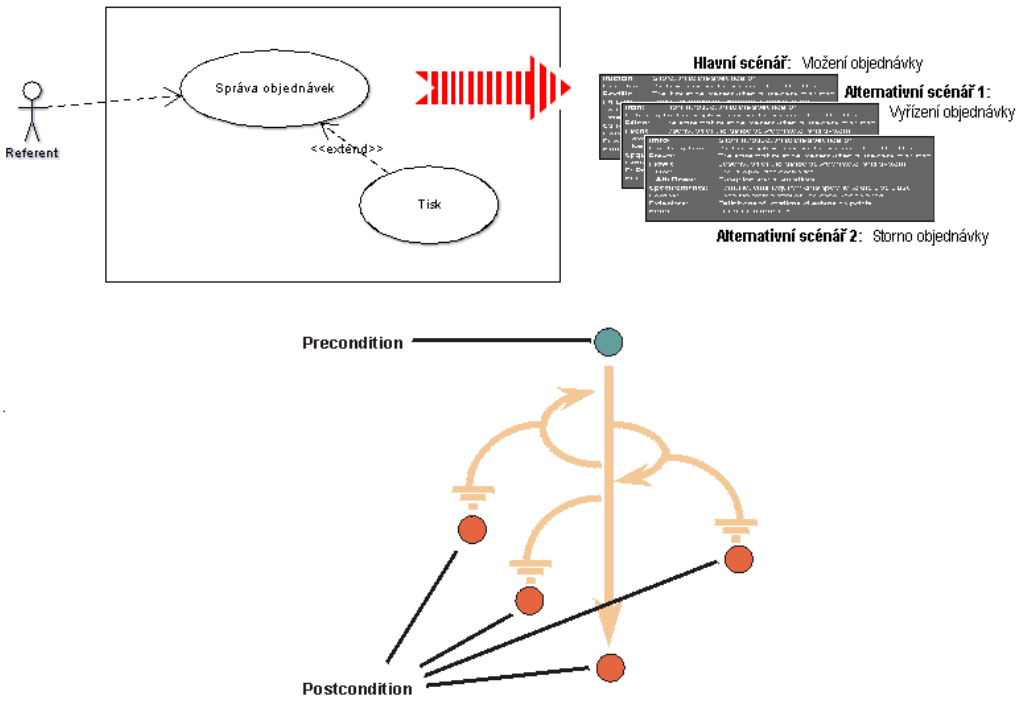
4.3.6 Souhrn – identifikace potřeb, projektový plán

V Inception fázi, v tzv. nulté iteraci projektu, tedy musíme porozumět tomu, co máme vytvořit (vize, definice rozsahu systému, jeho hranic), identifikovat zákazníka, tedy toho, kdo chce vytvářený systém a v neposlední řadě také to, co mu tento systém přinese. S daným zákazníkem či jeho/jejich zástupcem/ci identifikujeme klíčové funkcionality systému – tj. nejkritičtější use case. Ale pozor, ne všechny jejich scénáře a detailní popisy! Proto se ještě budeme v této kapitole stručně zabývat use casy a jejich scénáři jak tyto napasujeme do projektového plánu.

Následující obrázek ukazuje use case model, scénáře a toky scénářů. Use case model (vlevo nahoře) je pouze vysokoúrovňový model na systém, proto v něm

nezobrazujeme detaily, které by jeho čitelnost jen zhoršily a v této fázi projektu nám navíc nic podstatného nepřinesly. Model zachycuje use case „Správa objednávek“ a rozšiřující use case „Tisk“. Use case správa objednávek by šel zakreslit jako několik use case týkajících se objednávek, vytvoření objednávky, zpracování objednávky a storno objednávky. Měli bychom tedy již 4 use casy a velmi jednoduchý systém by už byl popsán docela složitým modelem. Ve skutečnosti tedy use case reprezentuje nějakou skupinu funkčností a tyto jsou potom detailněji popsány (ne však v use case modelu) textovými scénáři, jak to vidíme na obrázku. Scénáře jsou v našem případě tedy vytvoření, zpracování a storno objednávky (viz Obr. 4-12). Use case je možné si představit jako obálku a jednotlivé scénáře jako kartičky v ní zasunuté.

Navíc v následujících iteracích vývoje (hlavně v Elaboration) se například alternativními scénáři vůbec nebudeme zabývat a budeme je analyzovat, navrhovat, implmenetovat a testovat až v několika iteracích v Construction fázi.



Obr. 4-12: Vztah use case a jeho scénáře

Scénářů daného use case může být teoreticky nekonečně mnoho, obrázek výše znázorňuje toky scénářů a jejich varianty. Svislá šipka dolů značí hlavní tok, hlavní scénář (basic flow), ostatní šipky značí alternativní toky (alternative flow) a jejich návrat na hlavní linii nebo ukončení. Proto jsou také důležité ve slovních popisech scénářů počáteční (pre-condition) a koncové podmínky (post-condition), jelikož nám říkají, kde scénář začíná akde scénář končí.

Introduction:	Short introduction to the specification
Use Case Description:	Brief description conveys the purpose of the Use Case
Pre-condition:	The state that must be present when a use case may start
Flow of Event:	Description of the dialog between actor and system
Basic Flow:	The "Happy day scenario"
Alternative Flows:	Exception and alternatives
Special Requirements:	Nonfunctional requirements specific to the Use Case
Post-condition:	Possible states after at use case has finished
Extension Points:	Definitions of locations of extension points
References:	List of all references

Obr. 4-13: Možná forma slovního scénáře

Další chybou kromě výše popsané funkční dekompozice je také zahrnutí návrhových prvků do popisů scénářů, jedná se hlavně o pojmy jako obrazovka, tlačítko, listbox, databáze atd. Díky tomuto kroku už určujeme implementaci, když jsme teprve ve fázi analýzy! Což je zásadní problém, chyba.

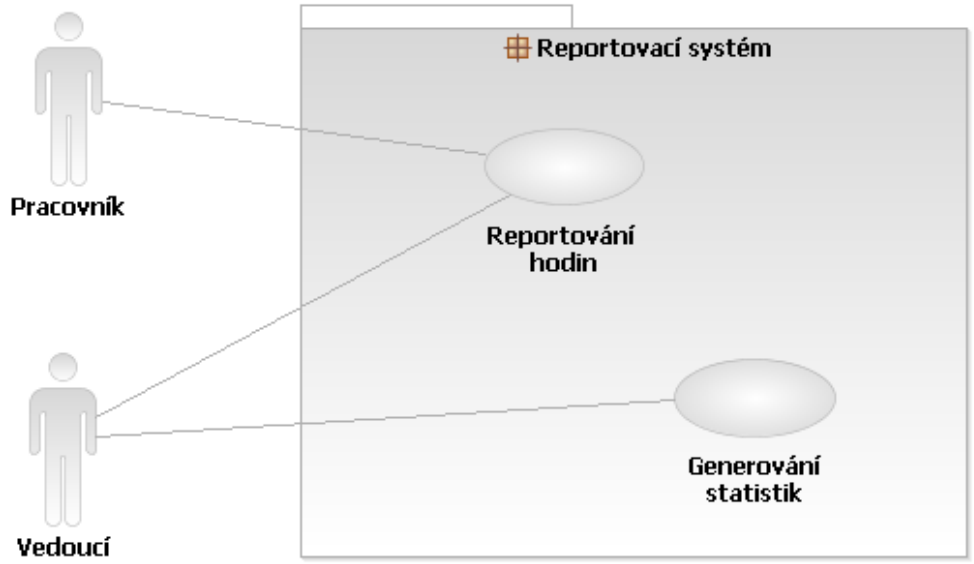
V dokumentu vize je ještě sumarizace potřeb uživatelů (v byznys doméně: např. potřebujeme zefektivnit vedení projektů) a jim odpovídající požadavky na software (tj. potřeby v technologické doméně: systém umožní vést a upravovat úkoly k definovaným projektům, …).

Potřeby	**Priorita**	**Požadavky**	**Plánovaná verze**
Automatické reportování hodin odpracovaných na projektech	vysoká	Systém umožní pracovníkům reportovat online hodiny strávené na projektech.	Verze 1.0
	vysoká	Hodiny je možné zadávat a aktualizovat pouze v rámci jednoho měsíce.	Verze 1.0
	nízká	Pro identifikaci uživatele je použito jednotné přihlášení z firemního LDAP serveru.	Verze 2.0
Jednoduché a přehledné vytváření statistik na základě reportovaných hodin	střední	Systém zobrazí vložené hodiny podle zadaného období a vybraného projektu.	Verze 1.0

Tabulka 4-5: Tabulka potřeb uživatelů a jim odpovídající požadavky na aplikaci

Spolu s budoucími uživateli (aktory) pak vypracujeme seznam use casů (s jejich scénáři prioritou a odhadem pracnosti), které je vhodné mít v nějaké formě tabulky či seznamu, jelikož s ním budeme neustále pracovat.

Identifikovaný UC model společně s aktory:



Obr. 4-14: Prvotně identifikovaný use case model

A následně rozpracovaný podrobný seznam use case a jejich scénářů s identifikovanými riziky, odhadem pracnosti, prioritou a spojenými riziky. Díky tomuto seznamu také lépe vidíme důležité věci z pohledu zákazníka (vysoká priorita) a také spojená rizika. Na základě kombinace těchto dvou hodnot jsem schopni identifikovat klíčové požadavky, které musíme implementovat jako první (z důvodu snížení rizik) a které budou tvořit architekturu systému, resp. na nichž bude architektura systému postavena.

UC	Scénář	Priorita	Odhad	Riziko	Nefunkční požadavky	Verze	
UC 1: Reportování hodin	BF (basic flow): Reportování hodin	vysoká	10 člověkodní	R1, R2	NF1 až NF5	1.0	
	AF1 (alternativní flow): Import dat	vysoká	8 člověkodní	R3, R4	Žádný, probíhá v dávkách	1.0	
	AF2: Vkládání více úkolů	nízká	8 člověkodní	-	NF1, NF4	1.0	
UC 2: Generování statistik	BF: Zobrazení hodin pro vybrané období	střední	5 člověkodní	R2	NF1, NF4, NF5	1.0	
	AF1: Export do Excelu	nízká	2 člověkodny	R3	Žádný, probíhá v dávkách	1.0	
	AF2: Bude upřesněno...	nízká	Bude upřesněno	-	…	2.0	

Tabulka 4-6: seznam use case a jejich scénářů

Identifikované scénáře use casů nyní podle priorit, hladiny rizika a odhadnuté pracnosti namapujeme do daných fází a jejich iterací. Podle rizik, kvality a

znalosti či neznalosti architektury a technologie, zkušenosti týmu apod. budeme také definovat počty a délku daných iterací.

Ukázka jednoduchého projektového plánu:

Fáze	Iterace	Primární úkoly	Datum od - do
Inception	I0	Sestavení týmu, rozpočtu Identifikace základních požadavků a rizik Výběr kandidátů architektury LCO (shoda na vizi, rozsahu, rozpočtu)	02.01.2008 - 14.01.2008 14.01.2008
Elaboration	E1	UC1 [BF]: Reportování hodin UC1 [AF1]: Import dat	14.01.2008 - 01.02.2008
	E2	UC2 [BF]: Zobrazení hodin pro vybrané období	01.02.2008 – 18.02.2008
	E3	Rezerva (5 dní)	18.02.2008 – 24.02.2008
		LCA (snížena rizika, implementována a otestována architektura)	24.02.2008
Construction	C1	UC1 [AF2]: Vkládání více úkolů	25.02.2008 – 09.03.2008
	C2	UC2 [AF1]: Export do Excelu	10.03.2008 - 23.03.2008
	C3	Rezerva (2 dny)	24.03.2008 – 25.03.2008
		IOC (beta release)	26.03.2008
Transition	T1	Akceptační testování zadavatelem ... PR: Předání projektu	13.05.2008 - 07.06.2008

Tabulka 4-7: Projektový plán

Z projektového plánu je zřejmé, že plánujeme opustit Inception fázi co nejdříve, ať můžeme ihned něco implementovat (= identifikovat a analyzovat detailněji vybraný scénář, navrhnout jeho implementaci, implementovat, sestavit a testovat) a také jsme se jsme zaměřili na rizikové scénáře, ty jsou implementovány nejdříve, abychom měli jistotu, že jsme snížili či odstranili dané riziko a scénář půjde implementovat. Jinak máme pořád ještě dost času se s nalezeným problémem vypořádat, změnit technologii, navrhovanou architekturu či rozsah řešení.

