Vysoká škola ekonomická v Praze

Katedra informačních technologií

RUP – Disciplína implementation

Semestrální práce

4IT421 Zlepšování procesů budování IS

ZS 2012/2013

Bc. Jan Bazala, xbazj03

Bc. Tomáš Gühl, xguht00

Obsah

Cíle práce

Cílem naší práce je seznámit čtenáře s disciplínou Implementation metodiky RUP, a to konkrétně s rolemi, činnostmi, produkty a návody, které disciplína obsahuje.

Dále chceme porovnat využití disciplíny velkými a malými projekty, případně uvést důvody rozdílných pojetí obou modelů.

Z práce také vyvodíme vlastní zhodnocení a případně návrh na zlepšení některých částí.

Úvod

Disciplínu Implementation jsme si vybrali, jelikož jí považujeme za stěžejní bod využívání metodiky RUP. Jedná se o jakýsi návod, jak vyvíjet, organizovat, provádět jednotkové testy a integrovat prvky závislé na specifikaci návrhu.

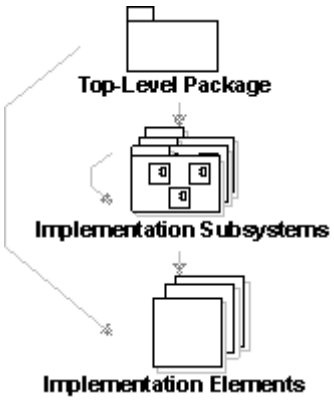
Důvod existence disciplíny Implementation přitom vidíme v následujících bodech:- je nutné nadefinovat jakousi organizaci kódu, a to hlavně v případech například modulárních řešení (subsystémy)

- je důležité realizovat jednotlivé designové prvky z hlediska implementačních prvků (zdrojové soubory...)
- je potřeba provádět jednotkové testy
- je nutné integrovat jednotlivé moduly (subsystémy) do konečného spustitelného programu (systému)

Disciplína směřuje k lepšímu pochopení fyzických částí projektu a tím i k lepšímu řízení a zajisté je neodmyslitelně propojena s ostatními disciplínami. Zpracovávat jí budeme převážně z dostupné metodiky RUP na stránkách KITSCM.

Model implementation

Model representuje fyzickou skladbu implementace v souladu s jejími subsytémy a prvky tak, aby mohly být lépe pochopeny a řízeny. Dále definuje dané jednotky integrace, pro které je třeba sestavit jednotlivé týmy.

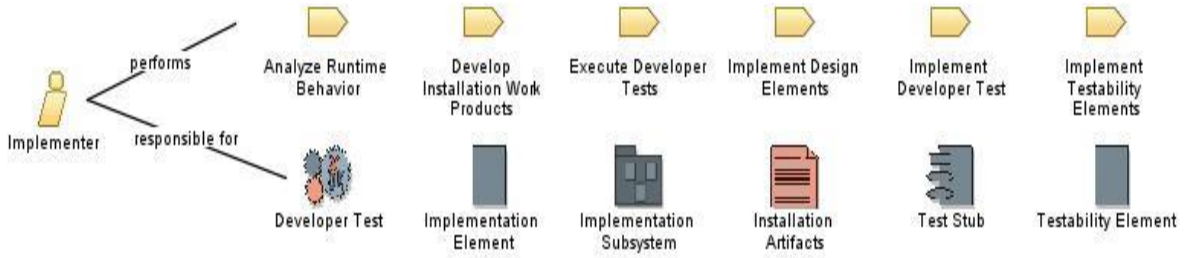


Obrázek 1: Model implementation

Dle disciplíny je pak možné volitelně vytvořit jednotný model pro fyzickou skladbu i návrh logiky což je vítáno v přístupech kombinujících soubory zdrojových kódů a Implementation/Design modelu. Při přesném mapování projektu v jeho začátku že každý implementační subsystém je zároveň designovým, což vede k jejich vzájemnému propojení i v rámci modelování.

Role

Implementer



Obrázek 2: Implementer

Činnost:

- vyvíjí softwarové komponenty a zadává testování pro integrací do větších subsystémů
- nese zodpovědnost za vývoj a testování komponent
- pokud je třeba vytvářet testovací komponenty, je zodpovědný i za jejich vývoj a testování včetně subsystémů

Skills:

- znalost systému nebo aplikace v rámci testování
- umí testovat a pracovat s testovacími nástroji
- zkušenosti s programováním

Designer

Obrázek 3: Designer

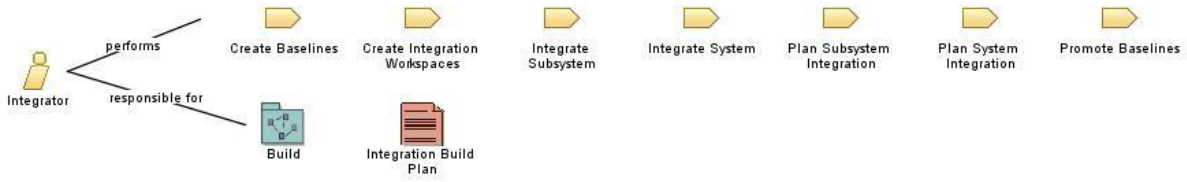
Činnost:

