**VYSOKÁ ŠKOLA EKONOMIE A MANAGEMENTU**
Nárožní 2600/9a, 158 00 Praha 5

# DIPLOMOVÁ PRÁCE




## MANAGEMENT FIREM

**Vysoká škola ekonomie a managementu**
info@vsem.cz / www.vsem.cz

# VYSOKÁ ŠKOLA EKONOMIE A MANAGEMENTU

Nárožní 2600/9a, 158 00 Praha 5

| NÁZEV DIPLOMOVÉ PRÁCE/TITLE OF THESIS |
| --- |
| Životní cyklus projektu vývoje SW v bankovním prostředí |

| TERMÍN UKONČENÍ STUDIA A OBHAJOBA (MĚSÍC/ROK) |
| --- |
| Červen 2020 |

| JMÉNO A PŘÍJMENÍ STUDENTA / STUDIJNÍ SKUPINA |
| --- |
| Vladimír Křížek / MF27 |

| JMÉNO VEDOUCÍHO DIPLOMOVÉ PRÁCE |
| --- |
| doc. Ing. Zita Prostějovská, Ph. D. |

| PROHLÁŠENÍ STUDENTA |
| --- |
| Odevzdáním této práce prohlašuji, že jsem zadanou diplomovou práci na uvedené téma vypracoval samostatně a že jsem ke zpracování této diplomové práce použil pouze literární prameny v práci uvedené.<br>Jsem si vědom skutečnosti, že tato práce bude v souladu s § 47b zák. o vysokých školách zveřejněna, a souhlasím s tím, aby k takovému zveřejnění bez ohledu na výsledek obhajoby práce došlo.<br>Prohlašuji, že informace, které jsem v práci užil, pocházejí z legálních zdrojů, tj. že zejména nejde o předmět státního, služebního či obchodního tajemství či o jiné důvěrné informace, k jejichž použití v práci, popř., k jejichž následné publikaci v souvislosti s předpokládanou veřejnou prezentací práce, nemám potřebné oprávnění.<br><br>Datum a místo: 24. 2. 2020, Praha |

| PODĚKOVÁNÍ |
| --- |
| Rád bych tímto poděkoval doc. Ing. Zitě Prostějovské Ph. D. za metodické vedení a odborné konzultace, které mi poskytla při zpracování mé diplomové práce. Dále bych rád poděkoval MUDr. Anně Angelovičové, Ing. Petře Nobilisové a v neposlední řadě Bc. Barboře Popelkové za poskytnuté konzultace a korekturu mé diplomové práce. |

# VYSOKÁ ŠKOLA EKONOMIE A MANAGEMENTU

Nárožní 2600/9a, 158 00 Praha 5



**SOUHRN**

**1. Cíl práce:**

Cílem diplomové práce je navrhnout konfiguraci metodiky Rational Unified Process pro potřeby vývoje robustního softwaru a následně definovat její vhodné zapracování pro IT projekt v bankovním prostředí. Práce se dále v teoretické části zabývá rozdílem mezi tradičními a agilními metodikami a jejich vhodností pro jednotlivé typy projektů. Praktická část práce obsahuje současnou charakteristiku vývoje softwaru ve společnosti XY, u které je následně provedeno její zhodnocení a navrhnutí doporučení pro zlepšení vývoje v budoucích projektech. O tato doporučení se následně opírá navržená konfigurace metodiky Rational Unified Process, dle které je následně vypracována případová studie zabývající se projektem v bankovním prostředí za pomoci navrhované konfigurace.

**2. Výzkumné metody:**

Diplomová práce je rozdělena do čtyř části, a to úvod, teoreticko-metodologickou část, praktickou část a závěr. Teoreticko-metodologická část diplomové práce je zpracována na základě komparace odborné české a světové literatury v oblasti životního cyklu projektu vývoje softwaru. Následně byl proveden rozbor hlavních přístupů jednotlivých metodik. V praktické části je za pomocí rozboru průběhu projektu vypracována charakteristika vývoje softwaru ve společnosti XY a provedeno její zhodnocení. Konfigurace metodiky Rational Unified Process je vypracována na základě syntézy pravidel pro vývoj softwaru popsaných v teoretické části práce a díky doporučením, která byla formulována v rámci hodnocení projektu vývoje softwaru ve společnosti XY. Poslední oddíl praktické části se zabývá zhodnocením aplikace navržené konfigurace v případové studii projektu, věnujícímu se vývoji softwaru v bankovním prostředí.

**3. Výsledky výzkumu/práce:**

Z výzkumu bylo zjištěno, že pro společnost XY je vývoj softwaru za použití současného postupu nevhodný a při jeho využití došlo k závažným nedostatkům. Tyto nedostatky se týkaly především testování vyvinutého kódu v jednotlivých prostředích, do kterých byl v rámci implementačních balíčků kód nasazen. To způsobilo pozdní odhalení a opravu chyb, což vyústilo k posunutí hramonogramu v čase. Dále došlo k absenci validace jednotlivých částí testovacích scénářů takzvanou kontrolou čtyř očí, což se negativně projevilo v průběhu samotného testování a při nasazení do produkce zvýšeným výskytem incidentů. Mezi problematickou část patří též nevhodný návrh architektury pro vývoj softwaru v rámci společnosti, který nezohledňuje její budoucí přepoužitelnost a možné snížení nákladů u budoucích projektů.

**4. Závěry a doporučení:**

Na základě detailní identifikace jednotlivých nedostatků, které se v průběhu projektu vyskytly, došlo k návrhu preventivních opatření zohledňujících slabá místa vyskytující se při vývoji softwaru ve společnosti XY. Tato opatření byla implementována do konfigurace metodiky Rational Unified Process a použita při implementaci nových funkčností v rámci projektu, který blíže popisuje případová studie. Díky těmto změnám v přístupu vedení vývoje softwaru ve společnosti XY došlo k odstranění původních nedostatků, které byly pro projekty v této společnosti běžné. Navržená metodika též pokrývá případný rozvoj a znovupoužití architektury v rámci budoucích projektů a tím možnost snížení nákladů na nové implementace v rámci společnosti.



**KLÍČOVÁ SLOVA**

Rational Unified Process, životní cyklus projektu, IT

# VYSOKÁ ŠKOLA EKONOMIE A MANAGEMENTU

Nárožní 2600/9a, 158 00 Praha 5



**SUMMARY**

**1. Main objective:**

The aim of this thesis is configuration of Rational Unified Process methodology for development of robust software and utilisation of this methodology in bank IT project. In theoretical part the thesis next focus on differences between iterative and agile software development in order to determine their fitness for certain types of projects. The practical part includes current definition of software development in company XY, evaluation of software development and sugestions for its improvement in future projects. These suggestions are incorporated in configuration of Rational Unified Process methodology and presented case study of software development.

**2. Research methods:**

The thesis is divided into four parts, those are introduction, theoretical part, practical part and conclusion. Theoretical part is based on references of recent technical czech and foreign literature concerning software development and compares their suggested approaches. In practical part is used case study of sofware development in order to refer the main characteristics of the process in company XY and evaluate the whole process. Configuration of Rational Unified Process methodology unites rules of software developments mentioned in practical part with final recommendations of case study. Last part evaluates practical use of suggested configuration in case study concerning software development in banking enviroment.

**3. Result of research:**

The conclusion of the research is that current methods of software development in company XY are inappropriate and incorporates severe flaws. Found flaws are concerning mainly inadequate code testing for its various settings in implementations pack what resulted in late identification and correction of errors with significant time prolongation in schedule. Further difficulies were due to absence of tested scenarios validation, called control of four eyes, what resulted in increased number of incidents during said testing and using in the production. Proposition of software architecture develepment also proved difficult as it did not regarded its possible further reuse and cost savings in next company projects.

**4. Conclusions and recommendation:**

Based on flaws found in course of software development in company XY were suggested preventive measures which were in next step implemented into configuration of Unified Process methodology and used for improvement of project progress shown in case study. Due to taken measures in software development in company XY, were reduced original flaws commom for all of this company projects. Suggested methodology concerns also further development and reuse of software architecture which can lead to significant cost savings in future company projects.



**KEYWORDS**

Rational Unified Process, project life cycle management system, IT

**JEL CLASSIFICATION**

M15 IT Management
O22 Project Analysis

**Vysoká škola ekonomie a managementu**
**Nárožní 2600/9a, 158 00 Praha 5**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| Jméno a příjmení: | Vladimír Křížek |
| Studijní program: | Ekonomika a management (Ing.) |
| Studijní obor: | Management firem |
| Studijní skupina: | MF 27 |
| Název DP: | Životní cyklus projektu vývoje SW v bankovním prostředí |
| Zásady pro vypracování (stručná osnova práce): | 1. Úvod<br>2. Teoreticko-metodologická část práce – přístupy k řízení projektů vývoje softwaru – agilní metody, vodopádový životní cyklus, Rational Unified Process, Business analýza<br>3. Praktická část – specifika IT projektů realizovaných v bankovním prostředí, návrh postupu řízení projektu ve specifickém prostředí, fáze projektu, zpracování požadavků, implementace, kontrola, doporučení<br>4. Závěr |
| Seznam literatury: (alespoň 4 zdroje) | • BUCHALCEVOVÁ, A., STANOVSKÁ, I. *Příklady modelů analýzy a návrhu aplikace v UML*. Praha : Oeconomica, 2013. ISBN 9788024519227.<br>• INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS. *Guide to Business Analysis Body of Knowledge (Babok Guide)*. Wettenberg : International Institute of Business Analysis, 2015. ISBN 9781927584026.<br>• SCHWALBE, K. *Řízení projektů v IT: kompletní průvodce*. Brno : Computer Press, 2011. ISBN 978-80-251-2882-4.<br>• ŠOCHOVÁ, Z., KUNCE, E. *Agilní metody řízení projektů*. Brno : Computer Press, 2019. 9788025149614. |
| Harmonogram | • Zpracování cílů a metodiky do 30. 11. 2019<br>• Zpracování teoretické části do 31. 1. 2020<br>• Zpracování výsledků do 28. 2. 2020<br>• Finální verze do 1. 5. 2020 |
| Vedoucí práce: | doc. Ing. Zita Prostějovská, Ph.D. |

prof. Ing. Milan Žák, CSc.
rektor

V Praze dne 29. 6. 2019

Prof. Ing. Milan Žák CSc.

Digitálně podepsal Prof. Ing. Milan Žák CSc.
DN: cn=Prof. Ing. Milan Žák CSc., c=CZ, o=Vysoká škola ekonomie a managementu, a.s., givenName=Milan, sn=Žák, serialNumber=ICA - 10393535

# Obsah

## Seznam zkratek

| | |
|---|---|
| BRQ | Business Requirements |
| CRM | Customer Relationship Management |
| CSS | Cascading Style Sheets |
| GDPR | General Data Protection Regulation |
| HTML | Hypertext markup Language |
| IMB | International Business Machines |
| IT | Information technology |
| MDA | Model-driven architecture |
| OOP | Object-oriented programming |
| RUP | Rational Unified Process |
| SCOPE | Rozsah projektu |
| SLA | Service Level Agreement |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SOS | Solution Screening |
| UML | Unified Modeling Language |

## Seznam obrázků

# 1 Úvod

V současné době se společnost ve vyspělých zemí setkává s nárustem softwarových produktů jak v rámci jejich počtu, tak co do složitosti jednotlivých systémů. Tento přírůstek aplikací a rychlost vývoje je umocňována využíváním informačních systémů pro obchodní aktivity korporátních společností i středně velkých podniků. Softwarové společnosti jsou na tyto systémy úzce zaměřeny a dochází k outsourcingu jednotlivých produktů, což v mnohých korporátních a bankovních prostředích způsobuje robustnost a složitost systémové architektury. Jak je ze současné praxe prokazatelné, dochází k nárustu počtu pracovníků, kteří jsou zapojeni do vývoje a údržby informačních systémů. Proto je žádané, s příchodem nových technologií a požadavků spjatých na vznikající produkty, optimalizovat způsob řízení vývoje softwaru včetně jednotlivých postupů a existujících návazností. Jedním z důvodů pro tyto optimalizace je především řada problémů, s kterými se softwarové inženýrství potýká. Mezi tyto problémy patří vysoké náklady na vývoj, kdy je systém příliš komplikovaný a dochází ke vzniku rizik. To mimo jiné může způsobit zpoždění časového plánu, tj. nemožnost dodání systému do dohohodnutého termínu. V některých případech mohou být rizika natolik závažná, že ochromí celý projekt a zabrání jeho úspěšnému dokončení, nebo proběhne dodání softwaru, který však nelze pro delší období validně užívat. Důvody těchto vad často spočívají v nesprávném pochopení nebo zadání požadavků klienta a jeho následné implementaci v rámci vývoje. Oprava je následně nákladná a mnohdy musí dojít ke změně samotné architektury systému a úpravě dotčených případů užití. Pomocí metodik je možné tato rizika snížit a proces vývoje softwaru ulehčit. Mezi pozitivně vnímanou metodiku pro vývoj softwaru patří Rational Unified Process. Nejedná se pouze o metodiku, ale především o procesní rámec pro vývoj softwaru. Rational Unified Proces zachytává jednotlivé vazby u hlavních, řídících a podpůrných procesů.

Cílem diplomové práce je provést vhodnou konfiguraci metodiky Rational Unified Software pro použití při vývoji robustního softwaru v bankovním prostředí. Pro pochopení, proč byla zvolena tato metoda, se práce zabývá rozdílem mezi tradičními a agilními metodikami a jejich vhodností pro jednotlivé typy projektu. Praktická část práce obsahuje charakteristiku vývoje softwaru ve společnosti XY. Na základě této charakteristiky byla navržena doporučení, které budou v u této konfigurace zohledněny. Dále je představena případová studie, která se zabývá projektem v bankovním prostředí a je zde použita navrhovaná konfigurace.

Diplomová práce je rozdělena na čtyři části, a to úvod, teoreticko-metodologickou část, praktickou část, případovou studii a závěr. Teoretická část práce reaguje na situaci popsanou v předchozím odstavci a v první části se věnuje problematice jednotlivých tradičních a agilních metodik. Následně se věnuje blížšímu zkoumání a analyzování metodiky Rational Unified Process. Dále představuje charakteristiku business analýzy, která je nedílnou součástí bankovních projektů a řídí se obecnými zásadami business modelování. Praktická část obsahuje charakteristiku vývoje softwaru ve společnosti XY, včetně doporučení, díky kterým byl proveden návrh samotné konfigurace včetně specifikace jednotlých rolí a fází u bankovních projektů. Konfigurace též obsahuje jednotlivé artefakty a činnosti, které v rámci odpovědnosti pod jednotlivé role spadají. V druhé polovině praktické části je tato konfigurace použita pro případovou studii, která se bude věnovat její implementací v bankovním prostředí.

# 2 Teoreticko-metodologická část práce

Jak uvádí Bruckner (2012, str. 100), v současné době se pro vývoj informačních systémů využívají především dva typy metodik. První z těchto metodik se označuje jako tradiční, neboli rigorózní, a jsou označovány jako takzvané „těžké metodiky“. Hlavní myšlenka vychází z přesvědčení, že jednotlivé etapy a postupy při vývoji softwaru lze analyzovat, popsat, naplánovat a následně řídit a měřit výstupy. Agilní metodiky se naopak označují jako „lehké metodiky“.

Tato kapitola se zaměřuje na popsání a vysvětlení rozdílů mezi tradičními a agilními metodikami. Popsány jsou fundamenty jednotlivých metodik včetně jejich historického kontextu v chronologickém pořadí. Především se však blíže zaměřuje na popis metodiky RUP, celým názvem Rational Unified Process, která se ve velké míře používá pro vývoj softwaru nejen v bankovním prostředí. Sommerville (2013, str. 21) uvádí, že metodika vývoje softwaru je specifické označení přístupů v rámci softwarového inženýrství. Označuje se tak přístup, který je použit pro řízení procesu vývoje softwaru. Metodologické rámce bývají vyvíjené a oficiálně dokumentované skupinou specialistů nebo organizací, kteří následně provádí jejich rozvoj, podporu a propagaci jejich využití.

Jak uvádí Bruckner (2012, str. 302), metody slouží pro standardizaci jednotlivých činností při vývoji informačních systémů. Bruckner (2012, str. 302) dále uvádí, že „*vycházejí ze společných základních principů. V metodikách se tyto principy promítají do všech oblastí analýzy a návrhu informačního systému a tvoří tak jádro těchto metod. Jednotlivé techniky a nástroje se mění a zdokonalují stejně jako jednotlivé metody a jimi doporučené postupy návrhu informačního systému, základní principy zůstávají stabilní. Jejich znalost pomáhá ke správnému pochopení a použití nástrojů a pravidel metod analýzy a návrhu.“* Bruckner (2012, str. 302) uvádí, že metody při vývoji informačních systémů stanovují, jaké kroky se musejí v jednotlivých fázích vývoje provést a jaké konkrétní přístupy je nutné dodržet. Jsou zpravidla spojeny s přístupem, který lze rozdělit na datový, funkční a objektový. Techniky následně blíže určují, jakým způsobem se postupuje při řešení konkrétního návrhu. Na rozdíl od metody je technika detailnější. Popisuje jednotlivé kroky činností včetně možných variant při rozhodování. Dále obsahuje nástroje, které slouží jako prostředník pro uskutečnění činností včetně popisu způsobu jejich použití.

Bruckner (2012, str. 110) dále uvádí, že tradiční metodiky se oproti agilním liší velkým důrazem na počátek projektu, kde dochází k obsáhlému sběru požadavků, jejich analýze a následně samotnému návrhu systému. Jako proměnné hodnoty jsou zde čas a zdroje, naopak fixní je samotná funkcionalita softwaru. Agilní metodiky si oproti tomu kladou za cíl dodat zákazníkovi v co nejkratším čase alespoň část produktu, který je kvalitní a lze jej používat nezávisle na budoucím vývoji. Jednotlivé funkce se tedy v aplikaci přidávají rychleji, nežli při tradičním vývoji. Při agilním programovacím přístupu je tedy fixní čas, za který se produkt vyvine a zdroje, které lze použít. Rozsah projektu, SCOPE, zůstává proměnný. Podmínkou těchto metodik je neustálá komunikace s interním uživatelem, který se podílí na návrhu a především na testováním aplikace. Bohužel v bankovním prostředí je příliš zainteresovaných stran a tento model se většinou nedaří splnit.

## 2.1 Tradiční metodiky pro vývoj softwaru

Jak uvádí Pergl (2008, str. 18), jako první zaznamenaný model používaný v počátcích vývoje moderního programování (50. léta) byl model „Napiš a oprav“. Tento model vychází z naprogramování aplikace, jejím následném uvedením do provozu a poté v opravě nalezených chyb. Jak autor uvádí, v modelu je patrná absence prvků analýzy a projektového řízení. Lze tedy říci, že model vznikl spontánně a byl katalogizován až dodatečně. V roce 1957 (Pergl, 2008, str. 19) vznikl první strukturovaný model, který svými fázemi odpovídá modelu vodopádového cyklu, viz Obrázek 1 v kapitole 2.2. Nevýhoda této metodiky byla v chybějící zpětné vazbě. Po dokončení jednotlivého kroku neprobíhalo žádné ověření a fáze revalidace probíhala po dodání aplikace. Počátek vývoje jednotlivých tradičních metodik započala softwarová krize na počátku 60. let 20. století. Důraz byl především kladen na způsob vývoje softwaru, kde bylo třeba definovat jednotlivé fáze. Po určení posloupností těchto fází byl jako první navržen vodopádový model životního cyklu. Z tohoto modelu následně vychází spirálový model, který zavádí iterativní přístup a analýzu rizik.

Vodopádový i spirálový model kladou velký důraz na projektové plánování, temíny a samotný rozvrh prací. Oba modely pracují s detailním naplánováním jednotlivých částí a jejich dodržením v harmonogramu implementace při realizaci projektu. Jak uvádí Bartošová (2011, str. 24), úkolem manažera projektu, potažmo projektového týmu, je dodržet rozpočet projektu, kvalitu výstupů a plnit harmonogram pro daný projekt. S těmito třemi proměnnými pracuje tzv. „Projektový trojimperativ“, který je grafickým znázorněním těchto tří parametrů projektu. Kvalita, čas a náklady jsou přeneseny jako vzdálenosti nacházející se na třech osách, které jsou v rovině. Při spojení těchto tří přímek dochází ke vzniku trojúhelníku, znázorňující propojení a závislost jednotlivých veličin. Pokud dojde ke změně u jednoho parametru, jsou ovlivněny zbývající dva. I přes správné řízení projektů vznikají v praxi nečekané komplikace, které mohou zapříčinit změny v projektovém plánu. To má za následek negativní dopady na jeden či více proměnných. V případě, kdy na projektu pracuje pět vývojářů a v rámci jiného projektu je nutné jednoho odvolat, zůstanou čtyři a existuje několik možností jak situaci řešit:

- navýšit zpět zdroje projektu – může být provedeno například nařízením přesčasů nebo získáním jiného pracovníka;
- provést změnu v rozsahu dodání – s vlastníkem projektu provedeme zmenšení dodávky projektu tak, aby jej bylo reálné dodat ve stanovený čas se zdroji, které jsou k dispozici;
- úprava harmonogramu projektu – v rámci dodávky projektu nastane změna v harmonogramu, kdy budeme pracovat se současnými zdroji, ale bude oddáleno datum dokončení projektu.

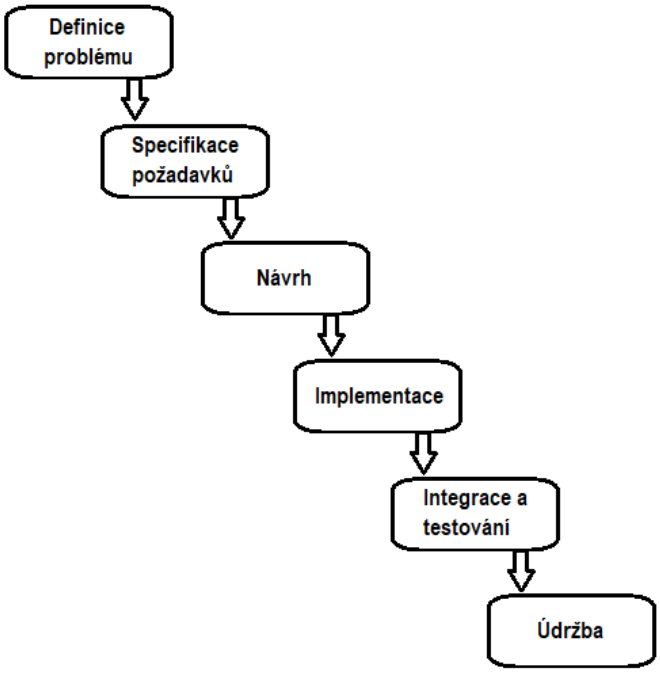
Pro minimalizování nepřesností v harmonogramu projektu, je použit Ganttův diagram. Jak uvádí Pasch (2011, s 99), diagram byl vytvořen kolem roku 1910 Henrym G. Ganttem avšak první verze dochovaného diagramu je z roku 1896 a jeho autorem je Karol Adamiecki. Lorenc dále uvádí (2013), že ke správnému řízení a kontrole projektu je třeba detailní a zároveň realistické plánování. Kromě jednotlivých kroků a jejich stavu plnění sledujeme i celkovou časovou náročnost projektu. Ganttův diagram pracuje na základě dvou částí, kdy na horizontální ose je zaznamenáno trvání samotného projektu, na vertikální ose se naopak nachází dílčí činnosti projektu, na které se projekt rozděluje. Pro každou činnost je definován samostatný řádek. V diagramu jsou jednotlivé činnosti zaneseny v čase pomocí obdelníků, kdy levá strana obdelníku indikuje začátek činnosti a pravá strana obdelníku její dokončení. Délka pruhů tedy závisí na délce jednotlivých činností v projektu. Výhody toto diagramu tkví v jeho nenáročnosti, přehlednosti a možnost rychlého grafického zpracování např. pomocí nástroje Microsoft Excel (Microsoft, 2019), který obsahuje pro tento druh diagramu bezplatnou šablonu.

Černocký (2014, str. 14) jako další výhodu uvádí možnost zobrazení časové rezervy, přehled o celkovém čase, který je potřebný pro realizaci daného projektu a jednoduchou kontrolu stavu dílčích úkolů. Mezi nevýhodu diagramu naopak autor uvádí obtížné zachycení rozsáhlého projektu mnoha úrovní.

## 2.2 Vodopádový model životního cyklu

Vodopádový model, též nazývaný jako vodopádový přístup, je tradiční a v praxi často využívaný pro vývoj informačních systémů. Cobb (2015, str. 18) uvádí, že model byl vyvinut Dr. Winstonem Roycem v roce 1970. Předpoklady pro vyžití tohoto přístupu je předem jasně daný plán, obsahující několik fází, viz Obrázek 1, který zahrnuje jednotlivé posloupnosti.

**Obrázek 1 Vodopádový model životního cyklu**




*Zdroj: Buchalcevová (2018, str. 48)*

Sherman (2015. str. 449) charakterizuje princip modelu jako sled fází, které mají jasně definovaný počátek a konec. Dále uvádí, že následující fáze může začít až po úspěšném skončení fáze předchozí. Jednotlivé fáze autor definuje a jejich proces dokumentuje v následujících bodech, které jsou také znázorněny na Obrázku 1:

- definice problému;
- specifikace požadavků;
- návrh;
- implementace;
- integrace a testování;
- údržba.

Sherman (2015. str. 450) uvádí, že ve fázi definice problému je hlavní cíl pochopit záměr zákazníka. Je tedy požadováno zjistit, k jaké konkrétní činnosti bude systém využíván, v čem má vyvíjený systém usnadnit zákazníkovi práci a také v neposlední řadě, jakou stávající činnost

tento systém nahradí. Petersen (2009, str. 3) upřesňuje, že při splnění tohoto cíle je kladen velký důraz především na nalezení společné řeči mezi dodavatelem a zákazníkem. Výstupem z této fáze je úvodní studie. Tento dokument shrnuje informace o zákazníkovi, potřeby a požadavky zákazníka a především důvody pro dodání systému. Druhá fáze se zabývá analýzou a specifikací požadavků zákazníka a jejím cílem je kvantifikovaně specifikovat přesné chování systému. V této fázi se klade důraz na detailní prozkoumání a pochopení daného problému. Výstupem této fáze je dokument, obsahující specifikace požadavků, které popisují aplikaci v jazyce zákazníka. Specifikaci je vhodné nechat zákazníka schválit a podepsat zadávací dokumentaci.

Petersen (2009, str. 4) uvádí, že fáze návrhu si klade za cíl navrhnout co nejvhodnější architekturu systému s použitím odpovídajících technologií. Dodavatel klade vysoký důraz na realizovatelnost samotného systému a dbá na předání dostatečně detailního materiálu pro implementaci, který obsahuje specifikace požadavků pro vývojáře. Výstupem je kompletní architektura systému, který je rozdělen na samostatné funkční celky. Jednotlivé celky obsahují definici chování modulů, použitý programovací jazyk a případně prerekvizity jako vývojové nástroje a zdroje třetích stran.

Petersen (2009, str. 5) uvádí, že ve fázi implementace je primární cíl naprogramovat jednotlivé moduly navrženého systému. Velký důraz se zde klade na dodržení systémové architektury. Při vzniku odchylek je nutné změny opětovně nechat potvrdit schvalovacím procesem, který je obdobný jako při návrhu systému. Výstupem v této fázi je naprogramovaný systém nebo v případě rozšíření současného takzvaný implementační balíček. Fáze integrace a testování se zaměřuje na odstranění chyb a odchylek v dodávaném systému. V současné době se klade velký důraz na komplexnost testování.

