ROČNÍKOVÝ PROJEKT 1

URČENO PRO VZDĚLÁVÁNÍ V AKREDITOVANÝCH STUDIJNÍCH PROGRAMECH

JAROSLAV PROCHÁZKA

ČÍSLO OPERAČNÍHO PROGRAMU: CZ.1.07
NÁZEV OPERAČNÍHO PROGRAMU:
VZDĚLÁVÁNÍ PRO KONKURENCESCHOPNOST
OPATŘENÍ: 7.2
ČÍSLO OBLASTI PODPORY: 7.2.2

INOVACE VÝUKY INFORMATICKÝCH PŘEDMĚTŮ VE STUDIJNÍCH PROGRAMECH OSTRAVSKÉ UNIVERZITY

REGISTRAČNÍ ČÍSLO PROJEKTU: CZ.1.07/2.2.00/28.0245

OSTRAVA 2012

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky

Recenzent: RNDr. Jaroslav Žáček, Ph.D.

Název: Ročníkový projekt 1
Autor: RNDr. Jaroslav Procházka, Ph.D.
Vydání: první, 2012
Počet stran: 71

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

Obsah

1 Úvod

Předmět Ročníkový projekt (jdoucí přes 2 semestry) je zaměřen hlavně na praktickou část vývoje podle RUP/OpenUP 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 1*) měla stačit.

Tento studijní text je podkladem pro vaši praktickou práci, abyste chápali průběh a cíle iterací, 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 to možné ukazujeme aplikace principů OpenUP na konkrétních příkladech, včetně použití technologií.

První semestr ROPR je zaměřen na pochopení problematiky vývoje softwaru a týmové práce, zmapování problémové domény, zachycení požadavků, prototypy, nastavení nástrojů a prostředí, identifikace několika málo klíčových use casů a jejich scénářů a jejich iterativní analýza, návrh, implementace, testování a sestavení. Jedná se tedy o fáze Inception a Elaboration.

Druhý semestr pak pokrývá hlavně dopracování a detailnější pochopení artefaktů z předchozí fáze Elaboration a přesnější, detailnejší postup podle těchto principů v několika málo iteracích fáze Construction.

Je důležité si uvědomit, že v prvním semestru budou akceptovány i méně kvalitní modely a návrhy, aby se týmy pohnuly kupředu. Budeme uznávat i ne zcela vhodné a ne zcela přesné modely a dokumentaci, jelikož hlavním cílem je pohnout se kupředu a získat základní povědomí o procesu tvorby softwaru. Týmy se také budou učit týmové práci, před kterou je celé dosavadní studium bránilo. Je třeba si tedy uvědomit, že dříve akceptované modely nemusí být vždy vyhovující a tudíž můžeme požadovat jejich přepracování. Jejich „kvalita“ vyhovovala potřebám rychlého naučení a postupu v prvním semestru, ale pro praktické využití by mohly způsobit mnoho problémů. Proto jedním z hlavních úkolů druhého semestru je:

- zrevidovat nevyhovující modely a „hloupé dětské obrázky“ přepsat do standardních UML modelů,
- refaktorovat existující kód,
- věnovat se více testování, zvláště automatizovanému,
- implementovat další agilní praktiky.

2 Projekt podle OpenUP/RUP

V této kapitole se dozvíte:

- Jaký použijeme model vývoje pro náš praktický projekt?
- Co je to iterace a jaký má průběh?
- Jaké jsou fáze, kterými projdeme včetně jejich kontrolních milníků?

Po jejím prostudování byste měli být schopni:

- Popsat základní fáze OpenUP/RUP včetně jejich náplně.
- Pochopit detailně náplň iterace.
- Představit si, co vás očekává v následujícím semestru.

Klíčová slova této kapitoly:

Iterativně inkrementální model, fáze, iterace, milníky.

Doba potřebná ke studiu: 1 hodinu

Průvodce studiem

Kapitola představuje iterativně inkrementální model vývoje softwaru pomocí kterého budeme vyvíjet náš praktický projekt. Připomeneme jednotlivé fáze a důležitost jejích milníků. Detailně se podíváme na průběh iterace.
Na studium této části si vyhraďte 1 hodinu.

V textu Informační systémy se zabýváme teorií: modely vývoje, principy a praktiky. Tento praktický text doplňuje problematiku iterativně-inkrementálního vývoje podle moderních trendů z praktické stránky a tudíž má sloužit studentům jako praktická kuchařka, podle které mohou postupovat v Ročníkovém projektu. Začneme připomenutím základních vlastností iterativně inkrementálního modelu vývoje, který budeme následovat.

2.1 Model vývoje

Jako vhodný start osvěžme iterativně inkrementálné model vývoje, který v našem projektu budeme následovat. Základní rysy tohoto modelu jsou následující:

- Vývoj a celý projekt je ***řízen hodnotou pro zákazníka***. Snažíme se doručit co nejdříve základní scénáře aplikace ve spustitelné formě, aby ji zákazník mohl co nejdříve využívat. Takže nepřemýšlíme v intencích jednotlivých technologií či softwarových vrstev, ale v intencích celků (scénářů), které jsou napříč všemi vrstvami a doručují hodnotu zákazníkovi.
- ***Komunikace*** uvnitř týmu a se zákazníkem. Svoje pochopení a doměnky ověřujeme dotazem přímo u zákazníka (či jeho zástupce, lektora kurzu) abychom svoje domněnky a pochopení ověřili nebo vyvrátili.
- Tato komunikace je demonstrována častými demonstracemi, interními demonstracemi a mikro milníky jako součást iterací.
- Dokumentace je nedílnou součástí projektu.

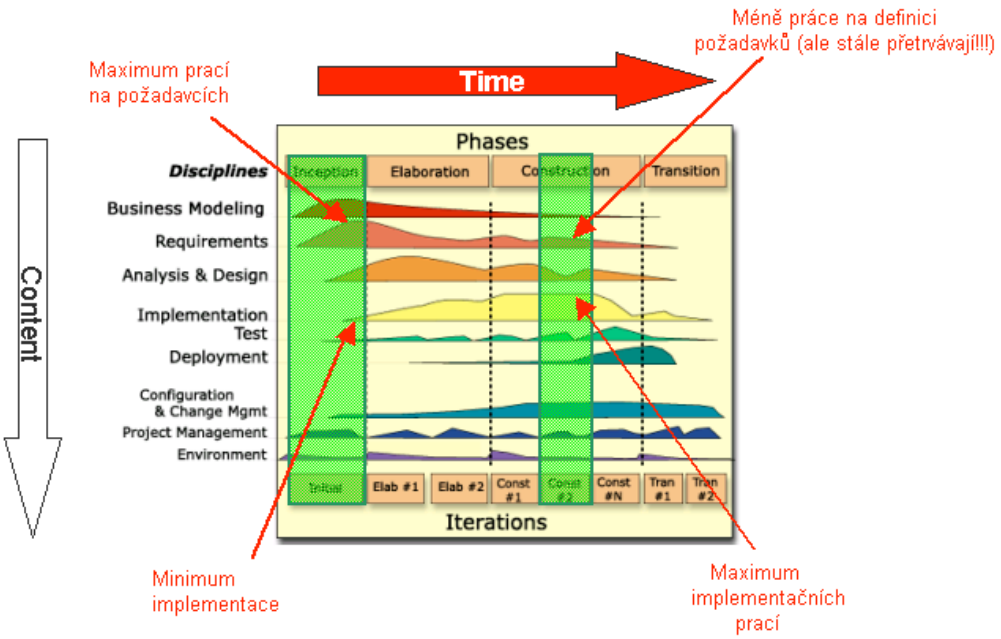
Základem tohoto modelu jsou ***iterace***, které jsou sdružovány do větších celků, fází, jelikož v určitých fázích projektu se zaměřujeme na jiné aspekty. Každý takový celek – fázi, pak ověřujeme kontrolním milníkem, který je součástí hodnocení poslední iterace v dané fázi.

Základní agilní praktiky, které použijeme v ROPR a které charakterizují tento model jsou následující:

- ***Time-boxing:*** časově omezené intervaly (iterace), které budou muset vždy doručit spustitelný výsledek.
- ***Dvouúrovňové plánování:*** projektový plán jako nástin směru a záměru s viditelnými cíli iterací a detailní iterační plány se seznamem úkolů.
- ***Use casy resp. scénáři řízený přístup:*** plánujeme podle UC, UC jsou cíle iterací, testujeme podle UC, navrhujeme podle UC.
- ***Definice hotovo:*** akceptační kritéria jsou součástí každého scénáře; součástí každé iterace je definice toho, co to znamená hotovo, jinak nevíme, co vše máme doručit.
- ***Testy řízený vývoj***, či alespoň částečné pokrytí kódu unit testy.
- ***Týmová a párová práce:*** klíčem je spolupráce v týmu, ti co umí programovat to učí v páru ostatní, ti pak mohou psát testy či jednodušší (opakující se) implementace.
- ***Refaktoring:*** pokud budete rozšiřovat aplikaci, je nutné upravit čitelnost a strukturu kódu beze změny chování aplikace, nutnou podmínkou bezpečného refaktoringu je dostatečné pokrytí kódu unit testy.
- ***Podpora nástroji:*** repozitář (repository) kódu a dokumentace je jediné bezpečné úložiště při práci více vývojářů; existující frameworky ušetří spoustu práce se základní obsluhou aplikace, mapovaním do databáze, komunikací, s tvorbou uživatelského rozhraní.

2.2 Fáze

OpenUP/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 každá fáze iterativně inkrementálního projektu obsahuje několik iterací a produkuje spustitelný kód. Smyslem fází je však seskupit iterace s podobným záměrem. Iterativně inkrementální projekt tedy není jen sledem stejných iterací. Dokonce i délka iterací v různých fázích se může lišit. Následující obrázek ukazuje základní fáze, jejich iterace a množství prací jednotlivých disciplín v nich prováděných:



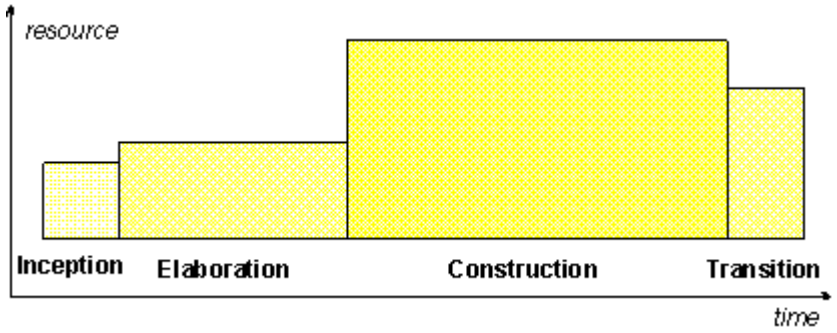
Obr. 2-1: Dvojrozměrný model OpenUP/RUP - fáze a disciplíny.

Smyslem fází OpenUP/RUP je tedy seskupit iterace do logických celků podle záměru jednotlivých úseků projektu. Tak například výsledkem Inception je pochopení problematiky, vize projektu, identifikovaná rizika, shoda na obsahu projektu a jeho financování. Výstupem Elaboration je spustitelná, otestovaná architektura ve formě fungující části aplikace a implementované rizikové scénáře, na kterých architekturu demonstrujeme. Výstupem Construction je již tzv. *beta-release* aplikace, relativně stabilní, opět spustitelná a 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, distribučních balíčků, školení apod. Zásadní tedy je, že každá fáze obsahuje několik iterací, v nichž se vždy provádí všechny disciplíny s různým objemem prací – což je reprezentováno plochou pod danou křivkou (viz Obr. 2-1), ale také s různým záměrem.

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, jen několik zkušených vývojářů a 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 si mohli ověřit dosažitelnost cílů a zákazníkovi co nejdříve poskytnout spustitelný software.

Účast lidí a množství potřebných zdrojů na této fázi je omezená. Celkově by Inception měla zabírat 10% celkového času, spíše méně.



Obr. 2-2: 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 2-1: Průměrné časy trvání jednotlivých fází RUP (zdroj: RUP)

Elaboration obsahuje o málo více lidí a zdrojů a časově je její trvání zhruba 30% celkového času. V Elaboration je zapojen malý „útočný“ tým (tiger team), který obsahuje zkušené borce schopné navrhnout a implementovat nejvhodnější návrh. Tito lidé se pak stanou architekty, senior programátory, team leadery dalších týmů zahrnutých do projektu. Z obrázku i tabulky je patrné, že nejvíce času a nejvíce lidí a 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 lidí a zdrojů je využíváno pořád hodně, jelikož dodělváme zbývající jednodušší rysy, řešíme finální vyladění produktu a opravu zbylých chyb, vylaďujeme výkonnost, kompletujeme dokumentaci a provádíme závěrečné testy. Toto je základní model pro typické projekty.
Projekty výzkumného typu a projekty nového charakteru budou mít zcela jiné poměry. Většinou bude maximumu práce v Elaboration se zaměřením na hodně iterací s prototypy s důrazem na stabilí architekturu a Construction už bude jen dodělávat podobné scénáře v menším rozsahu. ***Tento model bude i náš případ: Inception v našem výzkumném/inovativním projektu bude mít klidně více iterací pro modelování problémové domény, pochopení požadavků a vyzkoušení různých architektur v různých technologiích a frameworcích. Maximální pozornost a rozsah prací (rozuměj počet iterací) bude v Elaboration***. Elaboration bude mít tedy i více iterací než samotná Construction, bude delší a proběhne zde většina prací.