- v souladu s požadavky, architekturou a procesem vývoje navrhuje část systému
- určuje vztahy, atributy, operace a odpovědnosti při návrhu prvků
- zodpovídá za konsistentnost návrhu a architektury

Skills:

- znalost systémových požadavků
- znalost systémové architektury
- znalost technik návrhu, a to i objektově orientovaného
- znalost UML
- znalost technologií, jejichž pomocí bude prováděna implementace
- povědomí o potřebě detailu návrhu v souvislosti s následnou implementací

Integrator



Obrázek 4: Integrator

Činnost:

- vede plánování a zavádění prvků do kompletních buildů
- zodpovídá za rozdělení projektu na subsystémy a následnou integraci jednotlivých prvků do konečné podoby

Skills:

- znalost systému nebo části systému, který má být integrován
- potřebuje znát vzájemné závislosti mezi prováděním prvků a subsystémů
- znalost nástrojů pro integraci

Technical reviewer

Obrázek 5: Technical reviewer

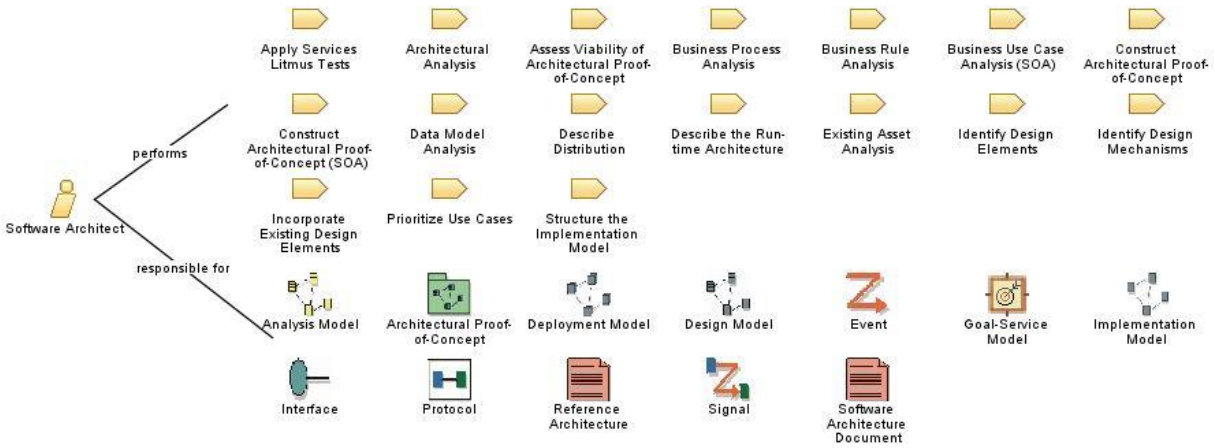
Činnost:

- přispívá technickou zpětnou vazbu na produkty práce na projektu. Tato zpětná vazba musí být včasná a vhodná.
- nese odpovědnost za plnění úkolů a jejich logické přerozdělování
- jednu roli může přiřadit několika lidem, stejně jako jeden člověk může zastupovat více rolí

Skills:

- znalosti a schopnosti potřebné k přezkoumávání projektů a produktů
- odpovědnost za vedlejší produkty, které daná projekt nějakým způsobem ovlivňují
- odpovědnost za úkoly následující po předání dané části projektu dále

Software architekt



Obrázek 6: Software architect

Činnost:

- sestavuje architekturu, což zahrnuje podporu klíčových technických rozhodnutí
- zodpovídá za podklady vedoucí k těmto rozhodnutím

Skills:

- vizionář s hlubokými zkušenostmi, které mu umožní rychle uchopit problematiku projektu
- člověk, schopný kritického myšlení i při nedostatku informací
- komunikativní a schopný podílet se spolu s project managerem na vedení projektu (každý po své linii)
- aktivní s tahem na bránu tak, aby přinášel viditelné výsledky
- radši by měl znát více technologií povrchně, než málo do hloubky

Návod

Důležitá rozhodnutí při implementaci

Návod popisuje některé důležité kroky, které bychom měli zvážit, přizpůsobujeme- li vzájemně implementaci a procesy.