Jak uvádí Page (2009, str. 88), mezi nejčastější využívání metod pro testování patří white-box a black-box. U prvního jmenovaného je k dispozici znalost vnitřní struktury. V opačném případě při metodě black-box známe pouze očekávané chování. Výstupem této fáze je otestovaný a ověřený systém, který je možný předat k užívání zákazníkovi. V závěrečné fázi provozu a údržby je hlavní cíl dodavatelské společnosti maximální spokojenost zákazníka, která je z velké části docílena důrazem na řešení případných problémů se softwarem. V rámci této fáze je požadovaným výstupem software, který je v nejlepším případě bezúdržbový. V případě požadovaných změn ze strany zákazníka probíhá specifikace nových požadavků a následný postup dle jednotlivých fází ve výše popsaném modelu.

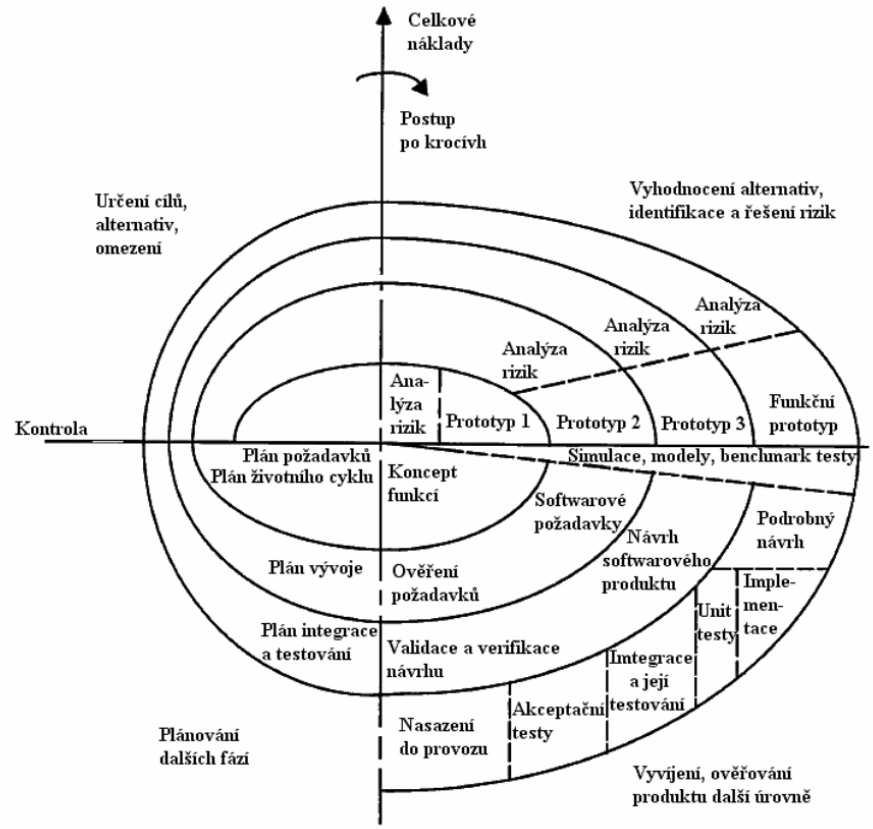
Jako hlavní výhody tohoto modelu Cobb (2015, str. 19) uvádí jednoduchost vedení projektu a možnost budoucího rozšíření vyvíjeného softwaru. Naopak jako nevýhody udává nepružnost projektového řízení tj. při změně nutnost návratu na začátek a riziko nepochopení zákazníka v jeho požadavcích. Vhodnost tohoto řízení je tedy pro malé organizace s jednoúčelovými projekty.

## 2.3 Spirálový model životního cyklu

Spirálový model životního cyklu vychází z vodopádového modelu a je postaven na opakovaných cyklech. Vyvinut byl Barrym Boehmem v roce 1985 a považuje se za další mezník ve vývoji metodik. Jak uvádí Boehm (2014, str. 25), tento model úspěšně pokrývá nedostatky vodopádového modelu životního cyklu tím, že zavádí iterativní přístup a analýzu rizik. Spirálový model se tedy zařazuje do skupiny tzv. přístupů, které se zaměřují na řízení rizik. Cílem tohoto modelu je na počátku odhadnout možná ohrožení projektu a připravit scénáře pro tyto budoucí události. Mezi největší rizika ze zde řadí kromě technických a projektových rizik předevšímdevším lidský faktor. V průběhu cyklu se nachází několik kroků,

které se neustále opakují, dokud není vývoj softwaru dokončen, viz Obrázek 2. Hlavní myšlenkou zde je implementace nových částí na již důkladně prověřené a funkční základy.

**Obrázek 2 Spirálový model životního cyklu**




*Zdroj: Boehm (2007, str.. 135)*

Boehm (2014, str. 25) popisuje, že při tomto postupu bývá problematické předem podrobně specifikovat veškeré požadavky, proto je snadnější určit obecný rámec a ten v jednotlivých iteracích postupně zpřesňovat. Díky tomu se tento model lépe vyrovnává s případnou úpravou požadavků v pozdější době projektu. Spirálový model životního cyklu je rozdělen do čtyř kvadrantů, které na sebe navazují a vytvářejí tak posloupnost jednotlivých kroků mezi cykly. V rámci prvního cyklu probíhá definice konceptu a hledání možných globálních rizik. V druhém cyklu dochází ke specifikaci a ověření požadavků zákazníka. V následujícím cyklu dochází k návrhu a vytvoření architektury dodávaného softwarového produktu. Poslední cyklus si klade za cíl provést úspěšnou implementaci produktu, jeho otestování a následnou integraci.

Boehm (2014, str. 25) dále uvádí, že mezi jednotlivé kroky tohoto modelu patří stanovení cílů, analýza rizik, realizace a plánování. V rámci stanovení cílů je nutné kromě vytyčení samotných cílů vývoje do plánu zapracovat omezující podmínky, kterými mohou být například cena, ale také alternativní řešení při nemožnosti splnění cílů dle počátečního zadání. V následujícím kroku je požadováno vyhodnotit jednotlivé alternativy vzhledem k jednotlivým omezujícím podmínkám. Analýza rizik, která probíhá před vytvořením každého prototypu, viz Obrázek 2, je dotčena druhem a množstvím rizik. Tato rizika mohou být vnějšího charakteru, např. výkon platformy nebo uživatelského rozhraní, nebo charakteru vnitřního, např. vnitřích vazeb. Fáze realizace si klade za cíl implemetaci konkrétních úkolů v rámci cyklu a verifikování dalších úrovní produktu. Podmínkou ukončení fáze realize je povedení testování a stanovení

dílčích výsledků pro možnost provedení zhodnocení. V kroku plánování je požadováno zhodnocení samotné implementace a následné přidělení zdrojů pro nadcházející cyklus. Výhoda spirálového modelu životního cyklu je přizpůsobitelnost při zavedení metodiky na konkrétním projektu. Model je komplexní a umožňuje včasné vyloučení případných nevhodných řešení. Je možné jej použít pro rozsáhlé, rizikové, či projekty s často se měnícími požadavky. Je vhodný též pro dodávky, kde je dodavatel současně zadavatelem, tj. jedná se o interní projekty nebo je kladen velký důraz na iteraci zákazníka s dodavatelem. Nevýhoda modelu tkví v komplikovanosti a absenci metodiky. Dále nelze provádět úpravy nebo zásahy do jednotlivých cyklů.

## 2.4 Agilní metodiky

Doležal (2016, str. 310), agilní metodiky, také nazývány jako „lehké metodiky“, označuje jako skupinu metod, které jsou využívány především v moderním pojetí iterativního vývoje softwaru. Tyto metodiky však neobsahují přesně určené směrnice, jakým způsobem v různých stádiích postupovat, nýbrž seznam doporučení, která lze flexibilně přizpůsobit dle konkrétního projektu. Jejich princip vychází z názoru, že nejefektivnější tvorba softwaru je jeho rychlý vývoj, následné představení zákazníkovi a na základě jeho připomínek jej dále upravovat. Zásadní rozdíl agilních metodik ve srovnání s tradičními metodikami je pojetí funkcionality softwaru jako proměnné hodnoty, oproti tomu zdroje a čas považujeme za fixní. Není tedy zaručeno dodání celé funkcionality, ale díky komunikaci s objednavatelem se jednotlivé části prioritizují v rámci požadavků. To souvisí s pravidelnou komunikací v rámci vývojového týmu, díky čemuž je možné zavčasu odhalit existující problémy.

Šochová (2014, str. 23) ve své publikaci uvádí, že hlavním důvodem ustoupení od rigorózních metodik a využití agilních přístupů je požadavek odběratelů softwaru na zvýšenou flexibilitu při implementaci změnových požadavků. Jako z dalších důvodů je výrazně větší efektivita práce v rámci pracovní skupiny než práce jednotlivců. V tomto ohledu je možné vybrat ze dvou frameworků. Techniku párového programování nebo postavení spolupracujícího týmu. Technika párového programování funguje na principu spolupráce dvou lidí na jedné činnosti, tj. například dva vývojáři pracují na jednom úkolu v rámci jednoho počítače. Naopak framework spolupracujícího týmu spoléhá na to, že jednotliví členové týmu spolupracují na jednom výsledku, v rámci tohoto procesu si pomáhají a sami se organizují. Je zde určité období, než si na sebe jednotliví členové týmu zvyknou, ale následně je možné při správně řízené funkcionalitě ušetřit až 80 % času. Myslín (2016, str. 30) dále doplňuje, že agilní přístup se snaží odstranit v maximální míře nadbytečnou byrokracii, bezdůvodné zaznamenávání každé nevýznamné události nebo aktivity. Nejedná se však o násilné odstranění za každou cenu. Tím se vyhneme vývoji softwaru v prostředí anarchie, který často vzniká při nesprávném zavedení agilního procesu a absencí zkušeného agilního kouče.

Šochová (2014, str. 76) dále doplňuje, že je vhodné zapojit do odhadů celý projektový tým. Dochází k postupnému zpřesnění odhadů a tím je možné lépe predikovat případně zpoždění v harmonogramu. Druhým aspektem je projekt rozdělit do menších časových úseků. Tímto krokem se zlepší předvídatelnost a v rámci bližších termínů dochází k provedení kritických rozhodnutí dříve. Bennett (2017, str. 9) dále upřeňuje, že v rámci projektu je možné kontrolovat cenu, kvalitu, harmonogram, rozsah, rizika a benefity projektu.

Šochová (2014, str. 90) dále uvádí, že neméně podstatné je vysoké cílení na kvalitu vyvíjeného softwaru jak ze strany kvality kódu, kdy je celý tým zodpovědný za konečný výsledek, tak ze strany zapojení zákazníka do vývoje produktu. Úzkou spoluprací s objednavatelem jsou získány hlubší poznatky o důvodech, co a proč chce, kdo a jak software bude používat. Pokud

je následně představována funkčnost po menších částech, dokážeme řídit jeho očekávání a nedojde případně k odmítnutí dodávky a požadavku na kompletní přepracování.

### 2.4.1 Manifest agilního vývoje softwaru

Myslín (2017, str. 24) uvádí, že základním stavebním kamenem přístupu lehkých metodik je Agilní manifesto (Agile Manifesto). Tito muži nabývali přesvědčení, že lze proces vývoje softwaru pojmout jiným způsobem a lépe než doposud. Agile Manifesto (2001) uvádí plné znění Agilního Manifestu včetně jeho autorů v několika jazycích, níže se nachází doslovný přepis českého překladu:

*„Objevujeme lepší způsoby vývoje software tím,*
*že jej tvoříme a pomáháme při jeho tvorbě ostatním.*
*Při této práci jsme dospěli k těmto hodnotám:*

***Jednotlivci a interakce** před procesy a nástroji*
***Fungující software** před vyčerpávající dokumentací*
***Spolupráce se zákazníkem** před vyjednáváním o smlouvě*
***Reagování na změny** před dodržováním plánu*

*Jakkoliv jsou body napravo hodnotné,*
*bodů nalevo si ceníme více.*

*Autoři:*

*Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland a Dave Thomas“*

Šochová (2014, str. 17) následně ve své publikaci detailněji popisuje jednotlivé principy a body manifestu:

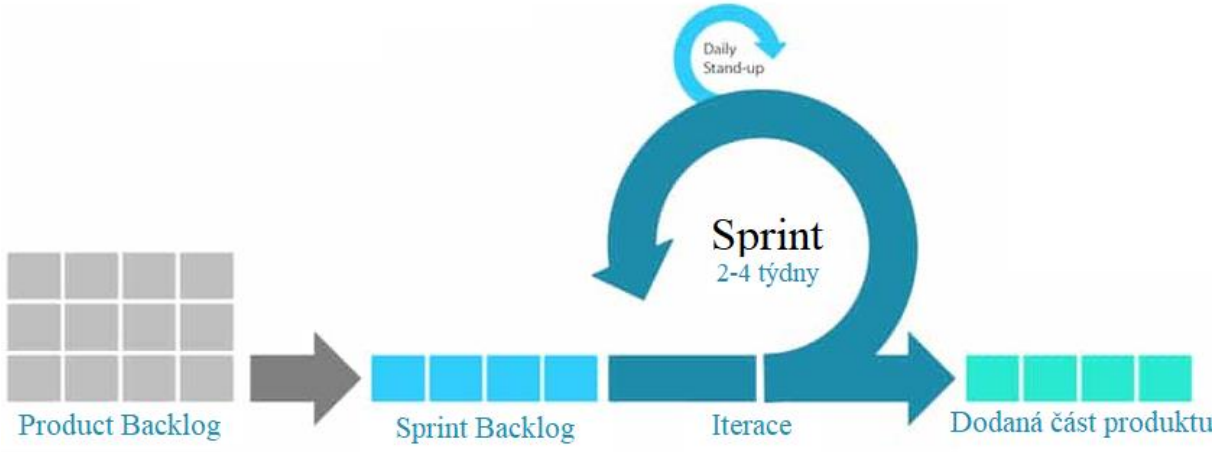
- Jednotlivci a interakce před procesy a nástroji – je všeobecně známo, že spolupracující lidé v týmu mají lepší výstupy než samostatně pracující jednotlivci. Manifest dále nezakazuje ani neomezuje použití procesů a nástrojů, doporučuje však možnost zvolit si a používat ty nástroje, které konzumentům pomáhají k dosažení kvalitních výsledků.
- Fungující software před vyčerpávající dokumentací – v agilním prostředí je primárním cílem fungující software a dokumentace je pouze nástroj, díky kterému se dostanu k cíli. Dokumentace se v agilním prostředí zaznamenává průběžně, stručně a obsahuje především klíčové informace. Nedoporučuje se dokumentaci jako takovou vynechat, ale zajistit aby užitek ze zaznamenané dokumentace odpovídal hodnotě pro zákazníka
- Spolupráce se zákazníkem před vyjednáváním o smlouvě – smlouvy jako takové jsou podstatné, ale nemají nahrazovat spolupráci a komunikaci mezi dodavatelem a odběratelem. Rozumná spolupráce je tedy v agilním prostředí preferována před vypracováním absurdně detailních smluv.
- Reagování na změny před dodržováním plánu – harmonogramy a projektové plány slouží jako vodítko, nemělo by však docházet k jejich dogmatickému dodržování, protože výsledek může uškodit oběma stranám více, než pokud by při vývoji došlo ke změně. Důvodem pro tento ústupek je neustálá změna technologií a tím i změny požadavků zákazníků a vznik nových problémů.

Jak je zmíněno v předešlé kapitole, lehké metodiky neobsahují žádné směrnice ale pouze doporučení. Stejně tak je tomu u Agilního Manifestu. Nelze jej chápat jako dokument nebo předpis, ale o prohlášení, které bylo sepsáno skupinou zabývající se vývojem softwaru.

### 2.4.2 Metodika SCRUM

Metodika SCRUM se řadí mezi agilní metodiky. Jak uvádí Šochová (2014, str. 23), využívá se u komplexních prostředí, kde je náročné vývoj naplánovat a je nutné flexibilně reagovat na změny, ale zároveň neupadnout v chaos a dokázat projekty strategicky řídit. Primárně se zaměřuje na organizaci vývojového týmu, nicméně ji lze aplikovat i mimo oblast vývoje softwaru. Čermák (2018, str. 118) dále doplňuje, že tato metodika funguje na bázi iterativního frameworku, který slouží k řízení komplexních projektů. Je navržena pro týmy o velikosti čtyř až deseti pracovníků, u kterých scrum master následně podporuje samokoordinaci v rámci týmu. Na Obrázku 3 jsou zobrazeny dílčí části procesu metodiky SCRUM.

**Obrázek 3 Proces SCRUMu**




*Zdroj: Axelos (2018, str. 11)*

Pro plánování a organizování práce se využívají backlogy uvedené na Obrázku 3, které slouží jako seznamy požadavků v kterých se schromažďuje práce. Jak uvádí Schwaber (2019, str. 15), backlogy se dále rozdělují na dva druhy. Na sprint backlog a product backlog. Sprint backlog je množina úkolů vybraných z produktového backlogu, které mají být implementovány v rámci současného sprintu. Autor jej blíže definuje jako „*seznam veškeré práce, kterou si vývojový tým vytyčil jako nezbytnou ke splnění cíle daného sprintu*“. Naopak produktový backlog lze charakterizovat jako prioritizovaný seznam všech požadavků, případně jednotlivých změn požadovaných v průběhu vývoje. Za jeho obsah, prioritizace a dostupnost přebírá odpovědnost vlastník produktu. Nelze jej v žádné fázi považovat za úplný, jelikož na začátku vývoje známe pouze základní požadavky. Lze tedy říci, že se produktový backlog vyvíjí v rámci toho, jak se vyvíjí samotná aplikace. Je to z toho důvodu, jelikož se jedná o živý dokument, který se váže k existenci produktu, a díky tomu se požadavky nepřestanou měnit. Úpravy v produktovém backlogu bývají nejčastěji způsobeny změnovými požadavky ze strany bysnysu, které vznikají z důvodu změny podmínek na trhu nebo často v případě mobilních platforem změnou technologií a designu zařízení. Jelikož je produktový backlog prioritizován, jsou položky nacházející se výše analyzovány dříve, než ty níže položené. To umožňuje v rámci nového sprintu přesunout tyto požadavky do sprint backlogu a díky přesnějšímu zadání je následně rozumně dokončit. Tento obsah úkolů je u vývojového týmu využit k tomu,

aby pomocí odhadu určil, jaké funkcionality budou v následujícím sprintu dodány a kolik časové kapacity pro tento vývoj bude potřeba. Díky tomu, že je seznam úkolů zpracován velmi podrobně, je možné sledovat jednotlivé změny na denní bázi v rámci daily scrumu neboli stand-upu. Schůzka trvající přibližně 15 minut, se provádí pravidelně každý den a slouží pro synchronizaci aktivit týmu a kontrolu odvedené práce. Je pravidlem, že probíhá na stejném místě, ve stejný čas a bez přítomnosti notebooků, případně mobilů. Jak uvádí Schwaber (2019, str. 16), každý člen týmu má za úkol odpovědět na tyto otázky:

- *„co jsem včera udělal proto, abych pomohl vývojovému týmu splnit cíl sprintu?“;*
- *„co budu dělat dnes proto, abych pomohl vývojovému týmu splnit cíl sprintu?“;*
- *„vidím nějaké překážky, které brání mně nebo vývojovému týmu ve splnění cíle sprintu?“.*

Jak doplňuje Myslín (2017, str. 112), tímto krokem dochází ke kontrole od vývojového týmu, zdali směřují k cíli sprintu a jeho úspěšnému dokončení. Tento denní stand-up zvyšuje pravděpodobnost úspěšného splnění cíle daného sprintu. Podstatné je, že scrum master tuto schůzku organizuje, ale za její průběh je zodpovědný sám tým. Je tomu tak především díky jeho vlastní sebeorganizaci, která napomáhá k vytvoření předpokládaného přírustku na konci sprintu. Samotné vyhodnocení sprintu následně analyzuje přírustek a v případě nutnosti dochází k upravení produktového backlogu. Při tomto vyhodnocení jsou přítomny jednotlivé zúčastněné strany a debatují nad výsledným stavem sprintu. Schwaber (2019, str. 12) dále doplňuje, že výsledkem vyhodnocení sprintu je revidovaný produktový backlog a předpřipravené požadavky, které mohou být zařazeny do dalšího sprintu. Jak bylo zmíněno v předchozím odstavci, product backlog není nikdy úplný dokument a může tedy dojít k přidání nových požadavků případně ke změnám u současných. Mezi vyhodnocením sprintu a plánovací schůzkou pro následující sprint dochází k retrospektivě ukončeného sprintu. Tato schůzka slouží týmu především k sebekontrole a naplánování bodů, díky kterým dojde v nadcházejícím sprintu ke zlepšení vývoje a procesů v týmu.

### 2.4.3 Extrémní programování

Autorem extrémního programování je Kent Beck, který tuto metodiku vytvořil počátkem devadesátých let dvacátého století. Beck (2005, str. 23) ve své publikaci uvádí, že *„principy a postupy užívané při extrémním programování vycházejí z těch, které se obvykle využívají při vývoji softwaru, jsou však dovedené do extrému“*. Smith (2016, str. 43) uvádí jako hlavní hodnoty extrémního programování komunikaci, jednoduchost, zpětnou vazbu a odvahu. Beck (2005, str. 24) dále uvádí, že extrémní programování klade vysoké požadavky na komunikaci mezi jednotlivými členy týmu a zákazníkem. Pro splnění tohoto požadavku je nutné používat vhodné praktiky, jako například párového programování, které zajistí častou vzájemnou komunikaci. Tím je podporován rychlý vývoj a snaha o implementaci jednoduchého systému, u kterého budou veškeré funkcionality využity, oproti složitému a komplexnímu řešení, které je díky provázanosti jednotlivých komponent časově náročné na vývoj i pochopení. Tato jednoduchost je umožněna kromě pravidelné komunikace i správně nastavenou zpětnou vazbou. U extrémního programování je zpětná vazba založena na principu „čím více validních informací, tím lépe“. Je poskytována z jednotlivých typů testování, ať již od programátorů a vývojářských testů přes jednotkové až po zákaznické testy. Pod poslední hodnotou extrémního programování, odvahou, se nachází především potřeba sdělovat nepříjemné informace. V extrémních případech může nastat nutnost odvedenou práci zahodit a začít znovu. Právě tehdy je nutné urychleně kontaktovat všechny zainteresované strany a dojít ke konsenzu. Pokračování v práci a oddalování rozhodnutí, případně vůbec nepředání těchto informací,

by v tomto případu bylo zbytečné a neekonomické. Výše uvedené hodnoty jsou promítnuty do následujících praktik:

- párové programování;
- společné vlastnictví kódu;
- neustálé testování;
- zákaznické testy;
- nepřetržitá integrace;
- velmi krátné iterace;
- malé verze;
- jednoduchý návrh;
- standardy pro psaní kódu.

Lynn (2016, str. 45) uvádí, že párovým programováním lze opravdu chápat práci dvou vývojářů u jednoho počítače. Bývá pravidlem, že jeden z dvojice analyzuje implementaci daného požadavku, zatímco druhý uvažuje o dalších testech, které je vhodné provést a zjednodušit tím implementaci. Během dne se tyto páry v pravidelných intervalech mění a to zpravidla podněcuje větší komunikaci v rámci týmu. Ačkoli tento typ vývoje nevypadá na první pohled efektivně, je potvrzeno, že párové programování zvyšuje kvalitu softwaru a především umožňuje plynule přenášet znalosti mezi jednotlivými členy týmu. Integrace a následný marge kódu se provádí několikrát za den. Při tomto procesu platí, že každý pár přebírá odpovědnost za svojí část vývoje včetně případů, kdy dojde v kódu k jakékoliv změně ze strany jiného člena týmu. Je však podstané, aby jednotliví vývojáři dodržovali standardy pro psaní zdrojového kódu, které podporují porozumění a komunikaci v týmu. To vyžaduje používání jednoduchého návrhu, kdy jsou komplexnější řešení nahrazovány jednoduššími. Pod pojmem neustálého testování se nachází nutnost zákazníkem specifikovat akceptační test pro každý jednotlivý požadavek. Díky tomu je umožněno provádět akceptační testy častěji, většinou na denní bázi. Samotná tvorba akceptačního testu poté umožňuje zadavateli lépe pochopit problém a následně jej komunikovat programátorovi. Testování, které bylo prováděno průběžně, zapříčiňuje snížení chybovosti a možnost nasazení balíčku do produkčního prostředí bez nutnosti dlouhého testování. I díky tomu lze provádět prakticky týdenní či měsíční iterace, které zákazníkovi přinášejí přidanou hodnotu. Umožněno je to především díky tomu, protože sám určil, které požadavky budou prioritně dodávány. Pro dlouhodobě udržitelný vývoj pomocí extrémního programování je nezbytné, aby byla dodržena pracovní doba čtyřiceti hodin týdně. Také je doporučeno nepraktikovat přesčasy dva týdny po sobě a neupřednosťovat vývojový plán před čerpáním plánované dovolené.

## 2.5 Metodika Rational Unified Process

RUP neboli Rational Unified Process je metodika pro vývoj softwaru, která byla vytvořena společností Rational Software Corporation, která je od roku 2003 součástí firmy IBM. Metodika vychází ze souboru osvěčených praktik a postupů při vývoji softwaru. RUP je možné díky své rozsáhlosti použít prakticky pro jakýkoliv rozsah projektu vzhledem k možnosti přizpůsobit metodiku specifickým potřebám. Doporučuje se však jeho použití spíše pro rozsáhlejší projekty, kde je zapotřebí klást větší důraz na analýzu, plánování a řízení zdrojů. Metodika byla nejdříve považována za tradiční, avšak postupně byla rozšiřována o praktiky agilního přístupu a nyní může být použita i na projekty, u kterých je nutné využití agilní metody. Samotná

metodika, jako komerční projekt, obsahuje několik tisíc stran textu a standardně se dodává v elektronické verzi. Obsahuje sadu nástrojů a webové rozhraní s intuitivní znalostní bází.

Kruchten též uvádí (2003. str. 25), že metodika obsahuje detailní návody pro zdárné dokončení vývoje softwaru dle požadavků zákazníka. Zároveň cílí na splnění dílčích úkolů, jako je termín dokončení zakázky, její rozsah a naplánovaný rozpočet. Bruckner (2012, str. 14) upřesňuje, že lze na metodiku nahlížet jako na framework, který lze v rámci konkrétního projektu konfigurovat. Může být rozšířen nebo redukován a dále upravován dle aktuálních potřeb. Cílem samotného procesu je dodávka produktu, v tomto případě softwaru, který je vysoce kvalitní a jeho dodání splňuje plánovaný časový rozvrh a odsouhlasený rozpočet klienta. Metodika RUP se řadí mezi rigorózní metodické přístupy, jelikož se zaměřuje především na vývoj nového řešení pomocí objektově orientovaného způsobu s pomocí přesné definice procesů a jednotlivých činností při vývoji softwaru.

### 2.5.1 Vývoj a historické milníky

Jak uvádí Bruckner (2012, str. 112), počátek vzniku metodiky Rational Unified Process se datuje k počátkům 80. lét 20. století a je s ním spojena firma Rational Software. Zakladateli společnosti jsou Paul Levy a Mike Devlin. Dokončena byla v roce 1995 a později vznikla její zobecněná verze Unified Process, kterou popsali Ivar Jacobson, Grady Booch a James Rumbaugh v knize Unified Software Development. Jejím cílem bylo úspěšně implementovat komplexní softwarové produkty, které byly doménou vládních organizací, především Ministerstva obrany Spojených států amerických. Dodávky se zaměřovaly na značkové harwarové platformy a prostředí pro Adu, což byl preferovaný jazyk na Ministerstvu obrany.