2.3 Iterace

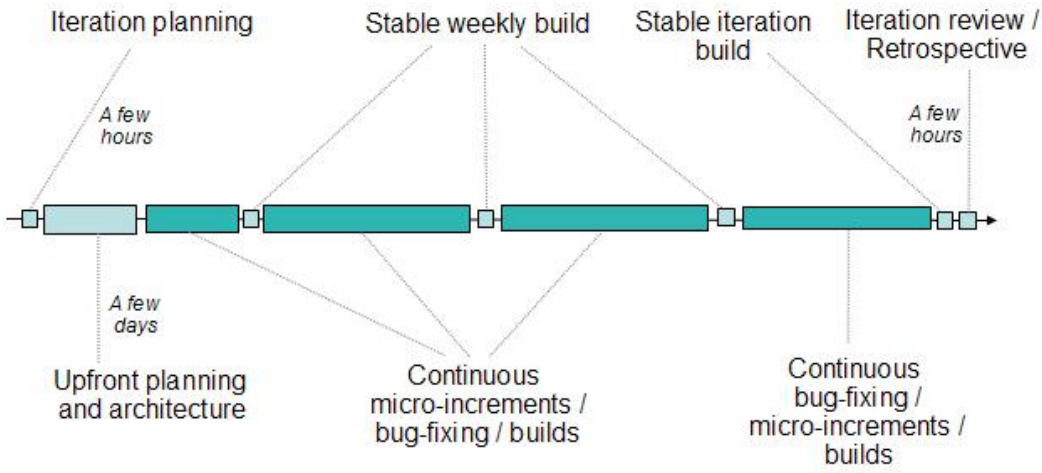
Jak již víte z předmětu SWENG, probíhá iterativně inkrementální vývoj v několika tzv. iteracích (opakováních), kdy:

- 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. Ptáme se, zda jdeme správným směrem. Zda jsme pochopili jeho potřeby dobře.

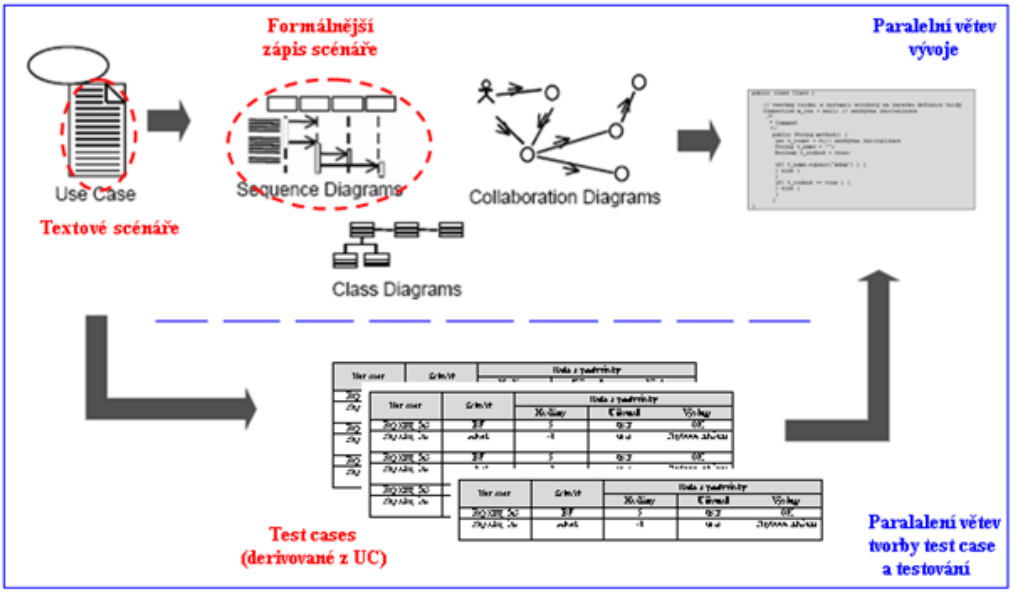
- Každá iterace má definovaný přesný cíl, který se snažíme naplnit, tímto cílem je opět konkrétní scénář nebo celý use case. Jako součást iterace pak provádíme paralelně analýzu, návrh, implementaci a testování vybrané nové funkčnosti s cílem doručení výsledku zákazníkovi na konci této iterace.
- Iterace je miniprojekt, což znamená, že má svůj začátek, konec a pevně definovaný časový rozsah (většinou 2 týdny) a úkoly pro jednotlivé členy týmu. Krátká doba iterace je vhodná, protože se předvídá a detailně plánuje lépe, než například 2 měsíce.
- 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 iterace je, že již po relativně krátké době má zákazník k dispozici nějakou verzi výsledného produktu, i když jde o nestabilní produkt bez některých scénářů a potřebných kontrol. Díky tomu si však s aplikací může zkoušet pracovat a již může říct, co se mu líbí, nelíbí, poskytne nám tedy velmi cenou zpětnou vazbu. V některých případech můžeme část aplikace i nasadit do provozu a aplikace může již vydělávat, i když neobsahuje zdaleka tolik rysů jako výsledný produkt. ***Pro vývojový tým je výborné, že si ověří, zda pochopení požadavků a použitý návrh odpovídají potřebám zákazníka***. Lidově řečeno, zda je to „to pravé ořechové“. Na Obr. 2-1 jsou 2 z iterací naznačeny zeleně, aby bylo zřejmé, že každá iterace obsahuje všechny disciplíny jen se liší množstvím práce podle dané fáze. Následující dva obrázky ukazují strukturu iterace ze dvou pohledů. Prvním pohledem je struktura iterace a jednotlivé mikro-milníky v rámci iterace.



Obr. 2-3: Struktura iterace a mikro-milníky (zdroj: OpenUP).

Druhý obrázek ukazuje iteraci z pohledu vyvíjených artefaktů a disciplín a jejich návaznost a paralelní práce v průběhu iterace.



Obr. 2-4: Artefakty, disciplíny a jejich návaznost a paralelní větve v průběhu iterace.

Detailní popis úkolů prováděných v iteraci bude součástí následujícího textu.

3 Fáze Zahájení (*Inception phase*)

V této kapitole se dozvíte:

- Jaké jsou úvodní kroky (iterace) projektu.
- Jaké jsou základní artefakty, které je vhodné vytvořit společně se zákazníkem.
- Jaký je kontrolní milník na konci této fáze.

Po jejím prostudování byste měli být schopni:

- Naplánovat, provést a vytvořit základní artefakty iterativně inkrementálního projektu.

Klíčová slova této kapitoly:

Vize, problémová doména, use case model, architektura, seznam rizik, projektový plán.

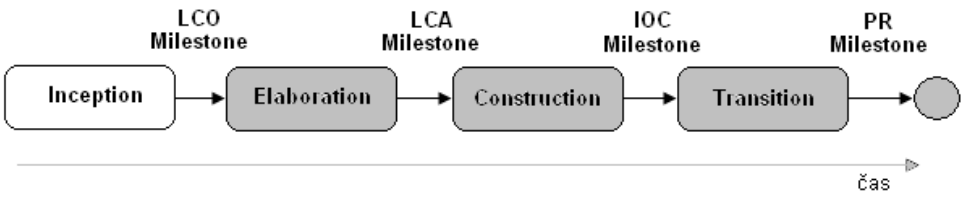
Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Kapitola představuje první fázi iterativně inkrementálního modelu vývoje softwaru, pomocí kterého budeme vyvíjet náš praktický projekt. Smyslem kapitoly je nastínit a pochopit cíle fáze zahájení, jaké základní aktivity provádíme a jaké artefakty vytváříme.
Na studium této části si vyhraďte 2 hodiny.

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. 3-1: 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.

3.1.1 Porozumění tomu, co vytvořit

Pochopení potřeb uživatele je často největším problémem softwarových projektů (viz graf Standish). 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 stanici vývojáře a reportuje určitý čas strávený prací na daném projektu (komplexnější šablona vize z OpenUP viz Příloha A – Vize):

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 3-1: 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.

Příklad: vytvořte vizi softwarového projektu, který má za cíl pomocí softwarové aplikace vytvořit osobní manažer úkolů a poznámek a tím odstranit nesourodé poznámky a připomínky v papírové formě, v mobilních zařízením, Google kalendáři či v e-mailových klientech. Aplikace by měla fungovat jako webová s přístupem přes PC i mobilní zařízení jako je mobilní telefon či tablet. Možností aplikace by měla být také synchronizace s těmito zařízeními.

Nezapomeňte si položit následující otázky:

- Co je skutečným problémem? Jaká je jeho přesná formulace?
- Co je řešením tohoto problému? Nezapomeňte, že řešení může být i změna způsobu práce bez použití softwaru, nás ale zajímá, jak software může pomoci tento problém vyřešit.
- Co jsou základní sady funkčností tohoto systému? Zajímají nás jen základní sady funkčností (tzv. use casy), žádné detailní popisy.

Vize: osbní manažer úkolů
Problém (co řešíme, co nefunguje, co nás trápí)
Řešení (jak tento problém popsaný v předchozím kroku budeme řešit)
Zainteresované strany (anglicky *stakeholders*)
Use casy (základní obecné funkčnosti)

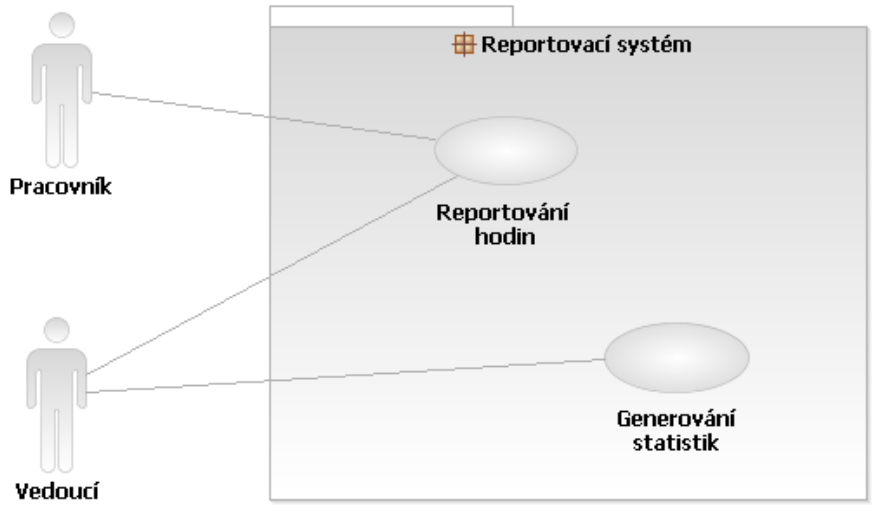
Tabulka 3-2: Šablona vize

3.1.2 Klíčová funkcionalita systému

Po definice cíle ve formě vize, je dalším bodem Inception fáze identifikace kritických use casů (dále jen UC). Většinou se jedná o 20 – 30% všech use casů, ale v technologicky složitějších projektech může jít až o 70% use casů. Kritické jsou z pohledu technologického. Tyto use casy tvoří technologickou kostru aplikace (tzv. architektura: tj. jednotlivé vrstvy, rozhraní, způsob komunikace či ukládání dat), nebo se jedná o funkčnosti, které musí aplikace nutně obsahovat. Kritické UC mají významný dopad na architekturu systému, jinými slovy ji tvoří (spolu s nefunkčními požadavky na systém). Analytiky je vytvořen jejich detailnější popis, ale je možné posunout zpodrobnění některých alternativních toků do dalších iterací, respektive do dalších fází (Construction), pokud nemají významný dopad na architekturu. Poté, co jsou identifikovány, jsou kritické UC představeny zbytku týmu a vysvětleny, proč jsou kritické. Ten by s nimi měl souhlasit.

Use case jako forma zápisu požadavků
Česky je někdy tato technika nazývána případy užití. Její výhodou je jednoduchost a grafické vyjádření požadavků na systém. Narozdíl od klasické specifikace reprezentované seznamem požadavků je tento přístup uživatelsky orientován. Ukazuje, co který uživatel (role) od budoucího systému očekává a jakým způsobem ho používá.

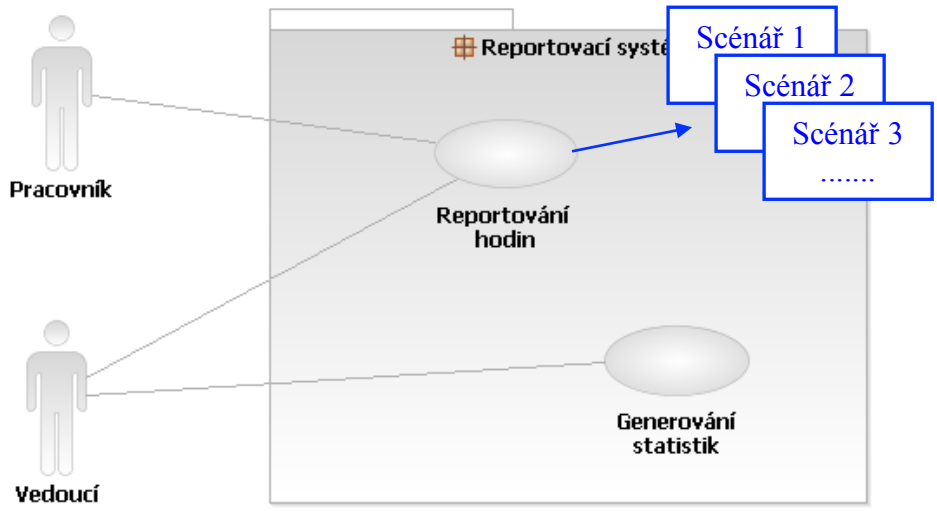
Tvorba use casů začíná identifikací aktorů – budoucích přímých uživatelů systému. S jejich pomocí poté identifikujeme očekávané vlastnosti a chování systému na abstraktní úrovni. Následující příklad ukazuje use case model našeho systému reportování času na projektu:



Obr. 3-2: Use case model: aktoři pracovník a vedoucí mohou používat zobrazené use casy.

Use casy popisují generické chování, které očekává uživatel od vytvářeného systému. Toto ***chování*** – use case – je pak ***detailně popsáno několika scénáři***. Scénář je mírně formalizovaný (strukturovaný) příběh, který zachycuje, jakým způsobem uživatel využívá konkrétní funkčnost systému. Pozor, jednotlivé

scénáře už se nezachycují v use case modelu! Pro názornost na následujícím obrázku ukážeme, jak se jeden use case rozpadá do více scénářů (zde se již nejedná o UML notaci!):



Obr. 3-3: Use case model ukazující vazbu use case a scénáře. Pozor, zde se již nejedná o standardní UML notaci! Scénáře do use case modelu tímto způsobem nezapisujeme.