Ještě zmíníme, že tento dokument není rozhodně zamrazen a již více neměněn. Projektový plán je aktualizován podle potřeby, kdykoliv je realita rozdílná od vytvořeného plánu, kdykoliv nám nějaký problém či riziko ztíží postup atd.

4.3.7 Milník LOM

Na konci Inception fáze následuje první milník projektu nazýván Lifecycle Objective Milestone (LOM). Milník je určen ke zhodnocení cílů celého projektu. Pokud nejsme schopni tohoto milníku dosáhnout, měl by být projekt zrušen nebo přerušen. Důvodem neshody může být neshoda na rozsahu

funkčností produktu, přílišné náklady, neexistující technologie schopná dostát našim požadavkům, nevýhodnost/nevratnost investice apod.

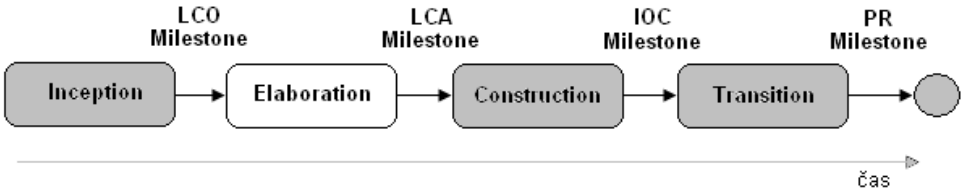
Pokud je produkt odsouzen k záhubě, je lepší ho ukončit dříve, než později, jelikož nás tím pádem připraví o míň peněz. Iterativní přístup společně s definovanými milníky nám umožňuje identifikovat tuto skutečnost dříve, než například ve vodopádovém přístupu.

LOM milník definuje následující evaluační kritéria:

- Shoda účastníků projektu (samozřejmě včetně zákazníka) na rozsahu projektu, počátečních nákladech a odhadu plánu, který bude dále upřesňován.
- Shoda na identifikaci správných požadavků a porozumění jim.
- Shoda na věrnosti odhadů nákladů a plánu, prioritách, rizicích a vývojovém procesu.
- Shoda na tom, že počáteční rizika byla identifikována a existují strategie na jejich snížení.

4.4 Elaboration phase

Po úspěšném projití první fází máme tedy představu o tom, co budeme vyvíjet, kdo budou uživatelé a co jim má nový systém přinést. Na této představě jsme se shodli se všemi účastníky projektu. Dále máme identifikovány klíčové funkčnosti a tyto stručně popsány. Navrhli jsme alespoň 1 možné architektonické a technologické řešení, vytvořili jsme hrubý plán projektu a identifikovali náklady. V neposlední řadě máme definovaný proces vývoje a nastavené prostředí a nástroje.



Obr. 4-15: Fáze Elaboration

Pokud jsme dosáhli prvního milníku (Lifecycle Objective Milestone – LOM), přechází projekt do druhé fáze zvané Elaboration. Cílem této fáze je definovat a nastínit architekturu systému, abychom na jejím základě mohli v následující fázi navrhnout a implementovat zbývajících 70-80% nekritických Use Casů (funkčností). Jak již bylo řečeno, architektura se vyvine z nejdůležitějších požadavků (z těch, které na ni mají dopad) a také z ohodnocení rizik. Cílem této fáze je hlavně snížení či odstranění rizik, a to ve čtyřech hlavních oblastech:

- Rizika spojená s požadavky na systém (Vyvíjíme správnou aplikaci?).
- Rizika architektonická (Poskytujeme správné řešení?).
- Rizika spojená s náklady a plány.
- Rizika procesní a spojená s prostředím a nástroji (Máme správný proces a nástroje?)

V následujícím textu se budeme podrobněji opět věnovat každému z cílů této fáze. Nejdříve však vysvětlíme, kolik iterací je vhodné naplánovat pro tuto fázi v případě různých typů projektů.

4.4.1 Iterace

Většinu rizik snížíme tím, že v této fázi vyprodukujeme spustitelnou architekturu našeho výsledného řešení. Tímto konkrétně demonstrujeme jeho proveditelnost a nespoléháme na možné mylné předpoklady. Pokud navrhujeme systém s použitím stejné technologie jako v předchozích projektech, jsme schopni cíle fáze naplnit většinou v jediné iteraci, jelikož existuje menší množství rizik, které potřebujeme odstranit. Navíc můžeme znovupoužít řešení či komponenty z předchozích projektů, což urychluje náš postup.

Naopak, pokud nemáme zkušenosti s danou problémovou doménou, pokud je systém velmi komplexní nebo pokud používáme novou technologii, bude zapotřebí dvou či tří iterací k vytvoření stabilní architektury a zmírnění největších rizik. Dalšími přispívateli k většímu počtu iterací jsou distribuovaný vývoj, velký počet uživatelů systému, bezpečnostní požadavky a další.

První iterace v Elaboration by měla zahrnovat:

- Návrh, implementace a testování malého počtu kritických scénářů, pomocí kterých identifikujeme typ architektury a potřebné mechanismy, rozhraní. Toto se snažíme provést co nejdříve z důvodu snížení největších rizik.
- Identifikace, implementace a testování malé množiny základních mechanismů v architektuře.
- Počáteční hrubý návrh logické struktury databáze.
- Detailní vypracování událostí hrubé poloviny Use Casů, které zamýšlíme detailně popsat v Elaboration fázi (podle priorit).
- Důkladnější testování pro ověření architektury a ujištění se, že největší architektonická rizika byla snížena na únosnou mez.

Druhá iterace v Elaboration by měla zahrnovat:

- Oprava všeho, co nebylo správné v první iteraci.
- Návrh, implementace a testování zbývajících architektonicky významných scénářů.
- Nastínění a implementace paralelismů, procesů, threadů na takové úrovni, která je potřeba k identifikaci potenciálních technických problémů. Zaměření této iterace je také na testování výkonnosti, zátěže a rozhraní mezi subsystémy, stejně jako na externí rozhraní.
- Identifikace, implementace a testování zbývajících mechanismů v architektuře.
- Návrh a implementace první verze databáze.
- Detailní popis zbývající poloviny Use Casů, které chceme blíže specifikovat v Elaboration.
- Testování, ověření a úpravy architektury do její stabilní verze, na ni pak můžeme dále navěsit další funkčnosti systému.

Pokud v těchto iteracích přijdou zásadnější zásahy do architektury např. vinou změnových požadavků, je vhodné přidat další iteraci, abychom měli jistotu (výsledky testů), že je architektura opravdu správná a stabilní. Toto pravděpodobně způsobí zpoždění projektu, ale je to o mnoho levnější, než celé řešení stavět na tekutých píscích (rozuměj na nestabilní architektuře).

4.4.2 Cíl 1 – podrobnější pochopení požadavků

V úvodní fázi jsme definovali vizi produktu a detailně popsali přibližně 20% těch nejdůležitějších Use Casů (z hlediska architektury). Tento cíl říká, že v průběhu Elaboration budeme podrobněji popisovat další Use Casy. Některé z nich mohou být natolik jednoduché nebo pouze používající jiná data, že je posuneme až do Construction fáze nebo dokonce nemusí být formálně vůbec popsány. Jejich detailní popis totiž nepřinese žádný přínos ke snížení nějakého rizika. Předmětem Elaboration může být také konstrukce prototypu uživatelského rozhraní pro významné Use Casy, nad kterým si budeme následně s uživateli vyjasňovat funkcionalitu, kterou mají Use Casy poskytovat.

Pokud se jedná o složitější problémovou doménu, můžeme vytvořit doménový model pro popis vztahů jednotlivých entit a samozřejmě doplnit či opravit doménový slovník, který byl vytvořen v Inception fázi.

Pro detailní popis jednotlivých Use Casů je vhodné si vytvořit časově omezený úsek (stejně jako v Inception), abychom nezabředávali do přílišných detailů. Pokud se jedná o menší projekt a implementaci bude provádět stejný člověk, který specifikuje Use Casy, je možné dokumentací jejich popisu strávit méně času a vrátit se k němu až po implementaci a otestování daných UC. Na konci Elaboration by mělo být popsáno přibližně 80% Use Casů, některé nové UC je možné nalézt i v Construction, nemělo by to však být pravidlem.

4.4.3 Cíl 2 – návrh, implementace a ověření architektury

Jelikož tento cíl říká, že v průběhu Elaboration máme navrhnout, implemenovat a ověřit základ celého řešení, tzv. architekturu, řekneme si nejdříve stručně, co to architektura je. Architektura je část systému, část řešení, řeší komunikaci, ukládání, uživatelské rozhraní, technologie a podobné věci, ale pouze v míře nezbytně nutné na základě kritických (nejdůležitějších) Use Casů. Při tvorbě architektury uvažujeme následující:

- subsystémy řešení (stavební bloky) a jejich rozhraní,
- jejich interakce za běhu programu pro naplnění identifikovaných scénářů,
- implementace a testování prototypu (rizika snížena, ověřeno řešení, výkonnost, škálovatelnost, náklady).

Proto abychom byli schopni ověřit správnost a proveditelnost navržené architektury, potřebujeme více než jen revidované modely či stránky papíru. Potřebujeme spustitelnou architekturu, kterou můžeme testovat, tj. spustitelný kód. Nyní představíme několik kroků, které je vhodné provést (samozřejmě

opět podle typu projektu) pro dosažení spustitelné, stabilní a testovatelné architektury.

Prvním krokem je definice subsystémů, klíčových komponent a jejich rozhraní, pomocí kterých budou komunikovat. V tomto kroku je vhodné uvažovat použití existujících frameworků (z předchozích projektů, či existujících na trhu) předtím, než budeme na zelené louce navrhovat architekturu vlastní. Potenciálními zdroji pro identifikaci komponent jsou doménové objekty z doménového modelu.

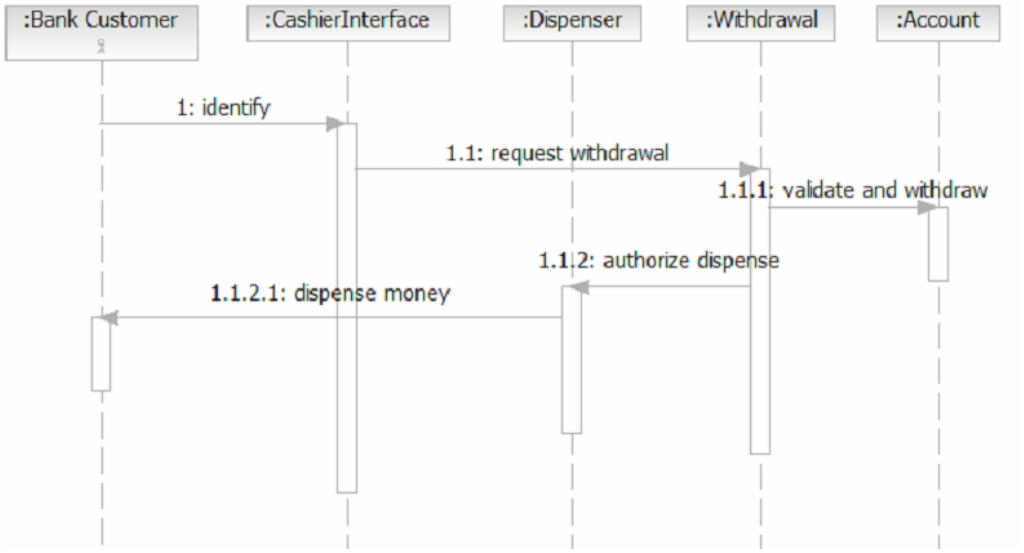
Dalším krokem je použití architektonicky významných Use Casů pro definici architektury. Jak již bylo řečeno, je to typicky 20-30% Use Casů, které jsou kritické. Je třeba brát v úvahu taktéž nefunkční požadavky a nalézt a implementovat Use Casy, které odhalí potencionální problémy a rizika a ověří jejich proveditelnost. Pokud mluvíme o implementaci Use Casů máme v těchto fázích na mysli pouze 1 nebo 2 scénáře. To znamená ideální („happy day“) scénář + například jeden chybový pro ověření způsobu zachycení a zpracování výjimek. Na paměti je třeba mít účel těchto všech kroků ve Elaboration fázi, tím je snížení rizik našeho řešení na akceptovatelnou úroveň či jejich úplné odstranění.