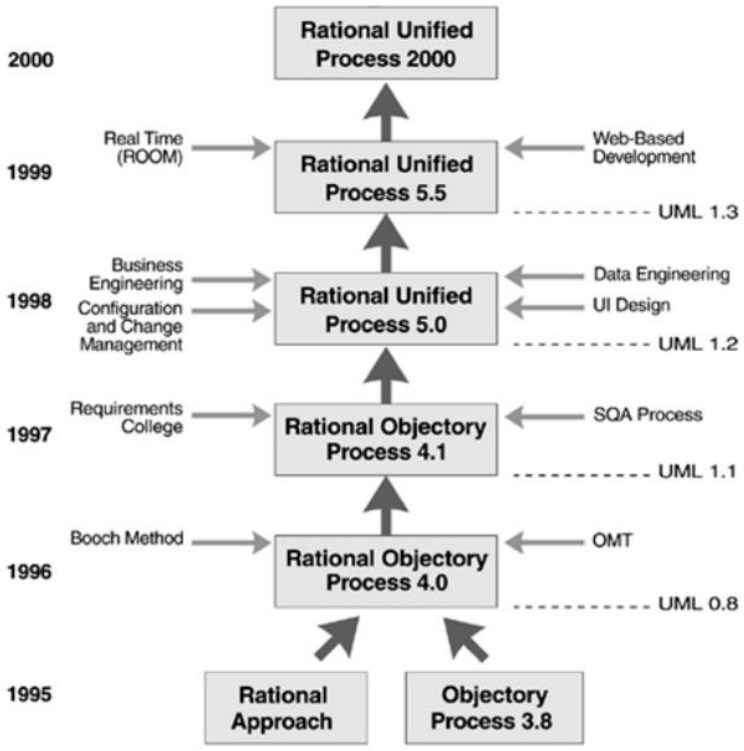
Bruckner (2012, str. 112) dále uvádí, že prvním milníkem ve změně strategie společnosti Rational Software na konci 80. let, který byl způsoben trendy na obchodních trzích, kdy
se značkové hardwarové platformy dostávaly do ústraní kvůli hardwaru s otevřenými systémy. Jejich zastaralost společně s vysokou cenou nemohly tomuto novému segmentu konkurovat. Otevřené platformy navíc zajišťovaly více operačního výkonu, který byl využit u rostoucího trhu osobních počítačů.

Jako druhý milník Bruckner (2012, str. 114) označuje enormní růst aktivit na komerčním trhu, kde byl z jedním preferovaných jazyků C++ i z důvodu zpoždění modernizace jazyku Ada. Ten byl v současné době příliš obsáhlý a komplikovaný, tudíž obtížně použitelný v moderním vývoji softwaru. Modernizace jazyku Ada proběhla až v roce 1995, což zapříčinilo skutečnost, proč nyní není bežným programovacím jazykem. Je však používán v odvětví, kde nejsou chyby akceptovatelné a bezchybná funkčnost aplikací je prvořadým cílem nehledě na zvýšené nároky a zdroje při vývoji. Jak již bylo řečeno, jedná se například o projekty Ministerstva obrany USA, NASA, případně společnosti vyvíjející satelitní a letové systémy. Po rozšíření dostupnosti webových stránek došlo k rozsáhlé prezentaci tisíců obchodních společností na internetu. Možnost této prezentace přispělo k tvorbě prvních webových aplikací, především pak eshopů jak je známe nyní. Tyto změny poskytly organiazci Rational Software nové možnosti, jakým způsobem maximalizovat zisk v oblasti nástrojů pro vývoj softwaru. Z toho důvodu společnost provedla několik investic a zakoupila sadu produktů, které umožňovaly vytvořit ucelený nástroj pro analýzu, návrh, podporu managmentu, testování a nástroj umožňující automatickou dokumentaci. Souběžně byl společností vytyčen cíl vytvořit jednotnou a především standardizovanou metodiku pro modelování informačních systémů. Jak uvádí Fowler (2009, str. 30), v počátcích 90. let. 20 byl však trh nejednotný a bylo náročné vyvinout nástroj, který by vyhovoval jednotlivých standardům pro vývoj softwaru. Proto Rational Software zaměstnala Grady Boocha, James Rumbaugha a Ivar Jacobsona, kterým byl svěřen konkrétní

cíl; vývoj jednoho univerzálního modelovacího jazyka. Tento cíl se „Třem amigos“, jak se o nich obecně hovoří, podařilo splnit a vyvinuli jazyk Unified Modeling Language, v IT branži známého jako UML.

Fowler (2009, str. 31) dále uvádí, že společnost současně při tvorbě UML vyvíjela metodiku, která by obsahovala množinu pravidel při vývoji a implementaci informačního systému a současně by byla kompatibilní s nástroji organizace Rational Software. Toto je oficiální počátek zrodu metodiky Rational Unified Process viz Obrázek 4.

**Obrázek 4 Evoluce metodiky Rational Unified Process**




*Zdroj: Anwar (2014, str.10)*

Samotná metodika je nástupcem, respektive vznikla sloučením přístupu Rational a metodiky Objectory Process. Původní název byl Rational Objectory Process, ale došlo k jejímu přejmenování na Rational Unified Process v roce 1998. Počínaje rokem 2003 byla Rational Software odkoupena americkou mezinárodní technologickou společností International Business Machines, známější pod zkratkou IBM. Metodika RUP byla od této obchodní transakce doplněna o způsoby agilního vedení projektu a je nadále rozvíjena tak, aby dokázala obstát v prostředí metodik zaměřující se na agilní vývoj.

### 2.5.2 Šest nejlepších praktik pro vývoj softwaru

Jak uvádí Anwar (2014, str. 14), metodika Rational Unified Process byla navrhnuta dle mnohaletých zkušeností vývojářů, kteří se v rámci své kariéry specializovali na vývoj komplexních informačních systémů. Díky praxí osvědčeným postupům a metodám, které se v metodice nacházejí, je dosaženo zlepšení komunikace mezi dodavatelským týmem a zadavatelem. Ke zlepšení komunikace a produktivity dochází i v rámci týmu dodavatele, což zrychluje a zefektivňuje samotný proces vývoje.

Mezi šest praktik (též nazývanými Best Practices), které jsou v rámci metodiky Rational Unified Process podporovány nástroji pro automatizaci specifických procesů, patří:

- iterativní vývoj softwaru;
- správa a řízení požadavků;
- použití komponentové architektury;
- vizuální modelování softwaru;
- průběžné kontrolování kvality;
- řízení změn.

**Iterativní vývoj softwaru**

Rational Software (2011, str. 1) uvádí, že současná náročnost projektů neumožňuje při vývoji softwaru postupovat sekvenční metodou tj. určit primární problém, navrhnou jeho řešení, úspěšně implementovat software a v poslední fázi jej testovat. Některé chyby lze nalézt až později, kdy je jejich samotná oprava násobně nákladnější, než kdyby k jejich odhalení a následné opravě došlo v rané fázi projektu. Dále se ve většině případů úvodní návrh zabývá pouze hlavními požadavky a to způsobuje budoucí odhalení nepřesností, nutnost jejich opravy a následné navýšení rozpočtu. I proto je při vývoji nevhodné používat vodopádový cyklus. Je tomu tak z důvodu, že není možné v počáteční fázi odhalit reálná rizika projektu. Vhodnou úpravou vodopádového cyklu je jeho změna na iterativní, kdy dochází k průběžnému zlepšování vyvíjeného softwaru. V jednotlivých iteracích, především pak na počátku projektu, dochází k odhalování jednotlivých rizik a díky tomu je na ně možné včasně reagovat. Tímto krokem zajistíme, že z jejich strany nedojde k budoucímu napadení projektu.

Jak uvádí Anwar (2014, str. 16), jednotlivé iterace se podobají vodopádovému modelu a skládají se z následujících bodů:

- plánování cíle iterace (funkčnost);
- doplnění / zpřesnění požadavků;
- dotváření návrhu;
- implementace přírůstku funkčností;
- integrace přírůstku – ověření, otestování;
- předání do provozu (interního / externího).

Anwar (2014, str. 16) dále uvádí, že doporučený počet iterací závisí na rozsahu projektu a velikosti týmu. Jejich počet se též odlišuje dle aktuální fáze vývoje. Doporučeny jsou však alespoň 3 iterace, která mají pevné datum ukončení. Při počátku každé iterace je nezbytné provést aktualizaci jednotlivých rizik, které byla v předchozí iteraci zjištěna a dle aktuální situace ohodnotit jejich aktuálnost a rozsah dopadu. Pro celkovou stabilitu projektu je nutné probíhající iterace uzavřít před změnami zvenčí. Délka jednotlivých iterací je určena podle velikosti projektů, kdy pro malé projekty je doporučeno na 1 - 4 týdny, pro velké projekty 3-6 týdnů a zřídka kdy se vyskytuje doba v řádech měsíců. Je to z toho důvodu, že lidé si pamatují překročené termíny, ne opuštěné vlastnosti a to je přiměje k včasnému provedení rozhodnutí a těžkých kompromisů. V rámci iterace je možné provést omezení plánované funkčnosti, co je však neakceptovatelné, je neprovést plánovaný release[1], změnit jeho datum nebo v rámci dodržení termínu nařídit přesčasy. Poslední uvedený důvod není v dlouhodobém rámci udržitelný a v krajních případech může dojít k fluktuaci či vyhoření vývojářů v týmu.

[1] Jedná se o vydání další verze aplikace, která obsahuje nové funkčnosti v rámci iterace

V rámci otestování při ukončení každé iterace dojde k objektivnímu ohodnocení projektu a lze odhalit případný nesoulad mezi schválenými požadavky a provedenou implementací. Pro maximalizování účinku fáze testování se doporučuje neustálé nasazení testovacího týmu, který díky tomu získává praktické zkušenosti z jednotlivých fází a postupně se zvyšuje jeho efektivnost.

### Správa a řízení požadavků

Anwar (2014, str. 16), specifikuje požadavek jako schopnost nebo vlastnost, kterou musí vyvíjený systém naplňovat. Požadavky se řídí na několika úrovních a to:

- business požadavky;
- uživatelské požadavky;
- funkční požadavky.

Bruckner (2012, str. 38) uvádí, že business požadavky přicházejí na přání zákazníků, které obchodní oddělení obdrží v rámci průzkumu nebo pomocí zpětné vazby, u mobilních zařízení např. v App Store nebo Google Play. Tyto požadavky jsou též vystavovány od business ownera, který má ve společnosti na starost vizi jednotlivých produktů. Jejich hlavní úlohou je objasnit co a proč se má realizovat. Uživatelské požadavky jsou upřesnění business požadavků a vyjadřují, co uživatel očekává od implementované části systému. Funkční požadavky naopak říkají, jakým způsobem dojde k jejich realizaci. Správa požadavků využívá pro jejich prioritizaci, utřídění, zdokumentování jednotlivých funkcionalit a v neposlední řadě pro sledování a zaznamenání dohod a rozhodnutí mezi zadavatelem a dodavatelem. Umožňuje nalézt nekonzistenci a v případě použití vhodných vývojových nástrojů uchovat všechny systémové požadavky včetně přiložených externích dokumentů. V rámci velkých a obsáhlých projektů je změna požadavků při vývoji mnohdy nevyhnutelná. Anwar zmiňuje (2014, str. 17), že je možné během prvních iterací zjistit, zdali produkt splňuje jednotlivé požadavky. Výhodou rozdělení vývoje do jednotlivých iterací je i rozsah požadavků, který je oproti celkovému projektu menší a je jednodušší na změny reagovat. Je nepravděpodobné, že požadavky získané na počátku budou shodné a vyhovující i na konci projektu. Způsobuje to i představa zadavatele, která se v průběhu projektu mění i dle vydaných konkurenčních aplikací. Proto je nutné si uvědomit, že trh v oblasti vývoje softwaru je dynamický a mnohdy se stává, že vyvíjený sofwtware je již v době uvedení na trh zaostalý oproti konkurenčnímu softwaru, který byl uveden na trh v nedávné minulosti. V mnoha případech nejsou zadavatelé schopni explicitně popsat, co od jednotlivých požadavků očekávají. Pokud však dostanou k dispozici část vyvíjeného systému, například dashboard platební aplikace, dokáží lépe popsat, jaké funkčnosti od něj očekávají a co je z jejich pohledu naopak přebytečné.

### Použití komponentové architektury založené na použití komponentů

Při vývoji systému a tvorbě softwarové dokumentace informačního systému je vyžadován pohled z více perspektiv. Je to především z důvodu, že se na projektu podílí několik rolí a každá z nich na systém pohlíží z jiného úhlu, který se během životního cyklu dále mění. Jak uvádí Eeles (2011, str. 24) architektura systému je pravděpodobně nejdůležitější částí projektu, která určuje jeho budoucí směr. Architektura informačního systému je žádaným komunikačním prostředkem managementu podniku s jednotlivými softwarovými architekty, v softwarovém inženýrství označovanými jako enterprise architekty. Je vyžadováno, aby byla názorná, srozumitelná, jednoduchá a slouží k vzájemnému pochopení investora, řešitele a uživatele. Architektura se též označuje za nástroj systémové integrace.

Eeles (2011, str. 86) dále uvádí, že mezi hlavními požadavky na architekturu se řadí opakované přepoužití konceptů, prvků a struktur při rozšiřování systému. Jednou z těchto technologických architektur je proto využívaná SOA, která má za úkol zjednodušit monolitický software na soustavu samostatných, leč propojených komponent. To spočívá v rozdělení složitějších operací a služeb na drobnější celky a zajistí tím jejich opětovné použití. Je tedy zřejmé, že tento způsob má zásadní vliv na menší finanční dopady. Další používanou architekturou je MDA, neboli Model Driven Architecture, která se primárně zaměřuje na vzájemné oddělení popisu business procesů a aplikační logiky od technologické platformy. Součástí architektury je též určení formátu pro výměnu dat, mezi které se řadí komerční nástroj UML.

Dílčí architekturu lze rozdělit následovně:

- technologická architektura;
- hardwarová architektura;
- softwarová architektura;
- datová architektura;
- procesní architektura.

Jak uvádí Rational Software (2011, str. 2), hlavním účelem technické architektury je systematické řízení rozvoje a provozu IT infrastruktury. To umožňuje dosáhnout odpovídající stability v IT infrastruktuře při jejím rozšiřování a to s akceptováním vzájemných vazeb. Hardwarová architektura následně určuje typy hardwarových komponent a vzájemné vazby mezi nimi. Oproti tomu se softwarová architektura zaměřuje na užití jednotlivých softwarových komponent a především na to, z jakých komponent bude systém tvořen. Na základě datové architektury se uskutečňuje návrh datových entit včetně jejich vazeb a atributů. Představuje rozvržení datových zdrojů, které společnost musí vlastnit, aby dosáhla naplánovaných cílů a potřeb. Procesní architektura poté slouží pro plánování budoucího stavu procesů v podniku. Tyto procesy jsou zachyceny např. pomocí data flow diagramů nebo síťových grafů.

**Vizuální modelování softwaru**

Důvodem pro využití vizuálního modelování je možnost tvorby menších a lépe pochopitelných modelů, které vyjadřují jednotlivé části systému. Předpokládaným výstupem je pro člověka zjednodušený a lépe pochopitelný návrh dodávaného systému, kdy jednotlivé části lze prezentovat v rámci schvalování architektury. Metodika RUP používá pro modelování standardizovaný modelovací jazyk UML. Fowler (2009, str. 23) tento jazyk charakteriuje jako *„Sjednocený modelovací jazyk je druh grafické notace podporovaný nezávislým meta-modelem, který umožňuje popisovat a navrhovat softwarové systémy, konkrétně systémy budované využitím objektovaně orientované metodiky.“* Díky využití tohoto jazyku v metodice RUP dochází k nezkreslené vzájemné komunikaci v rámci týmů a následně i při předávání dokumentace zákazníkovi.

Fowler (2009, str. 23) dále uvádí, že Unified Model Language se skládá především z těchto částí:

- stavební bloky – jedná se o základní prvky modelu, vazby a diagramy;
- společné mechanismy – obecné způsoby jazyku UML, pomocí kterých se dosahuje cílů;
- architektura – modelová vizualizace architektury vyvíjeného systému.

Anwar (2014, str. 9) uvádí, že UML nelze považovat za metodiku. Samotný jazyk nepředepisuje jednotlivé kroky při vývoji softwaru, ani jakým způsobem se má postupovat.

Avšak v případě sloučení praktik modelování a iterativního vývoje softwaru docházíme k závěru, že toto spojení přispívá k vyzdvihnutí a ohodnocení změn architektury a o zaznamenaných změnách je možné dále možné diskutovat na úrovni projektového či vývojového týmu. V případě zvolení adekvátních modelovacích nástrojů je možné provést v každé iteraci synchronizování jednotlivých diagramů a zdojových kódů. Díky těmto nástrojům je umožněna jednodušší a přehlednější správa modelů včetně funkcionalit jako i vyzdvihnutí určitých podrobností nebo naopak skrytí až příliš velkého detailu. Modelování dále pomáhá vývojovému týmu se složitostí systému díky tomu, že umožňuje udržovat konzistenci mezi artefakty systému. Mezi nejvyužívanější diagramy lze zařadit diagramy případů užití (Use Case), stavové diagramy a diagramy tříd. K největším výhodám modelování se řadí možnost specifikovat chování systému pomocí jasně definovaných scénářů a jednotlivých modelů případů užití. Ostatní modely dále zachycují návrh vyvíjeného softwaru, při němž je zachycena nemodulární a neměnitelná architektura. Při obsahově náročnějším projektu je poté možné lépe nalézt případné nekonzistence.

**Průběžné kontrolování kvality**

Jak ve své publikaci uvádí Bureš (2016, str. 16), defekt, který nalezeneme v produkci je v průměru 12,5 krát vyšší než finanční náklady, pokud by k jeho opravě došlo již v rámci vývojářských testů. Extrémním případem jsou například americké aerolinky, kde u modelu Boeing 787 Dreamliner byla v roce 2015 v produkci objevena chyba, u které pouze workaround provozovatel aerolinek vyčíslil na celkových 22 865 USD. Defekt se týkal přetečením rozsahu počítadla, ke kterému docházelo po 248 dnech nepřetržitého provozu. To způsobilo pád systému do nouzového režimu a ohrožení cestujících. Workaround spočíval v restartování systému servisním technikem, který má hodinovou sazbu 85 USD za hodinu. Toto muselo být provedeno u všech strojů, aby společnost získala čas pro implementaci opravy. Oprava samotného kódu včetně následných regresních a zátěžových testů byla odhadnuta na několik milionů dolarů. Nehledě na nepřímé finanční dopady, které mohou být až o řád či dva vyšší.

Bureš (2016, str. 41) uvádí, pro každý podstatný scénář je nezbytné vytvořit sadu testů, které zajistí validní kontrolu dané funkcionality systému. V následných krocích dochází ke zjištění, které scénáře při testování neuspěly, a proběhne analyzování důvodů selhání. V případě metodiky RUP, testy probíhají opakovaně v jednotlivých iteracích stejně jako samotný vývoj softwaru. Je však nezbytné vědět, jakým konkrétním způsobem budou požadavky ověřeny. K tomu v rámci testování slouží plán řízení kvality.

Nedílnou součástí tohoto plánu je plán testování, který se skládá z následujících bodů:

- cíle a požadavky testování;
- oblasti určené pro testování;
- kategorie testů;
- seznam testovacích případů;
- definice rizik pro testování;
- požadavky na testovací data.

Morgan (2010, str. 17) dále doplňuje, že tento systém testování je následně možné označit za průběžné kontrolování kvality. Autor označuje jako hlavní přínosy kontroly kvality možnost objektivně ohodnotit stav projektu na základě již provedených testů. Mezi další nesporné výhody lze označit zvýšení kvality vyvíjeného softwaru díky testování, které se zaměřuje na oblasti s největším rizikem chyb. Ty jsou během jednotlivých iterací nalezeny zavčas a to následně kladně ovlivňuje finance vynaložené na vývoj softwaru. To lze dále podpořit automatickým testováním, které zajišťuje testování spolehlivosti a výkonosti.

**Řízení změn**

Změny jako takové jsou v průběhu projektu nevyhnutelné ať ve formě změnových požadavků, které se následně projevují ve funkcionalitě vyvíjeného softwaru nebo v rámci rozčlenění vývojářů do týmů. Ty v dnešních podmínkách fungují z různým míst, mnohdy i v různých časových pásmech a přesto jsou schopni spolupracovat na stejných iteracích. Jak uvádí Anwar (2014, str. 20), disciplína změnového managementu je v metodice Rational Unified Process dělena do specifických oblastí. Jedná se o management konfigurace a změnový management. Management konfigurace poskytuje firmě nástroje, které ověří, že její produkt odpovídá s požadovanými potřebami. Jde se to tyto nástroje:

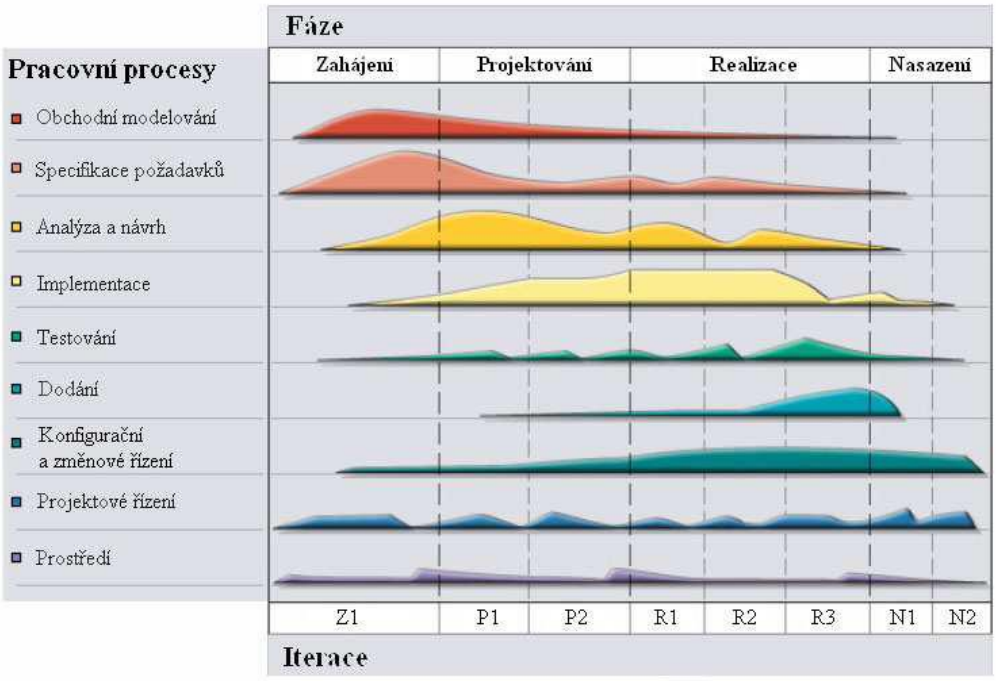
- sled kroků a postupů pro identifikaci konfigurace produktu;
- sled kroků a postupů pro řízení změn v jednotlivých etapách životního cyklu;
- sled kroků a postupů pro vykazování stavu konfigurace produktu;
- sled kroků a postupů pro přezkoumání konfigurace ve stanoveném milníku realizace.

Anwar (2014, str. 20) dále popisuje cíl změnového managementu v oblasti vývoje softwaru jako řádné verzování produktu se zaznamenáním veškerých implementačních změn, které byly provedeny. Řízení změn jako celek definuje pracovní proces změny požadavků a umožňuje jeho opakovatelnost. Díky tomu mají změnové požadavky určitý řád a dochází k ulehčení a zpřesnění komunikace.

### 2.5.3 Fáze životní cyklu

Jak uvádí Bruckner (2012, str. 113), životní cyklus, který obsahuje Metodika Rational Unified Process, je rozdělen do čtyř fází, jak je uvedeno na Obrázku 5. Fáze zahájení (Inception), fáze projektování (Elaboration), fáze realizace (Construction) a fáze nasazení (Transition).

**Obrázek 5 Fáze a disciplíny Rational Unified Process**




*Zdroj: Bruckner (2012, str. 114)*

Fáze obsahují rozdílné zaměření a tím se následně ovlivňuje náplň jednotlivých iterací. Na Obrázku 5 je patrný postup projektu v čase, který je znázorněn na horizontální ose. Pracovní

procesy jsou následně zachyceny na ose vertikální a společně s nimi je vyobrazen vývoj aktivit, které je nutné uskutečnit pro jednotlivé činnosti v rámci životního cyklu projektu.

**Fáze zahájení**

Tato fáze se zaměřuje především na úspěšný start projektu. Anwar (2014, str. 12) ve své publikaci následně upřesňuje její účel a cíle. Označuje ji za hlavní milník projektu, jelikož na konci této fáze by mělo být rozhodnuto, zdali se v projektu bude pokračovat nebo jeho vlastník dospěl k názoru, že projekt bude změněn či zrušen. Výstupem fáze jsou tyto skutečnosti:

- zúčastněné strany (stakeholders) by měly souhlasit s informacemi, které jim jsou projektovým vedením představovány;
- požadavky jednotlivých dotčených stran by měly být zcela pochopeny;
- odhad nákladů, harmonogram projektu a rizika projektu by měla být hodnověrná a schválená zákazníkem;
- je připraven dokument o vizi projektu a dokument obsahující jednotlivé požadavky projektu včetně jejich klíčových vlastností a hlavních omezení;
- měl by být připraven počáteční model případu použití (přibližně 10 % - 20 %);
- měl by být stanoven základní bod, který lze použít k porovnání skutečných hodnot výdajů oproti původně plánovaným výdajům.

Bruckner (2012, str. 113) dále uvádí, že iterace ve fázi zahájení projektu jsou obtížná a mohou mít formu ověřování, případně výzkumu. Není tedy možné garantovat úspěšné zakončení. Proto je doporučováno v počáteční fázi vytvořit model nebo jednoduchý prototyp, pomocí kterého dojde k ověření, zda je reálné s vybranými nástroji a pomocí zvolené technologie hlavní požadavky splnit. Artefakty, které jsou v tomto procesu vytvořeny, jsou nezřídka jednorázového charakteru a po použití zahozeny.

**Fáze projektování**

Ve fázi projektování je primárním cílem zvolit a validovat architekturu systému, který bude v projektu dále vyvíjen. Jak ve své publikaci uvádí Rational Software (2011, str. 4), hlavním cílem je analýza problémů jednotlivých domén, tvorba projektového plánu a eliminování největších rizikových elementů nacházejících se v projektu. Jako další aktivita, prováděna ve fázi projektování, je zaměření se na funkční, nefunkční a změnové požadavky, které se dotýkají architektury systému a mohou ji ovlivnit. Dle typů požadavků si musíme položit například tyto otázky:

- jakým způsobem je dimenzovaný navrhovaný systém a kolik uživatelů dokáže současně obsloužit;
- je navrhovaný systém schopen zpracovávat a odpovídat v požadovaném čase dle zadaných požadavků;
- jaké požadavky jsou kladeny na funkčnost a spolehlivost vyvíjeného systému? Je zajištěno SLA;
- je na vyvíjeném systému závislý lidský život.

Bruckner (2012, str. 113) dále uvádí, že realizovatelnost a udržitelnost architektury systému pomáhají ověřit jednotlivé iterace, které jsou prováděny ve fázi projektování. Je důrazně doporučováno, u některých typů projektů přímo vyžadováno, aby byl v rámci této fáze vytvořen prototyp, který umožní potvrzení všech použitých architektonických principů a poskytne

upřesnění pro návrh realizace systému. Pokud by nastala situace, kdy navrhovaná architektura nemůže vyhovět dodatečným požadavkům, systém následně nesplní očekávání, které byly na počátku projektu stanoveny. Zároveň by byla nepodstatná dosažená kvalita vývoje v následujících fázích projektu. Anwar (2014, str. 12) jako výstup z této fáze považuje tyto skutečnosti:

- model případu použití. Případy použití a aktéři by měli být identifikováni a samotný model by měl být přibližně z 80 % kompletní;
- detailní popis softwarové architektury včetně spustitelného kódu, pomocí kterého je možné validovat funkčnost systému a samotné architektury;
- business požadavky a s nimi související rizika by měla být revidována;
- plán rozvoje celého projektu;
- prototyp, který snižuje technická rizika zjištěná v předchozí fázi.

Výsledky projektové fáze jsou následně podrobeny analýze a poté jsou vyhodnoceny. Anwar (2014, str. 12) dále uvádí, že v případě nesplnění tohoto druhého důležitého milníku dochází ke zrušení projektu, změně návrhu v architektuře nebo úpravě problémových požadavků způsobující neshodu. V případě, kdy výstupní hodnoty vykazují uspokojivé výsledky, dochází k přechodu projektu do fáze realizace.