Následující šablona ukazuje detailní strukturu scénáře [Kr03a]:

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

Příklad základního toku use case („happy day" či „basic flow") pro námi zachycený reportovací systém vypadat následovně:

Název use case: Reportování hodin
Aktor: Zaměstnanec
Počáteční podmínka:

- Zaměstnanec odpracoval určitou dobu na projektu
- Zaměstnanec je přihlášen k systému jako role zaměstnanec

Tok událostí (basic flow):

1. Zaměstnanec vybere správu projektů.
2. Systém zobrazí správu projektů.
3. Zaměstnanec vybere projekt, na kterém pracoval.
4. Zaměstnanec zadá počet hodin a datum strávené na projektu.
5. Systém ověří maximální počet hodin a datum.

6. Zaměstnanec odešle zapsaná a ověřená data.
7. Systém uloží data a oznámí úspěšné uložení.

Chybové toky:

5a. Pokud zaměstnanec zadá špatný formát, systém údaj neuloží a informuje o chybně zadaném formátu uživatele s výzvou na vložení správného údaje.

7a. V případě problémů s uložením je o tom uživatel systémem informován.

Můžete si všimnout, že jsme uvažovali pouze scénář, kdy je vše v pořádku, nebrali jsme v potaz potenciální problémy při špatně zadaném datu (například novější než dnešní datum), při více zadaných hodinách atd. Těmito stavy se zabývají alternativní a chybové toky, které jsme jen nastínili. V úvodních iteracích v Inception fázi je to běžný krok. Detaily scénářů dopracujeme až ve fázi Elaboration.

Chyby při tvorbě use case

Běžné chyby, se kterými se lze setkat při tvorbě use case, jsou následující:

- ***Příliš mnoho UC a jejich funkční dekompozice*** – způsobeno nepochopením použití UC. UC vystupují jako jednotlivé scénáře a ne jako jejich zastřešující obálka! Tím ztrácíme celkový pohled na daný příběh a hlavně vazby mezi jednotlivými scénáři, což způsobuje špatnou interpretaci požadavku a možné redundance kódu (stejná věc psána 2x pro oba scénáře).
- ***CRUDL use casy a scénáře*** – nevhodně formulované use casy či jejich scénáře, které nenásledují kroky, které vykonává uživatel ale spíše kopírují databázové operace (Create, Read, Update, Delete, List). Takové UC a scénáře jsou často produktem programátorů. Nejedná se o celý scénář, ale pouze o fragment.

  Příklad chybného použití: UC „Správa uživatelů“ a připojené dekomponované use case – vložit uživatele, editovat uživatele, smazat uživatele. Ne! Use case je příběh, který popisuje celek používaný uživatelem formou příběhu s několika alternativními toky (místo vkládání, editace apod.). Tyto „scénáře“ budou jen součástí většího scénáře. Uvědommě si, proč a kdy chceme editovat uživatele? Proč a kdy jej chceme mazat? Tyto fragmenty většinou napojujeme na původní akci. Mažeme jej například v případě, když ukončil pracovní poměr, proto bude smazání součástí akcí jako je například scénář ukončení pracovního poměru.

- ***Zahrnutí návrhových rozhodnutí*** – tato chyba nás nutí dělat návrhová rozhodnutí unáhleně, předčasně. Návrhová rozhodnutí ale v Agilních projektech záměrně odkládáme do nejzazšího možného momentu, jelikož mají vliv na architekturu a mohou ovlivnit spoustu věcí, resp. způsobit spoustu problémů.

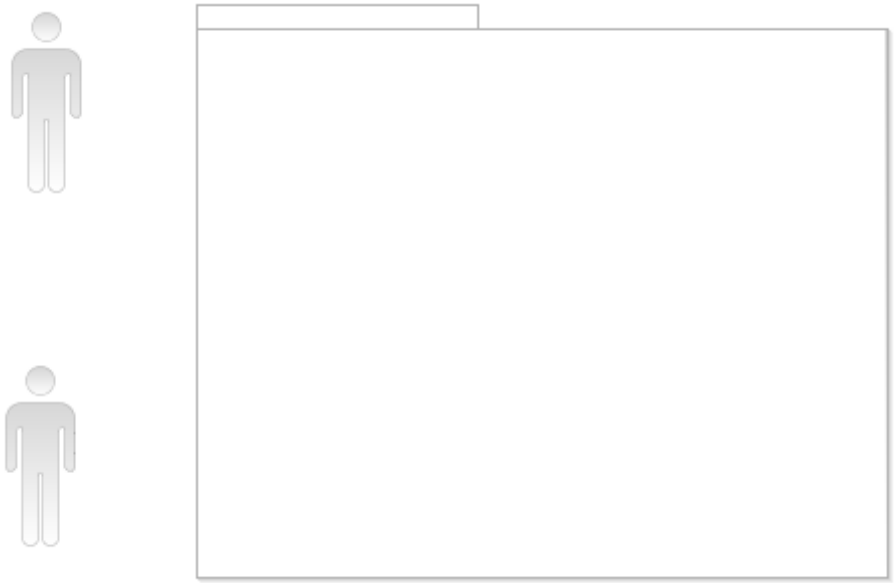
  Příklad chybného použití: Popis toku scénáře obsahuje pojmy jako: Klikneme na tlačítko, vybereme z databáze, vyhledávací klíč, seznam s rolovací lištou zobrazený červeně apod.

Identifikované scénáře potom zapíšeme například ve formě tabulky:

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: …	nízká	Bude upřesněno	-	…	2.0

Tabulka 3-3: seznam use case a jejich scénářů

Příklad: Vytvořte use case model a popište jeden scénář podle výše definované vize osobního manažera úkolů:



Název use case:
Aktor:
Počáteční podmínka:
Tok událostí (basic flow):

1.
2.
3.

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. Slovo *nefunkční* tedy neznamená, že uvedené body *nebudou fungovat*, ale že nejsou *funkcionalitami*, ale spíše schopnostmi či vlastnostmi. 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.

Tento zápis je tradiční plochá forma zápisu požadavků, která nebývá vždy přehledná, proto i pro nefunkční požadavky použijeme formu scénáře. Následující ukázka kvalitativního scénáře pro zápis nefunkčních požadavků pochází od SEI (Software Engineering Institute při Carnegie Mellon University), zdroj [SEI2], [SEI3]:

Stimul (Stimulus)	Podmínka, která ovlivňuje systém.
Zdroj (Source)	Entita, která generuje stimul.
Prostředí (Environment)	Podmínky za kterých se daný stimul objeví.
Artefakt (Artefakt)	Artefakt, který je stimulován.
Odezva (Response)	Odezva, kterou dostaneme jako odpověď na daný stimul.
Míra pro měření odezvy (Response measure)	Míra, jíž je daná systémová odezva měřena, ohodnocena.

Tabulka 3-4: Scénář kvalitativních atributů.

Takový scénář tedy budeme vytvářet pro všechny důležité atributy, které mají dopad na architekturu systému. Zdrojem možných kategorií jsou použitelnost, spolehlivost, výkonnost a mnoho dalších. Plný seznam možných viz [SEI3]. Pro zachycení a identifikaci těchto scénářů opět použijeme minimalistický princip, kdy v Inception některé tyto scénáře pouze identifikujeme a v průběhu iterací v Elaboration, jak začnou být dané nefunkční požadavky potřebné, resp. naplněny, je detailně popíšeme a průběžně implementujeme, resp. architekturu navrhujeme podle nich. Pro lepší obrázek ještě přikládáme příklad jednoho scénáře.

Scenario Refinement for Scenario N		
Scenario(s):		When a garage door opener senses an object in the door's path, it stops the door in less than one millisecond.
Business Goals:		safest system; feature-rich product
Relevant Quality Attributes:		safety, performance
Scenario Components	Stimulus:	An object is in the path of a garage door.
	Stimulus Source:	object external to system, such as a bicycle
	Environment:	The garage door is in the process of closing.
	Artifact (If Known):	system's motion sensor, motion-control software component
	Response:	The garage door stops moving.
	Response Measure:	one millisecond
Questions:		How large must an object be before it is detected by the system's sensor?
Issues:		May need to train installers to prevent malfunctions and avoid potential legal issues.

Obr. 3-4: Scénář kvalitativních atributů (zdroj [SEI3]).

Nefunkční požadavky posléze mapujeme v tabulce na funkční požadavky (use casy), jelikož všechny nefunkční požadavky nemusí platit pro všechny use casy. Mapování lze provést v jednoduché tabulce.

Název use case/scénář	NF1: přístup více uživatelů	NF2: VPN	NF3: různí klienti	NF4: odezva
UC1: reportování času [BF]	X	X	X	X
UC2: Statistiky [BF]			X	X

Tabulka 3-5: Mapování nefunkčních požadavků na use case.

V případě Ročníkového projektu se nebudeme nefunkčními požadavky tolik zabývat, jelikož cíl předmětu je naučit se postupovat podle procesu a pracovat v týmu. V praxi však tyto požadavky vytváří kostru systému a jejich opomenutí způsobuje velké problémy při vývoji a provozu systému, proto se o nich zmiňujeme.

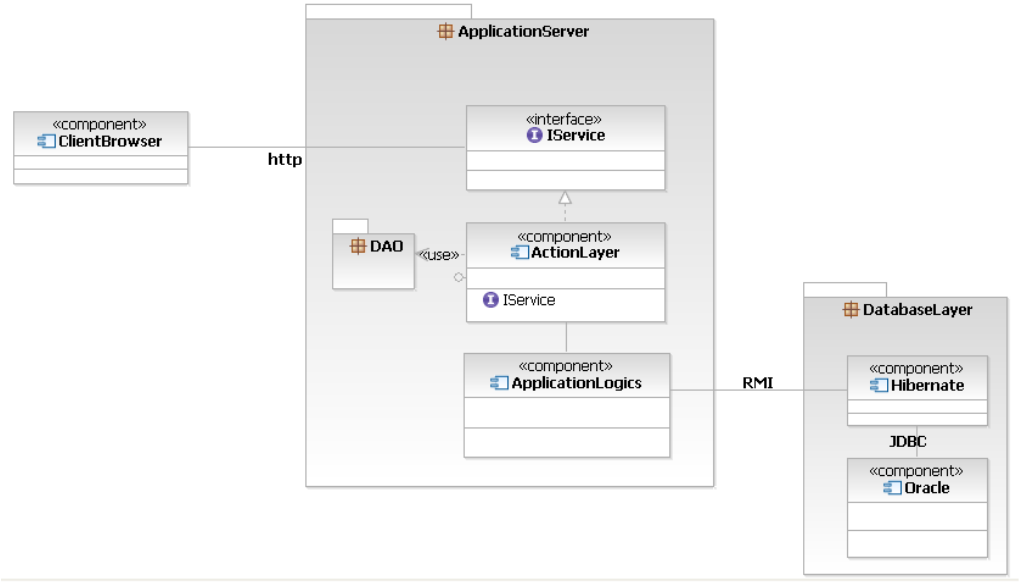
3.1.3 Návrh možného řešení

Jedním z hlavních cílů fáze zahájení 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 architekturou pak rozumíme detailnější pohled na strukturu, vrstvy a komunikaci mezi jednotlivými SW komponentami uvnitř těchto zařízení.

V našem případě reportování času 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

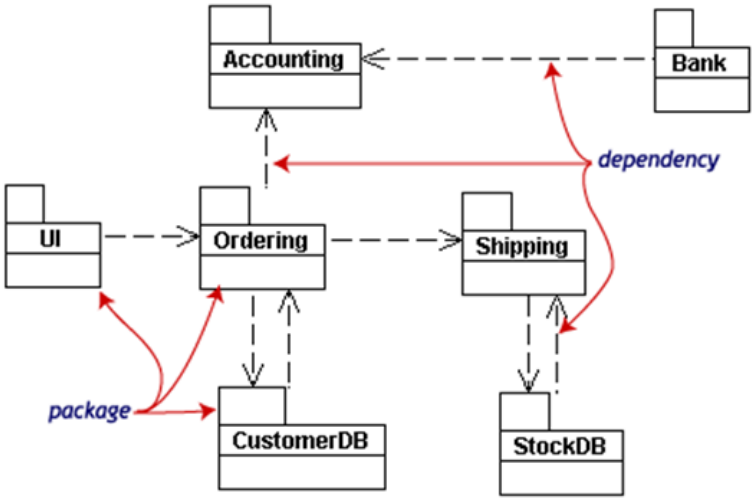
zůčastněným na projektu. Navíc víme, že máme také několik lidí v terénu (prodejci a analytici), kteří potřebují flexibilní přístup k aplikaci odkudkoliv v kteroukoliv dobu, a ti musí mít přístup i přes VPN, aby nemuseli reportovat až po příjezdu do kanceláře. Proto je důležité webové řešení s tenkým 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 nějakým ORM frameworkem napojeného na databázi Oracle. Podle potřeby je možné využít například post-relační databázi Caché či XML databázi.

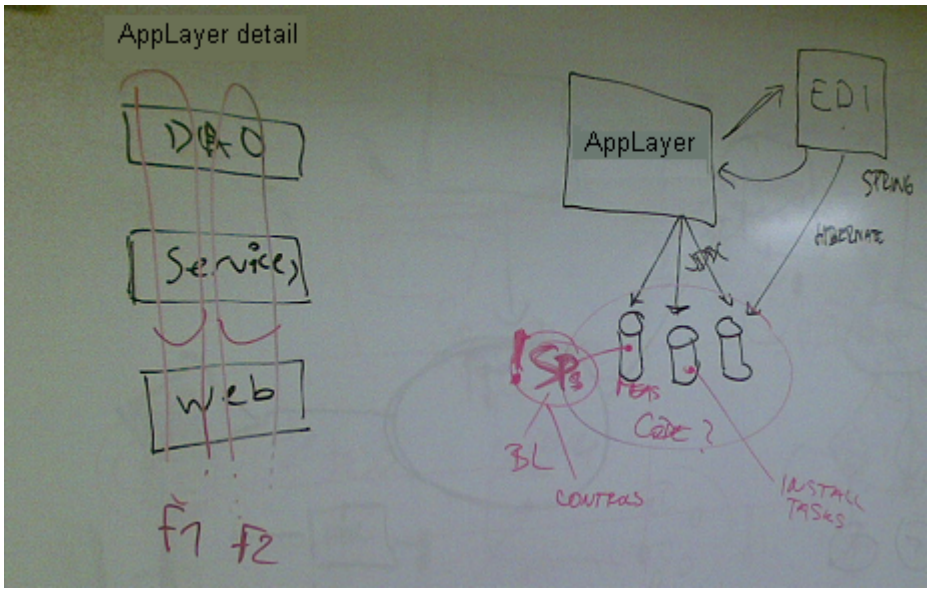


Obr. 3-5: Varianta klient-server (C/S) softwarové architektury popsané pomocí UML modelu komponent a balíčků.