Dalším krokem je ***návrh (design) kritických Use Casů***, jinými slovy také realizace Use Case. Ten popisuje, jak jsou konkrétní objekty realizovány v návrhovém modelu z pohledu jejich spolupráce. Toto může být provedeno formálně či opět neformálně, pouze náčrtky na tabuli nebo ve vizuálním modelovacím nástroji, Obr. 4-16 ukazuje zápis pomocí UML sekvenčního diagramu. Návrh probíhá v několika krocích:

1. Nástin analytických objektů.
2. Definice chování jednotlivých analytických tříd (jejich odpovědnost).
3. Detailní popis těchto tříd (přesnější pochopení odpovědností).
4. Návrh Use Casů – komunikace (pořadí, způsob, ...).
5. Rozpad (upřesnění) analytických tříd na návrhové.

Konsolidace a seskupení identifikovaných tříd do balíčků je dalším krokem. Tyto balíčky nebo subsystémy vytváříme podle několika aspektů:

- Předměty budoucích častých změn do 1 balíčku (např. rozhraní aktora).
- Pravidla viditelnosti (vícevrstvá aplikace nebude obsahovat v 1 balíčku třídy z více vrstev).
- Budoucí konfigurace produktu (výsledný produkt může být skládán z různých částí aplikace).



Obr. 4-16: UML sequence diagram – transformace slovních scénářů do formy analytických objektů.

Dalším důležitým krokem je návrh databáze, jelikož většina dnešních systémů využívá pro persistentní objekty některý ze systémů řízení báze dat (SŘBD). Cílem je pochopení, jak budou persistentní data ukládána a opět vyvolávána zpět (mechanismus a technologie).

Jak již bylo naznačeno v předchozím kroku, jsou důležitým aspektem architektury také různé mechanismy v architektuře. Jedná se o běžné situace a problémy, které jsou řešeny například pomocí návrhových vzorů (persistence, autorizace, komunikace, ...).

Integrace komponent je předposledním bodem, který zmíníme. Integrace určuje v jakém pořadí a jaké komponenty budou integrovány. Toto definujeme paralelně s identifikací analytických tříd. Integrace komponent je důležitá, jelikož sestavené a kompilované komponenty jsou předmětem testování, abychom viděli, zda splňují požadované chování a výkonnostní či bezpečnostní požadavky. Integrace je neustálou aktivitou, kterou provádíme v průběhu všech iterací a to většinou denně (např. night builds) minimálně však alespoň 2x týdně. Kritickými faktory této aktivity je fungující konfigurační management stejně jako automatizované buildy.

Testování kritických scénářů je posledním krokem splňující cíl 2. Testování je totiž kritickým aspektem Elaboration. Je to ukazatel postupu projektu v čase (co je již hotovo, otestováno, kde zbývají problémy apod.). Nejlepší cestou k přesvědčení se, že máme snížena všechna důležitá rizika, je otestovat spustitelnou architekturu, což znamená:

- Kritické scénáře byly správně implementovány a poskytují předpokládanou funkčnost.
- Architektura poskytuje dostatečnou výkonnost.
- Architektura podporuje a je schopna pojmout nezbytnou zátěž (např. 1000 současných uživatelů).
- Rozhraní s externími systémy fungují tak, jak bylo předpokládáno.

- Byly testovány také ostatní nefunkční požadavky, které nebyly zachyceny a popsány výše.

Je třeba si uvědomit, že pokud jsme prošli až sem, tak máme některé části systému vyvinuté v docela pokročilém stupni, stále ale máme implementováno pouze 10-20% celého řešení. Většinou se jedná o úspěšné scénáře 20-30% Use Casů. Provedli jsme tedy něco ode všech disciplín, ale stále nám zbývá nějakých 80% systému navrhnout a implementovat. Dobrou správou je, že tato část byla tou nejnáročnější z celého systému, díky tomu jsme snížili nejvýznamnější rizika projektu.

4.4.4 Cíl 3 – vytvoření přesnějšího plánu a odhad nákladů

Na konci Elaboration máme přesnější informace, které nám dovolí aktualizovat a upřesnit projektový plán a odhad nákladů. Máme totiž již:

- vytvořen detailní popis požadavků – rozumíme přesně jaký systém budeme vytvářet,
- implementovánu kostru řešení (architekturu),
- sníženu většinu rizik (což redukuje nadhodnocení či podhodnocení odhadů plánů a nákladů),
- přesnější porozumění, jak efektivně pracuje náš tým pomocí definovaného procesu s danými nástroji a s danou technologií (jelikož jsme prošli celý životní cyklus již minimálně jednou).

Nyní tedy zpřesníme odhady nákladů a plánů projektu (podle rozsahu projektu to mohou být následující dokumenty: Vision, Business Case, Software Development Plan, Project Plan), aktualizujeme seznam rizik a akcí na jejich odstranění/snížení.

4.4.5 Souhrn – příklad Elaboration iterace

Následující příklad ukazuje možný iterační plán pro první iteraci v Elaboration. Projektový plán (viz kapitola o Inception) zobrazoval jako náplň této iterace hlavní tok (BF) use case UC1: Reportování hodin v termínu od 14.1.2008 do 1.2.2008. Tento scénář implementujeme jako první proto, protože tím snížíme podstatná rizika a navíc přineseme uživateli největší hodnotu = hotová nejdůležitější funkčnost. Iterační plán by tedy mohl vypadat následovně:

Plán iterace E1
Začátek iterace (plánovací meeting dané iterace): 14.1.2008
Konec iterace (demo, assessment): 1.2.2008

Cíle iterace:

- Implementace UC1 [BF].
- Odstranění rizika R1 a R2.

Evaluační kritéria:

- 60 % kódu pokryto unit testy.
- 100 % unit testů prošlo.
- 70 % implementovaných funkčních testů prošlo.
- Sníženo riziko R1.

- Bylo předvedeno demo zákazníkovi.
- Zákazník demo akceptoval.

Úkoly:

Název / Popis	Priorita	Odhad (body)	Přiřazeno	Odhad (hodin)
Instalace build mechanismu (Ant)	2		Johan	70
Analýza scénáře UC1 (BF)	2	8	Lisa	
Návrh scénáře a implementace		8	Lisa, Ann, Johan	12
Implementace a testy server části			Ann	14
Implementace a testy klient části			Johan	28
Sestavení dema pro assessment	3	5	Johan	18
Vytvoření uživatelské dokumentace	2	5	Lisa	65
Vytvoření install manuálu	2	1	Lisa	5
Vytvoření release notes	2	1	Johan	4
Vytvoření online help stránek	3	2	Ann	22

Iterace začíná plánovacím meetingem jehož se účastní celý tým, ten poté rozpracuje cíle iterace na jednotlivé úkoly a jim se přiřadí řešitelé, dohodnou se se zákazníkem evaluační kritéria (+ možnost interních kritérií) a začně se daný scénář analyzovat, navrhovat, implementovat, paralelně s tím vznikají test casy (testovací scénáře), které jsou postaveny nad use casy, resp. znovupoužívají jejich scénáře. Výsledkem iteračního plánovacího meetingu jsou výše zmíněné cíle, evaluační kritéria a úkoly zachycené na wiki či v nějakém dokumentu a uložené v repozitory.

Na konci iterace, tj. v den kdy je stanoveno v iteračním plánu, je plánována ukázka dema zákazníkovi a proveden týmový assessment (ohodnocení) vůči definovaným kritériím. Pokud jsme náhodou nestihli něco naprogramovat, otestovat, vytvořit, pak konec iterace neposunujeme. Není to vhodné, raději definujeme iteraci jako neúspěšnou či pouze částečně úspěšnou. Na základě zjištěných příčin jsme pak schopni se poučit a přijmout opatření. Samozřejmě je nutné nehotové věci dodělat, a to buď v následující plánované iteraci (pokud je třeba pouze odstranit nějaké menší chyby) nebo musíme naplánovat novou iteraci s podobnými cíli (v případě velkých problémů či nehotového řešení). Výsledkem assessmentu je pak následující zápis opět na týmové wiki, sharepointu, word dokument či v nějaké jiné formě zachycené:

Assessment iterace E1 (vyplněn až na konci iterace při assessmentu)
Ohodnocení cílů podle evaluačních kritérií:

- 70% pokrytí kódu unit testy.
- 98 % unit testů prošlo.
- Riziko R1 sníženo, aplikace komunikuje s databází bez potíží, data jsou ukládána i zobrazena ve správném formátu (provedena kontrola uložení i v DB).
- Zákazník byl mile překvapen existujícími funkčnostmi takhle brzy (a akceptoval demo) a byl dohodnut pilotní provoz reportování již po skončení fáze Elaboration.

Neopravené chyby:

- ERR0012: Null pointer při určitém stavu… (ve formě odkazu na Bugzillu, Jiru či jiný nástroj pro vedení evidence chyb, tzv. issue tracking tool).

Jiné poznámky a poznatky:
Při ukázce dema zákazníkovi dne 1.2.2008 si reprezentant zákazníka uvědomil (díky demonstraci a krátké hře s aplikací), že zapoměl jako jeden z požadavků zmínit nutnost propojení budoucího systému s jeho existujícím logovacím systémem zaznamenávajícím činnost uživatelů. Zmínil pouze vazbu na LDAP server. Tento požadavek proto musí být zapracován do následujících fází projektu. Díky navržené rezervě jsme schopni tento důležitý požadavek realizovat v Elaboration (nutné zde – má vliv na architekturu) bez přesunu jiných požadavků do následující verze.

Lessons learnt (poznatky):
…

Zapracovanou skutečnost zobrazuje aktualizovaný projektový plán:

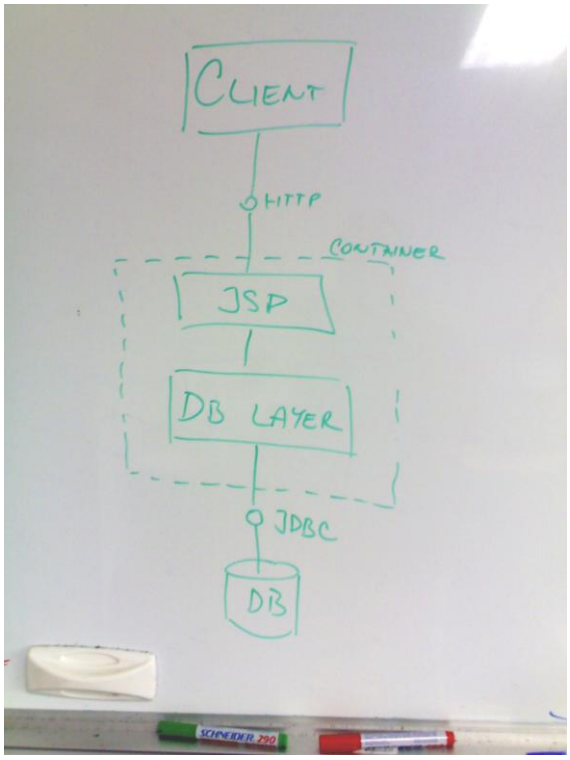
Fáze	Iterace	Primární úkoly	Datum od - do
Inception	I0	Sestavení týmu, rozpočtu Identifikace základních požadavků a rizik Výběr kandidátů architektury LCO (shoda na vizi, rozsahu, rozpočtu)	02.01.2008 - 14.01.2008 14.01.2008
Elaboration	E1 E2 E3	UC1 [BF]: Reportování hodin UC1 [AF1]: Import dat UC2 [BF]: Zobrazení hodin pro vybrané období Identifikován nový scénář => aplikována rezerva UC1 [AF3]: Vytváření projektů Rezerva (1 den) LCA (snížena rizika, implementována a otestována architektura)	14.01.2008 - 01.02.2008 01.02.2008 – 18.02.2008 18.02.2008 – 24.02.2008 24.02.2008
Construction	C1 C2 C3	UC1 [AF2]: Vkládání více úkolů UC2 [AF1]: Export do Excelu Rezerva (2 dny) IOC (beta release)	25.02.2008 – 09.03.2008 10.03.2008 - 23.03.2008 24.03.2008 – 25.03.2008 26.03.2008
Transition	T1	Akceptační testování zadavatelem ... PR: Předání projektu	13.05.2008 - 07.06.2008

Tabulka 4-8: Aktualizovaný projektový plán

V další iteraci musíme naplánovat nejdříve opravu chyb, pokud nějaké zůstaly z předchozí iterace, poté případný nutný refaktoring (zapracování návrhového vzoru) a až poté vlastní implementace nového scénáře. S touto prací jakoby navíc musíme počítat i v našich odhadech pracnosti na následující iterace.