**Fáze realizace**

Fáze realizace se zaměřuje na tvorbu operačního systému dle navržené architektury, která byla vytvořena a odsouhlasena v předchozí fázi. Rational Software (2011, str. 6) uvádí, že se především zabývá vývojem jednotlivých komponent systému a funkcí s nimi spjatými. To způsobuje spotřebu většiny časové kapacity a finančních zdrojů projektu právě ve fázi realizace. Hlavním cílem je na konci jednotlivých iterací provádět implementace nových funkcí, které je nutné v průběhu každé iterace otestovat nehledě na aktuální fázi. Je tedy žádané začít testování nové funkčnosti ihned po napsání kódu. To vyžaduje úzkou součinnost vývojářských a testerských týmů v projektu. Následné chyby a nedodělky, které odporují odsouhlaseným požadavkům, jsou zaznamenány ve formě úkolů společně s jejich prioritou. Díky tomu lze seznam těchto chyb prioritizovat a v rámci současné iterace provést opravu těch, které jsou pro projekt nejméně příznivé neboli blokační. Výstupem každé iterace musí být funkční a stabilní kód. Proto je možné chyby s nižší prioritou, které neovlivňují jeho spuštění a provoz, přesunout k opravě do následující iterace. Anwar (2014, str. 13) specifikuje výstupy fáze realizace jako:

- vytvořený / aktualizovaný uživatelský manuál;
- podrobný popis nové verze aplikace včetně změn, které implementuje;
- funkční a otestovanou verzi aplikace s novými funkcemi.

**Fáze nasazení**

Bruckner (2012, str. 114) ve své publikaci popisuje cíl fáze nasazení jako „*zajištění, aby uživatelé mohli systém používat*“. Bruckner (2012, str. 114) dále upřesňuje, že proto se iterace, nacházející se ve fázi nasazení, zaměřují především na opravu chyb, nastavení konfigurace jednotlivých systému a případně implementaci drobných vylepšení v uživatelském rozhraní. Rational Software (2011, str. 6) dále doplňuje, že je vhodné nové funkčnosti představit uživatelům v rámci beta testování nebo případně v produkční pilotní verzi. Tato skupina obsahuje segment seniorních uživatelů, pracující v dodavatelské či odběratelské společnosti. U určitých typů softwaru je nezbytné zajistit podrobný monitoring systému, aby bylo možné

případné incidenty v rámci produkčního prostředí flexibilně opravit, případně provést rollback neboli stažení aktuálně vydané verze kódu.

### 2.5.4 Role, aktivity a artefakty

Model metodiky popisuje, jaká osoba má daný proces provést, co v procesu tvoří, jakým způsobem dosáhne výsledku a v jaké časové fázi projektu. Eeles (2011 str. 50) uvádí shrnutí, že *„v podstatě lze říci, že každá efektivní metoda vývoje softwaru by měla stanovovat kdo, kdy, jak a co dělá.“* Eeles (2011 str. 50) poté upřesňuje, že Rational Unified Process je tedy reprezentován pomocí čtyř primárních modelových prvků:

- role: kdo;
- aktivity: jak;
- proces, iterace a činnosti: kdy;
- artefakt: co.

Následující odstavce se zaměřují na jednotlivé základní elementy metodiky, kterými je výše zmíněná role, aktivita, artefakt. Na samotný proces je z důvodu obsáhlosti zaměřena nezávislá kapitola.

#### Role

Eeles (2011 str. 52) ve své publikaci roli definuje jako „*odpovědnosti jednotlivce či skupiny jednotlivců, pracujících společně jako tým v rámci kontextu organizace, zabývající se vývojem softwaru*“. Eeles (2011 str. 52) definici vysvětluje tím, že za úkol případně sérii úkolů a jejich výsledek odpovídá daná role. Příkladem lze uvést roli business analytika informačních technologií, který má za úkol zjistit od jednotlivých zainteresovaných osob, neboli stakeholderů, požadované chování a vlastnosti implementovaného systému. Z těchto informací následně zpracuje seznam business požadavků, které definují popsané chování stakeholderů, jsou však rozpadnuté do jednodušší formy vhodné pro zpracování analytikem informačních technologií, případně pro finální zaznamenání do analytické dokumentace a následnému předání do vývoje. Zpracování této analytické dokumentace zajišťuje IT analytik. Pro odstínění technického řešení od očekávání zadavatele je vhodné do těchto rolí obsadit rozdílné osoby. Toto rozdělení obsahuje výhodu v přesně definovaném zadání pro IT analytika, které je možné upravit pouze na základě změnového požadavku na úpravu obchodních požadavků. Samotná metodika Rational Unified Process obsahuje celkem třicet jedna rolí. Seznam těchto rolí je uveden níže. Z důvodu všeobecné známosti těchto rolí v oblasti IT a v samotném procesu vývoje softwaru, není proveden překlad z anglického jazyka kvůli možné záměně významu:

- analytické role - Business-Process, Analyst Business Designer, Business-Model Reviewer, Requirements Reviewer, System Analyst, Use-Case Specifier,
- developerské role – Architect, Architect Reviewer, Capsule Designer, Code Reviewer, Database Designer, Design Reviewer, Designer Implementer, Integrator;
- testerské role - Test Designer, Tester;
- manažerské role - Change Control Manager, Configuration Manager, Deployment Manager, Process Engineer, Project Manager, Project Reviewer;
- doplňkové role - Any Worker, Course Developer, Graphic Artist, Stakeholder, System Administrator, Technical Writer, Tool Specialist.

Rational Software (2011, str. 8) doplňuje, že jednotlivé role samy o sobě nedefinují konkrétní osobu. V rámci jedné role se může nacházet více pracovníků, stejně tak jedna osoba se může

na projektu nacházet ve více rolích. Rolí tudíž není možné označit konkrétní osobu, ale popisuje chování a odpovědnost. Jaké osoby budou v jednotlivých rolích, určuje projektový manažer daného projektu. Jakým způsobem se následně mají chovat osoby dosazené do rolí, jsou definovány v aktivitě, za kterou je daná role odpovědná. Neplatí, že do jednotlivých rolích jsou obsazovány osoby pouze z vývojového týmu. Dochází k tomu tak například v případě outsourcingu vývoje softwaru externí společností.

### Aktivita

Anwar (2014, str. 14) definuje ve své publikaci aktivitu jako „*Činnost konkrétního pracovníka, která se označuje jako pracovní jednotka, v rámci které může být požádáno pracovní výkon provést. Činnost má jasný účel, obvykle vyjádřený z hlediska vytváření nebo aktualizace některého artefaktu, jako je model, třída nebo plán.*“ Rational Software (2011, str. 8) dále upřesňuje, že každá aktivita je vždy přiřazena konkrétnímu pracovníkovi nebo skupině pracovníků. Doporučuje se však mít takovou granularitu aktivit, aby bylo možnost jejich přiřazení vždy konkrétní osobě, která za její vypracování přebírá odpovědnost. Granularitou aktivity lze chápat složitost aktivity vázaný na čas, který je nutný pro její zpracování. Pro jednotlivé aktivity se doporučuje dodržet granuaritu od několika hodin do pár dní. V případě přiřazení aktivity jedinému pracovníkovi je nezbytné, aby daný úkol obsahoval pouze malý počet artefaktů, které budou činností ovlivněny. V rámci vývoje je možné vybranou aktivitu vykonat jedenkrát, ale mohou nastat případy, kdy je nutné její provedení opakovat. Pokud k tomu dojde, jedná se o opakovanou aktivitu. K tomu dochází při přechodu mezi jednotlivými iteracemi, kde jsou artefakty vytvářeny a dle potřeby dochází k jejich upřesnění a modifikaci během iterace. Jako příklady aktivit a rolí k nim přiřazených lze uvést následují:

- aktivita: vytvoření plánu iterací –> Role: projektový manažer;
- aktivita: tvorba architektury systému –> Role: solution architect;
- aktivita: definice business požadavků –> Role: business analytik;
- aktivita: definice případu užití a aktérů –> Role: IT analytik;
- aktivita: provedení performačních testů –> Role: tester.

### Artefakt

V rámci metodiky Rational Unified Process je artefakt nezanedbatelným pojmem. Rational Software (2011, str. 15) jej definuje jako „*informaci, která je vytvořena, modifikována nebo následně použita v procesu*“. Každý artefakt má v jednotlivých fázích vývoje přidělenou roli, která za něj přebírá odpovědnost. Artefakty jako takové vznikají napříč celým vývojem a jsou používány pro úspěšné dokončení projektu. Anwar (2014, str. 15) dále artefakty označuje za „*vstupy nebo výstupy jednotlivých aktivit*“. Následně Anwar (2014, str. 15) doplňuje, že každý artefakt má jasně specifikovaný životní cyklus, délka těchto cyklů je dle typu artefaktu odlišná. Může docházet k modifikaci vybraných artefaktů v rámci každé iterace, nebo jsou naopak využity pouze jednou a následně dochází k jejich zahození či případné archivaci. Při modifikaci artefaktů v průběhu jednotlivých iterací nebo nasazení nesmí být opomínáno na jejich verzování. Díky tomu je možné jednotlivé verze artefaktů porovnávat či v případě nutnosti, například při neúspěšné iteraci současného vývoje a zahození provedené práce, prohlásit předchozí verzi artefaktu za aktuální. Určité typy artefaktů lze považovat za nezbytnou součást vývoje. Mezi ně se řadí například diagram případu užití či rozhraní služby, ať už je definováno v jakékoliv podobě. Z toho tedy vyplývá, že artefakt nelze jednoznačně označit pouze jako dokument. Může to být například model, diagram nebo segment zdrojového kódu. V průběhu projektu je vhodné dodržovat nepsané pravidlo „méně je více“ platící

pro množství vytvořených artefaktů, které v konečném důsledku mohou způsobit vznik nekonzistence oproti reálnému stavu. Obecně lze říci, že artefakty slouží k podpoře projektu a jeho zdárnému dokončení. Nemělo by být tedy cílem, aby projekt byl nástrojem pro neúměrné zvýšení počtu artefaktů.

### 2.5.5 Proces

Rational Software (2011, str. 9) pracovní procesy definuje jako *„sekvenci různorodých aktivit, které provádíme za účelem získání hodnotných a měřitelných výsledků“*. Anwar (2014, str. 15) dále upřesňuje, že tyto pracovní procesy jsou složeny z vícero aktivit. Tyto aktivity lze zaznamenat pomocí notace UML. K tomuto účelu se používají sekvenční diagramy, activity diagramy nebo digramy spolupráce. V mnoha případech jsou jednotlivé aktivity provázány ve větší míře, než je zaznamenáno na modelech. Proto je nutné si uvědomit, že není vždy reálné zaznamenat veškeré propojení a závislosti mezi jednotlivými aktivitami.

Rational Software (2011, str. 10) uvádí, že metodiky Rational Unified Process obsahuje devět hlavních pracovních procesů, které se dále rozdělují na hlavní inženýrské a hlavní podpůrné pracovní procesy.

Hlavní inženýrské pracovní procesy:

- business modelování (Business modeling workflow);
- specifikace požadavků (Requirements workflow);
- analýza a návrh (Analysis and Desing workflow);
- implementace (Implementation workflow);
- testování (Test workflow);
- dodání (Deployment workflow).

Hlavní podpůrné pracovní procesy:

- projektové řízení (Project management workflow);
- konfigurační a změnové řízení (Configuration and Change management workflow);
- prostředí (Enviroment workflow).

Rational Software (2011, str. 10) upřesňuje, že názvy hlavních šesti inženýrských pracovních procesů sice připomínají jednotlivé fáze u vodopádového modelu, ale rozdíl tkví v tom, že v průběhu životního cyklu jsou procesy pozměňovány a fáze nacházející se v jednotlivých iteracích též nejsou shodné. V reálném projektu tedy dochází opakovaně k využití těchto devíti hlavních procesů s rozdílným důrazem a intenzitou dle aktuální iterace.

#### Business modelování

Rational Software (2011, str. 10) označuje jako jeden z primárních problémů při vývoji softwaru problémovou komunikaci mezi dodavatelem softwaru a zákazníkem. V krajních případech to může způsobit situaci, kdy na konci vývoje je implementovaný software pro zákazníka nepoužitelný nebo nesplňuje některá podstatná kritéria. Proto je vhodné využívat nástroje metodiky Rational Unified Process oběma stranami tak, aby se vztah mezi zákazníkem a dodavatel přibližoval synergickému efektu. Anwar (2014, str. 16) doplňuje, že business modelování vysvětluje, jakým způsobem popsat vizi společnosti, ve které bude vyvíjený systém používán a jakým způsobem následně využít tuto vizi jako podklad pro nastínění role, procesu a odpovědnosti. Jak uvádí International institute of business analysis (2015, str. 58) dokument, který je tímto modelováním vytvořen nazýváme jako obchodní případy užití. Tento způsob

prezentace zajišťuje, aby všechny zúčastněné strany napříč organizací porozuměly, jaké obchodní procesy je žádáno ve společnosti podporovat. Obchodní případy užití jsou dále analyzovány, aby došlo k pochopení, jakým způsobem má podnik podporovat jednotlivé obchodní procesy, částečně také pomocí nejlepších praktik pro vývoj softwaru. Tato podpora je následně dokumentována pomocí diagramů, zaměřující se na objektový model. Nicméně mnoho projektů se může uchýlit k tomu, že podnikové obchodní modelování nebude vůbec využívat. Jedná se tak především o menší společnosti.

**Specifikace požadavků**

Anwar (2014, str. 16) uvádí, že tato disciplína popisuje, jakým způsobem získat od stakeholderů, neboli zainteresových osob, jejich potřeby a požadavky na produkt. Následně dojde k jejich přeměně do sady detailně vypracovaných obchodních požadavků o nízké granularitě, které společně popisují vše, co se od systému očekává a musí splňovat. Tyto požadavky dále obsahují informaci o akceptačním kritériu, dle kterého bude rozhodnuto, zdali byl požadavek splněn či nikoliv. Dále je vhodné u každého požadavku uvést, jak velmi je pro zadavatele kritický a závažnost dopadu při jeho nesplnění. Rational Software (2011, str. 11) dále doplňuje, že cílem procesu získávání požadavků je explicitně popsat, co by měl systém dělat a umožňuje tím dosáhnout shody mezi vývojáři a zákazníkem u jednotlivých požadavků. Abychom této shody dosáhli, je nutné pravidelně organizovat schůzky k tomuto tématu a dokumentovat požadované funkce včetně veškerých omezení. Současně je nutné tyto změny sledovat a pravidelně zaznamenávat veškeré rozhodnutí a schválené kompromisy. Jako výstup tohoto procesu se očekává dokument, který obsahuje specifikaci všech požadavků. Souhrnně a přesně vyjadřuje účel nově vznikajícího systému, případně důvody jeho úpravy. Dokument jako takový je také stěžejním popisem funkčností systému pro skupiny jak na straně zákazníka (stakeholdery), tak i na straně dodavatele (analytiky či vývojáře). Obsahuje identifikaci všech aktérů, představující uživatele případně jiné systémy, které interagují s právě vyvíjeným systémem. Buchalcevová (2018, str. 85) uvádí, že v průběhu specifikace požadavků jsou tvořeny use case modely neboli modely případů užití. Tyto modely se zaměřují na potřebu z pohledu aktéra, díky čemuž je systém následně více relevantní pro budoucího uživatele. Každý z těchto případů užití musí být detailně popsán a je nutné definovat, jakým způsobem systém interaguje s aktérem a co přesně vykonává. Každý z modelů je následně schválen s obchodním vlastníkem systému a využit v následujících iteracích při IT analýze a vytváření testovacích scénářů.

**Analýza a návrh**

Rational Software (2011, str. 12) popisuje cíl této fáze ukázat, jakým způsobem bude probíhat implementace systému na základě požadavků, které byly vytvořeny v předchozí fázi. Analýza a návrh dále slouží pro vytvoření analytické dokumentace a návrhovému modelu, který se též označuje jako logický model nebo rozhraní služby. Vyvíjený systém musí umět provádět veškeré úkoly a obsahovat funkce, popsané v jednotlivých modelech případů užití. Dále se očekává, že splní popsané požadavky a bude v co největší míře robustní. Anwar (2014, str. 18) doplňuje, že samotný návrhový model je složen z návrhových tříd, které jsou uspořádané v rámci návrhových balíčků. Hlavním požadavkem pro tyto balíčky je jejich přesně definované rozhraní. Podstatnou částí návrhu je i zaměření se na tvorbu a validaci architektury. Tato činnost probíhá především během prvních návrhových iterací. Eeles (2011 str. 81) doplňuje, že architektura je reprezentována řadou architektonických zobrazení. Tyto pohledy na architekturu, v branži IT nazývanými jako big picture, pomáhají zachytit hlavní konstrukční rozhodnutí. Tyto architektonické pohledy lze v podstatě pojmout jako zjednodušení návrhu, u kterého jsou zviditelněny důležité vlastnosti a naopak detaily jsou

ponechány stranou. Architektura je podstatným prostředkem pro zlepšení kvality modelů, které jsou vytvářeny v průběhu vývoje systému.

### Implementace

Anwar (2014, str. 18) uvádí, že cíl implementační fáze se skládá ze čtyř částí. Jako první je nutné definovat, jakým způsobem bude organizován kód. Je nutné vzít v potaz implementaci ostatních subsystémů, kterých se vyvíjený software dotýká. Jako další cíl je implementace tříd a objektů z hlediska komponent. Výsledkem této části jsou spustitelné, binární a zdrojové kódy. V průběhu implementace je nezbytné nově vyvinuté části systému otestovat. Zde je vhodné použití přístupů extrémního programování, které napomáhá ke snížení času pro opravu identifikovaných chyb. V rámci posledního kroku dochází k integraci množin kódu, ve vývoji označeném pod pojmem „balíčky“. Tuto implementaci provádí tým nebo jednotlivci, kteří na vývoji systému spolupracovali. Následná kontrola kódu včetně tvorby nových verzí aplikací je v gesci team leadra vývojové skupiny. Jeho úkolem je též odpovědnost za juniorní členy vývojářského týmu. Bruckner (2012, str. 114) doplňuje, že metodika Rational Unified Process obsahuje postupy, jakým způsobem je možné opakovaně použít jednotlivé komponenty a jak implementovat komponenty nové. Obsahuje též způsoby, které jsou vodítkem pro navržení takové komponenty, kterou bude možné v budoucnosti znovu použít a vytvořit tak subsystémy.

### Testování

Rational Software (2011, str. 13) uvádí čtyři hlavní úkoly, které je nutné ve fázi testování provést. Na prvních příčkách zmiňuje ověření samotné interakce mezi implementovanými objekty a provedení kontroly, zdali byla implementace provedena dle plánu. Následně dochází k ověření, zdali jsou veškeré schválené požadavky obsaženy v dodávaném systému a je ověřena jejich funkčnost. Jako poslední bod stanovuje identifikování chyb systému, jejich zaznamenání a předání k opravě. Anwar (2014, str. 18) dále doplňuje pátý bod. Chyby, které byly v rámci testování odhaleny, musí být opraveny a provedeno retestování funkčnosti, se kterou byly spjaty. Následně dojde uzavření požadavku, popisující tuto chybu včetně informace, jakým způsobem byla provedena oprava. Metodika Rational Unified Process obsahuje hned čtyři role, které se zabývají činnostmi spjatými s testováním. První z rolí je test manager, který má na starosti zjistit co a proč se bude testovat včetně určení výstupních kritérií z testů. Test analytik následně stanoví, jakým způsobem se konkrétní testy budou provádět a zajistí konfiguraci aplikace tak, aby bylo možné testy provést. K tomu je vhodné vytvořit množinu testovacích uživatelů. Pro vytvoření samotných testů je v metodice určena role test managera, který provede tvorbu testů. Poslední rolí je tester, jehož úkolem je provedení smoke testů, neboli první otestování od počátku případu užití k jeho konci, a následnému provedení testů. Výstup je informace, zdali je provedená implementace shodná oproti stavu, který je očekávaný. Jakékoliv rozdíly je nezbytné v požadované formě nahlásit. V rámci takzvané kontroly čtyř očí se nedoporučuje jednotlivé role slučovat. Nejenom mezi jednotlivými testerskými rolemi, ale také napříč analytickými rolemi. Díky iterativnímu přístupu metodiky je možné testy automatizovat a provádět regresní testování s každou ukončující se iterací. Regresní testování je pojem, který se používá pro kontrolu původního kódu, který byl doplněn o nové části.

### Dodání

Anwar (2014, str. 18) charakterizuje fázi dodání jako *„pracovní postup, který má za cíl nasadit vyvíjený software na produkční prostředí a tím jej dodat pro používání koncovým uživatelům“*. Rational Software (2011, str. 13) dále tuto charakteristiku rozšiřuje o seznam aktivit, který

začíná migrací současných dat do nového systému, plánování a následně provádění beta testů, distribucí softwaru, instalací softwaru až po poskytování pomoci uživatelům při používání nového softwaru. Na přípravu dodání produktu se vyčleňuje kapacita již na konci fáze realizace.

### Projektové řízení

Bennett (2017, str. 9) projektové řízení charakterizuje jako umění vypořádat se s protichůdnými cíli a úspěšně zvládnout řídit rizika na projektu. Rational Software (2011, str. 13) dále doplňuje, že je nezbytné překonat omezení na projektu tak, aby bylo možné dosáhnout úspěšného dodání projektu, který bude splňovat požadavky zákazníků a uživatelů zároveň. Řízení projektu u metodiky Rational Unified Process lze rozdělit do dvou úrovní. Existuje fázový plán, který se zaměřuje na celkový průběh projektu a následně iterační plán, sloužící pro popsání vývoje v jednotlivých iteracích. Iterační plán se v metodice Rational Unified Process zaměřuje na podstatné aspekty iteračního vývojového procesu. Mezi tyto aspekty patří především plánování jednotlivých iterací na základě životního cyklu projektu, monitorování progresu v průběhu samotných iterací, obsazování rolí a řízení rizik. Metodika dále obsahuje sadu doporučení, jakým způsobem zaznamenat poučení z projektu a následné využití tohoto artefaktu v následujících projektech. Anwar (2014, str. 21) doplňuje, že projektové řízení naopak v metodice Rational Unified Process nepokrývá personální oblast, například nábor nových pracovníků a jejich zaškolení. Nevěnuje se též oblasti řízení rozpočtu nebo správou smluv se zákazníkem, případně vlastním dodavatelem.

### Konfigurační a změnové řízení

Anwar (2014, str. 20) uvádí, že konfigurační a změnové řízení je nepostradatelným prvkem každé metodiky, nevyjímaje metodiky Rational Unified Process. Některé metodiky tyto oblasti nedefinují zvlášť z toho důvodu, jelikož jsou si velmi blízké. Proto často bývá konfigurační a změnové řízení zahrnuto dohromady v konfiguračním managementu jako takovém. Konfigurační řízení odpovídá za systematickou strukturu jednotlivých produktů. Jednotlivé artefakty, jako například technické analýzy, musí být při jakékoliv změně verzovány a tyto úpravy je nutné v dokumentu patřičně zviditelnit. Při změnách je potřeba dbát na případné závislosti mezi artefakty a všechny dokumenty, které jsou dotčené změnami, je požadované odpovídajícím způsobem upravit a povýšit jejich verzi. Pro správu těchto požadavků se v rámci změnového řízení používá CRM, neboli Customer relationship management. Jedná se o software, který se používá pro řízení vztahu se zákazníky. Během implementace obvykle existuje hned několik verzí stejného artefaktu a pomocí tohoto systému je možné přehledně sledovat návrhy na jednotlivé změny. Díky této organizaci je možné zabránit případům, kdy na stejném artefaktu pracuje odděleně větší počet osob a poslední přispěvatel by smazal veškeré změny, které do dokumentu zanesli ostatní členové týmu. Rational Software (2011, str. 14) doplňuje, že koordinace tohoto paralelního vývoje je klíčovou součástí pro automatizovanou tvorbu buildu. Automatizovaný build je proces vytvoření změnového balíčku, který je automaticky sestaven z více změn od většího množství vývojářů. Tento proces též umožňuje rychlé nalezení chyby v případě, kdy se změna dotýká stejné oblasti v kódu. Hlavním cílem je však především vytvoření jednoho balíčku, který je na konci iterace nasazen do kódu k současné verzi softwaru.

### Prostředí

Anwar (2014, str. 19) uvádí, že pracovní proces prostředí se zaměřuje na nezbytné činnosti pro konfiguraci procesu pomocí metodiky Rational Unified Process pro konkrétní projekt. Jejím hlavním cílem je poskytnout pro účely projektu patřičné nástroje, které kladně podpoří

práci vývojářského týmu. Rational Unified Process je obsáhlá metodika sloužící pro vývoj softwaru a pracovní proces prostředí zajišťuje její přizpůsobení tak, aby byla vhodná pro daný typ projektu. Metodiku lze tedy libovolně upravovat nebo dokonce přizpůsobovat projektu na míru. Tímto krokem nebude vykonávána nadbytečná práce a vytvářeny nepotřebné artefakty. Díky tomu zůstane nadále jednoduchá do té míry, jak je to pro daný projekt možné. Pro plné využití metodiky je nutné, aby si její uživatelé uvědomili, že se kromě metodiky jako samotné jedná o procesní rámec usnadňující vývoj softwaru. Pokud tento krok nenastane, mohou jednotlivé osoby na projektu vnímat metodiku jako zbytečnou zátěž a nákladný proces. Rational Software (2011, str. 14) dále doplňuje, že tato konfigurace nezahrnuje určité aspekty, jako je výběr a získávání nástrojů pro vývoj. Neobsahuje též postupy pro jejich správné využití a propojení na konkrétních projektu.

### 2.5.6 Zhodnocení metodiky RUP

Jak je díky kapitolám výše zřejmé, metodika Rational Unified Process je systematická a lehce pochopitelná především díky modelu, na kterém je založena. To je možné z důvodu rozdělení na jednotlivé prvky, jimiž jsou role, aktivita, proces a artefakt. Tento způsob rozdělení v tradičních metodikách, jakými je vodopádový nebo spirálový model, ve většině případů úplně chybí. V lepších případech jsou tyto modely doplněny interními postupy, které jednotlivé artefakty upřesňují. Právě díky pohledu na jednotlivé prvky lze například zjistit, jaká role má odpovědnost nad jakým artefaktem, jaké činnosti které artefakty ovlivňují a v rámci jakých činností se konkrétní artefakty využívají. Dále díky rozdělení na fáze projektu a jednotlivé iterace se lze velmi dobře orientovat, v jakém životním cyklu vývoje se právě nacházíme a co je potřeba udělat.

Rational Software (2011, str. 2) uvádí, že díky dlouholeté zkušenosti předních odborníků na vývoj softwaru a softwarového inženýrství, mohla vzniknout metodika vycházející z reálných potřeb praxe a zároveň obsahující všechny základní praktiky pro vývoj softwaru. Jak již bylo zmíněno v předchozích kapitolách, jedná se konkrétně o iterativní vývoj softwaru, správu požadavků, využití komponentové architektury, vizuální modelování softwaru, průběžnou kontrolu kvality a využití řízení změn v jednotlivých fázích projektu. Metodika obsahuje též velké množství vzorů, nově vzniklých dokumentů a návodů, díky kterým lze projekt úspěšně dokončit ve vysoké kvalitě. Anwar (2014, str. 22) naopak k těmto materiálům uvádí, že mnohé z nich jsou směřovány na používání nástrojů od společnosti Rational. To skýtá sice výhodu využití nástrojů přizpůsobených pro tuto metodiku, avšak tyto nástroje nejsou součástí prodávané metodiky a jsou nabízeny samostatně za velmi vysoké částky. Obdobnou nevýhodu však obsahují i agilní metodiky, kdy pro jejich zavedení do podniku je nutné investovat vysoké částky za školení jednotlivých pracovníků a následné náklady za agilní kouče a případně za zkušené SCRUM mastery. Ani to v konečném důsledku nezajišťuje úspěšnou implementaci, jelikož agilní metodiky jsou velmi specifické a ve zkostnatělých korporacích je obtížné změny jako takové realizovat.