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. 3-6: 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. 3-7: Další, méně formální varianta modelu softwarové architektury. Vrstvený model vlevo je detailem aplikační vrstvy vysokoúrovňového modelu. Je však nutné doplnit legendu, aby bylo zřejmé, co znamenají bloky a spojnice.

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.

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 a dokumentovat:***

- 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, serveru)?
- 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***.

Příklad: Vytvořte model softwarové architektury řešení našeho osobního manažera úkolů:

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.

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. Tímto tématem se budeme zabývat v následujícím textu.

3.1.4 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. My se však v kontextu Ročníkového projektu budeme zabývat pouze riziky projektu a akcemi na jejich odstranění. Rizika projektu mohou pocházet z několika oblastí, nejběžněji se jedná o následující:

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

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 a včasné doručení.

Rizika jsou důležitá pro následující fázi Rozpracování (*Elaboration*), která je zaměřena na jejich snížení či odstranění. 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, akce 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.

Název rizika	Popis	Dopad	Pravděpodobnost	Důležitost	Vlastník
R1: Nedostatečné zapojení všech zúčastněných na projektu	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
R2: Integrace se systémem X	Není zřejmé, jak integrovat naši aplikaci s historickým systémem X.	3 vysoký	80%	2.4	Architekt Petr
R3: Tréninkové materiály	Nemáme oprávnění vytvořit kvalitní tréninkové materiály, což může vést k nekvalitnímu tréninku.	2 střední	100%	2.0	Manažerka Anička
R4: 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 3-6: Příklad seznamu rizik (tzv. *risk list*).

Je běžné, že seznam rizik v praxi často obsahuje spíše seznam faktů, věcí, které na projektu ***již nastaly*** a nejsou 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 „komerčního 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. Následující seznam rizik je již doplněný o konkrétní akce:

Název rizika	Popis	Strategie
R1: 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 pro demonstraci. Uspořádáme schůzku 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.
R2: 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 způsob může 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.
R3: 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).
R4: 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 .JEE a nalezení rezervy v rozpočtu na mentora 2 dny v týdnu po dobu prvních dvou měsíců trvání projektu. Nábor nového člena týmu se znalostí .JEE platformy. Vytvoření prototypu pro ověření použití .JEE platformy.

Tabulka 3-7: Příklad seznamu rizik (risk list) doplněný o akce na jejich snížení či odstranění.

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 externí firmy na konzultace atd.

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

1. Odstranění, snížení rizika – snížení závažnosti rizika.

2. Předejití riziku – modifikace projektu za účelem zamezení výskytu rizika.
3. 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 3-7).

Příklad: Vytvořte seznam rizik našeho projektu osobního manažera úkolů. Pamatujte na následující:

- Jsou akce opravdu konkrétní kroky, která může někdo vykonat?
- Uvedení faktu vám má pomoci identifikovat riziko.

Číslo a název rizika	Fakt	Dopad	Akce

3.1.5 Proces vývoje a použité nástroje

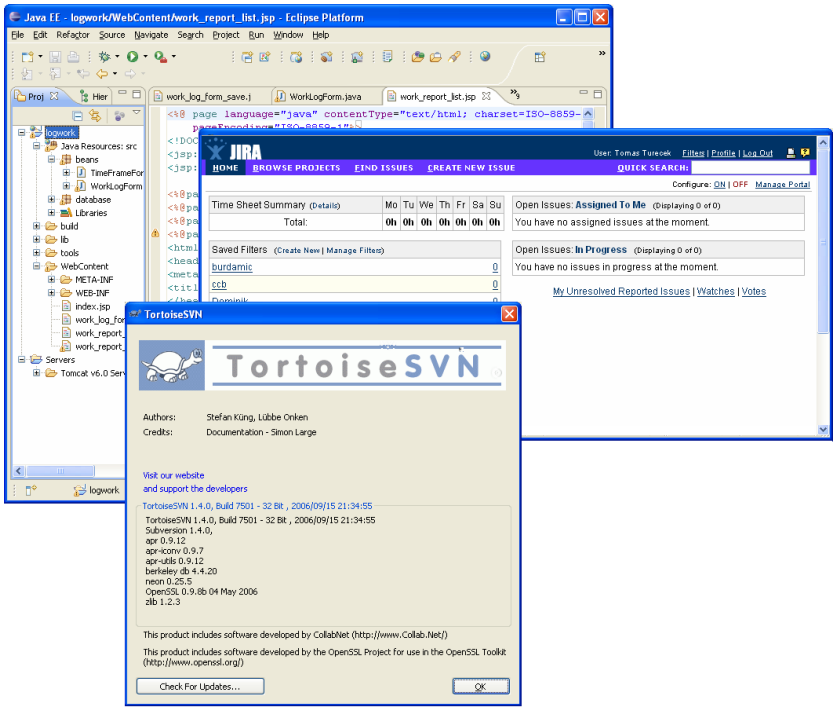
V úvodní iteraci/iteracích je také místo pro výběr a definici procesu, který bude vývojový tým dále v projektu následovat, stejně jako pro výběr a nastavení prostředí a nástrojů. Proces i nástroje by měly být vybírány podle specifických potřeb projektu. U procesu jde o množství činností a kroků, které 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. S definicí procesu vám z velké většiny pomáhají lektoři Ročníkového projektu. Nastavení prostředí je však více ponecháno na vás, proto jej nezanedbávejte. Uvažujeme následující oblasti:

- IDE – integrované vývojové prostředí (př. Eclipse, Visual Studio, NetBeans, …).

- Nástroje pro správu konfigurací a změn – Configuration and Change Management Tools (CVS, SVN, Jira) – velmi důležité pro sdílení zdrojových kódů.
- Nástroj pro správu požadavků, chyb (př. Rational Requisite Pro, Jira, Rational Team Concert) – zde nás zajímá hlavně správa chyb.
- Vizuální modelovací nástroje (př. Magic Draw, Enterprise Architect, ArgoUML, Borland Together, Eclipse pluginy, …) – není důležité, můžete modely kreslit, nástroje vám však pomohou s použitím standardních UML modelů místo vašich náhodných malůvek.

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 úkol Antu).
- Ukládání (*commit*) pouze hotových věcí do repozitory – nehotové věci mohou způsobit problémy a chyby.



Obr. 3-8: Eclipse IDE, SVN repository, Jira issue tracking tool.

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.

Příklad: Sepište použité základní nástroje pro váš projekt včetně jejich umístění pokud se jedná o online nástroj:

Oblast	*Nástroj*	*URL adresa, zdroj*
IDE		
Repository (úložiště zdrojových kódů)		
Úložiště dokumentace		
Build mechanismus		

3.1.6 Souhrn kapitoly a fáze Zahájení

Následující kapitola stručně shrnuje, co vše bychom měli znát na konci fáze Zahájení (Inception). Hlavním cíle fáze Zahájení je pochopení potřeb projektu a znalost plánu postupu. Ve fázi Zahájení (Inception), v tzv. nulté iteraci projektu, tedy musíme:

- porozumět tomu, co máme vytvořit (vize, definice rozsahu systému, jeho hranic například pomocí use case),
- identifikovat zákazníka, tedy toho, kdo chce vytvářený systém,
- a v neposlední řadě také to, co mu tento systém přinese,
- s daným zákazníkem či jeho zástupcem(ci) identifikujeme klíčové funkcionality systému – tj. nejkritičtější use case.

Na základě znalosti vize, use case, proveditelnosti projektu díky prototypu (architektura) a navrženým akcím na zmírnění rizik můžeme konečně vytvořit první návrh projektového plánu, který dále v projektu upravujeme podle reality a potřeb zákazníka:

Fáze	Iterace	Primární úkoly	Datum od - do
Inception	I0	Sestavení týmu, rozpočtu Identifikace základních požadavků a rizik Výběr kandidátů architektury a jejich ověření prototypy LCO (shoda na vizi, rozsahu, rozpočtu)	02.01. - 14.01. 14.01.
Elaboration	E1 E2 E3	UC1 [BF]: Reportování hodin UC1 [AF1]: Import dat UC2 [BF]: Zobrazení hodin pro vybrané období – prototyp Rezerva (5 dní) LCA (snížena rizika, implementována a otestována architektura)	14.01. – 01.02. 01.02. – 18.02. 18.02. – 24.02. 24.02.
Construction	C1 C2 C3	UC2 [BF]: Zobrazení hodin pro vybrané období – finální verze UC1 [AF2]: Vkládání více úkolů UC2 [AF1]: Export do Excelu Rezerva (2 dny) IOC (beta release)	Bude upřesněno Bude upřesněno 26.03.
Transition	T1	Akceptační testování zadavatelem ... PR: Předání projektu	13.05. – 07.06.

Tabulka 3-8: 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). V Elaboration jsme se zaměřili na rizikové scénáře, ty jsou implementovány nejdříve, abychom měli jistotu, že jsme snížili či odstranili dané riziko a scénář a celý systém půjde implementovat. Jinak máme pořád ještě dost času se s nalezeným problémem vypořádat, změnit technologii, navrhovanou architekturu či rozsah řešení.

Nezapomínejte, že každá iterace nemusí vždy doručit další nový hotový scénář daného UC. ***Jeden scénář může být nejprve doručen jako prototyp*** ověřující práci s daty a celkový koncept (musí ale prostupovat všechny softwarové vrstvy) a ***další iterace může doručit hotový scénář*** ale ***bez ošetřených výjimek*** (koncept v jednom případě ale musí být implementovaný - demonstrovatelný). Třetí iterace pak může doručit tento scénář ***hotový i s kontrolami*** a implementovanými výjimkami, tak jak vidíte v příkladu s UC2 v Elaboration a Construction fázi.

3.1.7 Milník LOM

Na konci Inception fáze následuje první milník projektu nazýván *Lifecycle Objective Milestone* (LOM), který je vyhodnocen na konci poslední iterace v Inception fázi. Milník je určen ke zhodnocení cílů celého projektu. Pokud nejsme schopni tohoto milníku dosáhnout, měl by být projekt zrušen nebo přerušen. Důvodem 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 méně peněz a času. 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 (use case) a porozumění jim.
- Shoda na tom, že počáteční rizika byla identifikována a existují strategie (jsou známé akce) na jejich snížení.

Kontrolní otázky:

1. Jaké artefakty vytváříme ve fázi Inception?
2. Je nějaký rozdíl, případně vazba mezi use case a scénářem?
3. Obsahuje projektový plán detailní akce?
4. Mění se projektový plán v průběhu projektu?

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 první fázi Inception. Hlavním účelem však bylo na příkladech představit základní artefakty iterativně inkrementálního projektu a vytvořit je pro definované téma do připravených šablon. Tyto artefakty jsou vize, use case model, scénáře, nefunkční požadavky, seznam rizik a akcí a konečně projektový plán.

4 Fáze Rozpracování (*Elaboration phase*)

V této kapitole se dozvíte:

- Co je hlavním zaměřením fáze Rozpracování.
- Jak dále rozpracujeme existující artefakty a prototyp.
- Co je kontrolním milníkem pro opuštění této fáze.

Po jejím prostudování byste měli být schopni:

- Minimalizovat možný dopad rizik.
- Doručit stabilní a otestovanou architekturu projektu.

Klíčová slova této kapitoly:

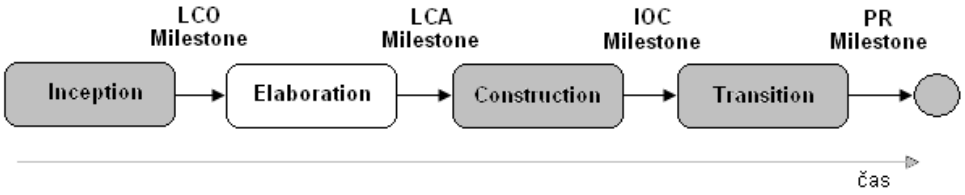
Snížení rizik, architektura, stabilní proces.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Kapitola představuje další fázi iterativně inkrementálního modelu vývoje softwaru zvanou Rozpracování (Elaboration). Hlavním smyslem iterací v této fázi je odstranit identifikovaná rizika a doručit spustitelnou a otestovanou architekturu.
Na studium této části si vyhraďte 2 hodiny.

Po úspěšném průchodu prvních iterací, respektive celé první fáze zvané Zahájení (Inception), máme 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, které jsou stručně popsány. Navrhli a implementovali jsme také prototyp minimálně jednoho možného architektonického a technologického řešení. Díky těmto krokům jsme byli schopni vytvořit hrubý plán projektu a identifikovat přibližné náklady projektu. Poslední zmíněný krok (náklady) není v našem případě tak důležitý. V neposlední řadě máme definovaný proces vývoje a nastavené prostředí a nástroje.



Obr. 4-1: Fáze Rozpracování (Elaboration)

Pokud jsme v předchozí iteraci úspěšně dosáhli prvního milníku (Lifecycle Objective Milestone – LOM), přechází projekt do druhé fáze zvané Elaboration. Cílem iterací této fáze je definovat a nastínit architekturu systému, abychom na jejím základě mohli v následující fázi navrhnout a implementovat zbývajících 70-80% nekritických use case (funkčností). Jak jsme již zdůraznili, architektura vyplývá z nejdůležitějších požadavků, tj. z těch, které na ni mají

dopad a formují ji, ale také například z identifikovaných rizik hlavně technologického charakteru. Cílem této fáze je 1) tvorba architektury a 2) 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 abychom doručili, co jsme slíbili?)