Aktualizujeme tedy nejen projektový plán, ale také risk list, UC model a seznam use casů!!!! V průběhu dané iterace pak také připravujeme hrubý iterační plán pro následující iteraci.

V průběhu Elaboration fáze řešíme architekturu systému a odstranění rizik převážně technického rázu. Jak tedy můžeme definovat architekturu, kterou budeme následně implementovat a testovat? Jednou z možností je následující:



Obr. 4-17: Další ukázka kandidáta architektury – logický pohled (vrstvy a technologie).

Není třeba vytvářet detailní modely v drahých nástrojích. Stejně tak nám poslouží v jednodušších případech či jako vysokoúrovňový model náčrtek na tabuli následně uložený v repozitory. Jelikož jde však o architekturu, mělo by být zřejmé, co která vrstva znamená (UI, aplikační logika, databázová vrstva, webová služba, vzdálená procedura, vnořený systém, …), jak spolu komunikují (protokol, rozhraní, vrstva) a také, kde která část běží (kontainer, klient, web server, …). Pokud model architektury takto nepopíšeme, nebude mít legendu, jedná se jen o bezcený cár papíru, který neřekne čitateli vůbec nic.

Dokumentace architektury

Jedná se o popis architektury systému, východisek, možných řešení. Opět nemusí být ve formě dokumentu ale například na wiki. Stává se ale základem

pro následnou údržbu a pochopení systému, takže by bylo dobré obsah tohoto dokumentu mít opravdu pokrytý v jakékoliv formě a také aktuálně udržovaný.

Obsahem by mělo být následující:

- Cíle a omezení systému z technického hlediska.
- Přehled všech use case a jejich scénářů – z důvodu definování UC, které mají dopad na architekturu.
- Možní kandidáti architektury, jejich výhody, nevýhody, omezení.
- Logický pohled (vrstvy a použité technologie) vybraného řešení či všech kandidátů.
- V průběhu projektu podle potřeby doplňujeme identifikované závislosti, návrhový model, model nasazení, kvalitativní atributy (výkonnost, dostupnost, spolehlivost, bezpečnost, …).

4.4.6 Milník LCA

Milníkem Elaboration fáze je tzv. Lifecycle Architecture milestone (LCA). Nyní máme detailně prozkoumány cíle a rozsah systému, vybranou architekturu a identifikována a snížena největší rizika. Opět platí, že pokud nejsme schopni dosáhnout tohoto milníku, je vhodné projekt ukončit.

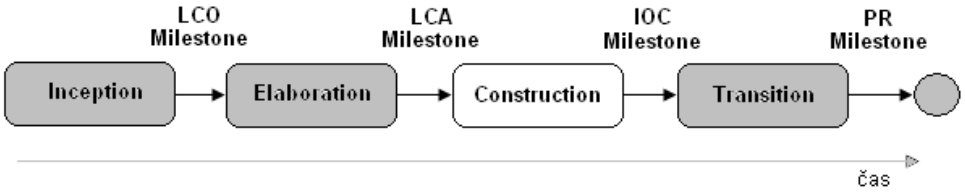
Zda jsme dosáhli tohoto milníku nám pomůže zjistit následující kontrolní seznam:

- Je vize produktu stabilní, jsou stabilní požadavky?
- Máme stabilní architekturu?
- Jsou klíčové postupy a přístupy, které budeme používat, otestovány a je dokázána jejich použitelnost?
- Ukázalo testování spustitelného prototypu, že jsou klíčová rizika identifikována a vyřešena?
- Máme definovány plány iterací pro následující Construction fázi v náležitých podrobnostech, abychom byli schopni podle nich postupovat?
- Jsou tyto plány podpořeny důvěryhodnými odhady?
- Naplněním plánu s použitím definované architektury dosáhneme cílů shrnutých ve vizi?
- Jsou aktuální náklady akceptovatelné vůči plánovaným?

Tato revize může trvat pro rozsáhlejší projekty den i více. Menší projekty mohou provést ohodnocení během hodinového sezení.

4.5 Construction phase

Fáze Elaboration byla ukončena interním releasem základní, spustitelné architektury systému, která umožnila identifikovat a implementací přímo ověřit největší technická rizika (soupeření o zdroje, výkonnostní rizika, zabezpečení dat, …). Následující fáze zvaná Construction, která je předmětem této kapitoly, je zaměřena na detailní návrh, implementaci a testování, aby bylo zajištěno zhmotnění kompletního systému.



Obr. 4-18: Fáze Construction

Předmětem prací v této fázi je návrh a implementace zbývajících přibližně 80% Use Case a finální implementace původních 20%, které představují kritické (hlavní) požadavky zákazníka. Dosud byla implementována pouze malá podmnožina z celkového kódu aplikace. Tato fáze je proto také časově nejnáročnější a účastní se jí největší počet lidí, hlavně programátorů a testerů. V průběhu Construction budou identifikována další rizika, na která se musíme zaměřit. Neměla by však být kritického rázu a tudíž by měla mít pouze malý vliv na architekturu systému. Pokud by tomu bylo naopak, značí to nekvalitní práci v předchozí fázi. Kritickými faktory úspěchu pro tuto fázi jsou zajištění celistvosti architektury, paralelní vývoj, správa konfigurací a změnové řízení (Configuration & Change Management) a v neposlední řadě automatizované testování. Zajímá nás také správná rovnováha mezi kvalitou, záběrem systému (jeho rozsáhlostí) časem a detailností či dokonalostí implementovaných požadavků.

Cíle Construction fáze lze definovat následovně:

- Minimalizace nákladů na vývoj, dosažení určitého stupně paralelního vývoje (pro efektivnější využití zdrojů).
- Iterativní vývoj kompletního produktu, který bude připravený k doručení uživatelské komunitě. (beta release – první funkční verze aplikace)

4.5.1 Iterace

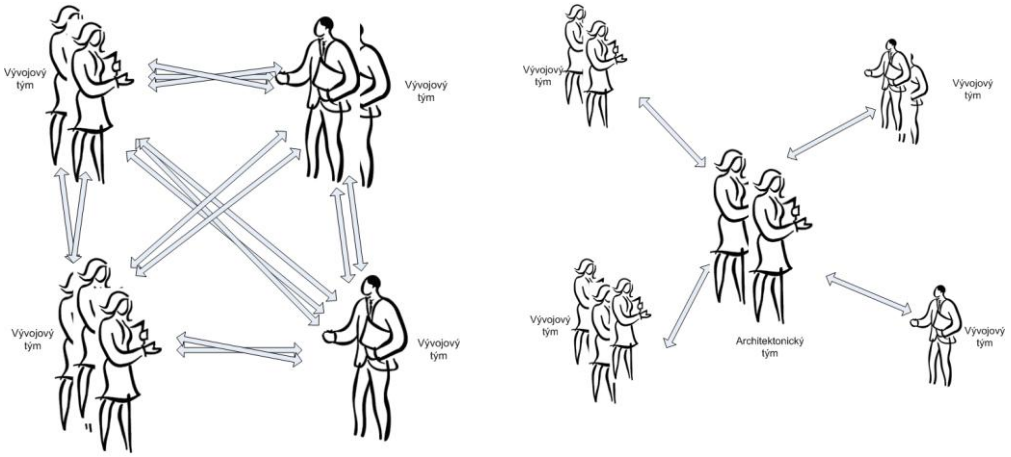
Počet iterací této fáze se bude opět lišit projekt od projektu, v zásadě lze říci, že jich bude více než v jiných fázích, typicky 2-4. Plánování iterací bude opět řízeno Use Casy, kdy nejdříve budeme implementovat ty nejdůležitější pro zákazníka, či technicky nejrizikovější, což může znamenat implementaci pouze některých scénářů (hlavně u technických rizik). Řečená zásada se týká hlavně první iterace v této fázi.

4.5.2 Cíl 1 – minimalizace nákladů na vývoj, paralelní vývoj

Cílem správně provedené Elaboration fáze je základní architektura systému, která je otestovaná a spustitelná. Cílem bylo vytvořit kostru komunikačních mechanismů, ukládání a správu dat a další. Pokud byla architektura navržena správně a je robustní, je nyní jednodušší pokračovat ve vývoji, jelikož tyto mechanismy můžeme využívat a znovupoužít, další kód je na tuto architekturu „navěšen“.

Jednou z výhod existence architektury systému je jasná definice odpovědností částí systému rozdělených do dobře definovaných subsystémů. To umožní

jednotlivým vývojovým týmům při paralelním vývoji nezasahovat si do svých subsystémů. Samozřejmě, vývojáři musí rozumět celému systému, ale měli by mít přidělenou určitou část, podsystém, na kterém pracují.



Obr. 4-19: Organizace kolem architektury minimalizuje přílišné komunikační zatížení

Tento způsob je nazýván organizace kolem architektury a snaží se efektivně nahradit komunikaci tváří v tvář, která je důležitá, ale v případě velkého vývojového týmu by neúměrně narostla (geometricky!) a snížila efektivitu vývojového týmu. Toto můžeme omezit existencí jednoho týmu, který je odpovědný za architekturu a několika podtýmů odpovědných za jeden nebo několik podsystémů. Komunikace mezi těmito týmy je pak zprostředkována týmem odpovědným za architekturu, jelikož může řešit problémy a těžkosti spojené s celkovým řešením, stejně jako s jednotlivými rozhraními a například mít poslední slovo (rozhodovat o jejich struktuře).

Velmi důležitým aspektem této fáze je také správa konfigurací (CM – Configuration Management). CM je definován a vybudován ve fázi Inception a vyladěn ve fázi Elaboration. V průběhu vývoje vzniká spousta různých souborů budoucí aplikace. Sledovat všechny jejich verze a změny je velmi složité, zvláště v případě iterativního vývoje, kdy neustále vytváříme nové verze, provádíme jejich integraci a testování. CM nám umožní jít zpět k posledním fungujícím verzím, umožní sestavovat build ze správných verzí či umožní přístup k některým souborům pouze vybrané skupině vývojářů. Existuje-li funkční správa konfigurací, mohou se vývojáři věnovat jen a pouze vlastnímu vývoji a tím zvýšit jeho efektivitu.

Proto, abychom mohli těžit z výhod architektury, je nutné architekturu aktivně prosazovat. Jak již bylo zmíněno, architektura jsou vlastně znovupoužitelná řešení v aplikaci, jedná se hlavně o komunikační kanály, ukládací či serializační mechanismy apod. Je třeba zabránit vývojářům, aby tyto mechanismy znovu programovali, vynalézali. Toho můžeme docílit tréninkem zaměřeným na architekturu společně s revizemi návrhu. Důležité je také hlídat každou změnu v rozhraní, což by mohlo způsobit problémy v komunikaci s jinými subsystémy. Jednoduché a elegantní řešení je mít rozhraní systému pod konfigurační správou (CM).

Pro zajištění nepřetržitého postupu při vývoji nové aplikace v Construction fázi je třeba naplnit krátkodobé cíle, mezi něž patří:

- Vytvoření jednoho týmu s jedním úkolem – je třeba předejít funkčním týmům, kde jsou analytici v jednom týmu a vytvořenou dokumentaci „hodí přes zeď“ vývojářům atd. Cílem je mít více-funkční týmy, kde každý cítí odpovědnost za aplikaci a za postup, k tomu pomohou mimo jiné také denní krátké schůzky, kde tým diskutuje současný stav a problémy a také na co se zaměřit v příštích iteracích.
- Nastavení jasných cílů pro vývojáře, co dokončit v této iteraci.
- Průběžně demonstrovat a testovat kód – toto je jediné měřítko postupu, ne říci 90% hotovo, pouze demonstrovatelná aplikace je brána jako měřítko postupu.
- Průběžná integrace – pokud je to možné, je vhodné dělat denní buildy.