Schwalbe (2011, str. 73) uvádí jako hlavní nedostatek vodopádovému a spirálovému modelu prediktivnost životního cyklu projektu a jeho neměnnost. U vodopádového modelu konkrétně uvádí stálost požadavků, u kterých se nepředpokládá modifikace v průběhu projektu. Dále doplňuje, že spirálový model vychází z vodopádového modelu a byl značně modifikován, i přesto u mnoha projektů dochází při použití tohoto modelu k přechodu od iterace a spirálového přístupu k lineární.

Anwar (2014, str. 8) dále uvádí, že vzhledem vlastnictví značky Rational obchodní společností International Business Machines lze očekávat budoucí podporu a rozšiřování metodiky Rational Unified Process díky rozsáhlému zázemí této společnosti. Důležité je též spojení

metodiky s modelovacím jazykem Unified Modeling Language. Tím, že je tento jazyk používán pro vývoj softwaru v rámci celého světa a je považován za nebytnou součást při tvorbě analytických modelů, existuje mnoho expertů vyměňujících si názory nejen na problematiku tohoto modelovacího jazyka ale celé metodiky jako takové. Tomu napomáhá způsob distribuce metodiky, který probíhá dodáním interaktivních webových stránek. Tyto stránky jsou vzájemně provázány, čímž je snadné přecházet z jednotlivých částí; např. od artefaktů k rolím, které je využívají neventuálně vytváří. Tyto stránky je možné implementovat do síťové infrastruktury podniku a umožnit tak přístup k těmto materiálům plošně tj. zajistit přístup pro každého zaměstnance. Technicky je však podporována i lokání forma instalace. Anwar (2014, str. 21) oproti tomu uvádí jako jednu ze slabších stránek této metodiky nulové zaměření se na zpracování rozpočtu projektu a vztahy, které jsou mezi jednotlivými stranami. Ať se již jedná o vztahy dodavatel versus zadavatel, nebo figuruje v daném procesu subdodavatel. Proto je nutné pro tyto činnosti navrhnout postup, který bude možné v rámci společnosti aplikovat na jednotlivé projekty a zároveň bude kompatibilní s metodikou, potažmo samotným iterativním vývojem. To stejné platí i pro životní cyklus systému, jeho rozšiřování, údržbu a v poslední fázi vyřazení celého systému nebo jeho částí z provozu a implementace nového. Mezi největší výhody metodiky Rational Unififed Process uvádí, patří především zkvalitnění risk managementu, kdy použitý inkrementální přístup dokáže odhalit rizika v dřívějším termínu. Je to z důvodu možnosti vyzkoušení produktu zákazníkem dříve, než například při použití vodopádového nebo spirálového modelu. I proto je metodika Rational Unififed Process používána v korporacích, kde se paralelní vývoj nových systémů ve velké míře dotýká současných komponent.

## 2.6 Metodika

Diplomová práce je rozdělena do čtyř částí, úvodu, teoreticko-metodologické části, praktické části a závěru. Teoretická část práce byla vypracována pomocí komparace literárních zdrojů, které se zaměřují na metodiky pro vývoj softwaru. Autor se při vypracování teoretické části zabýval rozborem hlavních přístupů jednotlivých metodik, pro což byly použity odborné knihy českých a zahraničních autorů, doplněny internetovými zdroji.

V rámci praktické části došlo k charakteristice vývojového procesu společnosti XY a na základě rozboru průběhu projektu k následné identifikaci nedostatků, které v průběhu projektu nastaly. Tyto nedostatky měly na projekt dopad z hlediska termínu dodávky a s tím souvisejících vícenákladů za jednotlivé osoby na projektu. Z tohoto důvodu bylo nutné získat finance od zadavatele projektu a zajistit prodloužení alokací jednotlivých osob na projektu. Detailní charakteristika nedostatků, včetně vyhodnocení jejich dopadu na projekt, se nachází v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Pro účely budoucích projektů byla navržena doporučení, které mají za úkol snížit možná rizika a těmto pochybením v co největší míře zabránit. Následný návrh konfigurace metodiky Rational Unified Process využívá literární rešerše v teoreticko-metodologické a doporučení obsažené v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Díky těmto poznatkům je v kapitole *3.2 Konfigurace metodiky Rational Unified* Process navrhnuta konfigurace metodiky Rational Unified Process pro projekty v bankovním prostředí. Jedná se o řešení, které obsahuje dostatečné informace pro období zahajovací fáze projektu až k fázi nasazení. Každá jednotlivá fáze je následně blíže upřesněna separátně a jejich stručné představení se nachází v této kapitole. Jednotlivé role, které budou využity při vývoji softwaru dle konfigurované metodiky, jsou uvedeny v kapitole *3.3 Role a artefakty metodiky Rational Unified Process*. Nejedná se pouze o hrubý seznam rolí, ale je zde k nalezení detailní charakteristika a jejich odpovědnost za jednotlivé artefakty a činnosti v průběhu implementace.

Dále je možné při aplikaci metodiky využít charakteristiku těchto rolí k určení vhodných osob, které mají požadované zkušenosti nebo vlastnosti a budou schopny tyto role zastat. Seznam výše zmíněných rolí, které jsou v rámci bankovního sektoru s drobnými odlišnostmi dle jednotlivých společností využívány, poskytl na základě interní analýzy prodeje externích pracovníků a požadovaných schopností v oblasti vývoje softwaru vedoucí obchodního oddělení J. N. společně s metodičkou pro vývoj softwaru A. K. ze společnosti YX. Oba zaměstnanci si nepřáli být jmenováni, proto jsou uvedeny pouze jejich iniciály. Podklady pro studii byly získány v rámci rozhovoru, který probíhal v pražské pobočce společnosti dne 13. 12. 2019. Společnost YX se na trhu vývoje softwaru pohybuje již 20 let a patří mezi největší poskytovatele konzultačních služeb v oblasti IT pro Českou republiku.

Soupis jednotlivých artefaktů potřebných pro úspěšný průběh projektu se nachází v kapitole 3.3 *Role a artefakty metodiky Rational Unified Process*, kde jsou mimo jiné uvedeny role, které za konkrétní artefakt přebírají v rámci vývoje odpovědnost. Jedná se však o doporučení a je možné při implementaci metodiky odpovědnost rolí uzpůsobit dle procesům dané společnosti. Detailní účely artefaktů, včetně konkrétních činností na nich prováděných v průběhu projektu, jsou uvedeny v kapitolách 3.4 až 3.7, které se jednotlivými fázemi zabývají. V kapitole 3.4 *Fáze zahájení* je následně uvedena případová studie projektu, u kterého byla využita konfigurace metodiky Rational Unified Process. Konfiguraci této metodiky mohou kromě bankovních institucí, kterých je dle České bankovní asociace (2019) dle udávané struktury celkem 18 (4 velké banky, 5 středně velkých bank, 9 malých bank), využít také společnosti zabývající se externím vývojem pro tyto instituce.

# 3 Praktická část práce

Autor se v rámci praktické části diplomové práce věnuje zkoumání současného procesu vývoje softwaru ve společnosti XY. Dle získaných zjištění následně došlo k identifikování chyb, které v důsledku nesprávných postupů při implementaci a vývoji vznikly. Jako řešení této situace a zlepšení vývoje softwaru ve společnost XY dojde k návrhu konfigurace metodiky Rational Unified Process, která bude následně využita u případové studie. Z důvodu robustnosti bankovních systémů je nutné provést přesnou charakteristiku jednotlivých rolí a činností, za které jsou v rámci projektu tyto jednotlivé role v týmu odpovědné. Role a artefakty, které nejsou pro zdárné dokončení projektu potřebné a zároveň se v kompletní metodice Rational Unified Process od společnosti IMB vyskytují, nebudou v konfiguraci zmiňovány. Využité role budou naopak mírně upraveny dle potřeb společnosti. Případová studie se dále věnuje praktickému využití navržené konfigurace pro projekt v bankovním prostředí. U zvoleného projektu jsou zdůrazněny využité změny v rámci metodiky, které pozitivně ovlivnily implementaci softwaru.

## 3.1 Charakteristika současného vývojového procesu ve společnosti XY

Tato podkapitola se zabývá charakteristikou současného procesu vývoje softwaru u společnosti XY, která působí ve finančním a bankovním sektoru. K tomuto účelu bude využit ukázkový projekt, který byl v nedávné minulosti implementován. Samotná charakteristika probíhá již zpětně, tedy v tomto případě po implementaci nové funkčnosti. V současné době společnost XY nevyužívá pro vývoj softwaru žádnou specifickou metodiku, postup se však v mnoha ohledech shoduje s přístupy vodopádového modelu. Projekt je tedy rozdělen do etap, které odpovídají vodopádovému cyklu uvedenému v kapitole *2.2 Vodopádový model*. Role, které byly použity v projektu včetně jejich odpovědnosti za jednotlivé artefakty, jsou následující:

- zadavatel – validuje jednotlivé obchodní požadavky, které jsou v průběhu projektu analytikem vytvořeny včetně schválení případných změn;
- projektový manažer – odpovědnost za projektový plán a akceptační protokol;
- architekt informačního systému – odpovídá za architekturu implementace;
- analytik – odpovědnost za obchodní požadavky a analýzu, která slouží jako zadání pro vývojáře a dále za testovací scénáře, dle kterých bude provedeno testování aplikace;
- vývojář – implementace funkčnosti dle dodané analýzy
- tester – odpovídá za komplexní otestování nově implementované funkčnosti včetně ověření, zdali funkce neovlivnila již funkční scénáře v aplikaci;
- service owner – přebírá dodanou funkcionalitu do produkce a provádí aktualizaci provozní dokumentace dle dodané analýzy, která je následně podkladem pro případné chyby vzniklé při užívání aplikace uživatelem.

V následujících kapitolách dojde k přiblížení výhod a nedostatků tohoto přístupu v praxi tak, jak probíhají jednotlivé etapy projektu. Cílem zkoumaného projektu byla implementace nové funkcionality, konkrétně možnost provést klientem autorizované zpětné volání na infolinku helpdesku. Tato funkčnost je v aplikaci znázorněna ikonou telefonu. Po jejím stisknutí uživatelem dojde k potvrzení volaného telefonního čísla a následně ke spojení s operátorem klientského centra. Díky umístění této ikony ve více místech aplikace je uživateli umožněno zažádat o telefonickou pomoc v těch krocích, kde by kvůli telefonátu musel tuto činnost ukončit. I díky tomuto umístění je možné klienta ihned přesměrovat na operátora, který tento problém řeší na denní bázi a není nutné jej přesměrovávat, případně ponechat výběr problému na automatizovaném robotu tak, jak tomu bylo doposud. Zadavatel projektu si od této

funkčnosti sliboval zefektivnění práce na straně klientského centra a zlepšení služeb, které jsou klientům poskytovány. Dále očekával zvýšení telefonického prodeje úvěrů a dalších služeb, u kterých bude tato funkčnost umístěna. Z důvodu přímých dopadů na klienta byla funkcionalitě přisuzována vysoká priorita.

### 3.1.1 Etapa specifikace problému

Zadavatelem této změny byl člen managementu klientského centra, který se ve společnosti zabýval zpětnou vazbou klientů. Hlavním požadavkem bylo zefektivnění práce operátorů klientského centra a zjednodušení žádosti o pomoc klienta, pokud se nacházel v rozpracované žádosti. V průběhu představení požadovaných vlastností systému, které zadavatel požaduje implementovat, provedl analytik hrubou specifikaci požadavků. Tyto konzultační schůzky proběhly celkem ve 4 termínech, kdy finální schůzka sloužila pro zafixování počtu požadavků, které budou v etapě analýzy dále upřesněny. Z důvodu uvolňování nových funkcionalit do produkčního prostředí pouze třikrát ročně, bylo datum počátku projektu stanoveno na 1. 12. 2016 a ukončení této etapy proběhlo 15. 12. 2016.

Pro úspěšný počátek projektu v plánovaném čase došlo projektovým manažerem k sestavení projektového týmu, který sestával z celkem sedmi členů. Mezi první obsazené role patřila pozice analytika a architekta informačních systémů. Tyto role byly v průběhu etapy specifikace problému nezbytné pro návrh funkční architektury a hrubé specifikování obchodních požadavků. Analytik v této etapě projektu vedl schůzky s ostatními zainteresovanými stranami a získané požadavky na novou funkci zaznamenal do strukturově uspořádaného seznamu. Každý požadavek byl následně ve fázi analýzy dále upřesňován. Tyto požadavky, které slouží pro validaci odpovídajících funkcí u zadavatele projektu, byly zpracovány dle interní metodiky firmy XY a měly následující podobu:

| | |
|---|---|
| **Identifikátor a název požadavku:** | *AC001 – Autorizované zpětné volání* |
| **Priorita požadavku:** | *1 (nejvyšší)* |
| **Zadavatel požadavku:** | *Klientské centrum, Z. V.* |
| **Popis požadavku:** | *Jako zadavatel požaduji, aby implementace funkce autorizovaného zpětného volání obsahovala možnost konfigurovat umístění této funkcionality napříč aplikací včetně jejího rozpoznání tak, aby bylo možné telefonát přepojit na odpovědného pracovníka klientského centra.* |

Hlavním úkolem projektového manažera byla v této etapě tvorba a následná aktualizace projektového plánu, která probíhala na základě výstupů analytika a architekta informačního systému. Projektový plán v této etapě obsahoval tyto činnosti:

- návrh architektury aplikace (architekt informačních systémů);
- vytvoření harmonogramu projektu (projektový manažer);
- sběr obchodních požadavků (analytik);
- identifikace rizik v projektu (projektový manažer);
- zajistit alokace osob pro nesledující etapy projektu (projektový manažer);
- definovat činnosti pro jednotlivé etapy projektu (projektový manažer).

Projektový manažer dle výstupů analytika a architekta informačních systémů definoval možná rizika, která mohou ohrozit plánovaný průběh projektu. Každé riziko bylo definováno

označením a jeho bližším popisem. Níže se nachází výčet rizik, která projektový manažer v rámci tohoto projektu identifikoval:

1) **Označení rizika:** *001 – Integrace jednotlivých služeb*
   **Detail rizika:** *Systémy společnosti v rámci projektu „Autorizovaného zpětného volání" využívají pro komunikaci mezi sebou technologii REST. Systémy klientského centra pracují pomocí technologie SOAP. Je nutné provést na konečném systému banky, nebo na prvotním systému klientského centra překlad struktury jednotlivých služeb.*

2) **Označení rizika:** *002 – Komunikace mezi vývojáři společnostmi*
   **Detail rizika:** *Z důvodu neexistující předchozí spolupráce mezi vývojáři banky a klientského centra nejsou stanoveny styčné osoby, které komunikaci a jednotlivé kompromisy zaznamenají a informují ostatní členy týmu. Hrozí tedy implementace částí systému rozdílným způsobem, než bylo v rámci analýzy plánováno. Toto může ohrozit celkové propojení jednotlivých sekvencí služeb.*

Po předání seznamu rizik a po jejich celkové akceptaci zadavateli projektu bylo možné navrhnout předběžný harmonogram projektu:

**Etapa analýzy**

Plánovaná délka etapy analýzy jsou 4 týdny (18. 12. 2016 – 19. 1. 2017). V průběhu etapy analýzy se počítá s těmito artefakty a činnostmi:

- dokončení návrhu architektury systému;
- příprava integračního a akceptačního prostředí pro implementaci;
- validace IT analýzy;
- dokončení implementace architektury včetně konfigurace systémů;
- příprava testovacích scénářů.

**Etapa implementace**

Plánovaná délka etapy implementace je 6 týdnů (22. 1. 2017 – 23. 2. 2017) a je rozdělena do dvou fází:

**Fáze 1 – vývoj aplikace na integračním prostředí** (22. 1. 2017 – 16. 2. 2017)

- implementace systému dle dostupné analýzy na integrační prostředí;
- akceptace aplikace pro přechod na akceptační prostředí;
- příprava akceptačního prostředí.

**Fáze 2 – nasazení aplikace do akceptačního prostředí** (9. 2. 2017 – 23. 2. 2017)

- nasazení aplikace na akceptační prostředí;
- dokončení testovacích scénářů;
- příprava konfigurace dat pro testování.

V případě úspěšného nasazení dat do akceptačního prostředí se projekt přesouvá do etapy testování, kde dojde k ověření implementace jednotlivých funkcí.

**Etapa testování**

Plánovaná délka etapy testování je 5 týdnů (26. 2. 2017 – 23. 3. 2017). Testování se provádí na akceptačním prostředí, stejně jako oprava nalezených chyb.

- testovaní aplikace dle testovacích scénářů;
- performační testování aplikace;
- oprava nalezených rozporů ve funkčnosti a jejich oprava;
- vyhodnocení výsledků testování;
- akceptování dodávky projektu a příprava aplikace pro nasazení do produkce.

**Etapa nasazení**

Plánovaná délka etapy nasazení je 1 týden (26. 3. 2017 – 30. 3. 2017). V průběhu této etapy probíhá především postupné nasazení na uživatele tak, aby bylo možné průběžně sledovat případné performační problémy.

- nasazení aplikace do produkčního prostředí;
- vytvoření nové verze aplikace, která bude uvolněna klientům;
- postupné uvolnění aplikace klientům;
- průběžné sledování zátěže jednotlivých systémů;
- oprava nekritických chyb pomocí hotfixu.

**Etapa projektového provozu a údržby**

Plánovaná délka této etapy je 1 měsíc (2. 4. 2017 – 30. 4. 2017). V průběhu této etapy probíhá průběžná kontrola funkčnosti aplikace, především pak zátěž jednotlivých systémů. Dále probíhají případné opravy aplikace, které nebyly nalezeny v průběhu testování a nyní jsou hlášeny uživateli jako incidenty z produkčního prostředí.

- průběžné sledování zátěže jednotlivých systému;
- oprava vážných a kritických chyb pomocí hotfixu.

Následně na konci této etapy (tj. na konci projektu) dospěl zadavatel a projektový manažer k rozhodnutí, že je projekt na základě předloženého harmonogramu, výčtu rizik a ostatních informací proveditelný. Zadavatelem tedy byla provedena akceptace zadání a projekt byl posunut do etapy analýzy.

### 3.1.2 Etapa analýzy

V počátku etapy analýzy byl dokončen návrh architektury systému včetně zadání, které obsahuje konfigurační parametry pro přípravu integračního a akceptačního prostředí. Tento dokument, včetně schválených požadavků od zadavatele, dále slouží jako podklad analytikovi k vypracování zadání pro vývojáře. V průběhu této etapy dále probíhala finalizace implementace architektury. V tomto konkrétním případě jsou však jednotlivé systémy vzájemně používány, proto se jedná pouze o dílčí úpravy a nastavení prostupů u jednotlivých služeb, které jsou uvedeny v dokumentaci projektu. Konkrétně se jednalo o tyto služby:

**getClientContacts@DB** – služba pro získání kontaktů daného klienta;

**createAuthConnection@BUS** – služba vytvoření autorizovaného zpětného volání.

Výjimkou byla komunikace mezi bankovním systémem a přístupovým systémem klientského centra. Zde proběhlo předání komunikační mapy, díky které bylo možné vytvořit zabezpečený prostup. Po dokončení analýzy a její validaci architektem informačních systémů došlo

k přípravě testovacích scénářů, které slouží jako zadání testerovi v testovací etapě. Na konci etapy dále došlo projektovým manažerem k validaci harmonogramu projektu a nebylo doznáno změn.

### 3.1.3 Etapa implementace

Etapa implementace byla rozdělena do dvou částí, kdy první část se zaměřuje na samotnou implementaci systému na integrační prostředí a následnou konfiguraci akceptačního prostředí. Ve druhé části došlo k nasazení aplikace do akceptačního prostředí a k přípravě dat pro její otestování v následující etapě.

V průběhu vývoje na integračním prostředí nevyvstaly žádné hrozby nebo incidenty, které by způsobily posunutí projektu v čase. Avšak na počátku implementace v akceptačním prostředí došlo ke zjištění, že neproběhl počátek vývoje (dle původní domluvy mezi bankou a klientským centrem) převodníku pro využití REST kódu na straně klientského centra. Proto bylo zadavatelem rozhodnuto, že tento převodník kódu bude implementován na straně banky i přes vzniklé vícenáklady za vývoj. Toto rozhodnutí bylo odůvodněno na základě návaznosti ostatních projektů na tuto implementovanou funkčnost. Nebylo tedy nutné upravovat plán projektu, avšak došlo k nárůstu celkových nákladů.

Na konci akceptační etapy došlo k dokončení testovacích scénářů, které byly využity v následující fázi pro ověření funkčnosti implementace. Dále došlo k provedení konfigurace u pěti stávajících testovacích uživatelů tak, aby bylo možné otestovat veškeré případy chování aplikace.

### 3.1.4 Etapa testování

Testování implementované funkčnosti probíhalo dle připravených testovacích scénářů. Prvotní ozkoušení aplikace proběhlo úspěšně a bylo možné úspěšně funkcionalitu začít testovat. V několika místech však již nebylo chování aplikace dle očekávaného scénáře. K datu 2. 3. 2017 proběhl test celkem 25 z celkem 40 scénářů, zbytek scénářů byl následně otestován v období 5. 3. 2017 až 9. 3. 2017, kdy reálná úspěšnost všech testů činila 77 %. Bylo tedy nutné provést opravu celkem 11 chyb, které se v rámci jednotlivých scénářů vyskytly. V průběhu oprav těchto chyb došlo k performačnímu testování aplikace v akceptačním prostředí včetně využití nově implementované funkčnosti. Konečný výsledek těchto testů neobjevil žádné problémy s nedostatkem operačního výkonu na systémech, které současná aplikace využívá. Nebyla zaznamenána ani žádná odchylka v rychlosti aplikace s implementovanými funkčnostmi oproti původní verzi aplikace. K datu 20 3. 2017 bylo ukončeno opakované testování fungujících a opravovaných funkcionalit. Výsledek těchto testů byl 100%, všech čtyřicet scénářů bylo úspěšně ozkoušeno metodou konec-konec. Dne 22. 3. 2017 proběhla zadavatelem akceptace implementovaných funkčností na základě testovacího protokolu, a bylo tudíž možné připravit nasazení nové verze aplikace do produkce. Po této validaci od zadavatele proběhla schůzka s vlastníkem produktu a předání podkladů v podobě analýzy pro potřebu aktualizace provozní dokumentace.

### 3.1.5 Etapa nasazení

Nasazení nové verze aplikace do produkčního prostředí proběhlo dne 26. 3. 2017 a doprovázely jej problémy týkající se nefunkčních bezpečnostních prostupů, které mezi bankou a klientským centrem nebyly nastaveny. Funkcionalita tedy v nové aplikaci byla dostupná, nikoliv však funkční. Tyto problémy byly v rámci hotfixu[2] dne 27. 3. 2017 opraveny a bylo možné provést

[2] Nasazení kritické opravy do produkčního prostředí, která zajistí správnou funkčnost aplikace

postupné uvolňování aplikace klientům. V průběhu zvyšování se počtu klientů s novou verzí aplikace nebyly zaznamenány žádné performační problémy. Dne 4. 4. 2017 došlo ke stoprocentnímu nasazení aplikace na klienty. Oproti plánovanému termínu se jednalo o nepatrné čtyřdenní zpoždění od projektovému plánu, které však bylo problematické a způsobilo výpadek některých úvěrových služeb z důvodu závislosti na dodávaném řešení.

### 3.1.6 Etapa projektového provozu a údržby

V průběhu této etapy je hlavním úkolem projektového týmu takzvaný babysitting[3], neboli údržba nových funkcionalit aplikace po nezbytnou dlouhou dobu, který zajistí případné opravy při zjištění závažných problémů. V případě této implementace došlo ve fázi babysittingu k závažným problémům, které způsobily přetížení konkrétních linek u klientského centra a delší odezvu klientům požadujících pomoc v rámci úvěrového procesu. Dle interních měření společnosti XY bylo zjištěno, že celkem 62 % klientů nebylo telefonicky obslouženo a jejich následné kontaktování v 45 % nebylo úspěšné, nebo bylo s velmi negativní reakcí.

Dle projektové dokumentace došlo v průběhu provozu a údržby aplikace k opravě celkem šesti závažných chyb, čtyř chyb se středním dopadem a dále k opravě dvanácti chyb grafického charakteru. Tento počet je oproti jiným projektům ve společnosti konstantní, nicméně vzhledem k propojenosti jednotlivých implementací velmi závažný. K datu (30. 4. 2017), tedy ukončení této etapy, došlo k opravě veškerých chyb a implementace byla předána pod plnou správu service ownera, který již bude další produkční incidenty řešit v rámci liniového týmu.

### 3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu projektu

V průběhu procesu vývoje projektu došlo k několika pochybením, která způsobila nekonzistenci mezi požadovaným a implementovaným stavem funkcionality. Tato subkapitola charakterizuje jednotlivé nedostatky a definuje doporučení, která je vhodné při konfiguraci metodiky Rational Unified Process zohlednit. Jmenovitý seznam nedostatků je uveden níže:

- na projektu došlo ke sloučení analytické a testerské role, což způsobilo přehlédnutí zjevných chyb na straně analýzy kvůli takzvané projektové slepotě;
- v rámci schvalování jednotlivých požadavků došlo ke změnám, které nebyly zaznamenány v rámci BRQ setu, ale došlo k jejich implementaci, což způsobilo nejasnosti a nekonzistentnost zadání;
- u jednotlivých požadavků není uvedeno akceptační kritérium, dle kterého je možné explicitně určit, zda byla splněna funkčnost požadovaná zadavatelem;
- při návrhu architektury nebyla zajištěna možnost přepoužití řešení nebo jeho částí v rámci jiných projektů, které se ve společnosti budou implementovat;
- u rizik uvedených v projektu chybí informace, o jak závažné riziko se jedná a specifikování preventivních opatření, které mohou riziko zmírnit nebo případně v určité fázi projektu odstranit;
- vzhledem k chybějícímu iteračnímu testování došlo k neodhalení závažných chyb, které způsobily opoždění nasazení implementace do produkčního prostředí;
- nasazení do produkčního prostředí doprovázely problémy týkající se nastavení jednotlivých prostupů a konfigurace mezi systémy banky a třetí strany;
- chyby, kterých se jednotlivé osoby na projektu dopustily, se mohou opakovat i v projektech navazujících a způsobit jejich zpoždění.

[3] Jedná se o nezbytně dlouhé období, kdy je nutná částečná alokace osob pro případnou analýzu a opravu produkčních chyb

Při definici jednotlivých pozic v týmu není vhodné provádět slučování analytických a testerských rolí, jak je uvedeno v kapitole *2.5.5 Proces*. Je to především z důvodu takzvané kontroly čtyř očí, kdy je velká pravděpodobnost, že analytik implementace nezohlední nutnost scénáře z důvodu projektové slepoty. Proto se doporučuje vyhradit pro projekt samostatnou roli test analytika, který zajistí přípravu testovacích scénářů dle vypracované analýzy. Z obdobných důvodů je dle kapitoly *2.5.4 Role, aktivity a artefakty* vhodné rozdělit analytickou činnost do dvou celků a určit pro zpracování dvě samostatné role. Jedná se o roli business analytika a IT analytika. Díky tomu dojde k zajištění, že návrh konečného řešení se opírá o schválené obchodní požadavky a pro případné změnové požadavky je nutné upravit stávající BRQ set, ze kterého IT analytik vychází. Pomocí tohoto kroku se můžeme spolehnout na validní verzování obchodních požadavků včetně patřičného odůvodnění, proč k této změně došlo. Dále dojde k rozpadu analýzy na dílčí artefakty, které zajistí potřebný detail pro přesnější implementaci. Jak dále uvádí kapitola *2.5.2. Šest nejlepších praktik pro vývoj softwaru,* je vhodné využít komponentovou architekturu, která umožní přepoužití jednotlivých služeb a systémů. K tomu je vhodné ve firemní struktuře zajistit roli, která bude model s touto architekturou udržovat aktuální včetně zajištění konzultací ze strany projektu. Způsob vedení architektury je nutné rozdělit na část obchodního modelování a technickou architekturu, přičemž jsou obě části na sobě nezávislé.