4.1.1 Iterace ve fázi Rozpracování (Elaboration)

Většinu rizik snížíme tím, že v této fázi ***implementujeme relevantní use casy, které lze spustit a demonstrovat na navržené architektuře***. Tímto krokem konkrétně demonstrujeme proveditelnost vybraného řešení a technologie a nespoléháme na možné mylné předpoklady a neověřená tvrzení. 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 ve fázi Rozpracování (Elaboration) by měla zahrnovat:

- Návrh, implementace a testování malého počtu kritických scénářů, pomocí kterých identifikujeme typ architektury a potřebné mechanismy komunikace a rozhraní. To 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 case, 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 ve fázi Rozpracování (Elaboration) by měla zahrnovat:

- Opravu všeho, co nebylo správné v první iteraci.
- Návrh, implementace a testování zbývajících architektonicky významných scénářů.
- Nastínění a implementace paralelismů, procesů, vláken (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 datového modelu, která nejlépe odpovídá navržené aplikační logice. ***POZOR! Datový model navrhujeme pouze pro use casy implementované v dané iteraci***.
- Detailní popis zbývající poloviny use case, které chceme blíže specifikovat v Elaboration.
- Testování, ověření a úpravy architektury do její stabilní verze, na kterou pak můžeme nabalit 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 nutnost odstranění méně prioritních funkčností z plánu projektu, případně nutnost dohody o posunutí konečného termínu doručení. Obě varianty jsou ale o mnoho levnější, než celé řešení stavět na tekutých píscích (rozuměj na nestabilní architektuře).

Poznámka: iterace ***nemusí vždy doručit kompletní nový hotový scénář daného use case***. Jeden scénář může být nejprve doručen jako prototyp ověřující práci s daty a celkový koncept architektury (musí ale prostupovat všechny softwarové vrstvy). Další iterace pak může doručit hotový scénář, ale bez ošetřených výjimek (koncept výjimek však v jednom případě musí být implementovaný – demonstrovatelný). Třetí iterace pak může doručit tento scénář hotový i s kontrolami a implementovanými výjimkami.

4.1.2 Podrobnější pochopení požadavků

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

Na konci fáze Elaboration by mělo být popsáno přibližně 80% use case, některé nové UC je možné nalézt i v Construction, nemělo by to však být pravidlem.

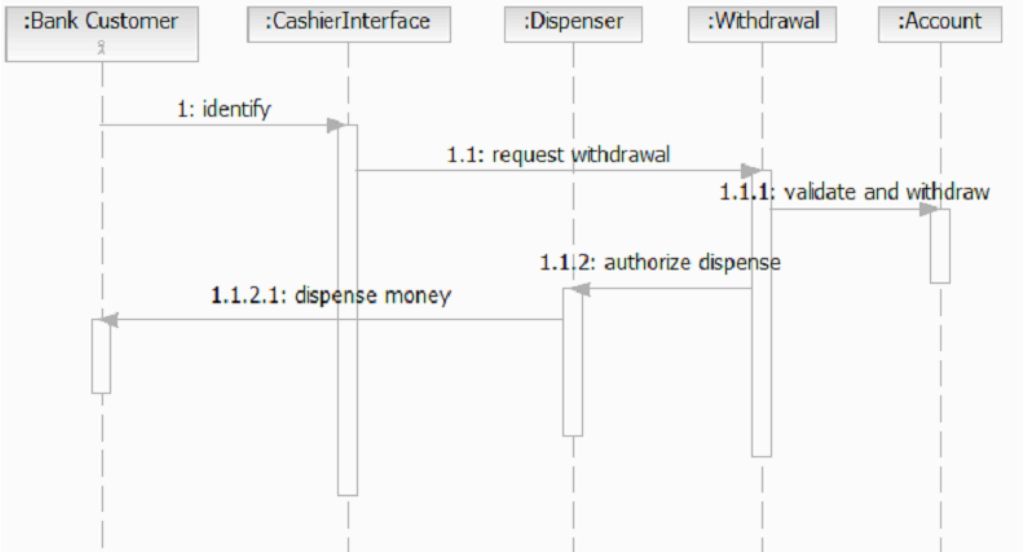
4.1.3 Návrh, implementace a ověření architektury

Architektura je část systému, část řešení, zabývající se komunikací, ukládáním dat, uživatelským rozhraním, technologií, ale pouze v míře nezbytně nutné na základě kritických (nejdůležitějších) use case systému. Při tvorbě architektury uvažujeme následující:

- nefunkční požadavky (běhové prostředí, použité OS, přístupy, stabilita běhu apod.), které vlastně tvoří architekturu,
- 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. Hlavní kroky iterací v této fázi jsou tedy:

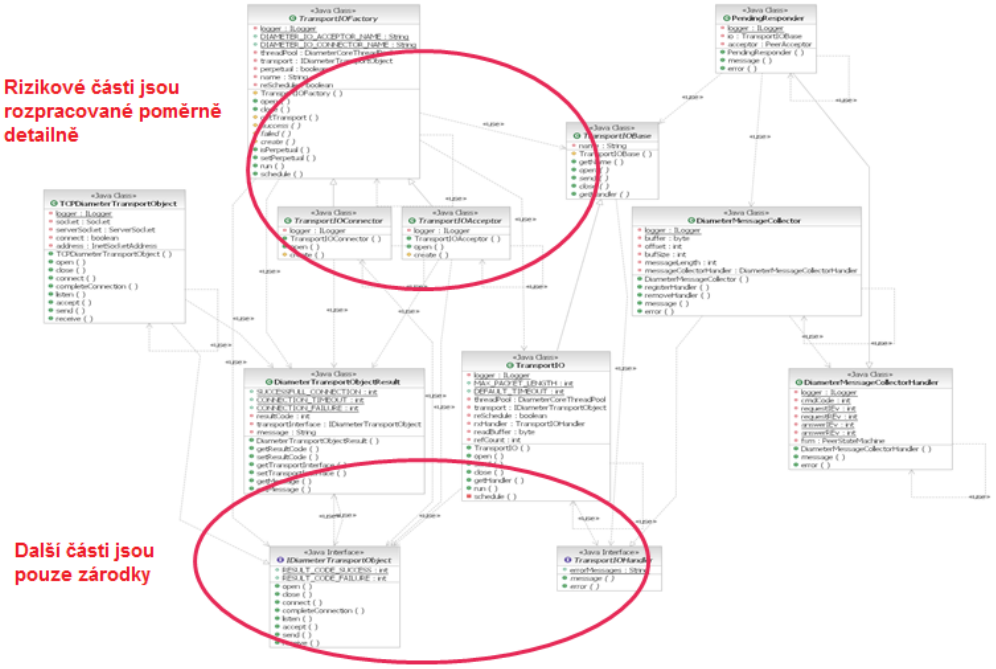
1. Definice subsystémů, klíčových komponent a jejich rozhraní, pomocí kterých budou komunikovat. V tomto kroku je vhodné uvažovat použití existujících frameworků (z předchozích projektů, či existujících na trhu) předtím, než budeme na zelené louce navrhovat architekturu vlastní. Potenciálními zdroji pro identifikaci komponent jsou doménové objekty z doménového modelu.
2. Identifikace architektonicky významných use case pro definici architektury. (typicky 20-30%). Je třeba brát v úvahu taktéž nefunkční požadavky a nalézt a implementovat use case, které odhalí potencionální problémy a rizika a ověří jejich proveditelnost. ***Implementací use case máme v této fázi na mysli pouze 1 nebo 2 scénáře***. To znamená ideální („happy day“) scénář + například ***jeden chybový pro ověření*** způsobu zachycení a zpracování výjimek. Důvodem je smysl této fáze: snížení rizik našeho řešení na akceptovatelnou úroveň či jejich úplné odstranění.
3. ***Návrh (design) kritických use case***. Ten popisuje, jak jsou konkrétní analztické objekty popsané v use case 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. 4-2 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.
4. ***Konsolidace a seskupení identifikovaných tříd do balíčků***. 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. 4-2: Use case realizace – transformace slovních scénářů do formy analytických tříd projektu (využit je UML sequence diagram).

5. Zdůrazníme, že ***až po návrhu aplikační vrstvy přichází na řadu 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).
6. ***Integrace komponent*** 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.
7. ***Testování kritických scénářů*** je posledním krokem, jedná se o kritický aspekt fáze Rozpracování (Elaboration). Otestované scénáře bereme jako 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.

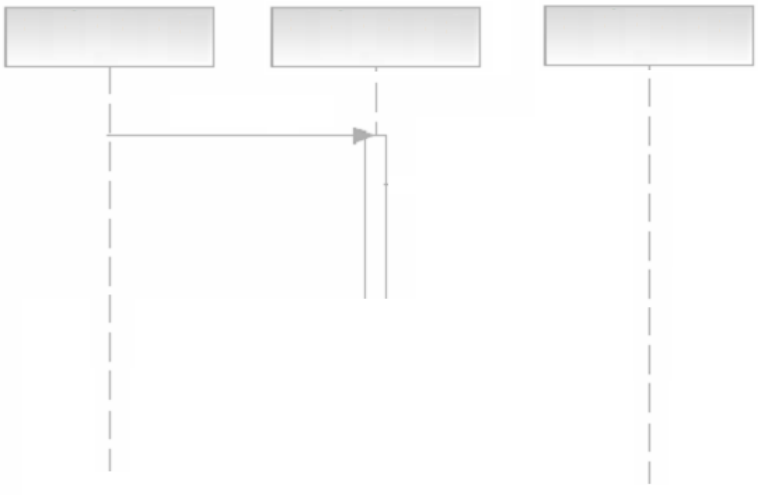
Je třeba si uvědomit, že pokud jsme prošli až sem, tak máme některé části systému vyvinuté v docela pokročilém stupni. Stále ale máme implementováno pouze 10-20% celého řešení. Většinou se jedná o typické scénáře 20-30% use case. Provedli jsme tedy něco ode všech disciplín, ale stále nám ***zbývá přibližně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



Obr. 4-3: Úroveň detailu rozpracování projektu ve fázi Elaboration. Kritické části z pohledu technologie jsou rozpracované, další nekritické části jsou pouze zárodky.

Příklad k vypracování:
Vyberte si jeden ze (slovních) scénářů, které jste popsali v předchozím úkolu, a pokuste se jej formalizovat pomocí use case realizací (příklad viz obr. 4.2 výše).

Identifikujte nejprve možné třídy (subjekty, objekty či podstatná jména vyskytující a opakující se ve scénářích) a poté jako šipky doplňte akce, které dané subjekty vykonávají či aktivují. Každý bod scénáře by měl reprezentovat jednu šipku (akci) v tomto modelu.



4.1.4 Vytvoření přesnějšího plánu a odhad nákladů

Na konci fáze 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: vize (*Vision*), případová studie (*Business Case*), projektový plán (*Project Plan*)), aktualizujeme seznam rizik a akcí na jejich odstranění/snížení.

4.1.5 Testování v Elaboration

Vývojářské a testerské testování je kritickým aspektem fáze Elaboration. Spustitelná architektura znamená nejen implementované scénáře, ale také otestované funkčnosti (testeři) a kód (vývojáři). Klíčovým testovacím prostředkem testování vývojářů je Test-Driven přístup (TDD) nebo alespoň implementace unit testů po implementaci kódu. Více o testování viz text Informačních systémů 1. Pojďmě alespoň zopakovat postup TDD:

1. Jako první krok vytvoříme nový unit test, který logicky nemůže projít, jelikož neexistuje implementace, kterou má testovat; důležitou myšlenkou v pozadí tohoto kroku je získat a ověřit si pochopení problematiky (požadavků) např. z detailních scénářů či use case. *Pozn.: aby však šel kód zkompilovat, musíme provést potřebnou implementaci potencionálních funkcí, které jsou v testu použity. Tehdy využijeme přirozených možností programovacího jazyka – tedy typicky výjimek typu* `NotImplementedException` *či* `UnsupportedOperationException` *apod.*
2. Spuštění všech unit testů je dalším krokem, test by neměl být úspěšný, nemůže projít, jelikož ještě neexistuje kód, který má testovat. Jedná se tedy o ověření jeho správnosti (odhalení případné chyby v testu).
3. Vytvoření zdrojového kódu, který má unit test testovat. Účelem tohoto kroku je pouze ověření úspěšného průchodu testem, nejedná se o finální implementaci! Ta může být „špinavá“ obsahující natvrdo zadané proměnné, bez komentářů a nestrukturovaná.
4. Následným krokem je pak spuštění testů a zjištění zda implementace úspěšně prošla, tedy zda kód splňuje požadavky. Pokud ano, můžeme začít psát finální implementaci.
5. Posledním a nezbytným krokem je refaktoring kódu, tj. zlepšení kvality kódu, jeho vyčištění, doplnění funkcionality, nahrazení kouzelných čísel či řetězců v kódu, který jsme napsali jako hrubou implementaci v kroku 3. Zanesení možné chyby je již kontrolováno spouštěním automatických testů.

6. Tento postup iterativně opakujeme, jak přidáváme nové funkčnosti, opravujeme chyby či implementujeme změny.

Pro unitové testování je vhodné využít různé automatizované nástroje či frameworky. Nejznámějším je pravděpodobně xUnit, který existuje ve variantách pro různé programovací jazyky, např. původní verze SUnit pro Smalltalk, JUnit pro Javu, PHPUnit pro PHP, CUnit pro jazyk C apod. Nyní již existují lepší a propracovanější frameworky, ale pro demonstraci a naučení je JUnit asi nejvhodnější.

Pojďme se tedy podívat, jak by mohl vypadat vývojářský test:

```
import org.junit.Test;
import static junit.framework.Assert.assertEquals;

public class BankAccountTestCase {

    @Test
    public void testDeposit() {
        BankAccount account = new BankAccount();
        double amountIn = 500.00;
        double result = account.deposit(amountIn);
        assertEquals(amountIn, result, 0);
    }
}
```

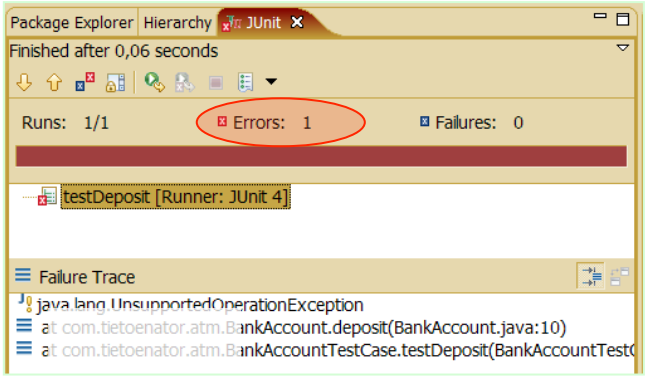
Anotace

Předpokládané chování

Ohodnocení výsledku

Implementace funkčnosti výběru z bankomatu pro daný test:

Pojďme test spustit, ať vidíme, jestli projde nebo ne. Neměl by ještě projít, jelikož nemáme implementaci, respektive měl by vrátit danou výjimku.

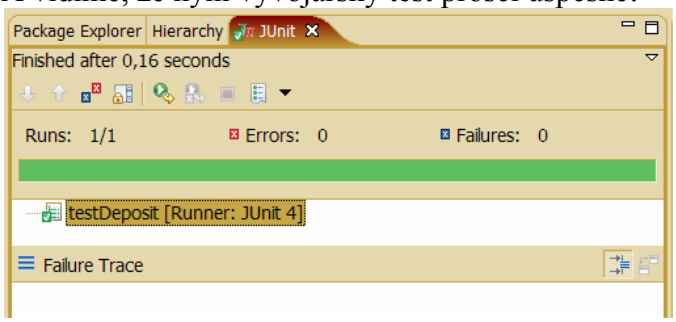


Výborně, test neprošel, protože Java vyvolala výjimku. Provedeme nyní tedy nejjednodušší možnou implementaci (*quick and dirty*) ať ověříme, zda test funguje:

```
public class BankAccount {

    public double deposit(double amount) {
        return amount;
    }
}
```

A vidíme, že nyní vývojářský test prošel úspěšně!



Vývojáři tedy testují svůj vlastní kód a rozhraní na jiné systémy a podsystémy. Testerské testování pak zdrojový kód nevidí a zaměřuje se na chování, funkčnosti aplikace. K tomu, co a jak testovat, existují tzv. testovací ideje, které jsou realizovány testovacími scénáři. Tyto scénáře mohou mít různou formu, pro manuální testování to může být dokument nebo wiki stránka, pro automatizované pak skript nebo „zdrojový kód“. Test casy a test skripty vznikají již v ranných fázích každé iterace. Analytik testů, návrhář testů (Test designer) a tester spolupracují s analytiky na pochopení požadavků (musí mít pochopení toho, co má systém dělat, když jej budou testovat) a formě jejich testování (co, jakým způsobem a v jakém rozsahu). Cílem je rychlá zpětná vazba pro vývojáře v rámci iterace, znovupoužitelnost use case ve formě tzv. testovacího scénáře (*test case*) a automatizace testovacích skriptů. Příklad znovupoužití use case v testovacím scénáři (*test case*) ukazuje následující příklad:

Use case	**Scénář**	**Data a podmínky**		
		Hodiny	**Uživatel**	**Výstup**
Reportuj čas	BF	5	user	OK
Reportuj čas	AF#1	-4	user	Chybové hlášení
...	...	...	...	...

Tabulka 4-1: Testovací scénář (test case) – základní princip znovupoužití use case.

Aby si student udělal obrázek, jak může vypadat komerční testovací scénář (*test case*) pro systém zasílání zpráv telekomunikačního operátora, přikládáme následující, pro někoho možná komplikovanou, ukázku:

Test Case Id	XDM-1.0-int-0200
Test Object	UE with XDMC, Aggregation Proxy, Shared XDMS
Test Case Description	Verify that the UE can create and retrieve an XML document from the Shared XDMS. TEST CASE GOAL: Verify that the creation of an URI List and/or addition of a member to that list creates an XML document in the Shared XDMS. The created document can be retrieved by the users.
Specification Reference	Refer to **Error! Reference source not found.**
SCR Reference	Refer to **Error! Reference source not found.**
Tool	Not available.
Test code	Not available.
Preconditions	Equipment: o 2 UEs (both with User1 credentials) o Aggregation Proxy o Shared XDMS Prerequisite for this test: o XDM-1.0-int-0100 executes successfully o Shared XDMS does not have any documents for User1 o UE1 and UE2 are powered off o UE1 and UE2 support the same applications that use URI Lists (e.g. oma_allcontacts, oma_buddylist)
Test Procedure	1. UE1 is powered on. 2. A URI list is created using UE1. 3. User2 member is added to the URI List using UE1. 4. UE1 is powered off. 5. UE2 is powered on. 6. The URI List modified in Step 3 is retrieved using UE2.
Pass-Criteria	3. UE1 displays a URI List with User2 as a member. 6. UE2 displays a URI List with User2 as a member.

Tabulka 4-2: Příklad komerčního test case z praxe: vytvoření XML dokumentu ze sdílené XDMS (XML Document Management Server v síti mobilních operátorů).

4.1.6 Příklad iterace v Elaboration fázi

Pro názornost si ukážeme příklad kroků a iteračního plánu pro první iteraci E1 ve fázi Rozpracování (Elaboration). Projektový plán představený v předchozí kapitole definuje jako cíl iterace E1 pouze scénář hlavní tok (BF) ***use casu UC1: Reportování hodin*** v termínu od 14.1. do 1.2.

Fáze	Iterace	Cíle iterace	Datum od - do
Elaboration	E1	UC1 [BF]: Reportování hodin	14.01. – 01.02.
	E2	…	…

Jediný scénář (BF = basic flow) daného use case UC1: BF Reportování hodin implementujeme jako první, protože přinese uživateli největší hodnotu = hotová nejdůležitější funkčnost. Ošetření chybových stavů a případně další

scénáře tohoto use case nyní neimplementujeme. Zároveň je smyslem iterací této fáze Elaboration odstranění rizik či snížení jejich dopadu na minimum. V Inception fázi jsme také definovali rizika projektu (viz následující tabulka, nebo kompletní Tabulka 3-7) a akce na jejich snížení. ***Tyto akce budeme nyní vykonávat jako jeden z cílů Elaboration iterací***.

Název rizika	Popis	Strategie
R1: Nedostatečné zapojení všech stakeholderů	Předchozí zkušenost nám říká že lidé z oddělení X nemusí pochopit či souhlasit s požadavky, což generuje požadavky na zásadní změny po Beta-releasu.	**Strategie: Snížení rizika** 1/UI prototyp pro demonstraci use case požadované tímto oddělením. 2/ Schůzka se zástupci oddělení X: demo prototypu a kreslení storyboardů. Sběr smysluplné zpětné vazbě od všech zúčastněných.
R2: Nedostatečná zkušenost s JEE	Je riziko, že vytvoříme druhořadé, méně technicky kvalitní řešení.	**Strategie A: Snížení rizika** 1/ Mentor JEE 2 dny v týdnu po dobu prvních dvou měsíců trvání projektu. 2/ Vytvoření prototypu pro ověření použití JEE platformy.

Iterační plán vytvořený na základě těchto dvou vstupů pak může vypadat následovně:

Plán iterace E1	
Začátek iterace (plánovací meeting dané iterace):	14.1.2013
Konec iterace (demo, assessment):	1.2.2013
Cíle iterace: • Implementace UC1 [BF]: Reportování hodin. • Odstranění rizika R1 (nedostatečná zkušenost) a R2 (zapojení všech).	**Status**
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.	**Status**

Úkoly iterace E1:

Název / Popis	Priorita	Odhad (body)	Přiřazeno	Odhad (hodin)
Instalace build mechanismu (Ant) a jeho propojení s IDE	2		Johan	16
Analýza scénáře UC1 (BF)	2	8	Lisa	24
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 instalačního manuálu		1	Lisa	5
Vytvoření release notes		1	Johan	4
Vytvoření online help stránek	3	2	Ann	22

Iterace začíná plánovacím meetingem jehož se účastní celý tým a může trvat od 30 minut do 4 hodin. Tým v rámci této schůzky rozpracuje cíle iterace na jednotlivé úkoly a k nim se přihlásí řešitelé, jednotliví členové týmu. Dále se dohodnou se zákazníkem evaluační kritéria a definují se interní kritéria doručení. Po tomto meetingu se začnou detailně analyzovat, navrhovat a implementovat plánované scénáře, které byly nastíněny v Inception iteracích. Paralelně s analýzou probíhá v iterativně inkrementálních přístupech také přípravná část pro testování. Tou je vytvoření testovacího prostředí, identifikace testovacích situací a scénářů, které jsou postaveny nad use casy, resp. znovupoužívají jejich scénáře a také příprava testovacích nástrojů. Výsledkem iteračního plánovacího meetingu jsou výše zmíněné cíle, evaluační kritéria a úkoly zachycené na wiki, v Jira 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 provedeno týmové ohodnocení vůči definovaným kritériím. Pokud jsme ***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). Ostatní úkoly pak musíme v projektovém plánu posunout, buď využitím rezervy nebo odsunutím méně významných scénářů do dalších verzí. Výsledkem týmového hodnocení iterace, přičemž členem týmu je i zákazník, je pak následující zápis opět na wiki, sharepointu, v Jira či MS Word dokument či v nějaké jiné formě zachycené:

Assessment iterace E1 (vyplněn až na konci iterace při assessmentu!)	**Stav**
Ohodnocení cílů podle evaluačních kritérií: • UC1[BF]: Reportování hodin implementován a demonstrován • 70% pokrytí kódu unit testy • 98 % unit testů prošlo • Riziko R1 odstraněno • Riziko R2: akce nastartovány	 Hotovo Hotovo Hotovo Hotovo Hotovo
Neopravené chyby: • ERR0012: Null pointer při určitém stavu… (ve formě odkazu naBugzillu, Jiru či jiný nástroj pro vedení evidence chyb, tzv. issuetracking tool).	

Jiné poznámky a poznatky:

Zákazník byl mile překvapen existujícími funkčnostmi takhle brzy (a akceptoval demo) a byl dohodnut pilotní provoz reportování již po skončení fáze Elaboration.

Při ukázce dema zákazníkovi dne 1.2.2013 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 některé z dalších iterací 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
Elaboration	E1	UC1 [BF]: Reportování hodin	14.01. – 01.02.
	E2	UC1 [AF1]: Import dat UC2 [BF]: Zobrazení hodin pro vybrané období	01.02. – 18.02.
	E3	Identifikován nový scénář => aplikována rezerva UC1 [AF3]: Vytváření projektů Rezerva (1 den) LCA (snížena rizika, implementována a otestována architektura)	18.02. – 24.02. 24.02.
Construction	C1	UC1 [AF2]: Vkládání více úkolů Dále beze změn…	25.02. – 09.03.

Tabulka 4-3: Aktualizovaný projektový plán

Dokumentace architektury a návrhových rozhodnutí

Důležitým výstupem iterací v Elaboration je dokumentovaná architektura a návrhová rozhodnutí. Tento popis by měl obsahovat možné varianty architektur, které šlo použít, ***důvody rozhodnutí*** pro naši variantu, dále vrstvy architektury a pravidla pro její implementaci. Opět nemusí být dokumentace ve formě dokumentu ale například na wiki. Tato dokumentace je velmi důležitá, jelikož se ***stává 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, …).

První body píšeme ve formě textu nebo tabulky. Logický pohled je vhodné popsat pomocí UML modelů či jejich „light“ verzí. Příklady takových více či méně formálních modelů jsme uvedli na obrázku 3.5, 3.6 a 3.7.

4.1.7 Milník LCA

Milníkem Elaboration fáze, který ohodnocujeme v rámci hodnocení poslední Elaboration iterace je tzv. Lifecycle Architecture milestone (LCA). Nyní máme detailně prozkoumány cíle a rozsah systému, požadavky se ještě mohou měnit, ale již bez dopadu na architekturu. Dále máme vybranou a implementovanou architekturu a identifikována a snížena největší rizika (díky implementaci asi 30% scénářů projektu). 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 a otestovanou architekturu?
- Jsou klíčové postupy a přístupy (nástroje, proces vývoje a testování), 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 (hlavně iterace C1 a cíle dalších iterací), 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.

Kontrolní otázky:

1. Co jsou hlavní dva cíle fáze Rozpracování (Elaboration)?
2. Jakým krokem se dostaneme od slovních scénářů k analytickým třídám?
3. Kdy začínáme poprvé v projektu testovat? Ve které fázi/iteraci projektu?
4. Jak definujete pojem „ověřená a spustitelná architektura“?

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:
Vytvořte seznam rizik a navrhněte vhodné akce na jejich snížení či odstranění pro projekt tvorby mobilní aplikace, která se má připojovat k IS Stag a umožnit zápis předmětů přes mobilní zařízení (smartphony, tablety). Uvažujte, že projekt bude implementovat studentský tým kolegů jako jste vy.

Shrnutí obsahu kapitoly
V této kapitole jsme se zabývali iteracemi druhé fáze zvané Rozpracování (Elaboration). Ukázali jsme si, jak se dostaneme od neformálních slovních scénářů k analytickým třídám (tzv. use case realizace), jaké kroky musíme udělat v návrhu a implementaci, jak paralelizujeme testovací úkoly již ve fázi analýzy. Klíčovým tématem této kapitoly byla rizika a architektura. Smyslem Elaboration je odstranit rizika technického charakteru a také implementovat a otestovat jádro scénářů, které tvoří architekturu. Ukázka postupů testování, plánování a hodnocení iterací, včetně příkladu plánu a hodnocení iterace E1 bylo nezbytnou součástí této kapitoly.

5 Fáze Konstrukce (*Construction phase*)

V této kapitole se dozvíte:

- Jaké jsou cíle této fáze projektu a jak vývoj v této fázi probíhá.
- O čem je princip organizace kolem architektury.

Po jejím prostudování byste měli být schopni:

- Správně organizovat tým a paralelizovat vývoj.
- Pracovat s příchozími změnami požadavků.

Klíčová slova této kapitoly:

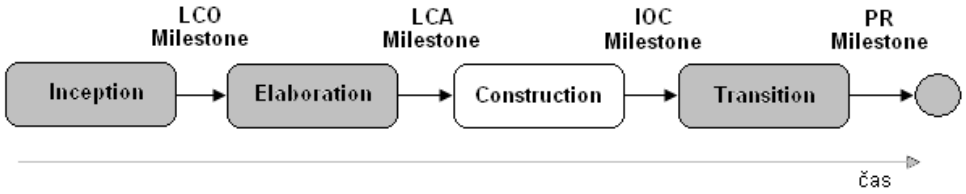
Organizace kolem architektury, paralelní vývoj, testování, beta release.

Doba potřebná ke studiu: 2 hodiny

Průvodce studiem

Kapitola představuje fázi konstrukce. Iterace této fáze jsou hlavní vývojovou fází projektu. V předchozí fázi jsme ověřili postup a technologii (resp. architekturu) a nyní již můžeme bezpečně pokračovat ve vývoji požadavků iteraci za iterací. V kapitole budeme mluvit o možnostech paralelizace vývoje a organizace kolem architektury a testování.
Na studium této části si vyhraďte 2 hodiny.

Předchozí fáze Rozpracování (*Elaboration*) byla ukončena interním releasem základní, spustitelné architektury systému tvořenou základními scénáři. Tento release umožnil identifikovat a implementací přímo ověřit největší technická rizika (soupeření o zdroje, výkonnostní rizika, zabezpečení dat, …). Následující fáze zvaná konstrukce (*Construction*), která je předmětem této kapitoly, je zaměřena na detailní návrh, implementaci a testování všech ostatních scénářů, aby bylo zajištěno zhmotnění kompletního systému podle přání a potřeb zákazníka.



Obr. 5-1: Fáze Construction

Předmětem prací v této fázi je návrh, implementace a testování zbývajících přibližně 80% use case a finální implementace původních 20%, které představují kritické (hlavní) požadavky zákazníka a nebyly třeba dokončeny kompletně (zbývající testy vstupů, zachycení výjimek apod.). ***Dosud byla implementována pouze malá podmnožina z celkového kódu aplikace, i když velmi důležitá***. 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 a nutnost návratu do této fáze. 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 fáze konstrukce (*Construction*) lze definovat následovně:

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

5.1.1 Iterace ve fázi konstrukce

Počet iterací této fáze se bude opět lišit projekt od projektu, v zásadě lze říci, že jich bude více než v jiných fázích, typický projekt ve známé doméně bude obsahovat minimálně 2-4 iterace. Plánování iterací bude opět řízeno use casy, tak jako v iteracích předchozí fáze. ***Nejdříve tedy budeme implementovat nejdůležitější use case z pohledu 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, další už by měly být řízeny hlavně hodnotou pro zákazníka.

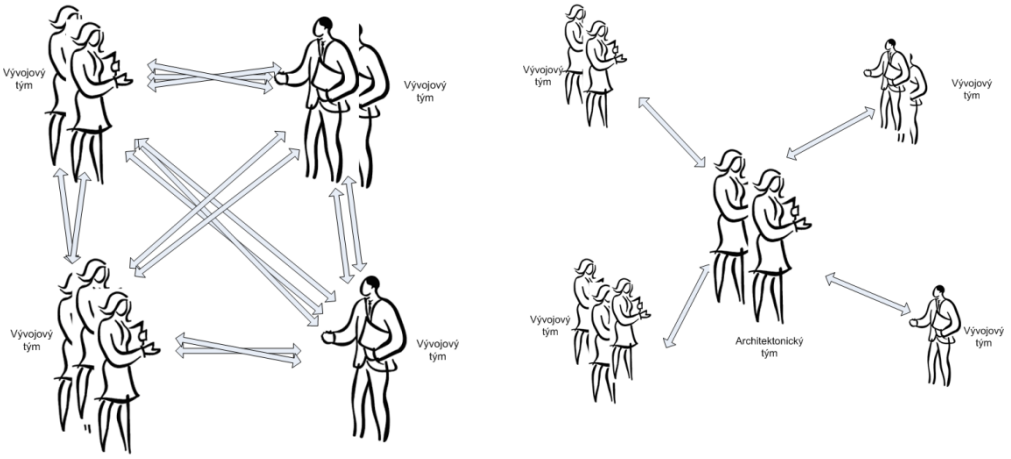
5.1.2 Minimalizace nákladů na vývoj, paralelní vývoj

Cílem správně provedené předchozí 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.

Příklad k vypracování: Vezměte si váš analytický diagram tříd či diagram balíčků a popište jednotlivé odpovědnosti základních balíčků vašeho řešení. Můžete popsat i detailnější odpovědnosti než je navržené rozdělení vrstev.

Tento způsob definování odpovedností a oddělení implementace 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 (každý tým odpovědný za nějaký z nich). 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 rozhodující slovo (rozhodovat o jejich struktuře) v případě neshod.



Obr. 5-2: Organizace kolem architektury minimalizuje přílišné komunikační zatížení.

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. Toto budování a ladění se týká nejen nástrojů a jejich nastavení, ale také procesu, jak a kam soubory nahrávat a jak konfigurace vytvářet a spravovat. V průběhu vývoje vzniká spousta různých souborů (zdrojové, konfigurační, spustitelné, skripty, …) 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 zahrnuje 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, tzv. *Definition of Done* (DoD).
- 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.

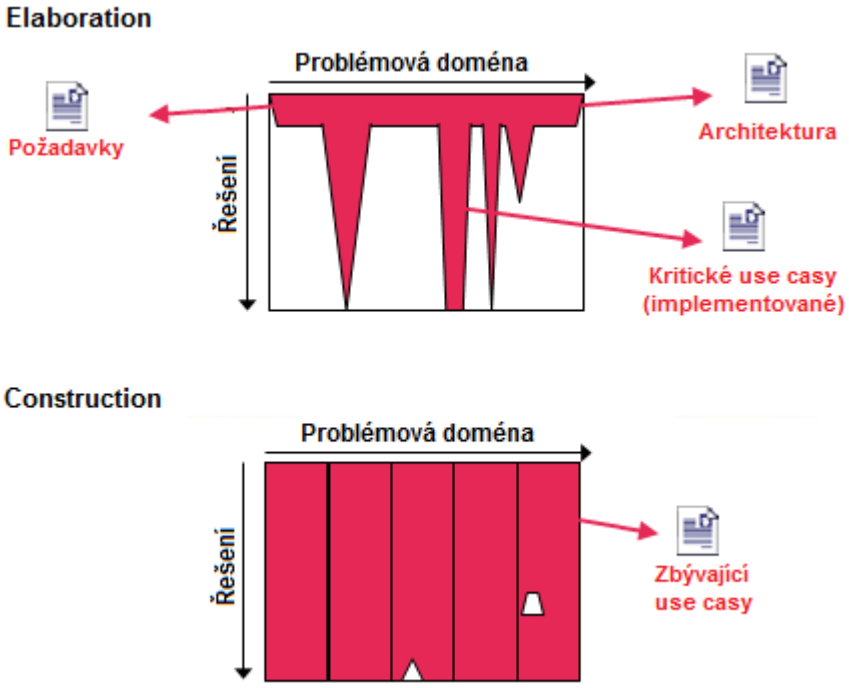
5.1.3 Iterativní vývoj kompletního produktu

V průběhu Elaboration jsme detailně popsali pouze kritické use casy nebo ty, které mají vliv na architekturu. Méně významné UC z pohledu architektury 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 byznysu 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 fázi, ale také v Inception a hlavně Elaboration.

V Elaboration fázi 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 casů. Situaci popisuje následující obrázek.



Obr. 5-3: Stav projektu a jeho rozpracovanost do hloubky ve fázi Elaboration a Construction.

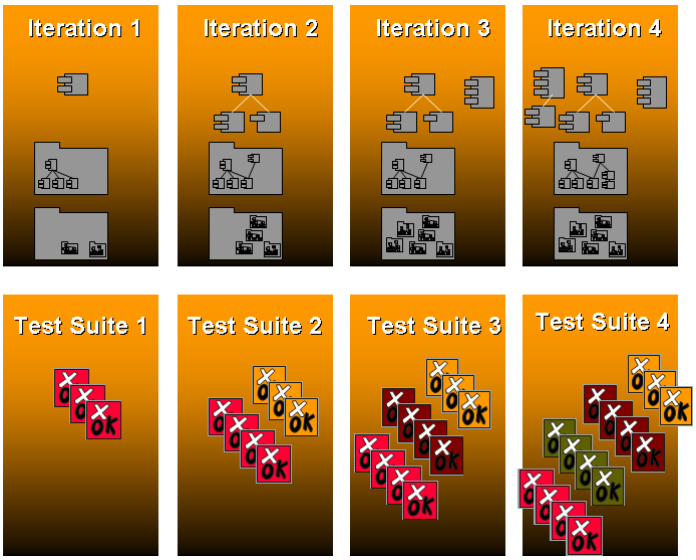
V Elaboration fázi jsme se zaměřili na implementaci kritických use case a to buď jejich úplnou nebo částečnou, která snižuje identifikovaná rizika. V Construction se zaměřujeme na úplnost. Cílem je krok za krokem (rozuměj iteraci za iterací) kompletně implementovat všechny scénáře, včetně alternativních a chybových toků, kontrol, ošetření vstupů, výjimek, nutné je také neustále aktualizovat dohodnutou programátorskou dokumentaci a také připravovat uživatelskou dokumentaci pro části, které se již budou používat v provozu.

V průběhu Elaboration jsme také na základě objektového modelu aplikační logiky navrhli a implementovali koncept databáze pro určité scénáře. 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 schéma a definované tabulky (struktury) databáze, to by svědčilo o nedostatečně stabilním návrhu architektury a tedy předčasném začátku Construction fáze.

5.1.4 Testování

Důležitým faktorem Construction fáze je ***opět průběžné ověřování chování*** implementovaných komponent/systémů. Pro tento účel je z pohledu vývojáře

využíváno již vytvořených 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 testovacích scénářů (test casů, které jsou odvozeny z use case a jejich scénářů. Jaká omezení (např. výkonnost) je třeba testovat najdeme v nefunkčních požadavcích. ***Testy vytváříme inkrementálně v jednotlivých iteracích, stejně jako scénáře.***



Obr. 5-4: Inkrementálně v jednotlivých iteracích nevznikají jen funkčnosti aplikace, ale také testy (jak manuální, tak automatické skripty).

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 chyb a odhalení příčin tohoto stavu. 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 na několik zásad:

- Cíle testování identifikujeme analýzou iteračního plánu, je třeba vědět, co je jeho cílem, abychom toto mohli následně otestovat.
- Je třeba identifikovat způsob testování: ručně, automatizovaně, nástroje apod.
- Musíme analyzovat způsob a výběr oblastí, pro které vzniknou testovací scénáře (*test case*).
- Provedeme testy pro každý testovací případ (test case).
- Analyzujeme testy, které selhaly a navrhujeme jejich 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. Proto provádíme nejpozději na konci každé iterace demonstraci zákazníkovi, či jeho reprezentantovi (například byznys konzultant).

Na konci Construction fáze ***provádíme také tzv. beta-relase***, jehož předmětem je testování do něhož jsou zahrnuti vybraní uživatelé[1]. 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, musí však fungovat všechny scénáře, nesmí se 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 školení uživatelů a správců aplikace.
- Přípravu prostředí pro nasazení (nákup nového HW, konvertování dat, zapojení do sítí a jejich konfigurace apod.) a přípravu dat.
- Příprava dalších aktivit zahrnujících marketing, distribuci, prodej.

5.1.5 Milník IOC

Tento milník je velmi důležitý, jelikož nám říká, zda je produkt připraven pro nasazení a beta-testování. Zkratka znamená *Initial Operation Capability*, což značí produkt schopný provozu. Zda jsme dosáhli tohoto milníku nám opět pomůže zjistit následující kontrolní seznam:

- Je produkt dostatečně stabilní a vyzrálý, aby mohl být rozeslán mezi komunitu uživatelů (odpovědí jsou výsledky testování, včetně beta-testování a zpětná vazba uživatelů)?
- Jsou aktuální výdaje na zdroje oproti plánovaným stále akceptovatelné?

Kontrolní otázky:

1. Co je to organizace kolem architektury a jaký je její smysl?
2. Jakým způsobem můžeme paralelizovat vývoj ve fázi Construction?
3. Co je to CM?
4. Co je to beta release a jaký je jeho smyslem?

Úkoly k zamyšlení:

Zamyslete se nad problematikou paralelních týmů. Jak musí být rozdělena aplikace, aby si nepřepisovali vzájemně kód? Jde toto nějak řešit? Mohou takové týmy fungovat i v případě, že nesedí v jedné místnosti (například jeden

[1] Určitě jste někdy zaregistrovali, že je možné si zdarma stáhnout nové připravované verze Windows s kódovým označením RC. Toto je přesně beta release. RC znamená Release Candidate a většinou ještě následuje číslo, například RC4. Smyslem této verze je rozšířit ji mezi komunitu, která je díky své velikosti schopná otestovat i situace a scénáře, které interní testovací tým Microsoftu není schopný pokrýt, ať již z důvodu jejich specifičnosti nebo naopak z důvodu množství uživatelů.

tým v Praze, druhý v Německu a třetí v Číně)? Jaké praktiky a pravidla byste zvolili, aby tento model fungoval?

Korespondenční úkol:
Popište, jak se liší iterace v Elaboration od iterací v Construction fázi. Mají stejné nebo jiné cíle? Jestli ano, proč? Jestli ne, proč? Na co se v nich zaměřujeme a jak je to s implementací scénářů? Implementuji v obou fázích kompletně celé use casy nebo jen některé jejich části?