4.5.3 Cíl 2 – Iterativní vývoj kompletního produktu

V průběhu Elaboration jsme detailně popsali pouze kritické Use Case nebo ty, které mají vliv na architekturu. Méně významné UC s malým dopadem na architekturu byly přesunuty do fáze Construction. Jedná se hlavně o UC, jejichž funkcionalita je podobná jako již implementovaných UC, ale využívají se při tom jiné entity, datové objekty, aktoři či rozdílné uživatelské rozhraní (UI).

Zdůrazníme, že v této fázi musí existovat pravidelná komunikace a otevřený dialog mezi analytiky a vývojáři. Analytici jsou odborníci na interpretaci nejasných požadavků businessu ve formě požadavků, ale nejsou znalí, či schopni přijít s optimálním řešením, kdežto vývojáři jsou schopni vidět či nalézt nové, progresivní řešení, proto je spolupráce těchto rolí klíčová nejen v Construction ale také v Inception a hlavně Elaboration.

V Elaboration jsme rozdělili systém na subsystémy a definovali jsme klíčové komponenty a jejich rozhraní a také mechanismy architektury. V každé iteraci v Construction se zaměřujeme na dokončení návrhu určité skupiny komponent a subsystémů a určité skupiny Use Case. V prvních iteracích Construction se zaměřujeme opět na snížení či odstranění nejrizikovějších věcí (jedná se hlavně o rozhraní, výkonnost, požadavky a použitelnost). V pozdějších iteracích v Construction se zaměřujeme na úplnost, kdy navrhujeme, implementujeme a testujeme veškeré nutné či možné scénáře vybraných Use Case.

V průběhu Elaboration jsme navrhli a implementovali koncept databáze. Nyní v Construction můžeme přidat další sloupce do tabulek, definovat nové pohledy (view), indexy pro efektivnější vyhledávání a optimalizaci výkonu apod. Neměli bychom však výrazněji předělávat existující struktury databáze, to by svědčilo o nedostatečně stabilním návrhu architektury a tedy předčasném začátku Construction fáze.

Testování

Důležitým faktorem v Construction fázi je průběžné ověřování chování implementovaných komponent/systémů. Pro tento účel je většinou využíváno unit testů a v další fázi integračních a systémových testů. Cílem je co největší

automatizace testování, jelikož tím omezíme chybovost, zvýšíme produktivitu a vývojáři dostanou okamžitou zpětnou vazbu o kvalitě svého kódu. Při tvorbě unit testů využíváme útržků kódu, které mohou simulovat další komponenty či interakci s nimi. Ty mohou být automaticky generovány vizuálními modelovacími nástroji. Automatizace a znovu-použitelnosti je využíváno také v případě Test Casů, které jsou odvozeny z Use Casů. Jaká omezení (např. výkonnost) je třeba testovat najdeme v nefunkčních požadavcích.

Pro zajištění očekávané funkcionality nejen jednotlivých komponent, ale hlavně celku, je nutné jednotlivé komponenty integrovat a testovat společně. K tomuto účelu slouží několik testovacích technik. Předně je nutné zajistit build aplikace, kdy jsou komponenty sestaveny ve správném pořadí do celku a poté testovány. Tento proces je velmi výhodné automatizovat a provádět ho nejlépe jednou denně (např. přes noc), minimálně však alespoň jednou týdně. Vývojáři poté mohou ihned po příchodu do práce vidět výsledky testů a věnovat se případným opravám. Stejně jako v případě unit testů dostávají okamžitou zpětnou vazbu, vidí například, že jejich zásahy či nové komponenty nijak neovlivňují původní chování aplikace. Při testování bychom měli pamatovat několik zásad:

- Cíle testování identifikujeme analýzou iteračního plánu, je třeba vědět, co je jeho cílem, abychom toto mohli následně otestovat.
- Identifikace způsobu testování.
- Analýza způsobů a výběr oblasti, pro kterou vzniknou testovací scénáře (Test Case).
- Implementace testů pro každý Test Case a jejich provedení.
- Analýza testů, které selhali a návrh změn.

Stejně jako je pro vývojáře důležitá zpětná vazba pomocí unit, systémových či integračních testů, je pro celý tým důležitá zpětná vazba od uživatele, který si „hraje“ s jednotlivými releasy aplikace, ověřuje její správné chování a poskytuje důležitou zpětnou vazbu.

Na konci Construction fáze provádíme také tzv. beta-relase, jehož předmětem je testování do něhož jsou zahrnuti vybraní uživatelé. V rámci Construction je třeba připravit úspěšně otestovaný beta-release. Je třeba, aby byly implementovány všechny rysy aplikace, mohou být však ještě nevyřešeny některé kvalitativní problémy, jako je menší dostupnost či odezva aplikace, nesmí se však ztrácet data apod. Stejně tak musí být připravena nápověda v aplikaci, instalační instrukce, uživatelské manuály a tutoriály, jinak nemůžeme dostat od uživatelů (beta-testerů) zpětnou vazbu. Pro některé projekty je také třeba připravit se v Construction na finální nasazení produktu, což zahrnuje:

- Tvorbu materiálů pro trénink uživatelů a správců aplikace.
- Přípravu prostředí pro nasazení (nákup nového HW, konvertování dat apod.) a přípravu dat.
- Příprava dalších aktivit zahrnujících marketing, distribuci, prodej.

4.5.4 Milník IOP

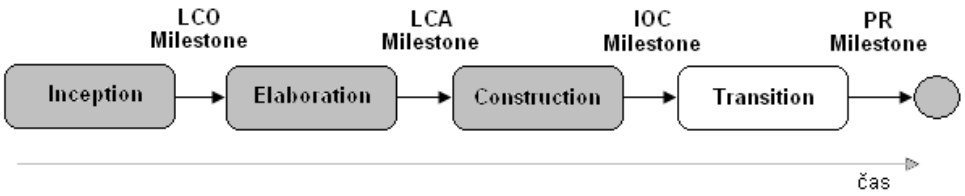
Tento milník je velmi důležitý, jelikož nám říká, zda je produkt připraven pro nasazení a beta-testování.

Zda jsme dosáhli tohoto milníku nám pomůže zjistit následující kontrolní seznam:

- Je produkt dostatečně stabilní a vyzrálý, aby mohl být rozeslán mezi komunitu uživatelů?
- Jsou aktuální výdaje na zdroje oproti plánovaným stále akceptovatelné?

4.6 Transition phase

Poslední fáze představovaného iterativního způsobu vývoje se nazývá Transition. Jejím cílem je především finální vyladění produktu a to nejen z pohledu funkcionality, ale také z pohledu výkonnosti, uživatelské použitelnosti a vůbec celkové kvality. Je také důležité si opět uvědomit, že artefakty, o kterých budeme opět mluvit, nemusí být vůbec formální (dokument či model v nějakém nástroji), je možné je mít ve formě fotek whiteboardu, či na něm přímo ponechané, dále ve formě náčrtků nebo je mít jen v hlavě.



Obr. 4-20: Fáze Transition

Beta-release, který byl nasazen mezi vybrané uživatele v rámci Construction fáze není finální produkt, je třeba ho stále ještě vyladit. Proto i zpětná vazba od uživatelů by měla zahrnovat jen body týkající se výkonnosti, instalace, použitelnosti. Žádné velké změny by neměly být v této fázi prováděny, již se s nimi nepočítá, např. nutnost změn v architektuře v této fázi jednoznačně indikuje špatně provedenou Elaboration a také částečně Construction a evokuje spíše vodopádový přístup, než správně pochopené a provedené iterace řízené riziky. Cílem Transition může být také kompletování některých scénářů, které byly z důvodu podobnosti s ostatními nebo kvůli jejich jednoduchosti přesunuty do fáze Transition (některé případně na konec Construction).

Jednoznačně se vymezíme od tradičních metodik. Cílem Transition není pozdní testování a integrace, které ve vodopádu teprve odhalují vzniklé problémy. Naopak, do této fáze již vstupuje relativně hotová integrovaná, spustitelná, stabilní a testovaná aplikace obsahující téměř všechny funkčnosti.

4.6.1 Cíle

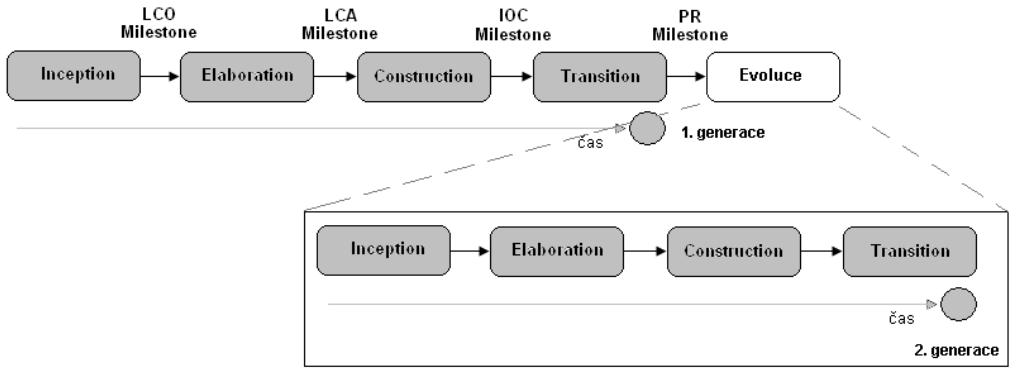
Cíle této fáze jsou následující:

1. Beta-testování – zjištění, zda jsme naplnili očekávání uživatelů.
2. Školení uživatelů a správců aplikace.

3. Příprava prostředí a dat.
4. Příprava dalších aktivit zahrnujících marketing, distribuci, prodej – tvorba letáků s popisem produktu, white papers, technical papers, case study, demo nahrávek, zpráv pro tisk.
5. Dosažení souhlasu uživatelů, že aplikace splňuje jejich představy zachycené v dokumentu Vize (Vision).
6. Zlepšení průběhu budoucích projektů díky ponaučením z tohoto projektu tzv. lessons learnt.

Transition fáze může být velmi jednoduchá, stejně jako velmi komplexní v závislosti na druhu projektu. Může zahrnovat provoz starého systému paralelně s novým, migraci a transformaci dat, školení uživatelů či přizpůsobení podnikových procesů. Typické projekty obsahují v této fázi pouze jednu iteraci, která je zaměřena na opravu chyb a vyladění aplikace.

Kromě dodání výsledného produktu je třeba také dodat zákazníkovi či třetím stranám, které budou provozovat, spravovat nebo dále rozvíjet stávající aplikaci, další artefakty jako je dokumentace uživatelská i technická popisující například architekturu systému. Aplikace na konci této fáze však neumře, pouze ukončíme vývojový cyklus, za kterým však mohou následovat další, viz následující obrázek.



Obr. 4-21: Vývojové cykly více verzí produktu

V případě evoluce, čímž je rozuměn vývoj další verze, jsou většinou překryty fáze Transition právě dokončované verze a Inception životního cyklu verze nové.

4.6.2 Testování

Součástí Transition fáze je také testování a to jak regresní, protože, jak již bylo řečeno i v Transition můžeme provádět návrh a implementaci některých rysů, tak akceptační. Pokud je vývoj ukončen, chyby opraveny a vytvořen build, je tento v Transition opět ještě testován podle standardního testovacího cyklu. Pořád však mějme na paměti, že se nejedná o vodopádový model, že testování a integrace neprobíhá až ve fázi Transition! Testování probíhá neustále v rámci minimálně Elaboration, Construction a Transition fází. Na konce každé iterace v těchto fázích je produkován spustitelný build, nové funkčnosti jsou integrovány a je provedeno unitové testování (provádí ještě samotný

programátor), integrační, systémové, funkční testy a také regresní, abychom si byli jisti, že jsme nenarušili dříve vytvořené funkčnosti.