Je nutné rozhodnout, jak používat výstupy, nejen mají-li být použity. V této fázi disciplína klade důraz na to, aby skutečně veškeré výstupy měli věcnou souvislost s projektem. Návod podává přehledný náhled na doporučené a volitelné výstupy. Uvedeny jsou pak tyto produkty:

Implementační model

- všechny projekty by měly mít model s prvky a s minimálním obsahem zdrojového kódu a spustitelných programů
- některé projekty zahrnují i subsystémy, knihovny a obrazové modely

Plán integrace buildu

- je volitelný
- doporučuje se využívat pokud integrace není zcela triviální
- určuje organizaci, jak by měly být jednotlivé části implementovány, které buildy je třeba při integraci systému vytvořit a jak by měly být hodnoceny

Je nutné se zaměřit na pokrytí jednotkovými testy. Nejde o rozhodnutí, zda testovat, či nikoliv, ale o určení míry pokrytí jednotkovými testy ve smyslu od neformální po 100%ní pokrytí testy.

Většinou pak záleží na potřebách integračních a systémových testerů, kterým jsou předávány jednotlivé výstupy. Ti jsou nejvíce závislí na kvalitě předávaného kódu. Pokud je kód příliš chybový, budou nuceni posílat ho zpět implementátorům příliš často, což se celkově velmi podepíše na kvalitě vývoje. Řešením je samozřejmě donutit implementátory více testovat přesto, že pak stále nelze očekávat 100%ní správnost kódu.

Pokrytí se také může lišit v různých fázích projektu a metodika uvádí například bezpečnostně-kritický projekt, který bude potřebovat 100%ní pokrytí testy ve fázi “construction and transition”, nikoliv však ve fázi “elaboration”.

Dále je třeba rozhodnout, jak kontrolovat a opravovat kód. Tato činnost má své výhody i nevýhody. Jako výhody metodika uvádí existenci možnosti sjednotit styl kódování a donutit tak všechny spolupracovníky psát kód stejně, vyhledání chyb, které nemůže automatické testování najít a sdílení znalostí napříč organizací nezávisle na zkušenostech týmu. Jako nevýhody pak metodika vyzdvihuje potřebu dalšího času a zdrojů a možnost, že přezkoumávání by bylo prováděno pouze z nutnosti a tudíž by nepřinášelo žádnou přidanou

hodnotu.

Přezkoumávání kódu je tedy důležité, dokážeme-li z něj získávat přidanou hodnotu a objektivně hodnotit jeho přínosy. V mnoha organizacích je však vývoj této oblasti zastaven například z důvodu nedostatečného množství shromažďovaných dat, nebo naopak jejich obrovským rozsahem, nebo například administrativní náročností v kombinaci s již zmiňovaným problémem, že někdo dělá přezkoumání jen proto, že musí vyplnit nějaký formulář.

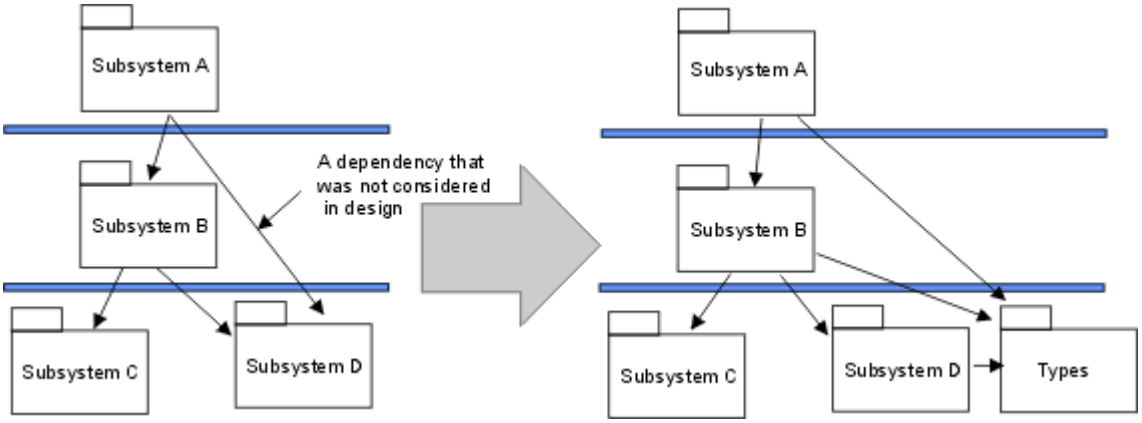
Činnosti

Strukturování vývojového modelu (Structure the Implementation Model)

Role: Software Architect (Softwarový architekt)
Vstupy: Návrhový model (Design Model)
Výstupy: Implementovaný podsystém (Implementation Substem)
Softwarová architektura (Software Architecture)
Model implementace (Implementation model)

Tato činnost popisuje, jakým způsobem by měla být určována struktura vývojových prvků v závislosti na přiřazených odpovědnostech pro vývoj pod-systémů a jejich obsahu

- **Určení struktury vývojového modelu**
- **Seřazení vývojových pod-systémů:** Upravení struktury modelu z pohledu organizace týmů nebo programovacího jazyka.