Shrnutí obsahu kapitoly
Tato kapitola se zabývala fází Konstrukce (*Construction*). Smyslem této fáze je iteraci za iterací implementovat všechny scénáře s využitím architektury definované a ověřené v předchozí fázi Elaboration. Klíčovými koncepty této fáze je organizace kolem architektury, paralelní vývoj, testování a beta release. Organizace kolem architektury řeší komunikaci více týmů, respektive vývojářů a vyžaduje oddělené komponenty s jedinečnou odpovědností (principy OOP). Dále jsme se zabývali paralelním vývojem pro který je nezbytná nejen architektura s nezávislými komponentami, ale také nástroje konfiguračního managementu. Nedílnou součástí iterací této fáze je opět vývojářské a systémové testování a také zahrnutí uživatelů v širším měřítku pomocí první (vydání schopné kompletní) verze zvané beta release.

6 Fáze Předání (*Transition phase*)

V této kapitole se dozvíte:

- Jaké dokončovací aktivity provádíme v této fázi.
- Jaká je nezbytná vazba na ostatní nevývojové aktivity (prodej, marketing, distribuce).

Po jejím prostudování byste měli být schopni:

- Připravit plán této fáze.

Klíčová slova této kapitoly:

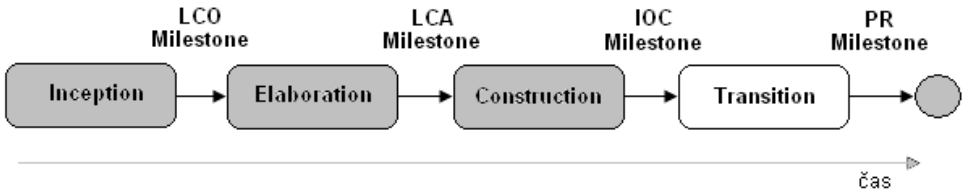
Školení, příprava prostředí, akceptační testování, ponaučení.

Doba potřebná ke studiu: 1 hodinu

Průvodce studiem

Kapitola představuje poslední fázi iterativně inkrementálního modelu vývoje softwaru zvanou Předání (Construction). Smyslem iterací této fáze je dokončení jednoduchých scénářů, které jsme posunuli do této fáze, oprava zbývajících chyb, vyladění běhu aplikace a příprava a předání do provozního prostředí. Důležitým krokem této fáze je akceptační testování, pilotní provoz a navázání na nevývojové aktivity jako je distribuce produktu, marketing či zaškolení a předání do údržby.
Na studium této části si vyhraďte 1 hodinu.

Poslední fáze představovaného iterativního způsobu vývoje se nazývá Předání (*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 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, to vše podle velikosti a formálnosti projektu.



Obr. 6-1: 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 otestovaná aplikace obsahující téměř všechny funkčnosti!

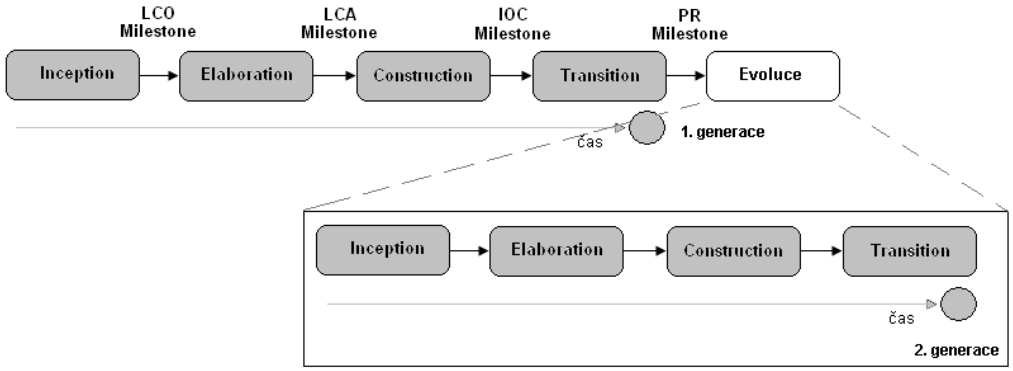
6.1.1 Cíle fáze Předání

Cíle této fáze jsou následující:

1. Beta-testování – zjištění, zda jsme naplnili očekávání uživatelů.
2. Dokončení uživatelské dokumentace, která byla vytvářena v každé iteraci pro dříve nasazené části aplikace
3. Školení uživatelů a správců aplikace, předání znalosti.
4. Příprava prostředí (HW, sítě, finální integrace rozhraní) a dat.
5. Příprava dalších aktivit zahrnujících marketing, distribuci, prodej – tvorba letáků s popisem produktu, white papers, technical papers, případové studie, demo nahrávek, zprávy pro tisk.
6. Dosažení souhlasu uživatelů, že aplikace splňuje jejich představy zachycené v dokumentu vize (*Vision*) formou akceptačního testování.
7. Zlepšení průběhu budoucích projektů díky ponaučením z tohoto projektu tzv. *lessons learnt*, které jsme sbírali a aplikovali na konci každé iterace.

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

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. 6-2: 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.

6.1.2 Testování a akceptační 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[2]. 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 všech iterací fází Elaboration, Construction i Transition. Na konci 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.

Akceptační testování je naopak záležitostí zákazníka. Smyslem je zahrnout klíčové uživatele, kteří se již účastnili demonstrací a systém znají. Díky tomu je vlastní akceptační záležitost spíše jen formální záležitostí, protože uživatelé již byli do vývoje vtaženi, systém znají a byly reflektovány jejich zásadní připomínky. Pokud tomu tak není, může se stát, že objeví zásadní připomínky ovlivňující architekturu systému. Smyslem je tedy ověření funkčnosti systému směrem k vizi a požadovaným změnám v průběhu projektu z pohledu zákazníka.

6.1.3 Předání do údržby

Důležitým krokem této fáze je předání aplikace do provozu a údržby. Existuje několik modelů provozu a pro každý z nich budou platit jiná doporučení. V zásadě jde o tyto scénáře:

1. Provoz a údržba je prováděna jiným týmem
    a. v rámci stejné organizace
    b. v jiné organizaci
2. Provoz a údržba je prováděna stejným týmem

[2] viz popisy testování v předchozích fázích Elaboration a Construction.

Základní doporučení je ***zahrnout členy budoucího týmu provozu a údržby do projektu vývoje co nejdříve*** (jako vývojáře, analytiky nebo testery), aby znali architekturu systému, procesy, které jsme následovali, principy a postupy konfigurací, testovací postupy a scénáře. Pokud to není možné je dobré je zahrnout alespoň do vzorové iterace v rámci Transition fáze nebo pro ně alespoň uspořádat několik praktických workshopů na předání této znalosti. Dalším možným scénářem je, že se někteří členové původního týmu vývoje stanou dočasnými nebo stálými členy týmu provozu a údržby. Možná je také časově omezená a pravidelná rotace členů mezi těmito týmy. Smyslem je předání znalosti struktury a organizace zdrojových kódů (arechitektury) a postupů vývoje. Bez této znalosti je provoz a údržba rozsáhlých systémů téměř nemožná[3].

6.1.4 Závěrečná ponaučení (*Lessons learnt*)

V průběhu celého projektu jsme na konci každé iterace provedli retrospektivu. Ptali jsme se v týmu, co fungovalo a co ne. Co nefungovalo jsme se snažili analyzovat a hned v příští iteraci zlepšit. Smyslem bylo neustálé zlepšování nejen aplikace, ale také vlastního procesu vývoje. Je možné, že jsme docela dobře vylepšili jak náš proces vývoje, tak i použité nástroje a jejich konfiguraci.

Smyslem (nejen) závěrečného sběru ponaučení je tyto lekce předat mezi další projekty a týmy, ale také je ***promítnout do procesů*** a *best practices* organizace. Výsledkem projektu mohou být také ***komponenty či patenty***, které lze použít v dalších projektech jak uvnitř organizace, tak u třetích stran. Můžeme například znovupoužít nastavení prostředí (jako struktura repository, nastavení nástrojů), některé komponenty apod.

6.1.5 Milník PRM

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

Primárními evaluačními kritérii Transition fáze jsou následující otázky:

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

Výsledkem této fáze je fungující nasazený systém v provozním prostředí zákazníka a případná vize a use case model dalších verzí.

[3] Více informací o těchto scénářích a možnostech předání či vybudování znalosti již provozovaného systému viz Procházka, Klimeš: *Provozujte IT jinak. Agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb*. Grada. 2011.

Kontrolní otázky:

1. Co jsou základní cíle fáze předání?
2. Co si představíte pod pojmem akceptační testování?
3. Jaký je smysl ponaučení a retrospektivy v projektu?
4. Co za aktivity zahrnuje příprava provozního prostředí?

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 s poslední fází iterativně inkrementálního modelu vývoje softwaru. Smyslem fáze Předání (*Transition*) je zaměření se na dokončení a doladění finální verze produktu, předání do provozního prostředí (příprava HW, instalace a propojení s existujícími systémy, migrace dat, konfigurace sítě, školení) a akceptační testování provedené zákazníkem. Výsledkem této fáze je fungující nasazený systém v provozním prostředí zákazníka a případná vize a use case model dalších verzí.

7 RUP vs. OpenUP

Učební text, který právě čtete, se věnuje iterativně inkrementálnímu vývoji pomocí procesního frameworku *OpenUP*. 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 od RUPu, který je maximalistický a obsahuje vše a my si vybíráme pouze to, co potřebujeme (což je právě to těžké, více viz Informační systémy), je OpenUP minimalistický, obsahuje pouze nutné minimum potřebné pro agilní vývoj software. Navíc je tento metodický framework/aplikace zdarma, dostupná každému ke stažení, odkaz viz níže.

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).

8 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.

[Pro11] Procházka, Klimeš: *Provozujte IT jinak. Agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb*. Grada. 2011.

[Pro12] Procházka, Vajgl: Informační systémy 1. Elektronický učební text. Ostravská univerzita. 2012.

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

[SEI2] Clements, P. et al.: *Documenting Software Architectures: Views and Beyond* (The SEI Series in Software Engineering). Addison-Wesley. 2002. ISBN 0201703726.

[SEI3] Barbacci, M., R. et al.: *Quality Attribute Workshop (QAWs)*. 3rd edition. *Technical Report* CMU/SEI-2003-TR-016. Dostupné na: http://www.sei.cmu.edu/pub/documents/03.reports/pdf/03tr016.pdf

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

9 Přílohy

Jako přílohy jsou uvedeny příklady nebo šablony některých artefaktů RUP, 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*).

10 Příloha A – Vize

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.*

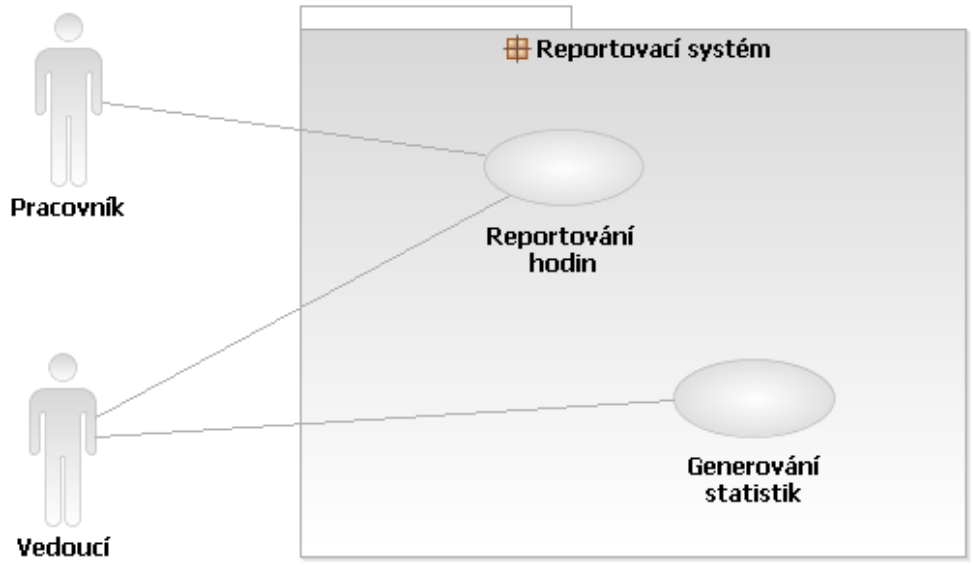
Define any specific documentation requirements, including user manuals, online help, installation, labeling, and packaging requirements.

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

Requirement	Priority	Planned Release

11 Příloha B - Use case model

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

12 Příloha C – Risk List

Příklad:

Risk ID	Risk Ranking/ Magnitude	Risk Description & Impact	Mitigation Strategy and/or Contingency Plan
R1	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.
R2	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.
R3	3	Incompatibility with internet browsers and specific configurations on client machines.	Address during Elaboration Phase by using more browsers for development and testing.
R4	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.
R5	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).

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

Akce na snížení těchto rizik byly popsány výše v textu.

13 Příloha D – Projektový plán

Příklad projektového plánu na počátku projektu. Iterace v Construction ještě nemají přesné datum, jelikož se projektový plán a vyvíjené požadavky mohou měnit. Je však již známé datum milníku IOP, jelikož k tomuto datu musíme mít hotovou beta verzi, abychom dodrželi dohodnutý termín dokončení projektu.

Fáze	Iterace	Primární úkoly	Datum od - do
Inception	I0	Sestavení týmu, rozpočtu Identifikace základních požadavků a rizik Výběr kandidátů architektury Implementace prototypu 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	Bude upřesněno
	C2	UC2 [AF1]: Export do Excelu	Bude upřesněno
	C3	Rezerva (2 dny)	Bude upřesněno
		IOC (beta release)	26.03.2008
Transition	T1	Akceptační testování zadavatelem ... PR: Předání projektu	13.05.2008 - 07.06.2008

14 Příloha E – Iterační plán

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 a R2.
- 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.

Retrospektiva a ponaučení:
Co fungovalo: …
Co lze zlepšit: …

- Konkrétní akce, vlastník akce, termín: …