Při tvorbě obchodních požadavků je vhodné zaznamenat, jak je uvedeno v kapitole *2.5.5 Proces,* kromě priority jednotlivých požadavků také jejich případný dopad, pokud nebudou do systému implementovány. Dále je nutné specifikovat akceptační kritéria, jelikož i při detailní specifikaci si lze výslednou funkci z různých pohledů vyložit rozdílně a může dojít k neakceptování implementované funkčnosti, která se na daný obchodní požadavek váže. Podobný nedostatek se nachází u artefaktu rizika, kdy je vhodné uvést též stupeň a prevenci rizika jak se uvádí v kapitole *2.5.2 Šest nejlepších praktik pro vývoj softwaru*. Tento artefakt se poté doporučuje v průběhu jednotlivých iterací aktualizovat. Nová rizika mohou v průběhu projektu nadále vznikat a naopak i díky uvedeným prevencím rizik následně zanikat. Je též vhodné provést zpřesnění jednotlivých časových odhadů, zejména pro implementační etapu, která v této společnosti obsahuje více iterací. To umožní projektovému manažerovi včas zaznamenat případné nedodržené termíny v oblasti implementace, které může způsobit celkové zpoždění projektu.

Jak se uvádí v kapitole *2.5.3 Fáze životního cyklu,* je vhodné v průběhu počátečních etap vytvořit architektonický a grafický prototyp aplikace. V prvním případě tento prototyp umožní potvrdit předpoklady architektury, která byla odpovědnými osobami navržena. U grafického prototypu naopak očekáváme zpřesnění obchodních požadavků díky možnosti lepší představy zadavatele, jakým způsobem bude požadavek uživateli reflektován. Díky tomu bude také zmírněno riziko budoucích změnových požadavků souvisejících s funkcionalitami grafického rozhraní, které jsou nezávislé na volání jednotlivých služeb.

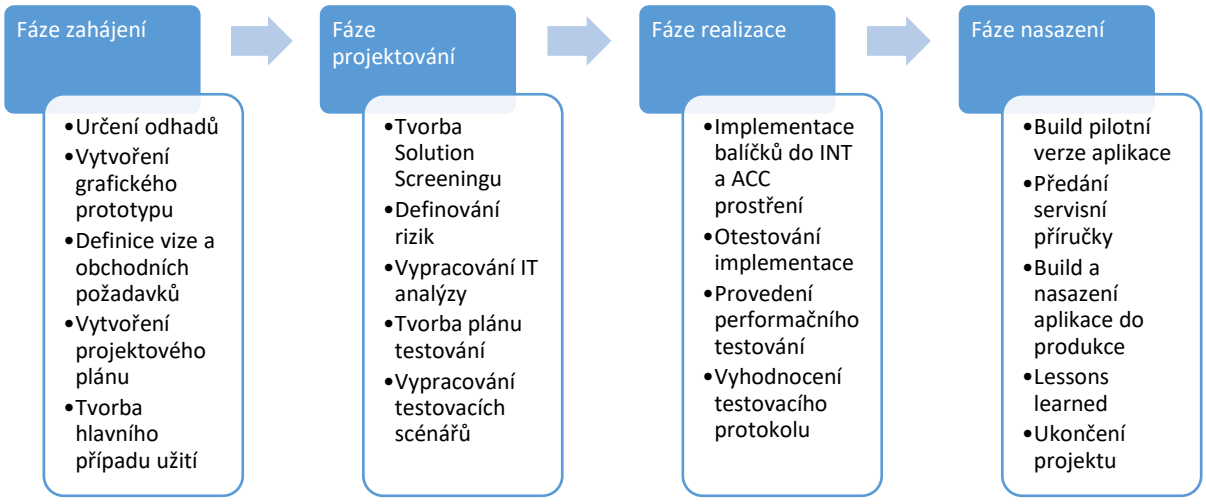
Vzhledem k tomu, že ve společnosti probíhá vývoj postupně ve dvou odlišných prostředích, kdy se verze aplikace z integračního prostředí na jeho konci fáze implementuje jako celek do akceptačního prostředí, je vhodné jak uvádí kapitola *2.5.2 Šest nejlepších praktik pro vývoj softwaru* provést změny v testovacím procesu. Metodika Rational Unified Process je založena na průběžném testování vyvíjeného kódu, tudíž je potřebné etapu implementace propojit s etapou testování. To lze provést posunutím testování nové funkčnosti již v etapě implementace ihned na počátku vývoje. Pokud bude vývoj správně řízen a bude probíhat formou end-to-end, je možné již testy provádět na prvních vyvinutých obrazovkách aplikace. Dále je doporučeno v počátcích implementace využít možnost testování pomocí SOAP testů, které mohou odhalit případné konfigurační problémy u jednotlivých volaných systémů a tedy nedostupnost jejich rozhraní.

Vzhledem k dopadům implementace na ostatní projekty v produkčním prostředí je vhodné aplikaci v první řadě vyzkoušet v rámci pilotního provozu tak, jak se uvádí v kapitole *2.5.3 Fáze životního cyklu*. Pilotní provoz v produkčním prostředí je běžný pro zjištění, zdali je implementace navzdory důkladnému testování funkční a neohrozí celou aplikaci jako takovou. Jako testeři jsou ve většině případů vybráni seniorní uživatelé z řad zaměstnanců podniku. Díky tomuto kroku je možné v případě kritické chyby provést její opravu a navzdory zpoždění u ostatních projektů zabránit vzniku incidentů, které budou mít negativní dopad na uživatele aplikace. Tato pochybení nebo naopak případné vhodně nastavená opatření je nezbytné zachytit na konci projektu, nejlépe formou strukturalizovaného dokumentu jak se uvádí v kapitole *2.5.5 Proces*. Pomocí této zkušenosti lze ovlivnit budoucí projekty a zabránit chybám, které se staly jejich součástí. V opačném případě lze naopak vhodné postupy zavést jako určitý projektový standard, který bude nutné dodržet v definovaných fázích a situacích.

## 3.2 Konfigurace metodiky Rational Unified Process

Konfigurace metodiky Rational Unified Process proběhla na základě závěrů z předchozí kapitoly, které se nacházejí v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Úpravy se zabývají především pochybeními a nedostatky, které se v průběhu definovaného projektu odehrály. Očekáváním od této konfigurace je navrhnout vhodnou formu metodiky, kterou bude možné použít pro budoucí projekty ve společnosti XY a zároveň díky její implementaci dojde ke snížení realizačních rizik a případné chybovosti na projektu. Konfigurace se nejdříve věnuje procesní části a následně specifikuje jednotlivé role a artefakty. Jednotlivé etapy, vyobrazené na Obrázku 6, byly nahrazeny fázemi projektu, tak jak se uvádí *2.5.3 Fáze životního cyklu*. Tato změna byla provedena především kvůli dodržení postupu vývoje dle metodiky Rational Unified Process, o který se navrhovaná konfigurace v rámci této práce opírá.

**Obrázek 5 Fáze metodiky Rational Unified Process**




*Zdroj: Vlastní zpracování*

V zahajovací fázi, jak je zřejmé z Obrázku 6, samotná implementace neprobíhá. Výjimkou je u některých typů projektů vytvoření modelu, případě jednoduchého prototypu, který pomůže k ověření, zda je reálné s vybranými nástroji a zvolené technologie požadavky splnit.

Projektový architekt a business architekt následně poskytnou jednotlivé odhady. Dle interní směrnice k projektovému řízení společnosti XY (2016, str. 25) je alokační návrh od business architekta pro IT business analytika ve většině případů stanoven s očekávanou odchylkou 20 %. Odhady projektového architekta, týkající se délky IT analýzy a budoucího vývoje, jsou s očekávanou odchylkou 75 %. Tímto je možné stanovit přibližnou cenu za vývoj požadovaného systému. Výstupem z této fáze, ve které hrají velkou roli takzvaní podnikoví inovátoři, je kromě jednotlivých odhadů především vize projektu společně se seznamem business požadavků. Tuto vizi je včetně požadavků nezbytné odsouhlasit zadavatelem samotného projektu. Jak se uvádí v kapitole *2.5.3 Fáze životního cyklu*, schválení se považuje za největší milník této fáze. Na konci zahajovací fáze je známo, zda budeme v projektu pokračovat, nebo bylo vlastníkem rozhodnuto, že bude projekt změněn nebo dokonce zrušen.

Ve fázi projektování dochází především k zaměření se na stabilní prototyp architektury systému. Díky tomuto prototypu je možné během jednotlivých iterací ověřit, zdali je zvolená architektura realizovatelná a udržitelná. Architektura systému je společně s hlavním případem užití popsána v dokumentu SOS, neboli Solution Screening. V tomto dokumentu jsou zaznamenány i jednotlivé dotčené systémy, veškerá rizika projektu včetně vyjádření od jejich vlastníků a odhady za jednotlivé fáze projektu. V této fázi projektu již potřebné technické odhady poskytuje projektový architekt, IT analytik a developer team leader. Tyto odhady se pohybují s odchylkou 50 % na počátku fáze a postupnou IT analýzou se zpřesňují až do finálních čísel. Též se zde vyskytují vyjádření od jednotlivých rolí, jakými jsou například bezpečnostní útvar nebo právní oddělení, kteřé jsou nezbytné pro kompletní dokončení dokumentu SOS. Schválení tohoto dokumentu, který je společně s IT analýzou hlavním výstupem z této fáze, podstupuje projektový architekt u enterprise architekta a zadavatele. Díky tomu dojde k doporučené kontrole čtyř očí, jak se uvádí v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu* čímž se sníží šance vniku chyb a nepřesností.

Ve fázi realizace probíhá samotná implementace systému dle IT analýzy, za kterou odpovídá IT analytik. Realizace je rozdělena do dvou hlavních iterací, při kterých je systém nejprve vyvíjen v integračním prostředí a následně je po testech nasazen do prostředí akceptačního. Tato prostředí jsou označovány zkratkou INT (integrační prostředí) a ACC (akceptační prostředí). Integrační prostředí slouží pro prvotní ozkoušení jednotlivých balíčků, které jsou v průběhu realizace dodávány vývojářským týmem. Balíček lze charakterizovat jako část implementace, díky které systém splňuje funkčnosti dle zadaných požadavků. Marge funkčního kódu z integračního do akceptačního prostředí probíhá po fázi integračního testování. Pokud je end-to-end testování (princip testování konec-konec) úspěšné, dochází k přenesení kódu do akceptačního prostředí a ve většině případů k opravě drobnějších chyb. Toto je největší rozdíl oproti projektu, který byl charakterizován v předchozí kapitole. Díky tomu také dochází ke splnění doporučení, které je uvedeno v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. V průběhu oprav chyb dochází též k performačnímu testování, které má za úkol odhalit případnou zvýšenou zátěž na systémy. Na konci této fáze je výstupem produkt, který neobsahuje kritické a vážné chyby. Tato skutečnost je v průběhu oprav v akceptačním prostředí zkoumána a testována seniorními uživateli, případně zadavateli ze strany businessu.

Fáze nasazení do produkčního prostředí, označovaném pod zkratkou PROD, probíhá v případě akceptování funkčnosti systému zadavatelem z předcházejícího prostředí. Systém tedy neobsahuje žádnou závažnou vadu a je možné jej poskytnout uživatelům. Pro tento krok je nejprve nutné vytvoření pilotní verze, která bude opět k dispozici pouze seniorním uživatelům, vlastnícím oprávnění do produkčních systémů. Díky tomuto kroku, který je též uveden jako doporučení v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*, dojde k ověření funkčností implementace a snížení rizika dodání nefunkční aplikace.

V případě správné funkčnosti systému v produkčním prostředí dochází k postupnému uvolňování systému na uživatele. To je provedeno konfiguračním nastavením, díky kterému je zajištěn menší dopad na koncové uživatele v případě neodhalené závažné chyby.

## 3.3 Návrh konfigurace rolí a artefaktů metodiky Rational Unified Process

V rámci konfigurace metodiky Rational Unified Process autor vybral 15 klíčových rolí z 30 celkových, které jsou uvedeny v kapitole *2.5.4 Role, aktivity a artefakty*. Výběr jednotlivých rolí proběhl na základě původně použitých rolí uvedených v kapitole *3.1 Charakteristika současného vývojového procesu ve společnosti XY* a na základě doporučení, která jsou uvedena v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Charakteristika jednotlivých rolí je popsána v následujících odstavcích a je nezbytná pro úspěšnou implementaci metodiky do procesu vývoje softwaru ve společnosti XY. U jednotlivých rolí bylo nutné definovat jejich povinnosti a vazbu k projektovým artefaktům. V neposlední řadě nesmí dojít k opomenutí určení činností, které má konkrétní role na starost.

### Zadavatel

Zadavatel, v tomto případě též sponzor projektu, je primární investor. V rámci korporátní organizace se nejedná o vlastníka společnosti, ale o osobu nebo osoby, kterým byl svěřen finanční rozpočet na rozvoj společnosti za určité období. V rámci konfigurace této metodiky se bude jednat o skupinu zainteresovaných osob, od kterých pochází nejvíce iniciativy, ať už se jedná o vůli projekt zahájit nebo splnit strategické cíle, jež byly na počátku projektu vytyčeny. Jak se uvádí v kapitole *2.5.2 Šest nejlepších praktik pro vývoj softwaru*, je tento výběr osob velmi podstatný a je vhodné cílit na osoby s dobrými komunikačními schopnostmi. Odvíjí se od toho budoucí kvalita sesbíraných požadavků. Zadavatel je též odpovědný za výstupy z každé proběhlé iterace, v tomto případě INT, ACC a PROD. Obsah těchto výstupů podléhá akceptaci zadavatele. Oproti současné charakteristice vývojové procesu se jeho úloha nemění.

### Business Architect

Tato role zastává v rámci projektu především konzultační pozici směrem k zadavateli projektu a projektovému architektovi. Jeho úkolem je dohlédnout, aby cíl projektu naplňoval business požadavky na úrovni společnosti. Dále slouží jako konzultant v případě změnových požadavků a při stanovení alokací na počátku projektu. Díky tomu jsou splněna doporučení uváděná v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu* ohledně role, která zajišťuje naplnění jednotné business architektury v rámci společnosti.

### Enterprise Architect

Role enterprise architecta slouží pro projektového architekta jako konzultační osoba, která schvaluje návrh implementačního řešení na úrovni solution screeningu. Jeho hlavní úlohou je, aby návrh implementace projektu odpovídal podnikové architektuře. V rámci této konfigurace budeme předpokládat, že osoba v roli enterprise architekta je též konzultant pro oblasti kybernetické bezpečnosti. Tato role je nutná pro naplnění doporučení týkající se technické architektury v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

### Projektový manažer

Role projektového manažera je v rámci projektu odpovědná za harmonogram, plánování činností v rámci jednotlivých iterací a koordinační součinnost v průběhu celého projektu. Jeho postavení a rozhodovací pravomoci nejsou omezeny pouze uvnitř projektu, ale funguje též jako prostředník pro komunikaci se zákazníkem, případně s externími dodavateli softwaru.

U projektového manažera je velmi kladně hodnoceno, pokud absolvoval proces vývoje softwaru i z jiných pozic, jako například IT analytik nebo vývojář. V případě chybějících zkušeností z těchto rolí je však žádané, aby osoba v rámci této pozice měla obchodní a prezentační zkušenosti, dokázala se v krátkém časovém úseku orientovat v dané problematice a byla schopna plánovat kapacitní a rozpočtové zdroje. Mezi absolutní nezbytnosti však patří široká orientace v rámci informačních technologií a vývoje softwaru. V současné době se též kladou vysoké požadavky na měkké dovednosti, jako například schopnost správně delegovat části práce, motivovat podřízené v týmu a schopnost rychle a správně se rozhodnout. Charakteristika této role vychází ze současného procesu vývoje softwaru společnosti XY a nedosahuje výrazných změn.

### Projektový architekt

Projektový architekt je odpovědný za produkt po softwarové stránce, odpovídá za způsob implementace a dodání. Jeho úlohou je v první řadě správně pochopit zadání a vymyslet takové řešení, aby splnilo nejen požadavky pro současnou funkčnost, ale bude také dostatečně bezpečné, nenáročné na operační výkon a později rozšiřitelné nebo upravitelné dle potřeby. Zároveň se předpokládá, že dojde k dodržení bezpečnostních, technologických, finančních a časových omezení. Od projektového architekta se očekává přehled v oblasti CASE nástrojů, jako například enterprise architekt, praktické znalosti s vývojem v roli programátora a zkušenost, jakým způsobem se provádí systémové integrace. Jako samozřejmost se považuje pochopení všech systémových požadavků, které vyvstaly při tvorbě zadání se zadavatelem projektu. V rámci této konfigurace se též očekávají zkušenosti projektového architekta s OOP, (objektově orientovanou architekturou) a SOA (architekturou orientovanou na služby) jak je uváděno v kapitole *2.5.2 Šest nejlepších praktik pro vývoj softwaru*. Tyto požadavky tkví především z výše zmíněných důvodů o přepoužitelnosti jednotlivých komponent v systému. Jeho doplňující úlohou je pomoc při tvorbě technických BRQ, které jsou v rámci business analýzy identifikovány.

### IT Business Analytik

Hlavní úlohou IT business analytika je především pochopit důvody, proč se společnost rozhodla pro danou implementaci a jakého cíle tím chtějí dosáhnout. Tyto informace jsou poskytovány od zadavatele projektu a ostatních zainteresovaných osob. I díky tomu je schopen identifikovat sadu BRQ, neboli obchodní požadavky, které slouží jako vstup pro IT analytika v rámci business analýzy. Důvodem pro zavedení této role je umožnění snazšího verzování obchodních požadavků, jak je uvedeno v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Díky této roli též dochází k odstínění technického řešení, které je zpracováno rolí IT analytika, od nefunkčních požadavků vznesených od zadavatele a zainteresovaných stran.

Je požadováno, aby IT business analytik byl dobře orientován v rámci dané problémové domény nebo si dokázal problematiku v krátké době nastudovat a osvojit. Osoba, která bude roli vykonávat, musí mít výborné komunikační schopnosti. Při konzultacích se zadavatelem je schopen porozumět jeho představám a díky tomu získat explicitní požadavky na vyvíjený systém. Tyto požadavky je následně schopen systematicky zaznamenat a připravit pro předání.

V této konfiguraci lze předpokládat, že společnost užívá rozšířený systém JIRA software od společnosti Atllasian. V současné době se jedná o standard v oblasti vývoje informačních systémů. Právě schopnost správy požadavků pomocí tohoto nástroje je v rámci role IT business analytika očekávána.

### IT Analytik

Role IT analytika, v oblasti vývoje softwaru zaměňovaná za roli datového analytika, se využívá pro analýzu jednotlivých systémů ve společnosti, které jsou v rámci projektu ovlivněny obchodními požadavky. Jak je uvedeno v kapitole *2.5.4 Role, aktivity a artefakty*, jeho hlavní úlohou je analyzování požadavků a jednotlivých procesů, které následně zaznamenává pomocí schématických diagramů. Výstupem této role je IT analýza, která obsahuje vyhodnocení současného stavu systému Následně dle dostupných technologických, finančních a ostatních kapacitních možností, navrhuje budoucí stav modifikovaného systému. Tato analýza se skládá ze třech samostatných artefaktů a to z modelu případu užití, sekvenčního modelu a logického modelu. Tento rozpad zajistí potřebný detail pro přesnější implementaci, jak je uvedeno v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

Samotná analýza se zabývá nejen požadavky, které byly předány jako podklad v BRQ sadě (tato sada obsahuje funkční a nefunkční požadavky), ale také technickým provedením včetně definovaných datových struktur, sekvenčních volání jednotlivých systémů a případu užití. Konzumentem IT analýzy, zpracované pomocí CASE nástroje Enterprise Architect od společnosti Sparx, je role vývojáře, test analytika a projektového architekta, který provádí její validaci a je přímo odpovědný za její obsah.

### Test manažer

Role test manažera je odpovědná za plán testování. Jak je uvedeno v kapitole *2.5.5 Proces,* samotným důvodem pro plánování testování je definovat, jaké komponenty případně systémy se budou testovat, jaký je požadovaný stav (souvisí s akceptačními kritérii) a v jaké míře je nutné provést regresní testování. Jeho úkolem je rozdělení prací na jednotlivé členy testovacího týmu, připrava návrhu harmonogramu testování a stanovení metrik, které budou využity pro zaznamenání průběhu testování. V případě velkých týmů, stejně tak jako v této konfiguraci, je doporučeno tuto roli přiřazovat pouze jedné osobě tak, aby nebyla zainteresována v rolích dalších.

### Test analytik

Role test analytika je odpovědná za kompletní přípravu testování. Důvodem pro zavedení této role v metodice je především zabránění projektové slepotě v oblasti tvorby testovacích scénářů, jak je uvedeno v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Při plánování testování je test analytik kapacitně k dispozici test manažerovi při specifikaci typů testů, revizi odhadů a určení testovacích technik. Očekává se, že osoba v této roli se již setkala s využitím testovacích nástrojů, je obeznámena s metodikou testování softwaru ve společnosti a v případě automatického testování má prokazatelné zkušenosti s programováním a psaním automatických testů.

Jak uvádí v kapitola *2.5.5 Proces,* hlavní úlohou je analýza a revize veškerých podkladů dostupných pro testování. V případě, kdy je v dokumentech nalezena neshoda nebo jakákoliv nesrovnalost, je nutné toto pochybení oznámit autorovi podkladu, který provede okamžitou korekci a případně doplní bližší detail. Po revizi těchto dokumentů test analytikem dochází k identifikaci podmínek, při kterých budou testy prováděny, návrhu jednotlivých testovacích

případů a v neposlední řadě proběhne konfigurace systémů a příprava testovacích dat pro ověření všech případů užití. Výstupem od role test analytika je sada testovacích scénářů, dle kterých může tester ověřit požadované funkčnosti systému.

### Tester

Role testera se v rámci projektu využívá při implementaci testů, kdy je k dispozici test analytikovi pro přípravu testovacích dat a následně ověřuje, zda je pro testování v rámci infrastruktury vše připraveno. U menších projektů je možné roli testera a test analytika sloučit v rámci jedné osoby.

Hlavní úloha testera v této metodice vychází z charakteristiky této role v kapitole *2.5.5 Proces*. Především se jedná o provádění testů dle testovacích scénářů, které byly připraveny test analytikem. V případě automatického testování se využívají k provedení jednotlivých testů připravené skripty. Po jejich provedení se jejich výsledek zaznamená do testovacího protokolu a nahlásí se do systému JIRA případné nalezené defekty. Velkým přínosem je proaktivita jednotlivých testerů v oblastech, které nejsou pokryty připravenými testy.

### Návrhář uživatelského rozhraní

Zavedení této role bylo provedeno v rámci doporučení, která jsou uvedena v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Role návrháře uživatelského rozhraní se v projektu využívá pro vizuální návrhy uživatelského rozhraní. Jedná se tak především o zachycení požadavků na grafické rozhraní, které lze graficky zaznamenat pomocí softwaru na tvorbu prototypů, jako například Axure RP Pro od společnosti Axure Software Solution. Samotnou grafickou tvorbu v rámci aplikace však provádí členové programátorského týmu v rolích vývojářů, nikoli návrhář uživatelského rozhraní. U této osoby je především vyžadováno, aby měla zkušenosti s grafickým designem a dobré grafické cítění. Poskytuje vývojovému týmu grafické podklady pro vývoj, které byly schváleny zadavatelem. Jedná se například o škály použitých barev, design jednotlivých komponent nebo ovládací trend, jakým je nyní palcové menu u mobilních aplikací.

### Developer Team Leader

Jak se uvádí v kapitole *2.5.5 Proces,* role vedoucího skupiny vývoje je odpovědná za programátory ve vývojářském týmu, především pak za méně zkušené kolegy, kterým poskytuje technickou podporu. Jeho hlavní úlohou je rozdělování úkolů mezi vývojáře v týmu, prioritizování jednotlivých požadavků v rámci vývoje a případný způsob řešení implementace dle používané technologie. Dále je plně odpovědný za code reviews, neboli kontrolu kvality kódu, od všech vývojářů v týmu a následné vytvoření buildu, neboli nové verze aplikace.

### Service Owner

Role service ownera je v rámci projektu především pro potřebu znalosti implementace v budoucím provozu. Osoba v této pozici má za úkol připomínkovat způsob provedení implementace z pozice správce produkčního prostředí. Jeho zkušenosti s údržbou aplikace v provozu jsou nápomocny pro výběr optimálního řešení.

### Akviziční analytik

Role akvizičního analytika se v závěru projektu využívá pro vytvoření dokumentace, která je předávána jako příručka technikům řešícím incidenty v produkčním prostředí.

**Vývojář**

Osoba, která se nachází v roli vývojáře, je v průběhu projektu zodpovědná za implementaci technických požadavků a způsobu užití jednotlivých softwarových komponent tak, jak bylo v rámci IT analýzy navrženo. Nutné je také dodržet architekturu, která byla navržena v Solution Screeningu. Při každém nasazení nové funkcionality musí vývojář provést testy, které potvrdí správné sloučení kódu z jednotlivých vývojových větví.

Základním požadavkem na znalost vývojáře je programovací jazyk, ve kterém je daný systém napsán. Dále je v rámci této konfigurace vyžadováno, aby byl vývojář znalý jazyka UML, který se v rámci projektů využívá pro modelování technické dokumentace v průběhu analýzy. Tato dokumentace je technickým zadáním pro implementaci systému. Předpokládá se také znalost technologií a programovacích jazyků souvisejících s platformou, která bude implementovaný systém využívat.

### 3.3.1 Artefakty v rámci projektu

V průběhu projektu dochází k tvorbě a modifikaci jednotlivých artefaktů. Za každý z artefaktů nese odpovědnost určitá role, v případě nutnosti postupu artefaktu schvalovacími procesy i rolí více. Detailní popis jednotlivých artefaktů, i rolí za ně odpovídajících, je uveden u každého artefaktu v následujících odstavcích.

**Projektový plán**

Odpovědná osoba za projektový plán je nehledě na aktuální fázi projektový manažer. Mimo projektového manažera se podílí na jeho vypracování také zadavatel, business analytik a projektový architekt. Kromě odhadů, díky kterým je možné vytvořit hrubý projektový plán, je jejich další činností ve fázi zahájení definovat akceptační kritéria projektu.

Jak je uvedeno v kapitole *2.5.5 Proces*, pro úspěšné dokončení projektu je nezbytné, aby projektový plán obsahoval detaily pro současnou a následující fázi včetně jednotlivých kroků v těchto iteracích. Konkrétně pro realizační fázi se silně doporučuje detailně naplánovat minimálně dvě iterace, aby byla zajištěna plynulá integrace bez možných problémů při přechodu mezi jednotlivými prostředími.

Plán iterací obsahuje detailní popis jednotlivých artefaktů, činností a kritérií nutných pro úspěšné dokončení iterací. Oproti tomu plán fáze vyobrazuje, kolik iterací je nutné absolvovat pro úspěšné dokončení dané fáze. Dále jsou v něm zaznamenány termíny iterací a kritéria pro splnění jednotlivých milníků.

Povinnosti projektového manažera v rámci tohoto artefaktu lze shrnout do tohoto seznamu:

- stanovení cíle a jednotlivé úkoly, které je nutné v rámci současné fáze a iterace splnit
- určení přibližné délky projektu a časový odhad doby nutné pro fázi zahájení
- definování úkolů a osob za ně odpovědných ve fázi zahájení včetně zajištění alokace
- stanovení akceptačních podmínek a milníků pro jednotlivé fáze
- zajištění alokace zdrojů pro následující fáze v rámci časového harmonogramu
- stanovení rizik pro následující fázi projektu
- průběžná kontrola reálného stavu projektu oproti harmonogramu času a práce
- řízení projektu na denní bázi, včetně kontroly, zdali byly dosaženy stanovené milníky

Jak je z následující charakteristiky zřejmé, cílem tohoto artefaktu v rámci konfigurace metodiky Rational Unified Process je zachycení hrubého projektového plánu, který bude možné

v průběhu celého projektu dále zpřesňovat. Mimo jiné artefakt přehledně informuje o stavu, v jakém se projekt v současnosti nachází a vyhodnocuje jeho hotové části. Díky těmto informacím je možné projektovému týmu a zainteresovaným osobám vykreslit stav projektu a jeho plánovaný postup.