4.6.3 Lessons learnt

V této fázi je také vhodné shromáždit data o projektu a strávit chvíli jejich analýzou, abychom zjistili, co fungovalo a co ne. Výsledkem mohou být doporučení pro příští projekty, abychom se vyvarovali opětovným chybám. Můžeme znovupoužít nastavení prostředí (jako struktura repository, nastavení nástrojů), některé komponenty apod.

4.6.4 Milník PRM

Posledním milníkem je tzv. Product Release Milestone, který ukončuje čtvrtou a poslední fázi životního cyklu RUP. Cílem milníku je zjistit, zda byly naplněny cíle, které jsme si předsevzali a zda můžeme/chceme začít další vývojový cyklus. V případě pokračování je tato fáze prováděna zároveň jako Inception dalšího cyklu.

Primárními evaluačními kritérii Transition fáze jsou následující otázky:

- Jsou uživatelé spokojeni?
- Jsou aktuální výdaje versus plánované akceptovatelné; pokud ne, jaké akce mohou být v příštích projektech provedeny, abychom tomuto problému předešli?

Kontrolní otázky:

1. Co je cílem fáze Inception?
2. Co je cílem fáze Elaboration?
3. Co je cílem fáze Construction?
4. Co je cílem fáze Transition?
5. Jaký je vztah mezi vodopádovým modelem a iterativním přístupem?
6. Kdy se snažíme odstranit technická rizika projektu?
7. Jaké jsou nejdůležitější principy, na kterých stavíme ve fázi Elaboration?
8. Co vše by měla obsahovat Vize (Vision) vytvořená v Inception?

Úkoly k zamyšlení:
Pokuste se zamyslet nad finančními přínosy iterativního a vodopádového přístupu. Kdy aplikace vyvinutá tím kterým způsobem může začít vydělávat a jaké jsou možnosti variabilního, rozloženého financování v čase?

Korespondenční úkol:
Vypracujte seznam rizik včetně priorit a akcí na jejich odstranění či zmírnění pro projekt výstupu na Mount Everest. Pro každé riziko napište, ve které fázi a v jaké iteraci této fáze budou odstraněna/snížena.

Shrnutí obsahu kapitoly
V této kapitole jste se seznámili se čtyřmi fázemi iterativního způsobu vývoje podle metodiky RUP a také s milníky, které tyto fáze uzavírají. Zásadním rozdílem oproti vodopádu je neustálá spolupráce všech zúčastněných na

projektu, fáze jsou spíše evolučními fázemi projektu, ne funkčně oddělenými bloky, v neposlední řadě je pak velmi brzy produkován hmatatelný výstup (spustitelná aplikace, ne jen vývojářské dokumenty), který je dán k dispozici zákazníkovi, čímž je umožněna zpětná vazba a jsou snížena určitá rizika plynoucí z nepochopení potřeb zákazníka.

5 Plánování iterací

V této kapitole se dozvíte:

- Co to je projektový plán?
- Co je iterační plán?
- K čemu slouží, co obsahují a jak se tvoří?
- Jak je to s počtem lidí na projektu.

Po jejím prostudování byste měli být schopni:

- Definovat pojem projektový a iterační plán.
- Vytvořit konkrétní projektový a iterační plán.

Klíčová slova této kapitoly:

Projektový plán, iterační plán, plánování.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Kapitola představuje dva hlavní druhy plánů v RUP. Jedním z nich je projektový plán, druhý iterační. Kapitola vysvětluje jejich místo v projektu, rozdíly a vazby. Dále je zde nastíněna problematika plánování iterací.
Na studium této části si vyhraďte 2 hodiny.

V předchozí kapitole jsme se věnovali fázím a iteracím. Řekli jsme si, co to jsou iterace, co je jejich cílem. Nyní se budeme věnovat jejich plánování, jelikož je toto podstatné pro úspěšně zvládnutý projekt.

V průběhu projektu připravujeme dva druhy plánů:

- hrubý plán (Road Map) – projektový plán (Project Plan) – pouze jeden pro projekt, projektový plán se zaměřuje na fáze a iterace a jejich cíle a na celkové počty zdrojů.
- detailní plán – iterační plán (Iteration Plan) – pro každou iteraci zvlášť, dává jednotlivé RUP aktivity a zdroje do souvislostí.

Projektový plán je jakousi road map, která nám říká v hrubých rysech, jak a kdy budou důležité milníky naplněny. Projektový plán je zaměřený na finální produkt, proto jsou v něm nejdůležitější zmíněné milníky a k nim vázané releasy. Obsahem projektového plánu jsou:

- Data nejvýznamnějších milníků, tj.:
    - LCO milník – konec Inception, projekt má definovanou šířku rozsahu, co vše bude zahrnuto a je odsouhlaseno a definováno jeho financování.
    - LCA milník – konec Elaboration, kompletní architektura, definovány základní požadavky.
    - IOC milestone – konec Construction, vytvořen beta release.
    - PR milník – konec Transition, konec vývojového cyklu, hotový produkt předán do užívání, následuje údržba nebo další vývojový cyklus.

- Potřebné personální zajištění (lidské zdroje) – sumarizuje, jaké zdroje a kolik bude potřeba v průběhu času.
- Data méně významných milníků – konce iterací včetně jejich základních cílů, pokud jsou již známy.

Pokud se bavíme o projektovém plánu, máme na mysli dokument v rozsahu 1-2 stran, který je produkován brzy v Inception fázi a je aktualizován tak často, jak je potřeba, není to tedy „mrtvý dokument“. Projektový plán odkazuje na Vizi, kde je definován obsah (scope) a předpoklady projektu.

Příklad:
Následující příklad projektového plánu pro projekt s jedním člověkem ukazuje jeho jednoduchou verzi a odkazuje se na vizi, kterou jsme představili v kapitole 4.3 o Inception fázi. Vize viz Tabulka 4-2. Složitější plán viz kapitola 4.3.

Osobní časovač (OČ): projektový plán

Pondělí	**Úterý**	**Středa**	**Čtvrtek**	**Pátek**
Inception Vize Plán Business Case Seznam rizik **LCO:** Souhlas od Garyho **Elaboration** Prototyp	Prototyp Zmírnění rizik **LCA:** Souhlas od Garyho Use Casy Testy	**Construction** Návrh Programování Testování Návrh Programování Testování	Návrh Programování Testování **IOC:** ukázat první beta verzi **Transition** Zlepšování Doručení	Časový buffer

Tabulka 5-1: Příklad jednoduchého projektového plánu

Zmínili jsme, že v projektovém plánu definujeme počty iterací v jednotlivých fázích, jejich stručnou náplň a jejich cíle, pokud jsou již známé. Příkladem cílů jednotlivých iterací může být pro systém Telefonního přepínače následující:

- Iterace 1: Hovor mezi lokálními stanicemi.
- Iterace 2: Přidání externích hovorů a správa účastníků.
- Iterace 3: Přidání telefonního záznamníku a konferenčních hovorů.
- Iterace 4: ...

Cíle iterací jsou dány především riziky, naší snahou je v prvních iteracích Elaboration snížit či odstranit největší identifikovaná rizika. Počty iterací v jednotlivých fázích se odvíjí od několika bodů. Například v Inception jde o to, zda je problémová doména neznámá a složitá, pokud ano, přidáme iteraci, jinak probíhá obvykle pouze jedna iterace. V Elaboration bude rozhodující počet a velikost rizik, stejně jako znalost či neznalost technologie. Podobně je tomu v dalších iteracích. Více o iteracích a jejich počtech v jednotlivých fázích bylo uvedeno v kapitolách týkajících se jednotlivých fází.

Iterační plán je narozdíl od hrubého projektového plánu detailní. Je také časově omezený, což vychází z časového omezení délky iterací, které se

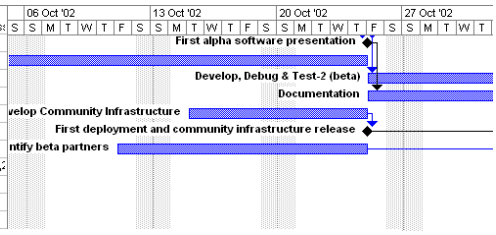
mohou měnit podle délky projektu, připomínáme, že jde o 2-6 týdnů. Jelikož je délka iterace v rozmezí několika týdnů, je jednodušší plánovat a předvídat možné nenadále události. Plán by měl zahrnovat pořadí jednotlivých úkolů a jejich přiřazení jednotlivým členům týmu. Každý by měl vědět, co je cílem iterace a rozumět, jakým způsobem k tomuto celkovému cílu přispívá, co je od něj očekáváno.

Projekt má v každém čase „aktivní“ dva iterační plány:

- Současný iterační plán (pro v daném čase prováděnou iteraci), který je používán ke sledování postupu prací v dané iteraci,
- Iterační plán pro nadcházející iteraci, který je vytvářen v průběhu této iterace a je hotov, připraven k použití na jejím konci.

Mezitím, jak tým provádí práce v současné iteraci, vytváří projektový manažer plán pro následnou iteraci. Cílem je, aby po dokončení a zhodnocení současné iterace měl tým k dispozici další plán a mohly být zahájeny práce na další iteraci bez dlouhých proluk. Je však nutné tento vytvářený plán aktualizovat podle současného stavu a brát ohled na veškeré změny i/hlavně na ty nejpozdější a s velkým dopadem. Pro tvorbu plánu se používají klasické techniky jako jsou Ganttovy grafy (viz následující obrázek) či PERT grafy.

ID	Task Name	Duration	Start	Finish	Predeces
19	**First alpha software presentation**	0 days	Fri 14-02-03	Fri 14-02-03	17,18
20	Research/develop new (2003) SOFI d	50 days	Fri 06-12-02	Thu 13-02-03	
21	Develop, Debug & Test-2 (beta)	25 days	Fri 14-02-03	Thu 20-03-03	18
22	Documentation	15 days	Fri 14-02-03	Thu 06-03-03	19
23	Develop Community Infrastructure	8 days	Tue 04-02-03	Thu 13-02-03	
24	First deployment and community infras	0 days	Fri 14-02-03	Fri 14-02-03	23
25	Identify beta partners	10 days	Fri 31-01-03	Thu 13-02-03	
26	**First beta software release**	0 days	Thu 20-03-03	Thu 20-03-03	22,24,25,2
27	First training and deployment session	7 days	Fri 21-03-03	Mon 31-03-03	26
28	Development, Debug & Testing-3	21 days	Fri 21-03-03	Fri 18-04-03	26
29	Integrate software and data	15 days	Mon 21-04-03	Fri 09-05-03	28



Obr. 5-1: Příklad Ganttova grafu

Iterační plán obsahuje jednotlivé aktivity, které je třeba vykonat (aktivity popsané v RUP), vzájemné návaznosti (například sestavení buildu a návazné testování) a přiřazení aktivit zdrojům, dále důležitá data jako významné buildy, doručení komponent od třetích stran, nejdůležitější revize. Definuje cíle iterace a kritéria pro ohodnocení jejich dosažení. Jednoduchý projekt může použít pouze jednoduchý list, co je třeba udělat, podobně, jak jsme to viděli na příkladu projektového plánu (viz Tabulka 5-1).

5.1 Počet pracovníků na projektu

Poté, co jsme v projektovém plánu definovali základní údaje projektu, data milníků, počty iterací, jejich náplně a cíle, definujeme počty pracovníků v průběhu celého projektu. Tyto počty se budou lišit, půjde-li o projekt tvorby zcela nového software („na zelené louce“) nebo o projekt údržby či evoluce existujícího produktu:

- Nový produkt – počet lidí na projektu se bude dost lišit, není vhodné, aby např. Vizi v Inception definovalo 50 lidí, proto je v úvodu zahrnuto méně lidí, další se přidávají až v Elaboration a hlavně v Construction, přibližné poměry viz Obr. 4-3 (osa *Resource* ukazuje počty lidí účastnících se projektu v daných fázích).

- Údržba a evoluce – stejný počet lidí může pracovat na projektu celou dobu, projekt (vykonávané práce) bude mít formu fáze Transition, případně Construction a Transition.