Obrázek 7: Seřazení podsystémů

- **Určení vstupů pro všechny vývojové pod-systémy:** Definování závislostí mezi jednotlikvými pod-systémy.
- **Rozhodnutí jak zacházet se spustitelnými programy:** Určení struktury konfigurovatelných prvků a jejich aplikace na vývojový model.
- Rozhodnutí jak zacházet s testovacími prvky
- **Aktualizace vývojového pohledu:** Úprava vývojového pohledu na softwarovou architekturu.
- **Hodnocení vývojového modelu**

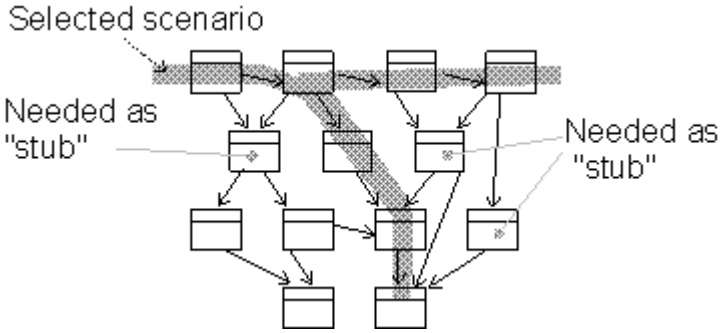
Plánování propojení pod-systémů (Plan Subsystem Integration)

Role: Integrator
Vstupy: Implementovaný podsystém (Implementation Substem)
Iteration Plan (Plán Iterace)

Výstupy: Plán integrace sestavení (Integration Build Plan)

Popisuje, jakým způsobem by se měl plánovat postup při propojování jednotlivých pod-systémů:

- **Určení sestavení:** Podle případů užití a scénářů se určí jednotlivé přírůstky integrace a ty se poté postupně propojují.
- **Identifikace tříd:** Je potřeba určit třídy, které budou potřeba pro implementaci určeného pod-systému. Některé již můžou být implementovány.



Obrázek 8: Identifikace tříd

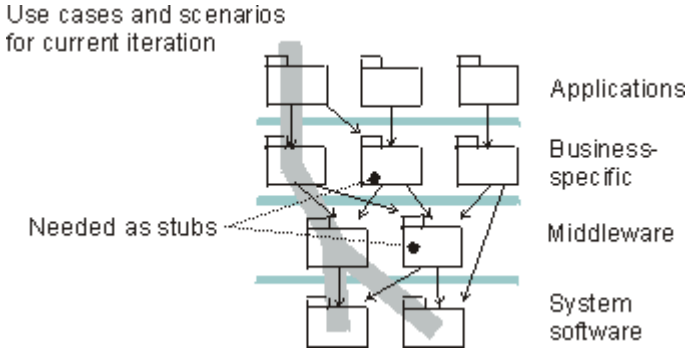
- **Aktualizace pod-systémových vstupů:** Pokud je kvůli novému sestavení zapotřebí integrace dalších podsystémů, je potřeba to zahrnout také do plánů.

Plánování propojení systému (Plan System Integration)

Role: Integrator
Vstupy: Plán iterace (Iteration Plan)
Výstupy: Plán integrace sestavení (Integration Build Plan)

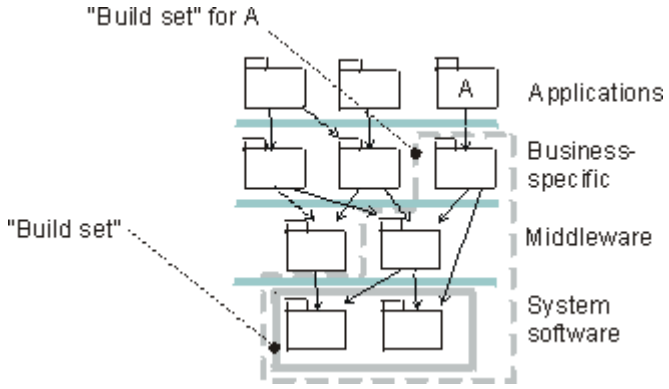
Činnost vysvětluje jak plánovat propojení celého systému.

- **Určení podsystémů:** V plánu iterace jsou popsány všechny případy užití a scénáře, které mají být v dané iteraci implementovány. Podle toho je poté potřeba určit všechny pod-systémy, které se mají implementovat.



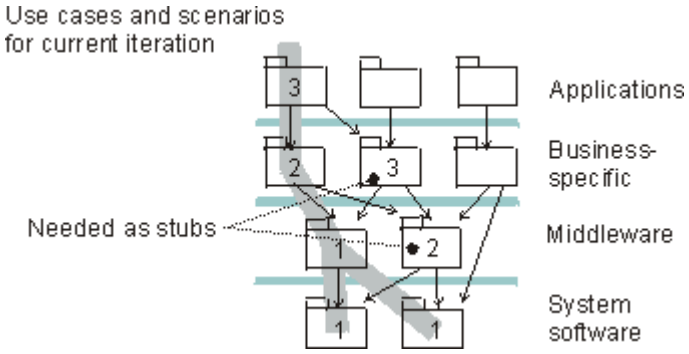
Obrázek 3: Určení všech pod-systémů pro propojení

- **Určení sad sestavení:** U rozsáhlých projektů je vhodné rozdělit systém na jednotlivé sady sestavení, které spolu nějakým způsobem souvisí a podle toho poté plánovat jejich integraci.



Obrázek 10: Určení Sad sestavení

- **Určení posloupnosti sestavení:** V jakém pořadí budou jednotlivá sestavení integrována. Většinou se začíná na nejnižší úrovni a pokračuje se výše s tím, že vždy musí být zahrnuty všechny potřebné podsystémy pro chod daného sestavení.



Obrázek 11: Určení pořadí sestavení

- **Hodnocení plánu propojení sestavení:** Provádí se hodnocení, zda postup integrace umožňuje jednoduché zjišťování chyb, potřeba podpůrných programů je minimální a jestli je postup integrace shodný s postupem vývoje

Rozhodnutí o provedení služeb (Document Service Realization Decision)

Role: Designer (Návrhář)
Vstupy: Refernce Architecture
Servisní model (Service Model)
Softwarová architektura (Software Architecture)
Výstupy: Návrhový model (Design model)

Účelem činnosti je poskytnout vývojářům model pro implementaci služeb aby mohla být poskytována nově požadovaná služba.

Skládá se z těchto kroků:

- **Určení přístupu ke zdrojům:** Určení jakým způsobem se bude přistupovat k novým nebo již novým zdrojům.
- **Určení přístupu k vývoji:** Doporučuje se provádět model služeb s použitím modelu návrhu.
- **Mapování existujících služeb:** Záznam všech služeb.

Vývoj navrhnutých prvků (Implement Design ELements)

Role: Implementer (Vývojář)
Vstupy: Návrhový model (Design model)
Vývojový prvek (Implementation Element)
Výstupy: Vývojový prvek (Implementation Element
Vývojový pod-systém (Implementation Subsystem)

Úloha popisuje, jakým způsobem postupovat při vývoji jednotlivých prvků, které byly předem namodelovány. Také popisuje jak napravovat případné chyby. Výsledkem je většinou nový nebo upravený zdrojový kód a datové soubory.

Jednotlivé kroky jsou:

- **Příprava na vývoj:** Jako první musí vývojář pochopit podstatu úkolu nebo problému, poté musí správně nastavit vývojové prostředí, provést analýzu již vyvinutých prvků a nakonec vše postupně implementovat.
- **Přeměna návrhu na kód:** V závislosti na typu modelu lze použít různé techniky.
- **Dokončení vývoje:** Pokud změny ovlivňují více prvků, je potřeba vše sjednotit.
- **Hodnocení vývoje:** Ověření, zda nové prvky naplnili očekávání.
- **Poskytnutí zpětné vazby:** Pokud se během vývoje objeví problémy, tak je někdy potřeba upravit i původní modely.

Analýza běžného chování (Analyze Runtime Behavior)

Role: Implementer (Vývojář)
Vstupy: Implementační prvek (Implementation Element)
Výstupy: Výsledky testů (Test Results)

Účelem činnosti je analýza běžného chování komponent systému během jejich vykonávání jejich běžných činností a snaha o nalezení případných slabých míst, která můžou být zlepšena. Detailní poznání chování komponent může napomoct právě při hledání chyb v implementaci. Skládá se z několika dílčích činností:

- **Výběr vhodného scénáře běžného použití:** Cílem je určit nejvhodnější scénář, který bude co nejvíce představovat alespoň část běžného použití systému. Při výběru je proto nejlepší konzultace přímo se samotnými koncovými uživateli systému. Další možností jak vybrat vhodný scénář pak může být analýza UseCase diagramů, případně konzultace s testery.
- **Příprava implementačních komponent na provedení pozorování:** Zahrnuje všechny kroky potřebné pro nastavení komponent do stavu, který umožňuje simulaci běžného provozu (například příprava frameworků nebo podpůrných nástrojů, zaznamenávajících průběh systémem)
- **Příprava prostředí pro provedení pozorování:** Zajištění požadovaného nastavení cílového prostředí.
- **Provedení simulace běžného chování a její záznam:** Po připravení komponent a prostředí se provede samotné pozorování a záznam výsledků. V závislosti na technice a nástrojích může tento krok proběhnout automatizovaně nebo s účastí pozorovatele.
- **Kontrola výsledků pozorování a prvotní identifikace nálezů:** Buď již v průběhu provádění pozorování, nebo po jeho skončení, se zaznamenávají výsledky, které můžou být chybami nebo se odlišují od normálu.
- **Analýza identifikovaných nálezů za účelem porozumění:** Detailní zkoumání příčin všech identifikovaných problémů a odlišností.
- **Identifikace a komunikace následujících akcí:** Zajištění dalších potřebných akcí k vyřešení nalezených chyb a problémů. Jedná se např. o další podrobnější zjišťování příčin chyb nebo přímo opravné akce.
- **Hodnocení výsledků:** Kontrola, zda byly všechny úkoly správně provedeny a výsledky práce jsou akceptovatelné.