**Akceptační protokol**

Projektový manažer je na konci každé iterace povinen akceptovat její stav u zadavatele projektu. V rámci fáze zahájení se jedná především o akceptování harmonogramu, alokačních odhadů jednotlivých osob ve všech fázích projektu a celkových nákladů ze strany externích dodavatelů. Jedná se však pouze o odhady, které jsou následně upřesněny po vypracování Solution Screeningu ve fázi projektování.

V průběhu ostatních fází tento protokol obsahuje především seznam defektů a přípustnou pass rate, neboli dovolený počet chyb dle jejich závažnosti. Jejich dopad a závažnost pro projekt stanovuje test analytik po konzultaci s projektovým manažerem. Tento seznam chyb je doplněn případnými změnovými požadavky, které z nich mohou vyplývat. Samotný artefakt je následně vytvořen vždy na konci každé iterace. Tyto akceptační protokoly jsou po skončení projektu využity pro takzvaný lessons learned, která probíhá za účasti všech osob na projektu. Díky tomu lze následně, ale i v průběhu projektu, hodnotit stav a úspěšnost projektu jako takového. Tímto krokem podpoříme splnění doporučení o poučení z projektů dle kapitoly *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

**Hlavní případ užití včetně vize projektu**

Artefakt, který se v rámci projektu označuje jako Core Use Case, představuje hlavní účel a přidanou hodnotu celého projektu. Kromě detailního popisu je vhodné připravit také grafickou notaci, která bude srozumitelná všem zainteresovaným stranám. Odpovědná osoba je zde business architekt, který odpovídá za jeho správný výklad a souvztažnost s businessovými strategiemi v podniku. Schválení tohoto artefaktu zadavatel je nezbytné pro pokračování projektu do dalších fází. Hlavní případ užití je též podstatný pro dokončení Solution Screeningu ve fázi projektování a nezbytný pro finalizování obchodních požadavků. Ve většině případů jsou tyto tři artefakty tvořeny souběžně s tím, že Core Use Case je pouze zpřesňován a drobné úpravy mají menší dopad na zbylé artefakty. Pokud však dojde k výrazným změnám v průběhu jakékoliv fáze, následkem bude časový posun dokončení projektu, alokační vícenáklady a nutnost revidovat harmonogram projektu včetně vzniklých artefaktů.

**Obchodní požadavky**

Artefakt obchodní požadavky, označován též zkratkou BRQ, je přímo odpovědný IT business analytik. Hlavní úlohou tohoto artefaktu, jak se uvádí v kapitole *2.5.5 Proces,* je explicitní zaznamenání všech obchodních a technických požadavků tak, aby na jejich základě bylo možné provést odhady a plány pro samotný proces vývoje systému. Tyto BRQ mají následující složení:

1) jednoznačný identifikátor a název požadavku v rámci projektu;
2) priorita požadavku;
3) riziko a následný dopad při neimplementování požadavku;
4) informaci, kdo tento požadavek požaduje (např. zadavatel, uživatel, právní oddělení...);
5) detailní popis požadavku;
6) akceptační kritérium požadavku.

Takto zaznamenané požadavky, pro následující použití ideálně připravené v programu Enterprise Architect pro využití IT analytikem, musí být dle priorit do konce fáze zahájení finalizovány. Výjimku mají ty požadavky, které nejsou prioritní, nemají vysoké riziko dopadu a nezasahují do softwarové architektury schvalované pomocí dokumentu Solution Screening. U těchto požadavků může dojít ke zpřesnění na počátku následující fáze.

Na počátku fáze projektování je nutné mít schválené veškeré obchodní požadavky včetně technických, které mohly vyplynout z průběžného upřesňování zadání při vypracování artefaktu Solution Screening a Core Use Case. Nicméně se i přesto doporučuje zajistit plnou kapacitu role business analytik pro celou dobu této fáze z důvodu možných změnových požadavků při tvorbě IT analýzy. Díky tomu je zajištěno verzování, které bylo na základě charakteristiky projektu ve společnosti XY uvedeno v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. Ve fázi realizace by však již ke změnám nemělo docházet. Výjimkou je úprava na základě změnového požadavku nebo nemožnosti dodržení části zadání z důvodu technologických bariér. Tyto informace jsou zpravidla zjištěny v průběhu jednotlivých iteračních testování aplikace a jsou zaznamenány do akceptačního protokolu.

**Rizika projektu**

Obsahem tohoto artefaktu jsou veškerá rizika projektu, kterých si je projektový tým vědom ve fázi počátku, a které jsou dle doporučení v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu* dále v průběhu následující fází aktualizována. Díky tomu je možné uzpůsobit plán projektu tak, aby byla rizika dle závažnosti postupně odstraněna nebo naopak došlo k jejich akceptaci. Cílem artefaktu je tímto přístupem dojít ke snížení negativních důsledků na projekt. Tento artefakt je projektovým manažerem vytvořen na počátku této fáze, nicméně v průběhu projektu dochází k jeho modifikaci, které se účastní celý projektový tým. Jak se uvádí v kapitole *2.5.2 Šest nejlepších praktik pro vývoj softwaru*, je potřebné u jednotlivých rizik uvést stupeň a možnou prevenci rizika.

Rizika lze dělit několika způsoby. V rámci projektu můžeme mít rizika technologická, kdy například v průběhu implementace můžeme zjistit, že technologie na zvolené platformě nefunguje dle našich očekávání nebo v rámci zvolené technologie nelze provést validní úpravu, která je žádaná v rámci změny od zadavatele. Dále se můžeme setkat s riziky obchodními, které mohou souviset se spoluprací u externích dodavatelů nebo rizika vnější, u kterých nemůžeme zajistit stoprocentní eliminaci. Jedná se tak například o často zmiňovanou změnu legislativy (např. GDPR, neboli ochrana osobních údajů).

**Solution Screening**

Artefakt Solution Screening se zaměřuje na návrh architektury využívaných systému a jednotlivých rozhraní, které jsou v rámci projektu použity. K tomuto přehledu slouží takzvaný Big Picture, který slouží pro zobrazení veškerých systému používaných na projektu pomocí jednoho přehledného diagramu. V průběhu celého projektu je dokumentace zpracovávána projektovým architektem a po jejím dokončení validována v rámci schvalovacího řízení enterprise architektem. Díky tomuto kroku je zajištěno dodržení jednotné podnikové architektury, jak se uvádí v kapitole *2.5.2 Šest nejlepších praktik pro vývoj softwaru*. Dále dojde k doporučené kontrole čtyř očí, jak se uvádí v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu* čímž se sníží šance vniku chyb a nepřesností.

Hlavním cílem tohoto artefaktu je zachycení závislosti jednotlivých systémů a seznam využitých služeb, které budou pro jejich vzájemnou komunikaci nezbytné. Architektura stanovuje jaké technologie a frameworky, které budou při implementaci systému použity.

Stanovuje též jaké postupy a pravidla nezbytné pro úspěšné dokončení. Artefakt musí obsahovat podrobné informace o databázovém prostředí a dále příručku, sloužící k nasazení a následné instalaci produktu. Počátek tvorby tohoto dokumentu nastává ve fázi zahájení, konkrétně po předběžném schválení artefaktu Core Use Case zadavatelem. Dokončení architektury implementovaného systému nastává ve fázi projektování, kdy na tento artefakt navazuje IT analýza, díky čemuž do projektu přináší potřebný detail a doplňuje technickou dokumentaci, která je nezbytná pro vývojářský tým.

**Změnový požadavek**

Změnový požadavek je ve fázi projektování využit pro případnou změnu v BRQ setu, pokud k jeho vyžádání dojde již po schválení obchodních požadavků zadavatelem. Vytvoření tohoto artefaktu provádí role projektový manažer, který společně se zadavatel definuje znění. Následně jej předá IT analytikovi případně projektovému architektovi k prověření, jak velký dopad na současné artefakty bude změna mít. Po tomto kroku dojde k předložení výsledků zadavateli a dle odhadnuté náročnosti na provedení změny dojde ke schválení požadavku případě jeho modifikaci či úplnému zrušení. Díky tomuto artefaktu je možné provést korektní verzování jednotlivých požadavků, jak je požadováno v rámci doporučení v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

**Plán testování**

Plán testování v rámci projektu určuje, jakým způsobem bude provedeno testování implementovaných případů užití. Dále stanovuje, jakým způsobem dojde ke zpracování výsledků testů a určuje ukazatele, dle kterých proběhne jejich vyhodnocení. Plán testování spravuje v průběhu projektu test manažer. Testovací strategii je nutné začít určovat již ve fázi zahájení. Je tomu tak především z důvodu možnosti využití automatického testování, případně SOAP testů, které umožňují testování jednotlivých služeb, nehledě na rozpracované části systému.

**Drátěný model nebo prototyp**

Artefakt ve fázi zahájení slouží jako podklad pro schválení grafické části aplikace zadavatelem projektu. Jak je uvedeno v doporučeních v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*, doporučuje se vypracování několika typů grafického návrhu, aby fáze schvalování proběhla v co nejkratším čase. Nejvhodnější grafický počin bude následně zadavatelem zvolen a dále rozvíjen. V tomto ohledu se k přiblížení vyvíjené skutečnosti používají drátěné modely, které jsou následně obohaceny o designové prvky nebo v doporučovanější variantě klikatelný prototyp. Při důsledném zpracování lze tento prototyp 1 : 1 považovat za budoucí vzhled aplikace. Neobsahuje však žádné funkcionality, nabízí pouze průchody mezi jednotlivými obrazovkami, které lze dle nutnosti doplnit specifickou interakcí. Nedoporučuje se však využívat alokaci návrháře uživatelského rozhraní pro zbytečný detail, jelikož bude prototyp po ukončení vývoje zahozen.

V průběhu finalizace schvalování je možné průběžně připravit grafickou dokumentaci na detailní úrovni pro vývojáře aplikace. Tato dokumentace je součástí prototypu nebo drátěného modelu a obsahuje přehled odstínů a kódů barev dle HTML[4] a CSS[5]. Díku tomu je možné provést přesnou definici barev dle odstínu. Hlavním milníkem ve fázi zahájení je tedy

[4] Značkovací jazyk pro hypertex
[5] Šablony kaskádových stylů

schválený prototyp aplikace. Výstupem z projektové fáze je dále grafický návrh obohacený o přehled odstínů a kódu použitých barev na jednotlivých obrazovkách.

### Model případů užití

Model případů užití je první artefakt, který je výstupem práce role IT analytika. Jeho schválení zadavatelem je nutné pro budoucí vypracování sekvenčního a logického modelu, který se z tohoto artefaktu odvíjí. Vypracování je možné na základě kompletního BRQ setu, který upřesňuje jednotlivé scénáře užití. Forma modelů případů užití je na zvyklostech organizace. Jako vhodnější forma se doporučuje zvolit diagram aktivit, který je v případě většího množství alternativních případů či chybových stavů pro zadavatele přehlednější než textová formaa je možné provádět jednodušší prezentaci. Toto představení lze kombinovat s prototypem aplikace a tím umožnit zadavateli lepší představu o uživatelském rozhraní, jak je uvedeno v doporučeních kapitoly *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*. V průběhu fáze projektování je nezbytné všechny modely případu užití schválit zadavatelem projektu. Jelikož je diagram aktivit úmyslně oproštěn od technikálií, je možné jeho schválení i méně technicky zdatnou osobou v roli zadavatele. Zbylé artefakty IT analýzy již schvalování zadavatele nepodléhají.

### Sekvenční model

Artefakt sekvenčního modelu je výstupem práce role IT analytika. Jak uvádí kapitola *2.5.5 Proces,* vychází z architektury obsažené v Solution Screeningu a detailně pomocí diagramu vyobrazuje sekvenci volání mezi jednotlivými systémy. Těmito voláními jsou myšleny zprávy mezi objekty, obsahující technické informace nepodstatné pro uživatele, ale též zobrazovaný textový obsah jako například textové popisy jednotlivých účtů v internetovém bankovnictví. U každého volání, které je zobrazeno na sekvenčním diagramu, je přiloženo plnění jednotlivých parametrů v mapovacím rozhraní služby. Na konci projektové fáze dochází ke schválení sekvenčního modelu projektovým architektem, který validuje použité systémy a služby. Výstupem je schválený model, který je použit jako podklad pro vývojáře ve fázi implementace. V následující fázi nejsou očekávány do tohoto modelu výrazné zásahy, vyjma změn schválených pomocí změnového požadavku.

### Logický model

Logický model se v IT analýze používá pro vyobrazení atributů a operací, které jsou využity v rámci jednotlivých případů užití. Je oproštěn od jakéhokoliv grafického náhledu. Atributy, které jsou v modelu zobrazeny v logických obrazovkách nebo dílčích logických obrazovkách, slouží pro propojení rozhraní služby s reálně plněnými daty v jednotlivých voláních. Tímto způsobem lze dojít k přehledu všech užitých parametrů, ať se již jedná o uživatelský vstup či technickou konstantu v konkrétním volání služby. Podobně jako u sekvenčního modelu, nejsou v následující fázi očekávány do tohoto modelu výrazné zásahy, vyjma změn schválených pomocí změnového požadavku.

### Implementovaný kód

Ve fázi realizace dochází ke zpracování IT analýzy vývojářským týmem a následnému naprogramování aplikace dle zadání. Každý samostatný případ užití je vhodné zadat jednomu programátorovi, který jej bude v průběhu vývoje verzovat a dále dle potřeby rozvíjet. Jak je uvedeno v kapitole *2.5.2. Šest nejlepších praktik pro vývoj softwaru*, díky použití SOA je vhodné jednotlivé části kódu programovat jako komponenty, které je možné v rámci

implementace spojit a tím zajistit jejich možnou přepoužitelnost v jiných částech aplikace. Výstupem vývojáře je tedy kód aplikace, který je uložen na jeho lokálním prostoru. Při potřebě vytvořit nový build aplikace dochází k mergi kódu, kdy jednotliví vývojáři svůj kód nahrají do nástroje pro automatickou správu kódu. Před úspěšným nahráním kódu do hlavní větve je nutné provést code reviews. Tato kontrola je prováděna developer team leaderem případně leaderem za platformu či programovací jazyk.

Ve fázi nasazení je artefakt dokončený a implementovaný kód již není dle projektového plánu nutné měnit. Výjimka může nastat v případě, kdy je nutné provést opravu kritické chyby v rámci pilotního testování nebo je vytvořen změnový požadavek.

### Build

Artefakt build lze chápat jako plně použitelnou verzi aplikace, kterou lze podrobit testování případně performačním testům. Odpovědnost za tento artefakt opět přebírá developer team leader nebo leader za platformu či programovací jazyk. Společně s každým buildem je přiložena průvodka změn a způsob, jakým má být nasazen. Buildy se provádí dle potřeby projektu, nejsou tudíž spjaty s jednotlivými iteracemi.

### Testovací scénáře

Na počátku fáze se dle průběhu implementace a testovacího plánu validuje, jaké scénáře užití se budou testovat prioritně. Toto ověření je nezbytné provést především v první iterační fázi, tedy na integračním prostředí. I přes stanovenou prioritu při vývoji může dojít k situaci, kdy scénář užití s vysokou prioritou není dokončen a je nutná změna plánu testování. V případě SOAP testů není nutné prioritu měnit, pouze zaznamenat nefunkčnost daných služeb a po jejich dokončení provést následné otestování. Jak se uvádí v doporučeních kapitoly v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*, správně naplánované SOAP testy mohou zavčasu odhalit případné chyby v rámci jednotlivých služeb.

### Testovací protokol

Testovací protokol je nezbytný pro fázi realizace. Jedná se o artefakt, který napomáhá projektovému manažerovi a zadavateli vykonat rozhodnutí, zda je termín dodávky aplikace dle projektového plánu reálný. Hlavními milníky bývají takzvané průchozí body, které požadují k předem definovanému datu procentuální protestovanost a funkčnost. Pokud tyto podmínky nejsou splněny, je nutné individualizovat termíny průchozích bodů a tuto skutečnost zaznamenat jak do testovacího protokolu, akceptačního protokolu, ale také do projektového plánu. V případě nesplnění podmínek při přechodu mezi jednotlivými iteracemi přes individuální plán, tedy při nasazování aplikace do akceptačního prostředí, dochází k automatickému posunu releasu do produkce a celkové změny harmonogramu.

### Servisní příručka

Za artefakt servisní příručka je přímo odpovědný analytik akvizice. Hlavním cílem tohoto dokumentu je systematický popis nových funkcionalit, které budou v rámci nasazení aplikace do produkce nově uvedeny do funkčnosti. Artefakt je vypracován na konci fáze realizace, kdy je již aplikace úspěšně nasazena v akceptačním prostředí a nepředpokládají se další výrazné zásahy do funkčnosti. Hlavním obsahem dokumentu jsou především artefakty, které byly vypracovány IT analytikem. Konkrétně se jedná o sekvenční model, logický model a model případu užití. Tyto artefakty jsou akvizičním analytikem doplněny o průvodku, sloužící jako

manuál pro řešení chyb, které díky nové implementaci mohou být na produkčním prostředí řešeny.

**Dokument o ukončení projektu**

Obsahem tohoto artefaktu je záznam o oficiálním ukončení projektu. Potvrzením projektového manažera a zadavatele dojde k formálnímu ukončení projektu. Podmínkou pro tento krok je kromě implementované aplikace též kompletní dokumentace, která obsahuje seznam a detailní popis veškerých artefaktů použitých v projektu. Zadavatel odpovídá za jeho akceptaci a v případě nutnosti jsou u jednotlivých artefaktů uvedeny připomínky.

**Poučení z projektu**

Tento artefakt, v IT branži nazýván jako lessons learned, je využívám nejen pro poučení a budoucí se vyvarování chyb vzniklých v projektu, ale také zároveň pro zaznamenání úspěchů a funkčních postupů, které lze aplikovat u jiných projektů v organizaci. Jak se uvádí v kapitole *2.5.5 Proces,* dokument by měl být správně strukturovaný a obsahovat věcné informace související s jednotlivými činnostmi, artefakty či rolemi v rámci projektu.

Je vhodné, aby každé takzvané poučení obsahovalo popis oblasti, které se týká (například analýza, integrace, komunikace v rámci projektu a další). Pokud je toto poučení katalogizováno pomocí softwaru, je vhodné zadávat výstižné názvy zkoumaného problému či úspěchu včetně klíčových slov, pomocí kterých lze provést hledání. Klíčovým bodem je následný popis události, která byla v průběhu projektu problémová nebo naopak přínosná. Následně je nutné charakterizovat, jaké dopady (časové, finanční nebo jiné) tato událost měla. Poté zbývá formulovat opatření, které v průběhu příštích projektů zabrání opakování chyby nebo díky tomuto doporučení bude možné proběhlý úspěch aplikovat, jak je uvedeno v rámci doporučení v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

## 3.4 Případová studie za použití konfigurace metodiky RUP

Cílem této případové studie je dokázat na ukázkovém projektu využití hlavních principů konfigurace metodiky Rational Unified Process. Zhodnocení průběhu projektu probíhá již zpětně, kdy byl úspěšně projekt dokončen a díky tomu je možné detailně popsat jednotlivé projektové fáze a iterace, které obsahuje. Případová studie nepopisuje projekt do úplných detailů, ale představuje vývoj softwaru v bankovní společnosti včetně jednotlivých rolí, artefaktů a aktivit, které jeho proces ovlivňují. Cílem projektu řešeného v této případové studii bylo implementovat novou funkci do mobilního bankovnictví, konkrétně možnost klientovi umožnit změnu svých osobních údajů bez nutnosti návštěvy pobočky banky. Projekt probíhal ve druhém pololetí roku 2019 tak, aby funkcionalita mohla být uvolěnana do provozu v termínu 30. 11. 2019. Nová funkčnost funguje na základě naskenování občanského průkazu pomocí telefonu. Tím pádem je možné provést kontrolu pravosti udávaných údajů a umožnit jejich změnu a případné pozdější vyhodnocení pravosti. Důvodem pro implementaci této funkčnosti je usnadnění změny povinných osobních údajů, které jsou nutné například při žádosti o půjčku, hypotéku či využití cestovního pojištění. Doposud tato změna závisela na osobní návštěvě bankovní pobočky, což z pohledu klienta není preferovaná varianta. Vyvíjená funkcionalita má tedy potencionálně přímé dopady na obchodní funkce aplikace a proto je projektu přisuzována vysoká priorita.

### 3.4.1 Fáze zahájení

Zadavatelem této změny byl člen vyššího managementu firmy, konkrétně z úvěrové sekce společnosti. Jeho hlavním požadavkem a očekáváním bylo navýšení prodejnosti produktů na mobilní platformě oproti ostatním bankovním institucím, které tuto automatickou funkci v mobilním bankovnictví nemají. Při představení požadovaných funkčností zadavatelem business architektovi a projektovému manažerovi došlo k obecné specifikaci požadavků. Tyto schůzky proběhly v několika v termínech a díky tomu bylo možné upřesnit zadání a odpovědět na otázky vzešlé z minulých setkání. Cílem těchto setkání bylo získat hlavní případ užití společně s vizí projektu a co nejlepší představu o tom, co od nové funkčnosti zadavatel očekává a zda jsou tyto předpoklady reálné. Jelikož společnost provádí nasazení nových funkcionalit v rámci produkčního releasu třikrát ročně, bylo stanoveno datum počátku projektu na 1. 7. 2019. Ukončení fáze zahájení se předpokládá 19. 7. 2019.

Jako další krok následovalo sestavení projektového týmu, což je úloha projektového manažera na projektu. Mezi první obsazené pozice patřila role IT business analytika a projektového architekta. Projektový architekt v této fázi analyzoval obecné požadavky získané na počátku této fáze a s konzultační pomocí business architekta určil hlavní rizika projektu vzhledem k bankovní architektuře systémů. V pozdější části této analýzy byly již k dispozici obchodní požadavky vypracované IT business analytikem. Bylo tudíž možné zahájit konzultace u enterprise architekta ohledně využití jednotlivých systémů ve vztahu k obecným požadavkům na projektu. Dále proběhla konzultace s ostatními zainteresovanými osobami, především pak se zástupcem bezpečnostního oddělení společnosti, který provedl zhodnocení prvního návrhu architektury. Účelem této schůzky bylo získat doporučení, na které konkrétní volání aplikace směrem k bankovních systémům se zaměřit, kvůli možným bezpečnostním rizikům.

IT business analytik v průběhu této fáze vedl schůzky se zadavatelem projektu a ostatními zainteresovanými stranami. Získané informace následně strukturovaně uspořádal a vytvořil seznam obchodních požadavků s jejich cíli, které v průběhu následujících sezení blíže specifikoval a snažil se o co největší upřesnění. Struktura požadavků vychází z metodiky kapitola *3.4 Fáze zahájení*. U některých požadavků je finální dokončení plánováno v počátcích fáze projektování.

Níže je uveden příklad požadavků, které byly v rámci fáze zahájení vypracovány. Obchodní požadavky jsou technologicky obecné a jejich typ (funkční či nefunkční požadavek) je následně rozlišen IT analytikem v následující fázi. Níže se konkrétně jedná o funkční požadavek určující hlavní záměr projektu:

| | |
|---|---|
| **Identifikátor a název požadavku:** | *ZOU001 – Modul pro změnu osobních údajů* |
| **Priorita požadavku:** | *1 (nejvyšší)* |
| **Riziko požadavku:** | *1 (blokační)* |
| **Zadavatel požadavku:** | *Obchodní oddělení, J. N.* |
| **Popis požadavku:** | *Jako zadavatel požaduji, aby v rámci implementované funkčnosti bylo možné provést změnu osobních údajů pomocí občanského průkazu klienta z mobilní aplikace. Vstup do této funkčnosti je možné beze změn přepoužít v různých částech aplikace (např. v uvěrovém procesu, pojistném procesu…).* |
| **Akceptační kritérium:** | *Funkční proces změny osobních údajů, který je možné využít napříč aplikací bez nutnosti úprav bankovních systému.* |

Projektový manažer následně v tomto období průběžně aktualizoval projektový plán na základě výstupů projektového architekta a IT business analytika. Projektový plán pro fázi zahájení obsahoval tyto aktivity a role za ně odpovědné:

- vypracovat hlavní případ užití (odpovědná role business architect);
- vytvořit harmonogram projektu a plán prvních iterací ve fázi projektování (projektový manažer);
- sběr obchodních požadavků a stanovení jejich akceptačních kritérií (business analytik);
- identifikovat rizika v rámci celého projektu (projektový tým);
- zajistit alokace pro následující fáze projektu (projektový manažer);
- definovat úkoly a odpovědné osoby pro následující fáze (projektový manažer).

Projektový tým na naplánované schůzce definoval rizika, která mohou projekt ohrozit. Projektový manažer následně vytvořil artefakt rizika projektu a společně s týmem provedl jeho validaci. Odpovědnost za artefakt však zůstává na projektovém manažerovi, nikoli na jednotlivých členech týmu. Každá položka v tomto artefaktu obsahovala popis rizika, stupeň rizika (nízké, střední nebo vysoké) a způsob jakým bude riziko ošetřeno. Níže je výčet rizik identifikovaných pro tento projekt:

1) **Detail rizika:** *Projekt „Změna osobních údajů“ je závislý na projektu „Těžba a zpracování dat“, který se přímo zabývá získáváním strojově čitelných dat. Pokud projekt nebude dokončen, hrozí výrazné zpoždění a vícenáklady.*

   **Stupeň rizika:** *Vysoké*

   **Prevence rizika:** *V průběhu projektu bude docházet k pravidelným schůzkám projektových manažerů těchto projektů. V případě potřeby bude alokace analytiků a vývojářů z projektu „Změna osobních údajů“ suplovat nedostatečné zdroje projektu „Těžba a zpracování dat“. Tímto krokem je možno lépe kontrolovat současný stav na projektech a případně zajistit snížení dopadu tohoto rizika.*

2) **Detail rizika:** *Výrazný počet klientů, kteří budou provádět změnu osobních údajů současně, může způsobit přetížení systému a výrazně zpomalit plynulost procesu v aplikaci.*

   **Stupeň rizika:** *Střední*

   **Prevence rizika:** *Ve fázi realizace budou provedeny performační testy. V případě nedostatečného operačního výkonu dojde k optimalizaci systému pro nahrávání dokumentů, případně k výkonnostnímu posílení hardwaru.*

3) **Detail rizika:** *Pro vývoj této funkcionality je nutná spolupráce s externím dodavatelem. Není možné reálně kontrolovat práci na straně dodavatele a případně zabránit možným zpožděním na jejich straně.*

   **Stupeň rizika:** *Nízká*

   **Prevence rizika:** *V průběhu fáze realizace je vhodné zvolit menší dodávky od externího dodavatele a v rámci pravidelných setkání (přibližně 3x týdně) validovat stav vývoje.*

Při skončení fáze zahájení již měl projektový manažer dostatečné množství informací pro sestavení hrubého harmonogramu, který bude přiložen do plánu projektu:

**Fáze projektování**

Plánovaná délka fáze projektování je 7 týdnů s počtem dvou iterací. Plánovaná délka první iterace je 5 týdnů, druhé iterace zbylé 2 týdny.

**Iterace 1** (22. 7. 2019 – 23. 8. 2019)

- dokončení solution screeningu;
- počátek implementace architektury včetně konfigurace systémů;
- vypracování grafického prototypu;
- vypracování IT analýzy.

**Iterace 2** (26. 8. 2019 – 6. 9. 2019)

- validace IT analýzy;
- schválení vizualizace dle grafického prototypu;
- předání a validace IT analýzy s externím dodavatelem;
- dokončení implementace architektury včetně konfigurace systémů;
- příprava integračního prostředí;
- dokončení testovacích scénářů;
- příprava servisní příručky.