Samozřejmě je nutné si uvědomit, že tyto počty se také odvíjí od finančních zdrojů, které mám k dispozici.

5.2 Iterační plánování

Plánování iterace se skládá z následujících čtyř kroků:

1. Určení obsahu iterace – co chceme v rámci této iterace dokončit.
2. Definice evaluačních kritérií dané iterace – říkají, jak na konci iterace objektivně ohodnotit dosažení cílů, konkrétně na jakém artefaktu budeme pracovat.
3. Definice aktivit iterace – co je třeba provést a s jakými artefakty.
4. Přiřazení odpovědností – přiřazení lidských zdrojů k definovaným aktivitám.

Pokud se jedná o tvorbu nového software („na zelené louce“), pak je při plánování v úvodních fázích třeba přihlížet ke zkušenostem z předchozích projektů, odhady provádíme způsobem shora-dolů, kdy v případě neznalosti nějakého problému, technologie, rizika uděláme krátkou analýzu (desítky minut, maximálně hodiny), abychom odhad více zpřesnili. Opět ale upozorníme na to, že není cílem dělat detailní plány předem (protože je ještě spousta nejasných věcí a spousta rizik), a nebo strávit danou analýzou dny. Plánování v dalších fázích už může být děláno způsobem zdola-nahoru, kdy na základě návrhových tříd, komponent či Use Casů definujeme pracnost, tyto odhady jsou ale svou povahou pesimistické. Hlavním zdrojem pro plánování projektů, fází, iterací jsou tedy empirická, historická data a jejich porovnání vůči aktuálnímu postupu v předchozích úspěšných projektech.

Aktivity, které je třeba provádět v jednotlivých fázích je možné vyčíst z RUP nebo OpenUP [OUP], jde o činnosti vedoucí k výslednému software. V jedné iteraci budeme tedy dle potřeby provádět několikrát například následující:

- Detailní popis nastíněných scénářů.
- Analýza a návrh scénáře.
- Implementace a unit testy scénáře.
- Automatická integrace, build a testování scénáře.

V rámci každé iterace pak určitě provádíme aktivity Plánování iterace (Iteration Planning) a vyhodnocení iterace (Iteration Review). První aktivita se zabývá tvorbou podrobného plánu pro danou iteraci, druhá pak hodnocením naplnění cílů, vyhodnocením iterace a vytvořením ponaučení (Lessons Learnt), pokud nějaké jsou.

V případě iterací v Elaboration budou cíle iterací a jejich pořadí řízeny riziky. Nejrizikovější věci (z pohledu podnikání i technologie) děláme nejdříve. To znamená, že scénáře pro zákazníka nejkritičtější a technologické mechanismy (jak ukládat data a komunikovat s DB, jsme schopni spojit se se systémem LDAP, jak zpracovávat a kontrolovat uživatelské požadavky = http requesty,

apod.) jsou definovány nejdříve, protože definují architekturu systému. Postup definování cílů pro iteraci v Elaboration (náplň iterace podle priorit):

1. Opravení chyb z předchozí iterace.
2. Zapracování prioritnějších požadavků (scénářů) definujících architekturu – zjištěno díky zpětné vazbě od zákazníka (user play s aplikací na konci předchozí iterace).
3. Vybrané scénáře prioritních Use Case – nejvíce rizikové.

V případě iterací v Construction je určování priorit jiné. Jelikož jsme v Elaboration rizika odstranili nebo je snížili na přijatelnou úroveň, určujeme priority podle přidané hodnoty pro zákazníka. V úvahu při plánování cílů iterace v Construction tedy bereme v úvahu následující body v daném pořadí:

1. Opravení chyb z předchozí iterace.
2. Zapracování prioritnějších požadavků (scénářů), které přináší větší hodnotu zákazníkovi – zjištěno díky zpětné vazbě od zákazníka (user play s aplikací na konci předchozí iterace).
3. Vybrané scénáře Use Case podle priorit – nejvíce přínosné pro zákazníka.
4. Refaktoring existujícího jednoduchého kódu (a testů) z předchozích iterací + zapracování návrhových vzorů – zlepšení kvality kódu.

Příklad plánu iterace E1

Začátek iterace (plánovací meeting dané iterace): 14.1.2008
Konec iterace (demo, assessment): 1.2.2008

Cíle iterace:

- Implementace UC1 [BF].
- Odstranění rizika R1 a R2.

Evaluační kritéria:

- 60 % kódu pokryto unit testy.
- 100 % unit testů prošlo.
- 70 % implementovaných funkčních testů prošlo.
- Sníženo riziko R1.
- Bylo předvedeno demo zákazníkovi.
- Zákazník demo akceptoval.

Úkoly:

Název / Popis	Priorita	Odhad (body)	Přiřazeno	Odhad (hodin)
Instalace build mechanismu (Ant)	2		Johan	7
Analýza scénáře UC1 (BF)	2	8	Lisa	
Návrh scénáře a implementace		8	Lisa, Ann, Johan	12
Implementace a testy server části			Ann	14
Implementace a testy klient části			Johan	28
Sestavení dema pro assessment	3	5	Johan	12
Vytvoření uživatelské dokumentace	2	5	Lisa	65
Vytvoření install manuálu	2	1	Lisa	5
Vytvoření release notes	2	1	Johan	4
Vytvoření online help stránek	3	2	Ann	22

Assessment iterace E1 (vyplněn až na konci iterace při assessmentu)
Ohodnocení cílů podle evaluačních kritérií:

- 70% pokrytí kódu unit testy.
- 98 % unit testů prošlo.
- Riziko R1 sníženo, aplikace komunikuje s databází bez potíží, data jsou ukládána i zobrazena ve správném formátu (provedena kontrola uložení i v DB).
- Zákazník byl mile překvapen existujícími funkčnostmi takhle brzy (a akceptoval demo) a byl dohodnut pilotní provoz reportování již po skončení fáze Elaboration.

Neopravené chyby:

- ERR0012: Null pointer při určitém stavu… (ve formě odkazu na Bugzillu, Jiru či jiný nástroj pro vedení evidence chyb, tzv. issue tracking tool).

Jiné poznámky a poznatky:
Při ukázce dema zákazníkovi dne 1.2.2008 si reprezentant zákazníka uvědomil (díky demonstraci a krátké hře s aplikací), že zapoměl jako jeden z požadavků zmínit nutnost propojení budoucího systému s jeho existujícím logovacím systémem zaznamenávajícím činnost uživatelů. Zmínil pouze vazbu na LDAP server. Tento požadavek proto musí být zapracován do následujících fází projektu. Díky navržené rezervě jsme schopni tento důležitý požadavek realizovat v Elaboration (nutné zde – má vliv na architekturu) bez přesunu jiných požadavků do následující verze.

Lessons learnt (poznatky): …

Kontrolní otázky:

1. Jaké 2 druhy plánů definujeme v RUPu?
2. Jak je nazývá a co je náplní prvního z nich?
3. Jak je nazývá a co je náplní druhého?
4. Jaké jsou 4 hlavní kroky plánování iterací?
5. Které aktivity provádíme v každé iteraci?

Úkoly k zamyšlení:

Zamyslete se nad problematikou obsazení pracovníků na projekt v různých fázích. Proč se počty lidí zúčastněných na projektu v jeho průběhu liší? Jsou případy, kdy se lišit nemusí, kdy pracuje na projektu stejný počet lidí po jeho celou dobu? Pokud ano, tak kdy?

Korespondenční úkol:

Naplánujte následující iteraci ve vývoji, když víte, že budete implementovat 2 scénáře, kdy každý vám zabere 2 týdny práce a musíte opravit chyby z předchozí iterace. Navíc přišel zákazník s novým požadavkem, který je pro něho důležitější než některé, které chceme implementovat nyní. Jak bude vypadat plán iterace, který zohledňuje tyto fakta? Jaké budou cíle iterace, jak bude iterace dlouhá? Je toto vůbec možné, není třeba úkoly rozdělit do více iterací?

Shrnutí obsahu kapitoly

Tato kapitola představila dva hlavní druhy plánů v RUP, projektový a iterační plán. Kapitola vysvětluje jejich místo v projektu, rozdíly a vazby. Dále je zde nastíněna problematika plánování projektů, fází a iterací, stejně jako problematika lidských zdrojů využívaných v průběhu celého projektu.

6 RUP vs. OpenUP

Učební text, který právě čtete, se věnuje procesnímu frameworku RUP, který byl místy nazýván metodikou. Jelikož je daný produkt, původně společnosti Rational, ve vlastnictví IBM, není se čemu divit, že je placený. Z tohoto důvodu není pro studenty zrovna přístupný. Druhým problémem je jeho přílišná rozsáhlost, nezkušení vývojáři jsou postaveni před problém, jak začít. Webová aplikace RUP obsahuje spoustu rolí, stovky artefaktu a ještě více činností. Není potom zřejmé, co je třeba vykonávat, který artefakt je důležitý a který už ne. Proto je velmi důležité mít v projektu po ruce mentora, což je zkušený člověk, který s implementací metodiky pomůže. V našich podmínkách však využijeme jinou možnost.

V rámci jedné z iniciativ projektu Eclipse.org vznikla agilní metodika, která je postavena na podobných principech jako RUP, je iterativní, produkuje releasy, je řízena riziky a Use Casy, definuje stejné fáze a podobné artefakty. Narozdíl o RUPu, který je maximalistický a obsahuje vše a my si vybíráme pouze to, co potřebujeme (což je právě to těžké), je OpenUP minimalistický, obsahuje pouze nutné minimum potřebné pro agilní vývoj software. Navíc je tato metodika/aplikace zdarma, dostupná každému ke stažení, odkaz viz níže v této kapitole.

OpenUP obsahuje celkem pouze:

- 7 rolí,
- asi 20 artefaktů (zde nazvány Work Product),
- disciplíny omezeny pouze na vývoj (chybí Environment, Business Modelling).

Součástí webové aplikace OpenUP jsou stejně jako v RUPu také šablony pro veškeré potřebné dokumenty jako je Vize (Vision), Projektový plán (Project Plan), Seznam rizik (Risk List), Iterační plán (Iteration Plan) a další nutné dokumenty.

OpenUP je podle potřeby možné rozšířit, aby vyhovoval specifickým potřebám, proto je mnohem vhodnější, zvláště pro začínající vývojáře.

OpenUp je možné stáhnou na adrese:
http://www.eclipse.org/epf/downloads/openup/openup_downloads.php
jedná se o soubor *OpenUP_published_XXXXXXX_zip* (velikost asi 5MB).

7 Literatura

[AgM] Manifest agilního vývoje. Dostupné na: [http://www.agilemanifesto.org/].

[Arl03] Arlow, J., Neustadt, I.: *UML a unifikovaný proces vývoje aplikací*. Computer press. Brno. 2003. ISBN 80-7226-947-X.

[Co05] Cockburn, A.: *Use Cases. Jak efektivně modelovat aplikace*. Computer Press. Brno. 2005. ISBN 80-251-0721-3.

[Jazz] Jazz homepage. Dostupné na: [http://jazz.net].

[Kli04] Klimeš, C., Procházka, J.: *Projektování informačních systémů 1*. Učební text pro distanční studium. Ostravská univerzita. Ostrava. 2004.

[Kr03a] Kroll, P., Kruchten, P.: *The Rational Unified Process. Made Easy*. Addison-Wesley. 2003. ISBN 0321166094.

[Kr03b] Kroll, P., Kruchten, P.: *The Rational Unified Process. An Introduction*. Addison-Wesley. 3. vydání. 2003. ISBN 0321197704.

[Kr05] Kroll, P., Royce, W.: *Key principles for business-driven development*. Dostupné na Rational Edge: [http://www.ibm.com/developerworks/rational/library/oct05/kroll/index.html?S_TACT=105AGX15&S_CMP=EDU].

[Kr06] Kroll, P., MacIsaac, B.: *Agility and Discipline Made Easy: Practices from OpenUP and RUP*. Addison-Wesley. 2006. ISBN 0321321308.

[OUP] OpenUP homepage: [http://www.eclipse.org/epf/general/getting_started.php].

[Pro06] Procházka, J.: *Procesní řízení realizace projektů*. Elektronický učební text. Systém dalšího vzdělávání pracovníků výzkumu a vývoje. Ostravská univerzita. 2006.

[RE] Rational Edge homepage (RUP články): [http://www-128.ibm.com/developerworks/views/rational/libraryview.jsp?topic_by=rup%20(rational%20unified%20process)].

[Sta02] Výzkum Standish Group. Dostupné na: [http://ciclamino.dibe.unige.it/xp2002/talksinfo/johnson.pdf].

8 Přílohy

Jako přílohy jsou uvedeny příklady nebo šablony některých artefaktů RUPu, zdroj [RUP], či OpenUP [OUP], jedná se o:

- Vize (Vision).
- Seznam rizik z Inception fáze (v dalších iteracích a fázích se mění).
- Use case model a seznam use case a scénářů.
- Projektový plán (Project Plan) z Inception fáze, také se dále mění.
- Iterační Plán (Iteration Plan).

9 Příloha A – Vision

Introduction

Positioning

Problem Statement

[Provide a statement summarizing the problem being solved by this project. The following format may be used:]

The problem of	*[describe the problem]*
Affects	*[the stakeholders affected by the problem]*
the impact of which is	*[what is the impact of the problem?]*
a successful solution would be	*[list some key benefits of a successful solution]*

Product Position Statement

[Provide an overall statement summarizing, at the highest level, the unique position the product intends to fill in the marketplace. The following format may be used:]

For	*[target customer]*
Who	*[statement of the need or opportunity]*
The (product name)	*is a [product category]*
That	*[statement of key benefit; that is, the compelling reason to buy]*
Unlike	*[primary competitive alternative]*
Our product	*[statement of primary differentiation]*

[A product position statement communicates the intent of the application and the importance of the project to all concerned personnel.]

Stakeholder Descriptions

Stakeholder Summary

Name	Description	Responsibilities
[Name the stakeholder type.]	*[Briefly describe the stakeholder.]*	*[Summarize the stakeholder's key responsibilities with regard to the system being developed; that is, their interest as a stakeholder.* *For example, this stakeholder:* *ensures that the system will be maintainable* *ensures that there will be a market demand for the product's features* *monitors the project's progress* *approves funding and so forth]*

User Environment

[Detail the working environment of the target user. Here are some suggestions:

Number of people involved in completing the task? Is this changing?

How long is a task cycle? Amount of time spent in each activity? Is this changing?

Any unique environmental constraints: mobile, outdoors, in-flight, and so on?

Which system platforms are in use today? Future platforms?

What other applications are in use? Does your application need to integrate with them?

This is where extracts from the Business Model could be included to outline the task and roles involved, and so on.]

Product Overview

Needs and Features

[Avoid design. Keep feature descriptions at a general level. Focus on capabilities needed and why (not how) they should be implemented. Capture the stakeholder priority and planned release for each feature.]

Need	Priority	Features	Planned Release

Other Product Requirements

[At a high level, list applicable standards, hardware, or platform requirements; performance requirements; and environmental requirements.

Define the quality ranges for performance, robustness, fault tolerance, usability, and similar characteristics that are not captured in the Feature Set.

*Note any design constraints, external constraints, assumptions or other dependencies that, if changed, will alter the **Vision** document. For example, an assumption may state that a specific operating system will be available for the hardware designated for the software product. If the operating system is not available, the **Vision** document will need to change.*

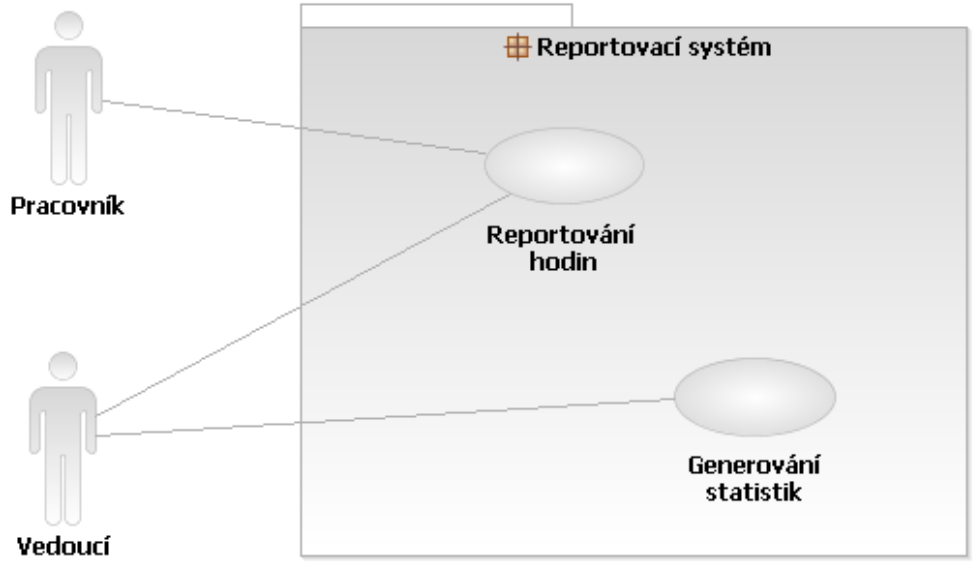
Define any specific documentation requirements, including user manuals, online help, installation, labeling, and packaging requirements.

Define the priority of these other product requirements. Include, if useful, attributes such as stability, benefit, effort, and risk.]

Requirement	Priority	Planned Release

10 Příloha B - Use case

Identifikovaný use case model:



A následně rozpracovaný podrobný seznam use case a jejich scénářů s identifikovanými riziky, odhadem pracnosti, prioritou a spojenými riziky.

UC	Scénář	Priorita	Odhad	Riziko	Verze
UC 1: Reportování hodin	BF (basic flow): Reportování hodin	vysoká	10 člověkodní	R1, R2	1.0
	AF1 (alternativní flow): Import dat	vysoká	8 člověkodní	R3, R4	1.0
	AF2: Vkládání více úkolů	nízká	8 člověkodní	-	1.0
UC 2: Generování statistik	BF: Zobrazení hodin pro vybrané období	střední	5 člověkodní	R2	1.0
	AF1: Export do Excelu	nízká	2 člověkodny	R3	1.0
	AF2: Bude upřesněno...	nízká	Bude upřesněno	-	2.0

11 Příloha C – Risk List

1. Příklad:

Risk Ranking/ Magnitude	Risk Description & Impact	Mitigation Strategy and/or Contingency Plan
7	R1 and R2 Releases may slip and not be available by September 1999 - the start of the Registration Period.	Monitor progress against the schedule & milestones. Update effort to complete and time to complete on a regular basis.
5	Interfaces to the old legacy Billing and Course Catalog Systems may introduce performance and response time issues.	Develop early prototypes to test all external interfaces.
4	Volume of students logged on during peak hours of the registration period may significantly degrade system performance.	Early prototyping and extrapolation of response time data should be done in the Elaboration Phase.
3	Incompatibility with internet browsers and specific configurations on client machines.	Address during Elaboration Phase.
3	The development team is relatively inexperienced with the Rational Unified Process (RUP) and Object Oriented Techniques. This could lead to lower efficiency and poorer product quality	Schedule training sessions for OO Development and the Rational Unified Process. Establish 'process mentors' who can assist the team in understanding the process and the development activities. Ensure all Design and Code is inspected.
2	Wylie College will be unable to fund the development as part of its 1999 budget.	Prepare a second option for financing which splits the development (and funding) across 2 years (1999 and 2000).

2. Příklad:

Název rizika	Popis	Dopad	Pravděpodobnost	Důležitost	Vlastník
Nedostatečné zapojení všech stakeholderů	Předchozí zkušenost nám říká že lidé z oddělení X nemusí pochopit či souhlasit s požadavky, výsledkem budou požadavky na zásadní změny po Beta-releasu.	3 vysoký	90%	2.7	Analytik Honza
Integrace se systémem X	Není zřejmé, jak integrovat naši aplikaci s historickým systémem X.	3 vysoký	80%	2.4	Architekt Petr
Tréninkové materiály	Nemáme oprávnění vytvořit kvalitní tréninkové materiály, což může vést k nekvalitnímu tréninku.	2 střední	100%	2.0	Manažerka Anička
Nedostatečná zkušenost s JEE	Je riziko, že vytvoříme druhořadé, méně technicky kvalitní řešení v důvodu nezkušenosti s platformou JEE.	2 střední	60%	1.2	Vývojář Tom

12 Příloha D – Project Plan

Projektový plán:

Fáze	Iterace	Primární úkoly	Datum od - do
Inception	I0	Sestavení týmu, rozpočtu Identifikace základních požadavků a rizik Výběr kandidátů architektury LCO (shoda na vizi, rozsahu, rozpočtu)	02.01.2008 - 14.01.2008 14.01.2008
Elaboration	E1	UC1 [BF]: Reportování hodin UC1 [AF1]: Import dat	14.01.2008 - 01.02.2008
	E2	UC2 [BF]: Zobrazení hodin pro vybrané období	01.02.2008 – 18.02.2008
	E3	Rezerva (5 dní)	18.02.2008 – 24.02.2008
		LCA (snížena rizika, implementována a otestována architektura)	24.02.2008
Construction	C1	UC1 [AF2]: Vkládání více úkolů	25.02.2008 – 09.03.2008
	C2	UC2 [AF1]: Export do Excelu	10.03.2008 - 23.03.2008
	C3	Rezerva (2 dny)	24.03.2008 – 25.03.2008
		IOC (beta release)	26.03.2008
Transition	T1	Akceptační testování zadavatelem ... PR: Předání projektu	13.05.2008 - 07.06.2008

13 Příloha E – Iteration Plan

Plán iterace E1
Začátek iterace (plánovací meeting dané iterace): 14.1.2008
Konec iterace (demo, assessment): 1.2.2008

Cíle iterace:

- Implementace UC1 [BF].
- Odstranění rizika R1 a R2.

Evaluační kritéria:

- 60 % kódu pokryto unit testy.
- 100 % unit testů prošlo.
- 70 % implementovaných funkčních testů prošlo.
- Sníženo riziko R1.
- Bylo předvedeno demo zákazníkovi.
- Zákazník demo akceptoval.

Úkoly:

Název / Popis	Priorita	Odhad (body)	Přiřazeno	Odhad (hodin)
Instalace build mechanismu (Ant)	2		Johan	70
Analýza scénáře UC1 (BF)	2	8	Lisa	
Návrh scénáře a implementace		8	Lisa, Ann, Johan	12
Implementace a testy server části			Ann	14
Implementace a testy klient části			Johan	28
Sestavení dema pro assessment	3	5	Johan	18
Vytvoření uživatelské dokumentace	2	5	Lisa	65
Vytvoření install manuálu	2	1	Lisa	5
Vytvoření release notes	2	1	Johan	4
Vytvoření online help stránek	3	2	Ann	22

Assessment iterace E1 **(vyplněno až na konci iterace při assessmentu)**
Ohodnocení cílů podle evaluačních kritérií:

- 70% pokrytí kódu unit testy.
- 98 % unit testů prošlo.
- Riziko R1 sníženo, aplikace komunikuje s databází bez potíží, data jsou ukládána i zobrazena ve správném formátu (provedena kontrola uložení i v DB).
- Zákazník byl mile překvapen existujícími funkčnostmi takhle brzy (a akceptoval demo) a byl dohodnut pilotní provoz reportování již po skončení fáze Elaboration.

Neopravené chyby:

- ERR0012: Null pointer při určitém stavu… (ve formě odkazu na Bugzillu, Jiru či jiný nástroj pro vedení evidence chyb, tzv. issue tracking tool).

Jiné poznámky a poznatky:

Při ukázce dema zákazníkovi dne 1.2.2008 si reprezentant zákazníka uvědomil (díky demonstraci a krátké hře s aplikací), že zapoměl jako jeden z požadavků zmínit nutnost propojení budoucího systému s jeho existujícím logovacím systémem zaznamenávajícím činnost uživatelů. Zmínil pouze vazbu na LDAP server. Tento požadavek proto musí být zapracován do následujících fází projektu. Díky navržené rezervě jsme schopni tento důležitý požadavek realizovat v Elaboration (nutné zde – má vliv na architekturu) bez přesunu jiných požadavků do následující verze.

Lessons learnt (poznatky): ….