Zavádění testovatelných prvků (Implement Testability Elements)

Role: Implementer (Vývojář)
Vstupy: Testovatelná třída (Testability Class)
Výstupy: Testovatelný prvek (Testability Class)
Test Stub

Popisuje, jakým způsobem implementovat speciální funkcionalitu pro podporu testovacích požadavků:

- **Zavádění ovladačů unit testů:** Určení a zavedení všech důležitých komponent, umožňujících funkcionalitu testů.
- **Zavádění rozhraní unit testů do automatizovaných testovacích nástrojů:** Určení rozhraní, která jsou nezbytná pro funkcionalitu automatizovaných testů.

Zavádění vývojových testů (Implement Developer Test)

Role: Implementer (Vývojář)
Vstupy: Vývojový prvek (Implementation Element)
Výstup: Vývojový test (Developer Test)

Popisuje jak zavádět testy pro ověření činnosti jednotlivých komponent během vývoje. Cílem je implementovat jeden nebo více testů a umožnit jejich pozdější zapojení do komplexních testů.

Kroky při zavádění:

- **Vylepšení rozsahu a identifikace testů:** Určení nejlepšího rozsahu testů.
- **Výběr vhodné techniky implementace:** Pokud jsou k dispozici různé techniky je potřeba vybrat tu nejvhodnější. Dvě základní techniky jsou automatické nebo ruční testování.
- **Implementace testu:** Účelem je implementovat všechny testy definované v prvním kroku.
- **Sestavení externích datových sad:** Vytvoření sady dat, která budou použita při testování. Je potřeba vybrat vhodné množství dat.
- **Ověření implementace testů:** Test testu - zda všechny testy pracují tak jak mají.
- **Zachování sledovatelnosti vztahů:** V závislosti na formálnosti, je někdy potřeba zachování sledovatelnosti vztahů.

Provedení vývojových testů (Execute Developer Tests)

Role: Implementer (Vývojář)
Vstupy: Vývojové testy (Developer Test)
Implementační jednotky (Implementation Element)
Výstup: Log testů (Test Log)

Úkol popisuje jak provádět a hodnotit testy na ověření funkčnosti komponent, před tím než jsou proveden hlavní testy. Cílem je ověřit funkčnost a vnitřní struktury testované jednotky.

Skládá se z těchto kroků:

- **Počáteční příprava:** Příprava pro implementaci unit testů. Unit testy se zde nemyslí pouze testy tříd z objektově orientovaného programování, ale také například funkce.
- **Provedení unit testů:** Jako první se musí nastavit prostředí (HW, SW, data, nástroje, atd.), poté se připravuje testovací prostředí a nakonec jsou provedeny samotné testy. Je důležité zda se jedná o automatizované nebo ručně prováděné testy.
- **Hodnocení provedených tesů:** Určení, zda testy proběhly v pořádku nebo jsou potřeba nápravné akce.

- **Ověření výsledků testů:** Zjišťuje, zda jsou výsledky opravdu v pořádku, případně určuje potřebné nápravné akce (v testech nebo přímo na pracovních produktech).
- **Oprava zaseknutých testů:** Pokud se testy zaseknou tak je potřeba je upravit a spustit znovu.

Kontrola kódu (Review Code)

Role: Technical Reviewer
Vstupy: Vývojové prvky (Implementation Element)
Výstupy: Záznam hodnocení (Review Record)

Jak by měl být kontrolován a ověřován kód.

- **Základní doporučení:** Soupis základních doporučení, které by měly být dodržovány při vývoji softwaru. Jako nejdůležitější je uváděna kontrola kódu pomoci techniky inspekce, procházení nebo čtení kódu
- **Určení kontrolních bodů vývoje:** Seznam záchytných bodů, které by měly být kontrolovány během vývoje. Rozdělené do tří kategorií: obecné, komentáře a zdrojový kód.
- **Připravení kontrolních záznamů a záznamu chyb:** Tvorba dokumentů zaznamenávající výsledky a chyby.

Propojení pod-systémů (Integrate Subsystem)

Role: Integrator
Vstupy: Vývojové prvky (Implementation Element)
Plán integrace sestavení (Integration Build Plan)
Výstupy: Sestavení (Build)
Vývojový pod-systém (Implementation Subsystem)

Činnost popisuje, jakým způsobem integrovat jednotlivé menší vývojové prvky do větších vývojových pod-systémů. Tyto pod-systémy poté slouží pro celkovou systémovou integraci:

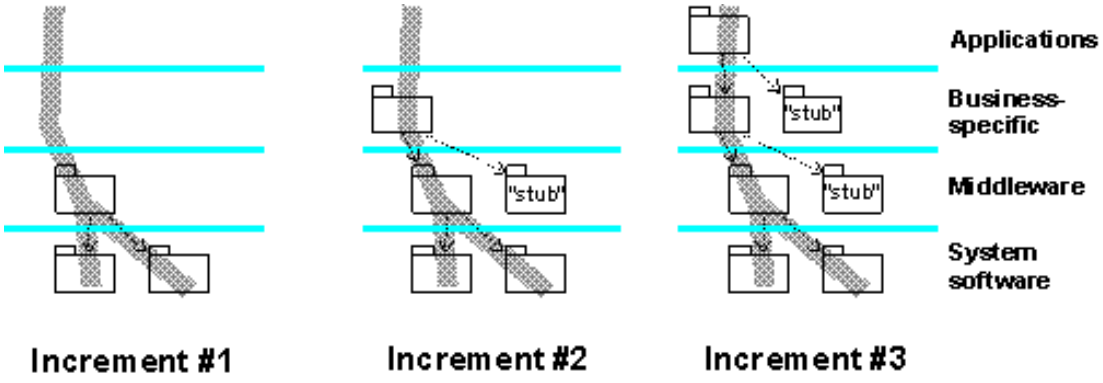
- **Integrace vývojových prvků:** Provádí se podle Plánu sestavení. Je potřeba ji provádět postupně (tzv. odspoda nahoru), aby byly minimalizovány chyby. Zvláštní pozornost je potřeba věnovat při sdílení kódu více vývojáři.
- **Doručení vývojového pod-systému:** Po integraci a otestování jednotlivých pod-systémů jsou tyto pod-systémy přesunuty a integrovány do jednoho systému.

Propojení systému (Integrate Systém)

Role: Integrator
Vstupy: Vývojové prvky (Implementation Element)
Plán integrace sestavení (Integration Build Plan)
Výstupy: Sestavení (Build)

Popisuje, jak integrovat jednotlivé vývojové pod-systémy do konečného sestavení (buildu):

- **Potvrzení pod-systému a vytvoření průběžného sestavení:** V závislosti na rozsahu systému je vhodné integrovat jednotlivé pod-systémy postupně do tzv. průběžných buildů.

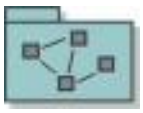


Obrázek 12: Znázornění pod-systémů

- **Kontrola základních požadavků:** Zjištění, zda sestavení splňuje nebo nesplňuje všechny předpoklady.

Produkty

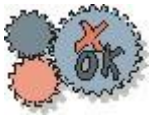
Sestavení (Build)



Artefakt produkující funkční verzi systému nebo jeho části. Sestavení zahrnuje jeden nebo více vývojových prvků. Účelem sestavení je poskytnutí testovatelné části běžných funkcí a kapacit systému. Sestavení můžou být tvořena ve všech fázích vývoje, podstatné jsou však ty, která jsou definována v plánu integrace sestavení, protože zde jsou uvedeny milníky potřebné pro dokončení iterace.

Sestavení můžou být tvořena z několika důvodů, například pro jednotkové testování. Nicméně hlavními jsou ty, které tvoří integrátor z identifikovaných verzí jednotlivých prvků vytvořených vývojáři a spojených do pod-systémů nebo uloženými v pracovním prostředí pro integraci.

Vývojový test (Developer Test)

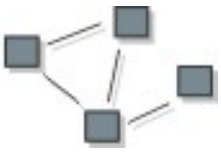


Zahrnuje výstupy, které se nejčastěji objevují při jednotkovém testování, integračním testování nebo také systémové testování. Účelem je správné a efektivní testování jednotlivých prvků. Každý vývojářský test by měl zahrnovat testování různých oblastí:

- Základní hardwarové požadavky (procesor, paměť, pevný disk)
- **Základní software** (operační systém, e-mail, kalendář)
- **Další vstupní a výstupní zařízení** (čtečka čárových kódů, tiskárna dokladů, pohybové senzory)
- Software pro vstupní a výstupní zařízení (ovladače, rozhraní, software)
- Soubor minimálně potřebného softwaru pro provedení testů, hodnocení a diagnostiku (diagnóza paměti, automatické provádění testů)
- **Požadované nastavení HW a SW** (rozlišení obrazovky, umístnění zdrojů, proměnné prostředí)
- **Další požadované existující vstupy** (stvrzenky z tiskárny účtenek, data o počtu obyvatel)

Většina vývojářských testů se tvoří přímo během programování, kdy je potřeba testování jednotlivých komponent. Testy spouštěné požadavky na změnu, jsou vyvíjeny až po vývoji komponent a posplnění svého účelu většinou zanikají.

Implementační model (Implementation Model)



Zobrazuje fyzické složení implementace pomocí pod-systémů a vývojových prvků (složky a soubory, zdrojový kód, data, spustitelné soubory). Díky tomu můžou být jednotlivé prvky lépe pochopeny a řízeny, protože dají každé samostatně verzovat, vyvíjet nebo nahrazovat.

Detailnější model zobrazující i zdrojový kód a výstupy se doporučuje pouze pokud funguje automatická synchronizace mezi modelem a soubory.

Plán integrace sestavení (Integration Build Plan)

Účelem je určení detailního plánu, v jakém pořadí budou jednotlivé komponenty vyvíjeny, které sestavení budou tvořena, kdy integrovat systém do celku a jak se bude průběh hodnotit. Plán integrace je důležitý pro:

- **Implementátoři:** Pro plánování v jakém pořadí implementovat prvky a co a kdy je potřeba poslat na integraci systému.
- **Integrator:** jako plánovací nástroj.
- **Návrhář testů:** pro definování co je potřeba testovat

V závislosti na velikosti projektu, je možné vytvářet i další pod-plány.

Porovnání RUP pro malé a velké projekty

Protože je metodika RUP velmi obsáhlou a komplexní, bývá její použití většinou výhodné pouze u velkých projektů, na nichž pracují velké týmy. U takovýchto projektů by naopak nepoužití některé z již v praxi zavedených a odzkoušených metodik bylo velmi riskantní. Proto společnost IBM vyvinula i metodiku RUP pro malé softwarové projekty. Ta metodiku RUP v mnoha věcech značně zjednodušuje a tím usnadňuje její použití. V disciplíně Implementation se konkrétně jedná o vypuštění činností Rozhodnutí o provedení služeb (Document Service Realization Decision) a Zavádění testovatelných prvků (Implement Testability Elements). Přesto však i takto upravená metodika zůstává velmi složitou a rozsáhlou. Navíc i tato její verze je placená. Pro malé projekty se proto může zdát jako nejlepší použití metodiky OpenUP, která také vychází z metodiky RUP, ale s jejím zjednodušením jde ještě mnohem dále a hlavně se jedná o open-source metodiku, každému zdarma dostupnou.

Vlastní zhodnocení

RUP obecně je dle našeho názoru dosti rozsáhlou a i z finančních důvodů hůře dostupnou metodikou. I přesto jí považujeme za velmi dobře navrženou a efektivní, avšak pro běžného uživatele téměř nepoužitelnou, jelikož orientace v levém rozklikávacím menu není rozhodně snadno pochopitelná, natož intuitivní.

Co se týče disciplíny Implementation samotné, je podle nás velmi dobře popsána a její využívání, kromě problémů s vyhledáváním v ní, není složité. Popisy jednotlivých činností, rolí a produktů jsou detailně popsány a mají tak velkou vypovídací hodnotu. Proto vidíme disciplínu jako velmi přínosnou, i když je podmínkou, že s ní pracuje kvalifikovaná osoba.

Závěr

Cíle práce byly podle nás splněny v plném rozsahu. Čtenář byl seznámen s disciplínou a jejími částmi, disciplínu jsme zhodnotili a porovnali. Práce nám přinesla nový pohled na metodiku jako celek i disciplínu samotnou.

Největší přínos této semestrální práce je pro nás pak alespoň částečné pochopení jak se orientovat v metodice RUP a k čemu slouží disciplína Implementation. Ta by totiž mohla být chápána v jiných souvislostech, než jak ji chápeme teď.

V závěru bychom také měli upozornit na některé pilíře. My jsme vybrali tyto:

- je důležité realizovat jednotlivé designové prvky z hlediska implementačních prvků (zdrojové soubory...)
- je potřeba provádět jednotkové testy, avšak pokrytí záleží na jejich potřebě
- je nutné integrovat jednotlivé moduly (subsystémy) do konečného spustitelného programu (systému)

Zdroje

Kitscm [online]. 2006 [cit. 2012-10-25]. Classic RUP for SOMA. Dostupné z WWW: <https://kitscm.vse.cz/RUP/LargeProjects/#core.base_rup/disciplines/rup_deployment_discipline_A1C86C42.html>.

Kitscm [online]. 2006 [cit. 2012-10-25]. RUP for small projects. Dostupné z WWW: <https://kitscm.vse.cz/RUP/SmallProjects/index.htm#core.base_rup/guidances/supportingmaterials/welcome_2BC5187F.html>.

ALDORF, Filip. Objekty - Objekty - Metodika RUP (Rational Unified Process) [online]. 2005, [cit. 2012-10-25]. Dostupné z WWW: <http://objekty.vse.cz/Objekty/RUP>