**Fáze realizace**

Plánovaná délka fáze projektování je 9 týdnů s počtem dvou iterací. Délka první iterace, která se zabývá úspěšným nasazením a otestováním aplikace v integračním prostředí je 5 týdnů, druhá iterace probíhající v akceptačním prostředí trvá zbylé 4 týdny.

**Iterace 1** (9. 9. 2019 – 11. 10. 2019)

- využití IT analýzy pro účely integrace;
- implementace aplikace v integračním prostředí;
- testování aplikace s 50% úspěšností testovacích scénářů;
- akceptace aplikace pro přechod na akceptační prostředí;
- příprava akceptačního prostředí.

**Iterace 2** (14. 10. 2019 – 8. 11. 2019)

- implementace aplikace v aplikačním prostředí;
- testování aplikace se 100% úspěšností testovacích scénářů;
- performační testování aplikace a dotčených systémů;
- vyhodnocení testovacího protokolu;
- předání servisní příručky service ownerovi;
- akceptace aplikace pro nasazení do produkčního prostředí.

**Fáze nasazení**

Plánovaná délka fáze projektování jsou 3 týdny s počtem dvou iterací. Délka první iterace je 2 týdny, druhá iterace zbylý 1 týden.

**Iterace 1** (11. 11. 2019 – 22. 11. 2019)

- nasazení aplikace do produkčního prostředí;
- vyhodnocení testování z hlediska pilotní verze aplikace;
- akceptace dodávky aplikace;
- připravení aplikace pro uvolnění klientům.

**Iterace 2** (25. 11. 2019 – 29. 11. 2019)

- oprava nekritických chyb pomocí hotfixu;
- uvolnění aplikace pro všechny uživatele.

Na konci fáze zahájení bylo zadavatelem a projektovým manažerem rozhodnuto, že je projekt proveditelný a je možné začít s vypracováním IT analýzy, která bude podkladem pro vývoj aplikace. Oficiální potvrzení od zadavatele bylo provedeno pomocí akceptačního protokolu a projekt byl posunut do fáze projektování.

### 3.4.2 Fáze projektování

V počátku fáze projektování došlo k dokončení artefaktu solution screening, který obsahoval návrh architektury. Dále probíhalo finální upřesnění požadavků s nízkým dopadem na ostatní artefakty. Souběžně byla započata práce na jednotlivých částech analýzy, konkrétně na modelu případu užití, který je nutné validovat u zadavatele v průběhu první iterace před tvorbou sekvenčního a logického modelu.

Na počátku první iterace vznikl grafický prototyp, který byl po odsouhlasení zadavatelem následně ve fázi realizace použit jako podklad pro vývoj uživatelského rozhraní. Po validaci artefaktu modelu případu užití, došlo IT analytikem k tvorbě sekvenčního a logického modelu. Průběžně s těmito pracemi došlo test manažerem k vypracování plánu testování. Jeho validování záviselo na dokončení solution screningu díky kterému bylo možné potvrdit, že testování bude prováděno manuálně bez pomoci automatizovaného testování. Po tomto rozhodnutí bylo možné test analytikem vytvořit sadu testovacích scénářů, které byly následně po dobu této fáze upravovány dle modifikací vzniklých v IT analýze. Zaměření těchto scénářů bylo především na jednotlivé funkčnosti u každé obrazovky tak, aby bylo možné objektivně zhodnotit procentuální funkcionalitu aplikace dle počtu úspěšně provedených scénářů.

Vzhledem k rozšiřování aplikace, byla základní architektura systému již funkční a bylo ji pro tento projekt možné pouze obohatit o systémy, které dosud nezahrnovala. Jednalo se především o napojení na komponentu zajišťující těžbu dat z dokumentu nahraného na servery banky. Ostatní systémy již byly v rámci architektury aplikace využívány, proto došlo pouze k navýšení počtu služeb, které aplikace konzumovala. Seznam těchto služeb a jejich konkrétní využití je uveden níže:

**getCheckChange@BUS** – služba se využívá pro zjištění, zda klient může měnit osobní údaje;

**postIdentityCard@BUS** – služba pro nahrání dokumentu do systému banky;

**getExtractionData@DB** – služba pro získání vytěžených dat z nahraného dokumentu;

**postConfirmData@BUS** – služba zajišťující validaci vytěžených dat uživatelem aplikace.

Na základě těchto informací uvedených jak v artefaktu solution screening, tak v sekvenčním modelu, mohl vývojář systému @BUS a @DB povolit využívání rozhraní těchto služeb. Toto nastavení bylo konfiguračního charakteru.

Na počátku druhé iterace proběhlo schválení sekvenčního a logického modelu projektovým architektem. Validací u zadavatele též prošel grafický prototyp aplikace. Díky tomu mohlo dojít k předávací schůzce pro externího dodavatele, které se účastnil celý projektový tým. V rámci této schůzky došlo k odevzdání artefaktů IT analýzy a grafického designu, nutných pro vývoj aplikace. Následně do konce druhé iterace došlo k dokončení implementace architektury, přípravě integračního prostředí a sběru dokumentů pro servisní příručku. Pro snížení rizik na projektu byla provedena jejich aktualizace, níže jsou uvedena nová a modifikovaná rizika:

3) **Detail rizika:** *Pro vývoj této funkčnosti je nutná spolupráce s externím dodavatelem. Není možné reálně kontrolovat práci na straně dodavatele a případně zabránit možným zpožděním na jeho straně.*

   **Stupeň rizika:** *Riziko bylo odstraněno*

   **Prevence rizika:** *Externímu vývojovému týmu aplikace byla přidělena projektová místa v budově společnosti. Dále byla určena osoba, která se bude za stranu dodavatele účastnit pravidelných projektových schůzek a informovat o stavu vývoje.*

4) **Detail rizika:** *Při neplánovaných změnách v logice nahrávání a těžbě dokumentu hrozí nutnost rozšíření služeb.*

   **Stupeň rizika:** *Střední*

   **Prevence rizika:** *Toto riziko nelze patřičně ošetřit. Při nutnosti úprav lze vytvořit novou verzi služby a tím snížit náklady pro vytvoření nového rozhraní.*

Projektový manažer provedl na konci fáze upřesnění pro jednotlivé iterace ve fázi realizace:

**Iterace 1** (9. 9. 2019 – 11. 10. 2019)

- využití IT analýzy pro účely integrace *(9. 9. 2019 – 20. 9. 2019)*;
- implementace aplikace v integračním prostředí *(9. 9. 2019 – 20. 9. 2019)*;
- testování s 50% úspěšností testovacích scénářů *(20. 9. 2019)*;
- akceptace aplikace pro přechod na akceptační prostředí *(4. 10. 2019)*;
- příprava akceptačního prostředí *(9. 10. 2019 – 11. 10. 2019)*.

**Iterace 2** (14. 10. 2019 – 7. 11. 2019)

- implementace aplikace v aplikačním prostředí *(14. 10. 2019 – 1. 11. 2019)*;
- testování se 100% úspěšností testovacích scénářů *(6. 11. 2019)*;
- performační testování aplikace a dotčených systémů *(6. 11. 2019 – 7. 11. 2019)*;
- vyhodnocení testovacího protokolu *(7. 11. 2019 – 8. 11. 2019)*;
- předání servisní příručky service ownerovi *(7. 11. 2019 – 8. 11. 2019)*;
- akceptace aplikace pro nasazení do produkčního prostředí *(8. 11. 2019)*.

### 3.4.3 Fáze realizace

Hlavní činností ve fázi realizace byla implementace navrženého systému dle analytické dokumentace. Dle projektového plánu byly naplánovány celkem dvě iterace, zaměřující se každá především na implementaci aplikace na jednotlivých prostředích a následném testování. U první iterace bylo očekáváno implementování kritických požadavků tak, aby bylo možné provést testování formou end-to-end. V rámci druhé iterace probíhá vývoj ostatních požadavků, které se zaměřují na uživatelské rozhraní.

V průběhu první iterace se testování zaměřilo na samotnou funkčnost vystavených služeb, které byly ověřeny pomocí SOAP a vývojářských testů. Tyto testy spočívaly v provolání systému patřičným rozhraním, proto i díky malému množství a vzájemné nezávislosti služeb nebylo nutné vytvářet testovací scénáře související s ověřením funkčnosti vystavených služeb. Implementace uživatelského rozhraní probíhala tak, aby bylo možné v návaznosti na vystavené služby testovat funkčnosti procesu změny osobních údajů od jeho počátku. Díky tomu bylo možné splnit průchozí body, které pro konec první iterace na integračním prostředí činily 50% úspěšnost testovacích scénářů. Reálná úspěšnost testů činila 75 % k datu 4. 10. 2019. Důvodem pro neúspěšnost ostatních scénářů byly chyby uživatelského prostředí. Dle těchto údajů provedl zadavatel akceptování aplikace v rámci první iterace a proběhla příprava akceptačního prostředí včetně všech nezbytných konfigurací.

Na počátku druhé iterace došlo k implementaci vyvinutého kódu do akceptačního prostředí a ozkoušení již funkčních testovacích scénářů z předchozí iterace. Při porovnání byla úspěšnost oproti první iteraci stoprocentní. V průběhu vývoje došlo ke změně uživatelského rozhraní, konkrétně přidání informačních boxů u obrazovky pro nahrávání dokumentů. Jelikož se nejednalo o stěžejní změnu a její zapracování proběhlo v řádech jednotek hodin, došlo k jejímu schválení zadavatelem projektu bez nutnosti změny časového harmonogramu v projektovém plánu. K datu 6. 11. 2019 došlo k ukončení vývoje aplikace a 8. 11. 2019 proběhlo vyhodnocení úspěšnosti testovacích scénářů. Úspěšnost testovanosti byla 96 %, přičemž bylo konkrétně kladně validováno 48 z 50 testovacích scénářů. Jednalo se o chyby grafického charakteru, konkrétně chybný odstín tlačítek a nevhodně zalomený text u starších typů telefonů značky Apple. Tyto rozdíly oproti analýze byly uvedeny zadavatelem do akceptačního protokolu. Tyto chyby bylo plánováno opravit v rámci produkčního hotfixu. Vzhledem k rizikům byly provedeny performační testy, díky kterým bylo zjištěno, že není nutné provést navýšení výpočetního výkonu. Tyto testy byly provedeny interním oddělením společnosti. Následně bylo zadavatelem shledáno, že aplikace dosahuje dostatečných kvalit pro její nasazení do produkční verze. Na konci iterace došlo k předání servisní příručky service ownerovi a bylo provedeno nezbytné školení směřující k novým funkčnostem aplikace dodávaných v tomto releasu.

Projektovým manažerem dále došlo k upřesnění jednotlivých iterací pro fázi nasazení:

**Iterace 1** (11. 11. 2019 – 22. 11. 2019)

- nasazení aplikace do produkčního prostředí *(11. 11. 2019)*;
- vyhodnocení testování z hlediska pilotní verze aplikace *(13. 11. 2019)*;
- akceptace dodávky aplikace *(20. 11. 2019)*;
- připrava aplikace pro uvolnění klientům *(22. 11. 2019)*.

**Iterace 2** (25. 11. 2019 – 29. 11. 2019)

- oprava nekritických chyb pomocí hotfixu *(25. 11. 2019 – 27. 11. 2019)*;
- uvolnění aplikace pro všechny uživatele *(28. 11. 2019 – 29. 11. 2019)*.

### 3.4.4 Fáze nasazení

V průběhu první iterace fáze nasazení došlo k uvolnění aplikace do produkce a bylo umožněno pilotním uživatelům testování nově implementované funkčnosti. V průběhu tohoto testování byla zaznamenána pouze jedna chyba grafického charakteru, tudíž byla zadavatelem aplikace s novou funkčností akceptována a proběhla její příprava pro postupné uvolnění klientům. Současně probíhaly práce na opravě tří nekritických chyb, které neovlivňovaly funkcionalitu aplikace a zadavatelem nebyly zhodnoceny jako překážka v užívání aplikace. Jejich nasazení do produkce není v rámci projektu plánované a jejich řešení bude přesunuto do provozních

aktualizací aplikace. Ve dnech 28. 11. 2019 – 29. 11. 2019 došlo k postupnému uvolnění aplikace všem uživatelům a byla sledována zátěž jednotlivých systémů. Dne 1. 12. 2019 bylo zhodnoceno, že nové funkcionality performačně neovlivňují zátěž jednotlivých systémů. Dne 3. 12. 2019 byla projektovým manažerem svolána schůzka ohledně vytvoření dokumentu, který slouží k poučení se z projektu a dává možnost tyto zkušenosti čerpat i projektům v budoucnosti. Při kontrole jednotlivých artefaktů bylo projektovým manažerem konstatováno, že došlo k jejich úplnému dokončení a je možné provést ukončení projektu a jeho následnou archivaci.

**Vyhodnocení implementace pomocí konfigurace metodiky Rational Unified Process**

Při použití konfigurace metodiky Rational Unified Process došlo k podchycení chyb, které se vyskytovaly v rámci projektu, kterým se zabývala kapitola *3.1 Charakteristika současného vývojového procesu ve společnosti XY*. Seznam nedostatků, kterým se projekt v případové studii vyvaroval, je uveden v níže strukturalizovaném seznamu:

- na projektu došlo ke kontrole čtyř očí a díky tomu byly podchyceny i alternativní testovací scénáře, které by jinak byly opomenuty a projevily by se jako incidenty v produkčním prostředí;
- díky změně průběhu zpracování analýzy, konkrétně rozdělení role analytika na dvě samostatné (business analytik a IT analytik) došlo k vypracování detailnějšího obchodního zadání a dále k zajištění správnému verzování jednotlivých obchodních požadavků v případě vzniku změnových požadavků na projektu;
- vzhledem k detailnější charakteristice rizik a návrhu prevence pro jednotlivá rizika bylo možné těmto rizikům lépe předejít a ve dvou konkrétních případech díky tomu došlo k jejich úplnému eliminaci;
- díky změně strategie testování, konkrétně počátku testování v průběhu integrace a využití SOAP testů, byla aplikace v akceptačním prostředí z velké části odladěna, což se projevilo v následném nasazení do produkčního prostředí, kde byl zaznamenán pouze jeden incident grafického charakteru;
- po skončení projektu došlo k vytvoření artefaktu „*Poučení z projektu*“, čímž se zajistilo vyvarování se stejných či obdobných chyb v průběhu následujících projektů, díky využití poznatků jednotlivých rolí, které se účastnili tvorby tohoto dokumentu.

Díky vyvarování se zmíněným nedostatkům a dodržení harmonogramu projektu lze konstatovat, že proběhlo úspěšné implementování doporučení, které se nacházejí v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu*.

### 3.4.5 Vzešlá doporučení pro implementaci metodiky v bankovním prostředí

Pro úspěšné dokončení projektu dle navržené metodiky bylo nutné dodržet činnosti jednotlivých rolí a význam artefaktů, za které nesou přímou odpovědnost. Pokud by došlo ke slučování rolí, může dojít ke zhoršení kvality artefaktů, které bude zapříčiněno takzvanou chybějící kontrolou čtyř očí. Tato kontrola byla využita například u spolupráce IT analytika a projektového architekta při tvorbě a validaci vytvořených artefaktů související s IT analýzou a architekturou systému. Odpovědnost za verzování a aktualizaci dokumentů však vždy spadalo do odpovědnosti pouze jedné odpovědné osobě.

Pro hladký průběh projektu se předpokládalo, že osoby zvolené do jednotlivých rolí již mají zkušenosti s vývojem softwaru, znalost jazyku Unified Modeling Language a jsou s konfigurací této metodiky seznámeny při nástupu do společnosti. Proto na projektu nebyla nutná role procesního inženýra, která má za úkol provádět dohled nad tím, jakým způsobem probíhá vývoj.

Pokud by došlo k neplánované potřebě proškolit konkrétního pracovníka, lze v rámci větší velikosti týmu nalézt vhodný zástup na nezbytně dlouhou dobu. Nedojde tedy k ohrožení projektu a nutnosti provést změnu harmonogramu projektu. V rámci tohoto projektu však k tomuto případu nedošlo.

Díky artefaktu poučení z projektu lze metodiku dle potřeby v organizaci dále modifikovat pro různé typy projektů. Je tedy možné, aby konfigurace metodiky Rational Unified Process byla převzatá na jednotlivých projektech s drobnými úpravami obdobně, jako na tomto projektu. V počátcích je nutné vést dokumenty na vysoké formální úrovni, aby nedošlo k ohrožení jednoznačnosti komunikace. Po skončení prvních projektů lze následně vyhodnotit, které body je vhodné změnit a zda metodice nechybí podstatná role či artefakt, který je specifický pro danou společnost. Jak bylo zmíněno v teoretické části, metodika Rational Unified Process se nezabývá obchodním řízením projektů a konfigurace tuto vlastnost přebírá. Je proto nutné obchodní a personální záležitosti řešit jiným způsobem nebo v případně jednotných procesů v rámci všech projektů metodiku doplnit o postup v těchto oblastech. Nicméně vzhledem k různorodosti dodavatelských společností, upřednostňování typu komunikace a smluvních povinností je tato varianta možná spíše v rámci dlouhodobé údržby a modifikaci produktu než u samostatných projektů, které na sebe nenavazují.

# 4 Závěr

Hlavním cílem diplomové práce bylo vypracování návrhu konfigurace metodiky Rational Unified Process dle její specifikace, která je uvedena v kapitole *2.5 Metodika Rational Unified Process*. Teoreticko-metodologická část diplomové práce se zabývala rozdílem mezi tradičními a agilními metodikami a jejich vhodností pro jednotlivé typy projektu. Této činnosti se věnují kapitoly *2.2 Vodopádový model životního cyklu, 2.3 Spirálový model životního cyklu a 2.4. Agilní metodiky*. Dále se blíže věnovala samotné metodice Rational Unified Process, která je detailně popsána v kapitole *2.5 Metodika Rational Unified Process*. Zde autor čtenáře seznámil s jednotlivými aspekty metodiky, počínaje představením použitých praktik v metodice, charakteristikou jednotlivých fází při využití metodiky v projektu až k definování jednotlivých rolí aktivit a artefaktů potřebných pro úspěšné dokončení projektu.

V praktické části diplomové práce byla následně v kapitole *3.1 Charakteristika současného vývojového procesu u společnosti XY* provedena charakteristika vývoje softwaru u společnosti XY, která se zaměřuje na vývoj softwaru ve finančním a bankovním sektoru. Z tohoto zkoumání současného procesu došlo k identifikaci chyb a nedostatků, které v průběhu projektu nastaly. Tyto nedostatky byly charakterizovány v rámci kapitoly *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu projektu* a současně došlo k definici doporučení, která byla následně použita pro vypracování konfigurace metodiky v kapitole *3.2 Konfigurace metodiky Rational Unified Process.* Konfigurace této metodiky se opírala o definovaná doporučení a především o specifikaci samotné metodiky, která byla charakterizována v teoretické části diplomové práce v kapitole *2.5 Metodika Rational Unified Process*. Dále se zabývala především její procesní částí a to konkrétně návrhem jednotlivých fází na projektu. Každá z těchto fázi obsahovala přesnou definici jednotlivých kroků, které je nutné vykonat pro její úspěšné dokončení. Ke specifikování rolí a artefaktů došlo v kapitole *3.3 Role a artefakty metodiky Rational Unified Process*. Při vypracování byl kladen autorem důraz na detailní charakteristiku jednotlivých rolí a jejich vzájemné propojení s artefakty, které jsou potřebné v průběhu samotného projektu. Pro zpřesnění potřebných vlastností u jmenovaných artefaktů v subkapitole *3.3.1 Artefakty v rámci projektu* byla využita vzešlá doporučení uvedená k současné charakteristice vývojového procesu ve společnosti XY, která jsou definována v kapitole *3.1.7 Identifikace nedostatků a chyb v rámci vývojového procesu projektu*.

V kapitole *3.4 Případová studie za použití konfigurace metodiky RUP* byla provedena aplikace konfigurace metodiky Rational Unified Process na reálném projektu v rámci případové studie. Tento projekt se věnoval implementaci nové funkčnosti do současné aplikace společnosti XY, konktrétně umožňoval změnu osobních údajů pomocí nahrání osobního dokladu. V kapitole *3.5 Vzešlá doporučení pro implementaci metodiky v bankovním prostředí* následně došlo k vyzdvihnutí podstatných skutečností, které v rámci projektu nastaly. Důvodem pro jejich převzetí i do budoucích projektů je skutečnost, že projekt s využitím konfigurace metodiky Rational Unified Process byl úspěšně dokončen v daném termínu se stanovenými náklady oproti projektu, který byl založen na původním procesu vývoje. U projektu využívajícího prvky vodopádového modelu byly zjištěny nedostatky především v oblasti nesprávného navržení iterací a nevhodném rozdělení osob do rolí, kvůli kterému nebylo možné provést nezaujaté testování formou kontroly čtyř očí a tím zabránit projektové slepotě.

Lze tedy vyvodit závěr, že implementace této metodiky je pro společnost XY v dlouhodobém měřítku vhodným řešením pro zabránění podobných nedostatků a pochybení, která negativně působí na konečnou cenu vyvíjeného softwaru. Díky možnosti tuto konfiguraci dále upravovat dle velikosti a typu projektu je možné pokrýt různorodé potřeby společnosti napříč dodávkami, ať se již jedná o optimalizace aplikace u jednotlivých platforem nebo tvorba nových robustních bankovních systémů.

# Literatura

### Odborné knihy a časopisy


AXELOS. *Directing successful projects with PRINCE2*. Praha: TSO, 2018. 236 str. ISBN 9780113315727.

BECK, K., ANDRES C. *Extreme programming explained: embrace change*. 2nd ed. Boston, MA: Addison-Wesley, 2005. 224 str. ISBN 0321278658.

BENNETT, N. *Managing successful projects with PRINCE2*. Praha: TSO, 2017. 425 str. ISBN 9780113315338.

BUCHALCEVOVÁ, A. *Zlepšování procesů při budování informačních systémů*. Praha: Oeconomica, nakladatelství VŠE, 2018. 228 str. ISBN 978-80-245-2235-7.

BOEHM, B. W. *Software engineering: Barry W. Boehm's lifetime contributions to software*. Wiley-IEEE Computer Society Press. 2007. 832 str. ISBN 978-0-470-14873-0.

BOEHM, B. W., LANE J. A., KOOLMANOJWONG S., TURNER R. *The incremental commitment spiral model: principles and practices for successful systems and software*. Upper Saddle River, NJ: Addison-Wesley. 2014. 336 str. ISBN 9780321808226.

BUREŠ, M. et al. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. 232 str. ISBN 978-80-247-5594-6.

BRUCKNER, T. et al. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. 360 str. ISBN 978-80-247-4153-6.

COBB, CH. G. *The project manager's guide to mastering agile: principles and practices for an adaptive approach*. Hoboken, New Jersey: John Wiley. 2015. 383 str. ISBN 978-1-118-99104-6.

ČERNOCKÝ, J. *Řízení výroby ve stavebním podniku*. Bakalářská práce. Olomouc: Moravská vysoká škola Olomouc. 2014. 63 str.

DOLEŽAL, J. *Projektový management: komplexně, prakticky a podle světových standardů*. Praha: Grada Publishing, 2016. 424 str. ISBN 978-80-247-5620-2.

EELES, P. a CRIPPS P. *Architektura softwaru*. Brno: Computer Press, 2011. 328 str. ISBN 9788025130360.

FOWLER, M. *Destilované UML*. Praha: Grada, 2009. 176 str. ISBN 978-80-247-2062-3.

GILB, T. *Competitive engineering*. London: Approx, 2005. 176 str. ISBN 9780750665070.

INTERNATIONAL INSTITUTE OF BUSINESS ANALYSIS. *Guide to Business Analysis Body of Knowledge*. Oakville: International Institute of Business Analysis, 2015. 512 str. ISBN 1927584027.

KRUCHTEN, P. *The rational unified process: an introduction. 3rd ed.* Boston: Addison-Wesley, 2004. 336 str. ISBN 0321197704.

LYNN, S. Agile Software Development with C#, Scrum, Extreme Programming, and Kanban Revised Edition. London: CreateSpace Independent Publishing Platform. 2016. 284 str. ISBN 978-1540671325.

MORGAN, P., SAMAROO A. *Software testing: An ISEB Foundation*. London: BCS The Chartered Institute, 2010. 224 str. ISBN 9781906124762.

MYSLÍN, J. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. 256 str. ISBN 978-80-251-4650-7.

PAGE, A., JOHNSTON K., ROLLISON B. *Jak testuje software Microsoft*. Brno: Computer Press, 2009. 384 str. ISBN 978-80-251-2869-5.

PASCH, O. *Microsoft SharePoint 2010: praktický průvodce uživatele*. Brno: Computer Press, 2011. 279 str. ISBN 978-80-251-3177-0.

PERGL, R. *Metody řízení softwarových projektů využívající moderní paradigmata*. Disertační práce. Praha: Česká zemědělská univerzita v Praze. 2008. 148 str.

SHERMAN, R. *Business intelligence guidebook: from data integration to analytics*. Amsterdam: Elsevier, Morgan Kaufmann is an imprint of Elsevier. 2015. 510 str. ISBN 978-0124114616.

SCHWALBE, K. *Řízení projektů v IT: kompletní průvodce*. Brno: Computer Press, 2011. 632 str. ISBN 978-80-251-2882-4.

SOMMERVILLE, I. *Softwarové inženýrství*. Brno: Computer Press, 2013. 680 str. ISBN 978-80-251-3826-7.

ŠOCHOVÁ, Z., KUNCE E. *Agilní metody řízení projektů*. Brno: Computer Press, 2014. 175 str. ISBN 978-80-251-4194-6.

**Ostatní zdroje**

AGILE MANIFESTO. *Manifest agilního vývoje softwaru [online]*. 2001 [cit. 05-10-2019]. Dostupné z WWW: https:// http://agilemanifesto.org/iso/cs/manifesto.html.

ANWAR, A. *International Journal of Software Engineering (IJSE), Volume (5) : Issue (2) [online]*. 2014 [cit. 18-08-2019]. Dostupné z WWW: https://www.cscjournals.org/manuscript/Journals/IJSE/Volume5/Issue2/IJSE-142.pdf.

BARTOŠOVÁ, H. *Projektový management [online]*. 2011 [cit. 04-08-2019]. Dostupné z WWW:http://files.vsrr.webnode.cz/200000020-5b25a5c1fc/Projektov%C3%BD%20management_OPPA_2012_Barto%C5%A1ov%C3%A1%20a%20kol.pdf.

ČERMÁK, P. *Metodiky vývoje software [online]*. 2018 [cit. 12-10-2019]. Dostupné z WWW: https://mvso.cz/wp-content/uploads/2018/02/Metodiky-v%c3%bdvoje-software-studijn%c3%ad-text.pdf.

ČESKÁ BANKOVNÍ ASOCIACE. *Český bankovní sektor [online]*. 2019 [cit. 28-12-2019]. Dostupné z WWW: https://czech-ba.cz/o-bankovnim-sektor.

SPOLEČNOST XY. *Projektové řízení [online]*. 2016 [cit. 25-2-2020].

MICROSOFT. *Prezentace dat v Ganttově diagramu v Excelu [online]*. 2019 [cit. 05-08-2019]. Dostupné z WWW: https://support.office.com/cs-cz/article/prezentace-dat-v-ganttov%C4%9B-diagramu-v-excelu-f8910ab4-ceda-4521-8207-f0fb34d9e2b6.

PETERSEN, K. *The Waterfall Model in Large-Scale Development [online]*. 2009 [cit. 17.11.2019]. Dostupné z WWW: https://www.researchgate.net/publication/30498645_The_Waterfall_Model_in_Large-Scale_Development.

RATIONAL SOFTWARE. *Rational Unified Process Best Practices for Software Development Teams [online]*. 2011 [cit. 02-11-2019]. Dostupné z WWW:

https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf.

SCHWABER, K., SUTHERLAND J. *The Scrum Guide [online]*. 2019 [cit. 13.10.2019]. Dostupné z WWW: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf.