Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informačních technologií

Analýza a návrh informačního systému s využitím agilních přístupů k vývoji softwaru

Modelování informačního systému jazykem Unified Modeling Language

Diplomová práce

Autor: Adam Krátký
Studijní obor: Informační management

Vedoucí práce: Ing. Tereza Otčenášková, BA

Hradec Králové srpen 2016

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 15.8.2016 Adam Krátký

Poděkování:

Rád bych chtěl tímto poděkovat Ing. Tereze Otčenáškové BA za velkou podporu a cenné rady při vedení mé diplomové práce. Dále bych chtěl poděkovat MUDr. Martinu Langrovi, který mi poskytl neocenitelné rady při zpracování práce a ukázal mi, co být agilní opravdu znamená.

Anotace

Informační systémy jsou v dnešním světě charakterizovaném propojením byznysu, ekonomiky a technologií neoddělitelnou součástí většiny firem. Společnostem výrazně napomáhají využít svůj potenciál, ale zároveň zvyšují jejich závislost na informačních technologiích. Proces tvorby informačních systémů a softwaru obecně tak čelí novým výzvám v prostředí, pro které je příznačná rychlost změn, kratší čas na vývoj a častější nasazování produktů.

Předkládaná práce věnovaná procesu tvorby informačních systémů představuje specifika vývojového procesu a navrhuje softwarové řešení pro správu úkolů, které má podobu modulu zavedeného do již existujícího podnikového systému. V teoretické části práce jsou vymezena specifika tvorby informačních systémů se zaměřením na agilní přístupy, které poskytují vhodný soubor principů a praktik pro moderní vývojový proces softwaru. Tento proces je následně demonstrován v reálném prostředí společnosti Ders, s.r.o.

Pro návrh softwarového řešení systému pro správu úkolů nazvaného Projekty je nejdříve v teoretické části popsán modelovací jazyk Unified Modeling Language. Při vlastní analýze a návrhu systému je nejprve provedena analýza požadavků, při níž jsou využity agilní přístupy User Experience a User Centered Design. Systém je následně modelován výše uvedeným modelovacím jazykem a pomocí příslušných modelů analyzován a navržen odpovídající design systému pro správu úkolů.

Klíčová slova: Agilní metodika, analýza a návrh softwaru, informační systém, Unified Modeling Language

Annotation

Title: Analysis and Design of Information System using Agile Approaches to Software Development

In current world characterized by interconnections among business, economy and technologies, information systems are an integral part of most companies. They significantly help them to achieve their potential. On the other hand, they increase their dependency on information technologies. The development process of information systems and software generally face new challenges in current environment typical for quick changes, shorter development time and more frequent product deployment.

The diploma thesis is dedicated to the development process of information systems. It presents the specifics of the development process and proposes software solution for task management. The latter is represented as a module implemented into existing enterprise system. The theoretical part of the thesis outlines the specifics of the development of information systems. It is focused on the agile approaches which provide a suitable set of principles and practices for modern software development process. Subsequently, this process is demonstrated in the real company environment of Ders, Ltd.

Firstly, the chosen modeling language, Unified Modeling Language, is described. It is used for the design of software solution for task management module, called Projects. The first phase of the system analysis and design includes the analysis of the requirements. For these purposes, the agile approaches User Experience and User Centered Design are employed. Then the system is modeled using the aforementioned modeling language. Afterwards, the appropriate models help to analyze and propose the relevant design of the system.

Keywords: Agile Methods, Information System, Software Analysis and Design, Unified Modeling Language

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V současné době, kterou je možné charakterizovat dynamikou a rychlostí změn, propojeností a sdílením znalostí, technologickým pokrokem, důrazem na inovace a především masivním využíváním informačních technologií, lze považovat postavení informačních systémů jako nedílnou součást a zásadní složku fungování podniků i celé lidské společnosti. Význam informačních systémů podtrhuje vymezení dnešní společnosti španělským sociologem Manuelem Castellsem (2001), jenž ji označil jako *Společnost sítí* a popsal ji jako: *společnost, kde jsou stěžejní struktury a aktivity společnosti soustředěny kolem elektronicky zpracovávaných informačních sítí.*

A právě proto přístup k samotnému procesu vytváření informačních systémů nabývá důležitosti. Vývoj informačních systémů, a softwaru obecně, často podléhá rychle se měnícímu prostředí a požadavkům na systém, co možná nejkratším termínům dodání produktů a omezením ze strany rozpočtu softwarových projektů. Pro zajištění hladkého průběhu vývojového procesu a úspěšného nasazení informačního systému existuje řada metodických přístupů k tvorbě softwaru. Od primitivních modelů, spontánně vzniklých v prvopočátcích vývoje softwaru, přes detailně zpracované robustní rigorózní (tradiční) metodiky, až po agilní metodiky a přístupy k vývoji softwaru. A jsou to právě agilní metodiky, které primárně vznikaly v reakci na potřebu řešit nové otázky při tvorbě softwaru. Jejich použití se však neomezuje pouze na zefektivnění a zlepšení softwarového vývoje, ale jsou vhodné i v dalších oborech, kde je pro vznik produktu využita kreativita, flexibilita a potřeba umět rychle reagovat na změny.

Pokud má být dosaženo správně fungujícího informačního systému, je nutné se soustředit na jednotlivé aktivity procesu vývoje. Především při analýze systému by měla být věnována značná pozornost jeho uživatelům, kteří jsou významným zdrojem požadavků na informační systém. Analýzou a modelováním těchto požadavků lze na určitém stupni abstrakce a z různých pohledů odhalit strukturu a chování systému. Detailnější specifikací některých modelů lze vytvořit návrh informačního systému, jenž může velmi zjednodušit práci při jeho implementaci.

Tyto modely lze vytvářet pomocí standardizovaného unifikovaného modelovacího jazyka Unified Modeling Language a zobrazovat pomocí předepsaných diagramů, které do značné míry usnadňují komunikaci vývojářů se zákazníky a předávání znalostí a informací mezi vývojáři samotnými.

Úvodní kapitoly této diplomové práce pojednávají o stěžejních pojmech a principech, kterým je tato práce věnována a na které odkazuje. Vývojový proces informačních systémů je nejprve představen pomocí modelů životního cyklu, jež vhodně demonstrují základní rozdělení vývojového procesu a klíčové aktivity, které jsou v jeho rámci vykonávány. Od modelů životního cyklu informačního systémů se odvíjí metodické přístupy k budování informačních systémů, které poskytují postupy, návody a doporučení pro vykonávání aktivit jednotlivých fází životního cyklu systému. Pozornost je věnována především výše zmíněným agilním metodikám, jež představují moderní a adaptivní přístup k vývoji softwaru. Použití agilních metodik a jejich přístupů je demonstrováno na reálném prostředí v praktické části. Poslední kapitola je věnována unifikovanému modelovacímu jazyku UML, pomocí kterého je v praktické části modelována případová studie. Na závěr jsou uvedena doporučení a závěry shrnující diskutovanou problematiku.

2 Cíl práce

Cílem diplomové práce je představit proces tvorby informačních systémů a z analytického pohledu vytvořit návrh pro implementaci softwarového řešení, v podobě nového modulu Projekty, do již existujícího podnikového systému.

Vzhledem k tomu, že si diplomová práce klade za cíl představit i samotný proces tvorby informačních systémů, jsou v teoretické části práce přehledně zpracovány teoretické poznatky o metodických přístupech k budování informačních systémů, které k tomuto procesu poskytují konkrétní postupy a návody. Pozornost je, s přihlédnutím k současným trendům ve vývoji softwaru a praktické části, věnována agilním metodikám a dále přístupům a technikám s nimi souvisejícími. Proces vývoje softwaru je v praktické části popsán a analyzován v reálném prostředí společnosti Ders, s.r.o. Popisem modelovacího jazyka Unified Modeling Language a specifikací vybraných diagramů, jsou vytvořeny podklady pro modelování softwarového řešení, které je dále předloženo v praktické části diplomové práce.

3 Metodika zpracování

Při vypracování diplomové práce byla nejprve zpracována rešerše odborné literatury. Hlavním zdrojem publikací vztahujících se k danému tématu byly katalogy knihoven. Zdroje byly následně doplněny o aktuální články ze zahraničních odborných elektronických databází. Analýzou a spojováním znalostí do souvislostí z těchto zdrojů informací a doplněním dalších informací z relevantních internetových zdrojů, vznikala především teoretická část této práce.

Pomocí agilních metodik Scrum a Kanban byl v praktické části popsán proces vývoje softwaru ve společnosti Ders, s.r.o. Vedením semistrukturovaného rozhovoru s vedoucím pracovníkem konkrétního týmu společnosti a analyzováním zjištěných poznatků, byly identifikovány nedostatky a odchylky od teoretické definice metodiky Scrum. Následně byly specifikovány pravděpodobné příčiny těchto rozporů a navrženo jejich možné řešení.

Za využití základních praktik User Experience a best practices (osvědčených postupů) společnosti Ders, s.r.o. pro vedení rozhovoru s klientem, byly specifikovány dříve předeslané požadavky na novou funkcionalitu stávajícího systému. Při rozhovoru byla zároveň využita technika pro tvorbu person, díky které lze lépe pochopit potřeby a chování uživatelů systému. Metody a praktiky User Experience byly také využity při návrhu uživatelského rozhraní.

K modelování navrhovaného systému byl využit modelovací jazyk Unified Modeling Language a modelovací nástroj Enterprise Architect. Při tvorbě modelů, respektive diagramů, bylo postupováno podle obecně přijímaných postupů a best practices.

Všechny výše popsané odborné pojmy a názvy jsou vysvětleny v následujících kapitolách zabývajících se teoretickými východisky. Některé cizojazyčné výrazy byly ponechány v původním znění z důvodu obtížnosti zachování jejich významu v českém překladu a z důvodu jejich ustáleného používání i v českém jazyce.

4 Teoretická východiska

V kapitole teoretická východiska jsou nejdříve definovány základní pojmy a principy používané při tvorbě informačních systémů. Pomocí modelů životního cyklu informačního systému je popsán proces tvorby softwaru. Jejich rozdělením do jasně definovaných částí a vymezením posloupností a vazeb mezi nimi, lze lépe pochopit činnosti a aktivity, které se při tvorbě softwaru vyskytují. Zvláštní pozornost je také věnována sběru a analýze požadavků na systém, které jsou primárním zdrojem informací při budování informačních systémů. Proto jsou na úvod této práce zmíněny obecné principy, úskalí a přínosy při práci s požadavky na systém. Dále se práce zabývá metodickými přístupy při budování informačních systému a jejich dvěma hlavními proudy – rigorózními a agilními metodikami. V následujících kapitolách jsou podrobně popsány agilní metodiky a přístupy k budování softwaru Scrum, Kanban a další agilní přístupy jako User Experience a User Centered Design, jejichž konkrétní aplikace je předvedena i v praktické části. V závěru teoretické části je popsán objektově orientovaný standardizovaný unifikovaný modelovací jazyk (Unified Modeling Language, zkráceně UML) a jeho modely, které slouží k podpoře procesů analýzy a návrhu informačního systému.

4.1 Vymezení základních pojmů a principů tvorby IS/ICT

Dříve, než jsou analyzovány specifické detaily vývojového procesu informačních systémů, jsou v následujících kapitolách vymezeny základní pojmy a principy tvorby informačních systémů tak, jak budou v této práci používány a chápany.

4.1.1 Systém, byznys jako systém, informační systém

Systém je prvotním pojmem, který je třeba definovat. Řada složitých věcí je jako celek více než jen souborem částí, ze kterých jsou složeny (Aristoteles, uvedeno v Bruckner a kol., 2012). Oproti prostému souboru částí, bývá celek složitých věcí definován svou kvalitou, mívá určitý význam, záměr či cíl anebo specifické poslání. Takovéto složité celky popisujeme výrazem systém (Bruckner a kol., 2012). Podobně definuje systém Bureš (2011), který v něm spatřuje synonymum pro utříděnost,

organizovanost a zároveň i složitost. Pro lepší představu jsou vymezeny ještě dva pojmy, systém a „hromada“ (2000, uvedeno v Bureš, 2011). Rozdíly mezi těmito dvěma pojmy a jejich specifikace jsou uvedeny v tabulce č. 1.

Systém	Hromada
Recipročně propojené části, které se chovají jako funkční celek.	Soubor, kolekce seskupených částí.
Při změně části dojde ke změně systému. Rozdělením či půlením nezískáme dva systémy odpovídající velikosti, ale poškozený systém, který bude zřejmě nefunkční.	Pokud přidáme či odebereme některé části, podstatné vlastnosti nebudou změněny. Pokud budeme hromadu půlit, dostaneme dvě menší hromady.
Zásadní je uspořádání částí.	Uspořádání částí je nepodstatné.
Aby části společně fungovaly, musí být propojeny.	Části pracují samostatně a nejsou propojeny.
Struktura je kritická a závisí na ní chování systému. Změna struktury přináší změnu chování.	Pokud zde existuje nějaké chování, závisí na velikosti a počtu částí.

Tabulka č. 1 – Rozdíl mezi systémem a „hromadou“ (zpracováno podle Lampi, 2000 uvedeno v Bureš, 2011)

Dále je nutné si uvědomit, že způsoby, kterými systémy strukturujeme na části, se liší podle účelu zkoumání, úrovně detailu a z jakého hlediska na systém nahlížíme, případně podle toho, jakou máme o věci představu. Celkový pohled na systém i jeho části a jejich vzájemné působení je také závislý na subjektu, který zkoumání realizuje (Bruckner a kol., 2012).

Pro tvorbu informačních systémů je nahlížení na byznys jako systém základním předpokladem. Pod pojmem byznys rozumíme český ekvivalent slova podnik, ovšem výstižnější je výraz byznys, do kterého můžeme zahrnout i instituce státní správy a neziskové organizace. Byznys systém označuje byznys, který představuje systém či celek, jehož integritu tvoří především byznys cíle a záměry. Prvky systému jsou kromě jiného lidé, aktivity, které uskutečňují pro dosažení stanovených byznys cílů

a zdroje, jež k tomu využívají. Vztahy mezi nimi představují jejich vzájemná komunikace, struktura kompetencí, vztahy služební podřízenosti, kontinuita akcí a podobně. Z důvodu otevřenosti byznys systémů je třeba zkoumat i vnější prostředí, jež tvoří klienti, obchodní partneři, konkurence, stát či normotvorné organizace, veřejnost a podobně. (Bruckner a kol., 2012).

Bruckner a kol. (2012) dále uvádí, že pokud definujeme byznys systém, můžeme vymezit podobný pojem - informační systém (zkratka IS). Informační systémy se většinou uplatňují na organizacích, které jsou určitou formou sociálních systémů. Tyto sociální systémy se vyznačují tím, že jejich části jsou často tvořeny množstvím vzájemně komunikujících lidí. Prvky informačního systému jsou většinou shodné s prvky byznys systému. Ovšem v IS je spíše podstatná informace o daném prvku (lidech, materiálu a podobně) než onen prvek byznys systému samotný. Informační systém je tak z tohoto hlediska součástí byznys systému. Lze tedy usuzovat, že informační systém a byznys systém mohou být tvořeny shodnými prvky, ale odlišují se svým účelem. Účelem informačního systému je podporovat předávání správných informací na správné místo a ve správný čas. Adresátem dodávaných informací jsou většinou lidé, kteří jsou prvkem byznys systému (uživatelé IS). Měřítkem správnosti je vhodnost podpory byznys systému k dosažení jeho cílů (u podniků především dosahování zisku).

Podle Laudona a Laudona (2009, uvedeno v Procházka a Klimeš, 2011, s. 19) lze informační systém chápat jako: „*Sociálně-technický systém, ve kterém lidé, technologie, podnikové procesy a organizace na sebe navzájem působí ve snaze sbírat, zpracovávat, archivovat a distribuovat informace s cílem podpořit řízení, koordinaci a rozhodování v organizaci.*“

Rozsahem bývá obvykle informační systém shodný s byznys systémem (IS pokrývá podnik). Je však příhodné obsáhnout do IS i část okolí byznys systému. Důležité informace pro byznys systém jsou totiž vytvářeny a využívány i objekty z vnějšího okolí (dodavateli, zákazníky, konkurencí a podobně). V rámci byznysu je nemusíme jako prvky systému (podniku) vnímat, ovšem z pohledu informačního systému se jako součást systému mohou jevit. Jelikož navrhovat a spravovat IS celého podniku

není vůbec jednoduché, jsou v dnešní době obvykle vytvářeny IS, které pokrývají buď část podniku (například výrobní či skladové systémy), nebo IS zahrnující části více byznys systémů (odběratelsko-dodavatelské systémy a podobně). Toto je možné díky skutečnosti, že na určité části byznys sytému lze nahlížet jako na celky (Bruckner a kol., 2012).

Doucek (2006) nabízí obecnější pohled na IS, a to jako na systém, kde se vazby mezi komponentami systému a vazby s okolím uskutečňují pomocí předávání dat a informací. Autor dále upozorňuje na fakt, že se informační systém, jako obecný pojem, vyznačuje přenosem dat, a to bez ohledu na použitou technologii a způsob ukládání, zpracování, přenášení a prezentaci dat. To vede k myšlence, že IS existovaly v nějaké formě již ve společnosti prehistorické, agrární, industriální a informační. Tyto systémy se od dnešních neliší v základním principu fungování IS (vkládání, zpracování, popřípadě uchování a poskytování dat), ale především v technologii použité k činnosti informačního systému.

Funkcionalitu a potřeby IS zajišťují informační a komunikační technologie (zkráceně ICT z anglického Information and Communication Technologies). Voříšek a kol. (2008) zformulovali ICT jako hardwarové a softwarové prostředky určené pro získávání, přenášení, ukládání, zpracování a šíření informací a pro vzájemnou výměnu informací lidí a technických prostředků IS. Z tohoto důvodu je často používána pro informační systémy podporované informačními a komunikačními technologiemi zkratka IS/ICT.

4.1.2 Objektově orientovaný přístup

Dalším pojmem, na kterém tato práce staví a na který bude odkazováno, je objektově orientovaný přístup (OOP). U objektově orientovaného přístupu je na svět nahlíženo pomocí objektů, které v sobě zahrnují data, popisující určitou entitu reálného světa, a funkce (metody), jež předepisují chování dané entity. Na rozdíl například od procedurálního přístupu, kdy je systém rozdělen na část datovou a část funkční, které jsou analyzovány, navrhovány a implementovány samostatně. U aplikací navržených objektově orientovaným přístupem je dále charakteristické řízení

na základě nastalých událostí, na které objekty reagují. Hlavními znaky objektově orientovaného přístupu jsou (Buchalcevová a Pavlíčková, 2002):

- **Existence objektů**

  Objekty představují základ OOP. Reprezentují uzavřenou strukturu s vlastní identitou a s vnitřní pamětí (atributy, data), která je z vnějšku neviditelná. Dále obsahují metody (procedury, funkce), jež zajišťují určitou činnost nad atributy objektu. Objekt může přijímat a zpracovávat zprávy z vnějšího okolí a obsahovat i jiné objekty.
- **Třídy objektů**

  Třída (class) je vzorem pro vytváření objektů, respektive instancí dané třídy, se stejnými vlastnostmi, chováním, strukturou a vztahy k dalším objektům. Objekty z dané třídy musí mít stejný sémantický význam, metody a stejné atributy, jejichž hodnoty se ale mohou lišit.
- **Zapouzdření**

  Jedním z nejdůležitějších principů v OOP je zapouzdření (encapsulation). Data a funkce jsou sloučeny do jedné entity – objektu. Objekt obsahuje vše potřebné pro své fungování a to umožňuje jeho znovupoužitelnost. Data jsou skryta v objektu a přístup k nim zajišťují metody objektu. Jiné objekty nemusejí znát vnitřní strukturu dat a změny jsou prováděny pouze v jediné třídě. Data jsou tímto způsobem navíc i chráněna.
- **Abstrakce**

  Technika vytváření jednoduché reprezentace složité skutečnosti, zaměřením se na relevantní detaily a vypuštění nepodstatných aspektů.
- **Dědičnost**

  Na úrovni tříd lze vytvářet třídy s podobnými vlastnostmi odvozením z již existujících tříd. Nová třída (potomek) dědí všechny vlastnosti (atributy) a chování (metody) původní třídy (předka) a může přidat své vlastní. Dědičnost tak umožňuje znovupoužitelnost těchto tříd.

- **Polymorfismus**

  Polymorfismus neboli mnohotvarost v OOP představuje skutečnost, že atributy nebo metody se stejným názvem mohou být definovány ve více třídách, ale jejich implementace se v každé třídě může lišit.
- **Komunikace objektů**

  Objekty mezi sebou komunikují pomocí zasílání zpráv (požadavků). Objekt na požadavek reaguje spuštěním příslušné metody (programového kódu), čímž je požadavek uspokojen. Objekt může obsloužit pouze požadavky, které má definované ve svém takzvaném rozhraní objektu.
- **Skrývání informací a implementací**

  Díky zapouzdření dochází u objektů k omezení viditelnosti informací či způsobu implementací z vnějšku. Metody objektů tak lze využívat bez nutnosti znalosti jejich implementace. Z hlediska uživatele je zpřístupněno jen to, co je pro něj relevantní, tedy rozhraní objektu, respektive metody, které může používat.

4.1.3 Modely životního cyklu informačního systému

V následující kapitole je popsán životní cyklus informačního systému a jsou uvedeny jeho základní typy modelů. Jak uvádí Kadlec (2004), dnes jsou již původní modely životního cyklu, jako je například vodopádový model, z hlediska metodiky budování informačních systémů, považované za zastaralé a překonané a nelze je srovnávat s moderními metodikami, kterými ani nejsou. Nicméně uvedeny jsou z toho důvodu, že přinášejí základní dělení softwarového procesu vývoje na jednotlivé aktivity a vymezují jejich logickou návaznost. Buchalcevová (2009) uvádí tři základní typy modelů životního cyklu podle normy ISO/IEC TR 15271 - vodopádový, inkrementální a evoluční. Přičemž inkrementální a evoluční model představují základní typy modelů životního cyklu pro iterativní vývoj. Při iterativním vývoji je software tvořen po menších přírůstcích, které představují samostatně fungující části systému. Je také jedním ze základních principů agilních metodik, které budou popsány v dalších kapitolách.

Jelikož je lidská schopnost myšlenkově řešit komplikovanější struktury omezená, je nutné si vypomoci zavedením přehledných hierarchických nebo lineárních struktur pro řešení komplexních problémů a rozlišení nejasných hranic oblastí problému. Takto lze dosáhnout zvýšení intelektuálního výkonu díky přehlednosti, zřetelnosti a zapamatovatelnosti (Polák, Merunka a Carda, 2003). U dobře strukturovaných problémů, je často nejlevnějším a nejefektivnějším způsobem nalezení řešení, pokud je daná úloha nejdříve popsána. Na základě popisu je poté sestaven model a na něm jsou veškeré změny testovány. Právě modelový přístup je základem při řešení nejen vědeckých problémů a používají ho jak vědy technické, tak i vědy společenské. Pokud nahlížíme na tvorbu informačního systému jako na kompletaci nehmotného produktu, je použití modelového přístupu o to naléhavější (Doucek, 2006).

Softwarové projekty bývají často velice komplexní a složité, proto je zpravidla dělíme na několik jasně definovaných fází. Proces vývoje systému postupně prochází všemi fázemi, od vývoje, přes nasazení, údržbu až po ukončení provozu. Postupný sled těchto fází je potom označován jako životní cyklus informačního systému (Polák, Merunka a Carda, 2003). Slovník softwarové terminologie (IEEE Standard Glossary of Software Engineering Terminology) definuje životní cyklus jako časový úsek, jehož počátek je v záměru vytvořit systém a končí, když systém přestane být používán (IEEE, 1990). Doucek (2006) vymezuje životní cyklus jako formalizovaný popis činností a aktivit s předem vymezenou časovou posloupností jejich plnění a definovanými vazbami mezi nimi. Model životního cyklu se tedy vyznačuje začátkem, předem stanovenými částmi a jejich hierarchickou strukturou i svým koncem. Podle standardu ISO/IEC 12207 (2008, uvedeno v Buchalcevová, 2009) je model životního cyklu rámec procesů a aktivit, které jsou spjaty s životním cyklem. Tento rámec může být organizován do stupňů a slouží jako reference pro komunikaci.

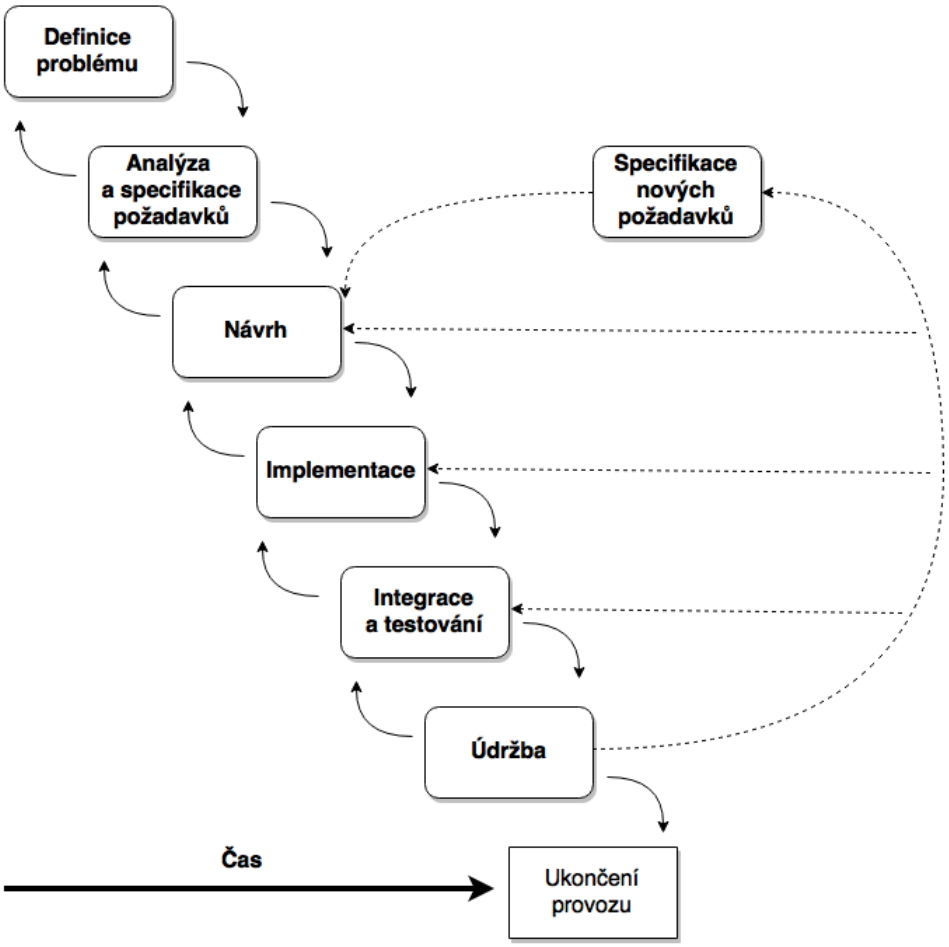
Řepa (1999) dále doplňuje, že přehled o struktuře a obsahu životního cyklu informačního systému je základním východiskem veškerých úvah v metodikách tvorby informačního systému. Veškerý obsah metodiky se poté k tomuto přehledu váže a od něj odvíjí. Fulbright (2013) dodává, že všechny modely vývoje softwaru obsahují

nějakou základní formu aktivit životního cyklu, které jsou potřebné k budování a udržování úspěšného softwarového řešení.

4.1.3.1 Vodopádový model

Vodopádový model životního cyklu IS byl rozšířený především v 70. a 80. letech dvacátého století. Model, jenž se inspiroval postupy v průmyslu, ve své době představoval významný pokrok, jelikož rozdělil proces vývoje do samostatných fází a poskytl systematický postup vývoje softwaru, který lze opětovně aplikovat na dalších projektech (Bruckner a kol., 2012).

Kadlec (2004) definuje vodopádový model jako model procesu vývoje softwaru, ve kterém jsou všechny fáze životního cyklu prováděny v předem určeném pořadí. Mezi základní fáze vodopádového modelu typicky patří definice problému, analýza a specifikace požadavků, návrh a vytváření architektury, implementace, integrace a testování, a dále provoz a údržba systému (viz obrázek č. 1). Pro vodopádový model je charakteristické rozdělení vývoje do sekvenčně uspořádaných etap, které korespondují s jednotlivými vývojovými aktivitami (definice, analýza, návrh, implementace, testování, provoz) a striktní posloupnost jednotlivých kroků a fází. Postup vývoje je tedy stanoven tak, že teprve po dokončení každé fáze je započata následující fáze a nelze vykonávat více fází současně. Doucek (2006) podotýká, že jednotlivé etapy jsou prováděny postupně po dokončení etapy předchozí. Výsledky ukončené etapy jsou verifikovány a slouží jako vstupy do etapy následující. Guntamukkala, Wen a Tarn (2006) doplňují, že v tomto modelu je hlavní důraz kladen na přípravu podrobné specifikace návrhu v raných fázích vývoje a poté efektivní provedení na základě této specifikace. Kadlec (2004) dále uvádí, že model nepředepisuje žádné iterace či doplňující kroky, nezabývá se prototypy ani průběžnou komunikací se zákazníkem. Vodopádový model je velmi významným softwarově-inženýrským mezníkem, který se stal inspirací mnoha dalších autorů. Proto vznikla celá řada variant tohoto modelu, které se prakticky vždy liší jen konkrétní množinou fází, které definují.



Obrázek č. 1 – Vodopádový model životního cyklu (zpracováno podle Kadlec, 2004 a podle Doucek, 2006)

Kadlec (2004) také upozorňuje na fakt, že ačkoliv se ve vodopádovém modelu smí postupovat pouze po jednotlivých fázích, z obrázku č. 1 jsou patrné i zpětnovazebné smyčky. Vysvětluje to tím, že ve všech šesti fázích, se pohybujeme v kategorii definice vývoje. V jejich rámci dochází k vytváření návrhu vývojovým týmem ve spolupráci se zákazníkem, implementaci, testovým případům a podobným relevantním náležitostem, které jsou s vývojem softwaru spjaty. Lze se tedy v rámci těchto fází pohybovat po vodopádovém modelu vždy o jednu fázi dopředu nebo zpět. Pokud tedy dojdeme ve fázi implementace k názoru, že se v návrhu vyskytuje určitý problém, například pokud některý prvek architektury nelze nebo lze jen velmi obtížně převést do implementace, je možné vrátit se o jednu fázi zpět a provést úpravy. Důležité však je, že po provedení nezbytných úprav, by měly projít všechny nové dokumenty opětovným schvalovacím procesem. Návrat do libovolné fáze vývoje a provedení jakékoli úpravy je ve vodopádovém modelu možné, až když je celý produkt hotový, tedy

po skončení fáze testování a předání zákazníkovi. To je na obrázku č. 1 naznačeno přerušovanými šipkami a specifikací nových požadavků.

V následujících odstavcích jsou stručně popsány jednotlivé fáze vodopádového modelu.

Definice problému

V této fázi životního cyklu vodopádového modelu je nejdůležitějším úkolem porozumět záměru zákazníka (Kadlec, 2004). Podle Poláka, Merunky a Cardy (2003) v úvodní fázi životního cyklu dochází k definování samotného problému, vymezení jeho rozhraní, identifikování základní procesů, které se v systému mají uskutečňovat. Kadlec (2004) upřesňuje, že zde nedochází k detailní specifikaci požadavků, ale prozatím jde pouze o poznání zákazníka a jeho potřeb. Cílem je navázat dialog se zákazníkem a pochopit, v čem mu má systém usnadnit práci, k čemu bude používán a jaké fáze jeho práce má systém usnadnit, co od systému očekává, ale i jak dané činnosti vykonává teď, před nasazením systému.

Analýza a specifikace požadavků

Ve fázi analýzy a specifikace požadavků se již podrobně zkoumá, co konkrétně by měl systém vykonávat a jak by toho měl dosáhnout. Požadavky by zde měly být popsány exaktně, pomocí přesných termínů, verifikovatelných formulací a kvantitativních charakteristik. V rámci analýzy by měl být přesně pochopen problém, který zákazník požaduje vyřešit zavedením systému. Vodopádový model vychází z předpokladu, že po dokončení fáze analýzy bude dokončena specifikace požadavků, která se již nebude dále měnit až do předání systému (Kadlec, 2004). Více o specifikaci požadavků v kapitole 4.1.4.

Cílem fáze analýzy je vytvořit logický model vyvíjeného systému a zaznamenat jej za pomocí grafických technik (Richta a Sochor, 1998). Buchalcevová a Pavlíčková (2002) k fázi analýzy poznamenávají, že základem je u objektově orientované analýzy definování případu užití a tvorba diagramů užití (Use Case Diagram). Na základě analýzy jednotlivých případů užití je dále vytvořen analytický návrh struktury tříd a sekvenční diagramy pro každý případ užití.

Návrh a vytváření architektury

Fázi návrhu lze rozdělit na systémový návrh a objektový návrh. Z hlediska systémového návrhu se volí implementační prostředí a architektura aplikace. V rámci objektového návrhu se definují třídy, vztahy mezi nimi, algoritmy metod a uživatelské rozhraní (Buchalcevová a Pavlíčková, 2002). Také se zde řeší problémy spjaté se znovupoužitelností jednotlivých softwarových komponent a napojení vyvíjeného systému na dosavadní komponenty většího celku (Polák, Merunka a Carda, 2003).

Implementace

V této fázi by v ideálním případě měly specifikace z fází analýzy i návrhu popisovat systém natolik dokonale, že by implementátoři měli mít jasnou představu o tom, co a jak mají vytvořit ke spokojenosti zákazníka. V případě vodopádového modelu by neměli programátoři přidávat nové funkce nebo měnit kterékoliv parametry uvedené ve specifikacích. Pokud je nutné provést nějakou změnu ve specifikaci či návrhu, je nezbytné, aby znovu proběhl celý návrhový a schvalovací proces (Kadlec, 2004).

Smyslem této fáze je tedy tvorba požadovaného softwaru, za pomocí programování, sestavování nebo automatického generování využitím CASE nástrojů (Computer-Aided Software Engineering – nástroje pro tvorbu programového vybavení pomocí počítače). Zahrnuje i konverzi dat nebo refaktoring kódu předešlé verze systému (Polák, Merunka a Carda, 2003). Důležité je také stanovit pořadí a prioritu implementace jednotlivých tříd a popsat, jak se vypořádat se vzájemnými závislostmi tříd. Dále by měl být definován způsob, jakým rozdělit práci mezi členy týmu (Buchalcevová a Pavlíčková, 2002).

Integrace a testování

Při fázi testování je cílem, aby aplikace fungovala správně a splňovala očekávání zákazníka. Je potřeba zaměřit se nejen na základní funkčnost, ale měla by být otestována každá část zdrojového kódu. V praxi není zcela možné ověřit všechny aspekty kódu a na konkrétních příkladech nasimulovat veškeré kombinace připadající v úvahu (Kadlec, 2004).

Testování lze definovat jako srovnání určitých předem vymezených charakteristik nebo vlastností daného produktu s jejich reálným stavem (Pol, 1998, uvedeno v Doucek, 2006).

Provoz a údržba

Převzetím aplikace zákazníkem začíná fáze provozu a údržby. Jedná se o proces kontinuálních úprav, oprav, vylepšování a ladění (Kadlec, 2004). U výsledného produktu předaného do běžného užívání v této etapě, jsou za provozu opravovány některé jeho nedostatky, chyby nebo nevhodně sestavené moduly. Nemusí se přímo jednat o chyby, ale o nedostatky, které uživatelům ztěžují práci, jako nepřiměřeně dlouhá doba odezvy, nevhodně strukturovaná data a podobně. Náklady na údržbu, v závislosti na době používání systému, tak mohou několikanásobně překročit původní náklady na vývoj projektu (Chroust, 1992, uvedeno v Doucek 2006). Zavedení systému do rutinního užívání neznamená jen údržbu systému, ale ve finále i jeho stažení z provozu. Především v podmínkách firem vybavených dílčími informačními systémy je ukončení životního cyklu IS a jeho stažení nebo transformace či reengineering obzvláště aktuální (Doucek, 2006).

I přes současnou kritiku vodopádového modelu, je v praxi nadále používán. Pokud je možné ve fázi specifikace požadavků přesně určit všechny požadavky na produkt a během vývoje nedojde k jejich podstatné změně, dosahuje model dobrých výsledků a dává dobrou představu o rozsahu řešení (Bruckner a kol., 2012). Nejvhodnější použití pro vodopádový model a jeho varianty je tedy v prostředí, kde jsou uživatelské požadavky a požadavky na technologie dobře popsány a pochopeny (Guntamukkala, Wen a Tarn, 2006). Četné uplatnění nalezl tento model také na úrovni řízení projektu. Stal se ideálním nástrojem pro sledování celkového průběhu informačního projektu díky svému charakteru, jednoduchosti, srozumitelnosti a prosté transformaci etap a fází do časové osy (Doucek, 2006). Kadlec (2004) dodává, že vodopádový model je vhodný pro řízení, jelikož jsou jasně stanoveny všechny dokumenty a výstupy i produkty a poměrně snadno lze sestavit rámcový harmonogram projektu. Postup do další fáze vyžaduje schválení té předchozí. Lehce lze sledovat, zda je projekt v časové tísni, zda je dodržována specifikace požadavků

nebo není překročen rozpočet. Nicméně je důležité zůstat přiměřeně skeptický, jelikož při postupu vývoje podle vodopádového modelu vznikají dosud netušená rizika, která mohou veškeré odhady výrazně zkreslit.

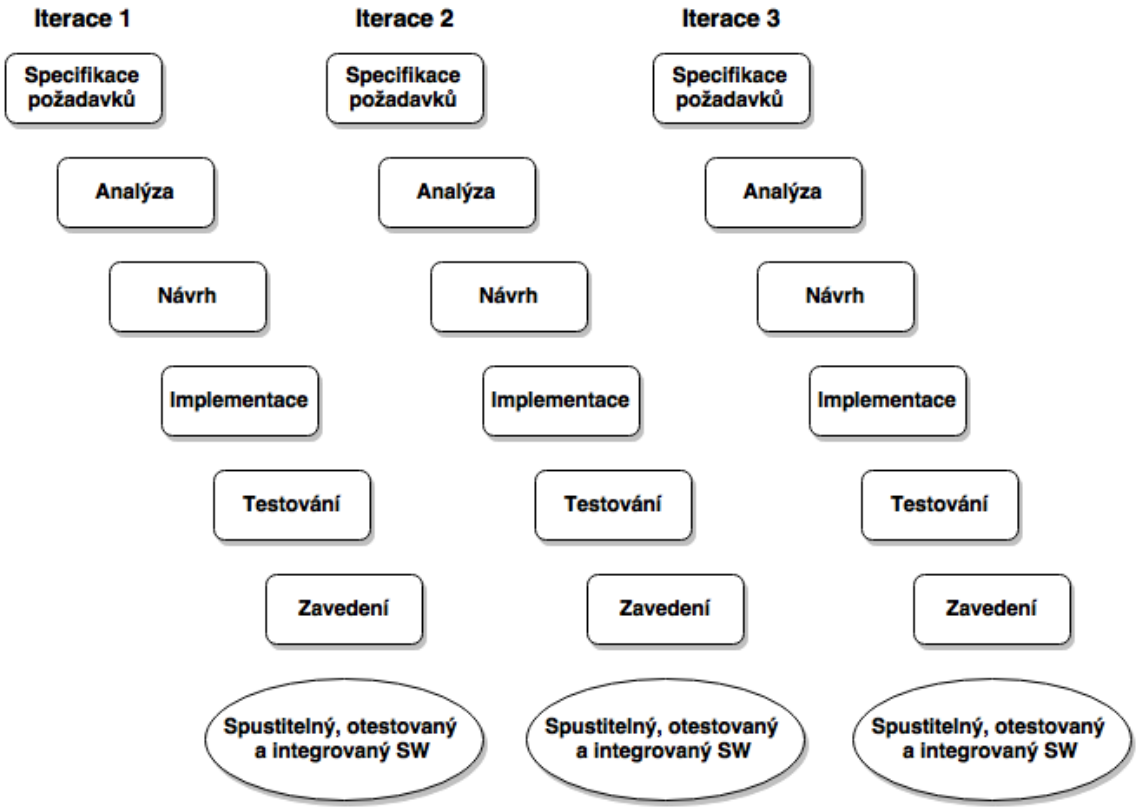
Nevýhodou vodopádového modelu je jeho značná nepružnost. Nelze nijak vstupovat do vývoje a postupuje se striktně podle dané posloupnosti fází. Toto je problémové především v situacích, kdy zákazník přijde s novými požadavky. Potom je nutné projít znovu fází analýzy, vytvořit nové dokumenty a znovu podstupovat schvalovací procesy. Významně se tak prodlužuje doba vývoje systému (Kadlec, 2004). Problémy model přináší v situacích, kdy nelze specifikovat všechny požadavky na začátku projektu, nebo kdy během vývoje dochází k častým změnám. Nevhodné je také zapojení zákazníka do vývoje pouze na začátku a konci procesu, kdy nemá kontrolu nad postupem projektu. Nejzásadnějším problém je však pozdní integrace programového systému, která je provedena až po naprogramování všech modulů. Ve fázi integrace často dochází k odhalení problémů, které vyžadují změny návrhu, přeprogramování a tím dochází k celkovému zpoždění projektu (Bruckner a kol., 2012). Kadlec (2004) dodává, že přítomnost zákazníka pouze v úvodu a na konci projektu přináší řadu nevýhod. Zákazník může mít pocit, že se mezi specifikací požadavků a dodáním nic „neděje“, jelikož není nijak spjat s vývojem. Nemůže tak ovlivňovat například uživatelské rozhraní nebo funkcionalitu aplikace. Požadavky se také mohou v čase vyvíjet a měnit nebo mohou být na začátku projektu špatně pochopeny. Mimo jiné je důležité, aby zákazník cítil sounáležitost s dodavatelem systému a byl vývojovému týmu loajální. Obtížné může být i nasazení kompletního systému najednou, namísto průběžného zavádění. Pokud nebude projekt nakonec realizován, zákazník neobdrží ani částečně funkční systém nebo prototyp.

4.1.3.2 Modely pro iterativní vývoj

Vodopádový model vývoje čelil krátce po svém vzniku řadě poznámek a kritických komentářů. Nevýhodami se postupem času začala zabývat i tehdejší softwarově-inženýrská komunita, jejíž nejvýraznější osobností je Barry Boehem, který obohatil vývojářský svět o řadu vynikajících myšlenek, nápadů, technologií a postupů. Právě Boehem jako první zavedl v souvislosti s modelem vývoje softwaru přelomový koncept

iterativního přístupu, když definoval spirálový model (Kadlec, 2004). Během posledních 30 let, iterativní metody vývoje softwaru, jako jsou například spirálový model, Rational Unified Process, agilní metodiky vývoje software či SCRUM, vznikaly jako reakce na slabé stránky tradičního sekvenčního vývoje vodopádového modelu (Fulbright, 2013).

Iterativní vývoj je založen na faktu, že pro člověka je lepší řešit menší problémy, a proto rozebírá celý projekt na řadu dílčích projektů neboli iterací. V každé iteraci jsou přitom obsaženy všechny vývojové etapy – plánování, analýza, návrh, integrace, testování a zavádění. Po dokončení každé iterace by měla vzniknout fungující část systému, která je testována a integrována s výsledky předešlých iterací (Larman, 2004, uvedeno v Bruckner a kol., 2012). Ilustrace iterativního vývoje je na obrázku č. 2.



Obrázek č. 2 – Model iterativního vývoje (převzato z Bruckner a kol., 2012, s. 109)

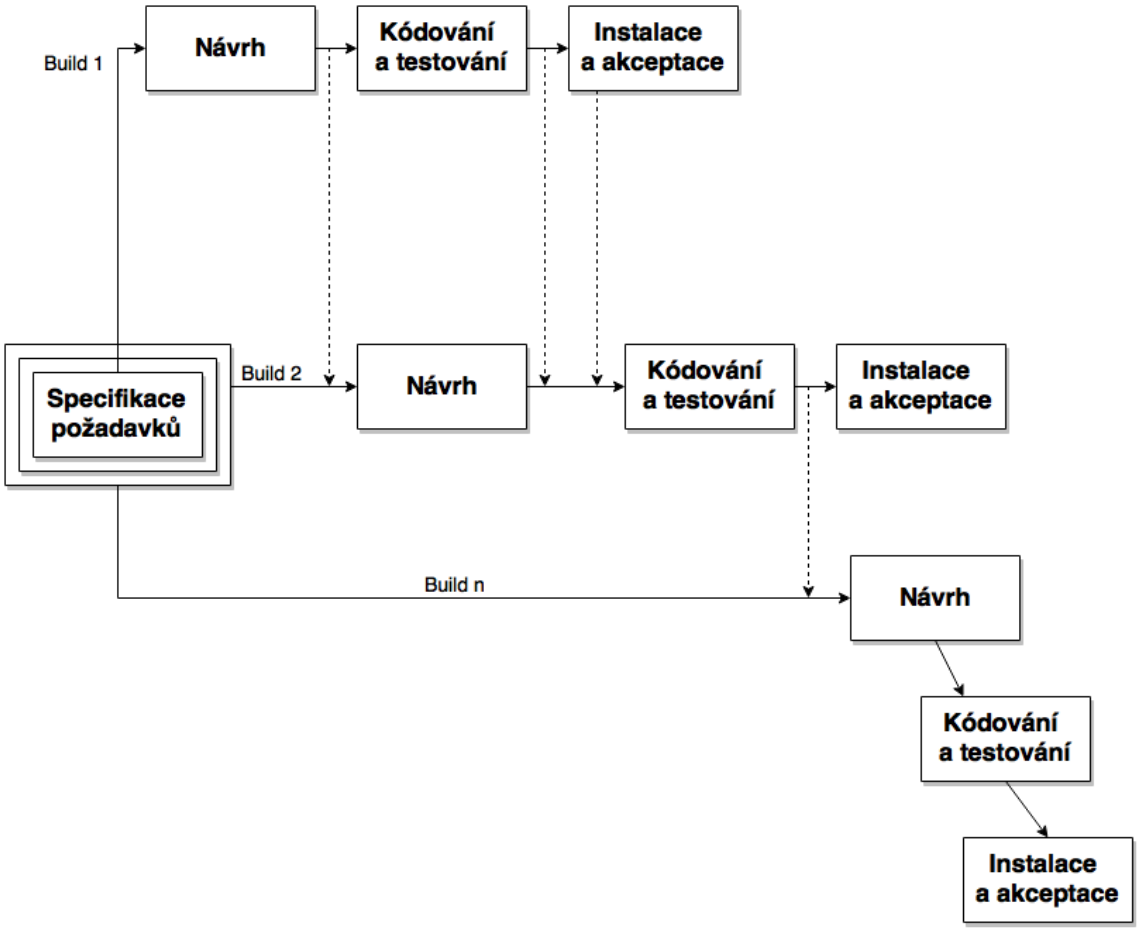
Díky iterativnímu vývoji je dosaženo podstatného snížení rizik při vývoji softwaru, jelikož již po první iteraci lze zjistit, zda je vývojový tým schopen danou funkcionalitu nejen naprogramovat, ale i integrovat. První iterací je také prověřena práce vývojového

týmu ve všech fázích vývoje. Vede také k včasnému odhalení problémů, které lze následně účinněji řešit. Umožňuje i lépe sledovat postup projektu během celého procesu vývoje. Na konci každé iterace by mělo dojít ke zhodnocení vlastní práce vývojovým týmem a zavedení případných změn vedoucích ke zlepšení procesů při vývoji. Významným přínosem je redukce rizika nespokojenosti zákazníka díky možnosti rychlé zpětné vazby od zákazníka a všech zúčastněných stran. Již po provedení první iterace může zákazník vidět fungující část softwaru, na kterou může reagovat. Dochází i k významnému zlepšení při řešení problému ze strany změn požadavků (Bruckner a kol., 2012). Fulbright (2013) doplňuje, že pro metodiky založené na iterativním modelu, je charakteristické „navrhnout trochu, vybudovat trochu“, tedy postupovat po menších částech. To umožňuje dynamickou zpětnou vazbu od uživatelů a projektantů, čímž je dosaženo neustálých změn, jež vedou ke zmírnění rizik a snížení plýtvání se zdroji.

Iterativní vývoj lze realizovat pomocí dvou základních modelů životního cyklu – inkrementálního a evolučního (Bruckner a kol., 2012).

Inkrementální model

Slovník systémového a softwarového inženýrství (Systems and software engineering – Vocabulary) inkrementální vývoj definuje jako techniku vývoje softwaru, ve které jsou specifikace požadavků, návrh, implementace a testování prováděny překrývajícím se, opakovaným způsobem, což vede k přírůstkovému tvoření celého softwarového produktu (ISO/IEC/IEEE , 2010). Bruckner a kol. (2012) inkrementální model popisují tak, že na začátku jsou definovány na hrubé úrovni požadavky na systém. Dále je systém rozdělen na samostatně proveditelné celky neboli přírůstky (na obrázku č. 3 označeny jako build), které představují část plánovaného systému. Každý přírůstek prochází všemi fázemi procesu vývoje zvlášť. Přípustné jsou i případné synchronizace mezi jednotlivými verzemi (buildy), které jsou znázorněny přerušovanými čarami na obrázku č. 3.



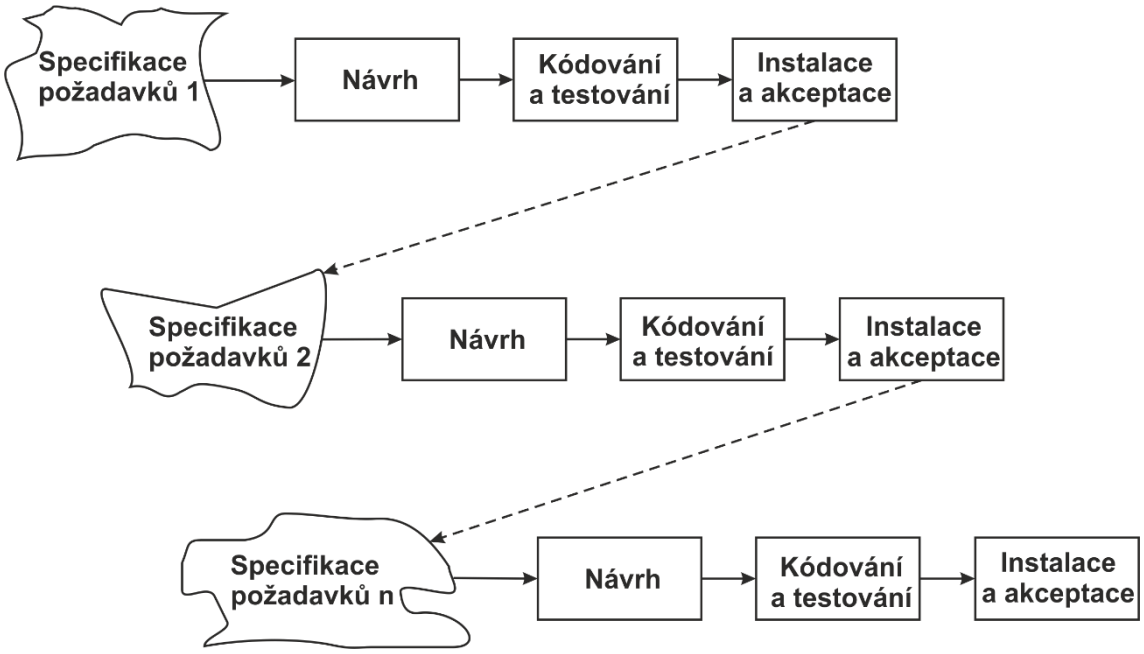
Obrázek č. 3 – Inkrementální model (převzato z ISO/IEC TR 15271, 1998 uvedeno v Buchalcevová, 2009, s. 52)

Buchalcevová (2009) spatřuje hlavní výhodu inkrementálního modelu v tom, že se jednotlivé části řešení zavádějí poměrně rovnoměrně v průběhu projektu. Zákazník tak má lepší přehled o tom, zda se řešení vyvíjí požadovaným směrem a může řešiteli poskytnout rychlejší zpětnou vazbu. Bruckner a kol. (2012) jako další výhodu uvádějí možnost přírůstové spotřeby personálních zdrojů. Při využití inkrementálního modelu je velice důležité správné určení přírůstků co do velikosti, ale především i možnosti vyvíjet přírůstek nezávisle na ještě nerealizovaných částech. Buchalcevová (2009) dodává, že ne vždy je také možné rozdělit systém na přírůstky. Slabou stránkou je také to, že jsou požadavky specifikovány pouze na počátku projektu a tím dochází k obtížnější realizaci pozdějších změn. Stoica, Mircea a Ghilic-Micu (2013) upozorňují, že inkrementální přístup vyžaduje jasnou a kompletní definici celého systému předtím, než je rozdělen na přírůstky. Hůře se i odhalují a odstraňují chyby v návrhu. Inkrementální přístup se také může lehce obrátit v model „napiš a oprav“ neboli model

„Code and Fix“. Tento model byl používaný v prvopočátcích vývoje softwaru. Spočíval v sestavení aplikace, jejím zavedením do provozu a následných opravách chyb. Vznikl spontánně a nejedná se o žádný model životního cyklu či metodiku.

Evoluční model

Evoluční model (viz obrázek č. 4), též přímo označován jako iterativní, se také vyznačuje tím, že se vyvíjejí jednotlivé verze produktu, ale na rozdíl od inkrementálního modelu nejsou všechny požadavky předem specifikovány. Předem jsou vymezeny jen hrubé požadavky, které se zpřesňují po odevzdání každé verze. Kromě časté zpětné vazby od zákazníka je výhodou specifikace požadavků na počátku každé iterace. Lze tedy lépe realizovat změny, a je možná i jejich časná a častá integrace. Nevýhody tento model přináší v podobě špatné představy o rozsahu celého projektu, což může být problémem u projektů s pevně stanovenou cenou. Nutný je také častý kontakt se zákazníkem – nejlépe denně. Nákladná může být i akceptace a instalace jednotlivých verzí (Bruckner a kol., 2012).



Obrázek č. 4 – Evoluční model (převzato z ISO 15271, 1998, uvedeno v Buchalcevovvá, 2009, s. 53)

4.1.4 Požadavky na systém

Sběr a analýza požadavků na systém jsou kritickými činnostmi při vývoji IS. Z toho důvodu jim je v této kapitole věnována zvláštní pozornost. Pokud jsou tyto činnosti zanedbány, podceněny či špatně pochopeny, dochází k závažným chybám v návrhu informačního systému, jako je špatná funkcionalita systému či zcela opomenuté, nebo naopak zbytečné funkce. Takovýto informační systém nereflektuje potřeby uživatelů, nesprávně popisuje a nepodporuje podnikové procesy a není dosaženo stanovených firemních cílů. Proto jsou v následující kapitole zmíněny základní charakteristiky, definice, nástrahy a přínosy specifikace požadavků na systém.

Pro ilustraci je uveden příklad z praxe, který popsal Karel E. Wiegers (2008) ve své knize Požadavky na systém:

Zachycuje zde rozhovor mezi pracovnicí osobního oddělení Marií, která hovoří s vývojářem Filipem, který se podílel na dodání zaměstnaneckého systému. Problém spočíval v tom, že jedna ze zaměstnankyň si změnila jméno a tato změna jména nešla zadat do systému. Filip se totiž při analýze požadavků mylně domníval, že změna jména bude provedena pouze při změně rodinného stavu osoby. Nepředpokládal situaci, že by si někdo mohl změnit jméno jen tak. Tuto chybějící funkcionalitu systému dal za vinu Marii, která ho měla o potřebě této funkcionality informovat při analýze požadavků na systém. Marie samozřejmě požadovala urychlenou nápravu tohoto nedostatku a bránila se tím, že přece každý si může změnit jméno, kdy chce a Filip toto přece musí vědět. Vývojář nicméně trval na tom, že chybějící funkcionalita není jeho chybou a příště mu mají být tyto požadavky řečeny předem a písemně a zjedná nápravu, až budou hotovy projekty, které má právě rozdělané. Frustrované Marii, která potřebovala systém používat, nezbylo nic jiného než zlostně konstatovat: tak tohle je přesně to, kvůli čemu nesnáším počítače.

Wiegers (2008) k výše uvedenému příkladu uvádí, že podobné problémy se softwarem vychází z nedostatků při získávání, dokumentaci, akceptaci a změnách požadavků na systém. Jako příčiny problému uvádí neformální sběr informací, mlčky míněnou funkcionalitu, mylné nebo zatajené předpoklady, neuspokojivě vymezené požadavky a neformální, neočekávané změny. Tyto chyby vzniklé při analýze požadavků mohou

za 40-60 % veškerých chyb softwarových projektů (Davis, 1993; Leffingwell, 1997, uvedeno v Wiegers, 2008). Významnou příčinou neúspěchu softwarového projektu je nedostatečné zapojení uživatele, který představuje zásadní zdroj požadavků. Rozsah jeho absence při získávání požadavků bývá přímo úměrný neúspěchu celého softwarového projektu (Arlow a Neustadt, 2003). Výše uvedené nedostatky při práci s požadavky řeší sada praktik a přístupů User Experience (UX), často označovaná českým ekvivalentem „uživatelský prožitek". User Experience bude blíže popsáno v následující kapitole o metodikách.

Wiegers (2008) také upozorňuje na nedostatek jednotných definic v softwarovém průmyslu. Z různých úhlů pohledu může být tatáž věta popsána jako uživatelský požadavek, funkční požadavek, podnikatelský požadavek, systémový požadavek a podobně. To, jak požadavek definuje zákazník, může znít vývojáři jako velmi abstraktní zadání. Naopak zákazníkovi může připadat vývojářův pohled na požadavky jako podrobná specifikace zadání například z oboru návrhu uživatelského rozhraní.

Wiegers (2008) pojem požadavek vymezuje jako nezbytnou vlastnost systému, která přináší určitou hodnotu některému z účastníků systému. Výraz účastník zde nahrazuje pojem uživatel, jelikož ne všichni účastníci systému jsou i uživateli.

Arlow a Naustadt (2003, s. 47) požadavky definují jako „*specifikaci toho, co by mělo být implementováno*". A dále rozlišují dva typy požadavků:

- **Funkční požadavky** (functional requirements), které vymezují poskytované chování systému.
- **Nefunkční požadavky** (non-functional requirements), které vymezují vlastnosti nebo omezující podmínky systému.

V této práci bude k požadavkům přistupováno podle Wiegerse (2008), který je dělí do třech různých skupin, a to na **požadavky podnikatelské, uživatelské a funkční**.

- **Podnikatelské požadavky** definují na nejvyšším stupni cíle organizace či zákazníka, který systém požaduje. Určují, proč organizace systém potřebuje a tím vymezují cíle, kterých má být prostřednictvím systému dosáhnuto.

- **Uživatelské požadavky** definují cíle uživatelů a úlohy, které by měly být uživatelé schopni prostřednictvím systému vykonat. Definují tedy, co bude uživatel se systémem schopen vykonávat. Mezi vhodné formy zápisu uživatelských požadavků patří případy užití, scénáře a tabulky popisující reakce na různé děje.
- **Funkční požadavky** předepisují softwarovou funkcionalitu. Ta musí být v systému obsažena, aby mohly být realizovány uživatelské úlohy, díky kterým jsou splněny i požadavky podnikatelské. Funkční požadavky jsou někdy označovány jako behaviorálními požadavky, tedy požadavky na chování.

Vedle výše uvedených skupin požadavků má každý systém ještě soubor takzvaných parametrických požadavků. Ty se ovšem odvíjejí z prostředí systému, nikoli z jeho funkce (Wiegers, 2008).

Důležité je si i uvědomit, že požadavky jsou vyjádřením toho, co by měl systém vykonávat, ne však toho, jak by to měl vykonávat. Požadavky tedy vymezují, co by měl systém provádět a jaké chování by měl nabízet, aniž bychom se zabývali způsobem, jak bude daná funkce realizována (Arlow a Neustadt, 2003). V softwarových projektech se objevují i další požadavky, jako jsou například požadavky na vývojové prostředí, termín dokončení, rozpočet a podobně. V těchto případech se jedná o požadavky na samotný projekt, nikoliv na vyvíjený systém (produkt) a mezi požadavky, jak jsou definované v této kapitole, nepatří (Wiegers, 2008).

Cílem procesů zahrnujících práci s uživatelskými požadavky je sběr požadavků, jejich dokumentace a opodstatnění jejich existence vzhledem k cílům a schopnostem organizace. Hodnocení oprávněnosti požadavků tak zkoumá nejen jejich užitečnost vzhledem k celkové podstatě obchodu, ekonomice a hodnototvornému procesu organizace, ale snaží se i objevit jejich duplicity. Zdvojení požadavků se může vyskytnout především ve velkých systémech, kde uživatelé definují nové požadavky, aniž jsou si vědomi, že požadovaná funkcionalita je již aplikována ve stávajících programových řešeních, nebo je již zahrnuta v právě řešených projektech (Pour, 2006).

Smyslem inženýrství požadavků je vymezení služeb, jež by měl budoucí systém poskytovat včetně odhalení toho, k čemu uživatelé daný systém potřebují. Stejně tak jsou identifikovány restrikce, za nichž systém musí pracovat. K tomu je nutný sběr požadavků od uživatelů budoucího systému a utřídění jejich priorit. To s sebou nese i nutnost vyjednávání, jelikož se obvykle vyskytuje řada protichůdných požadavků, pro které je třeba najít řešení (Arlow a Neustadt, 2003).

Pokud chceme dosáhnout dobře navrženého a funkčního systému, zákazníků, kteří systém s radostí využívají, a spokojených vývojářů, je nutné správně pojmout průnik, kde se střetávají zájmy všech účastníků softwarového projektu. Tyto zájmy jsou konfrontovány právě v analýze požadavků. Špatně pojatý průnik požadavků účastníků může být zdrojem nedorozumění, zklamání a sporů mající vliv na kvalitu systému a jeho význam pro zákazníka. Důležité je, aby se všichni účastníci zavázali, že s požadavky budou pracovat efektivně, jelikož jsou základem pro vývoj softwaru a veškeré úkoly spojené s řízením projektu (Wiegers, 2008).

Náročnost spojená se získáváním požadavků na systém vyplývá ze značné části z toho, že dochází ke styku objednatele s dodavatelem systému, jejichž zaměření se různě liší a musejí nejdříve najít společný jazyk (Richta a Sochor, 1998). Požadavky by měly být popsány v jazyce, kterému zákazník rozumí, neměl by být používán specifický programátorský žargon. Výsledkem analýzy požadavků by měl být dokument, který obsahuje specifikace požadavků (Kadlec, 2004). Jelikož zadavatel a uživatel systému nemusí být stejnou osobou, je nutné rozlišovat jejich, někdy protichůdné, pohledy na systém. Zadavatel požaduje maximální efektivitu vynaložených prostředků, kdežto uživatel požaduje maximální pohodlí při své práci. Další příčinou komplikující analýzu požadavků je častý výskyt vícero uživatelů, kdy každý má o systému trochu jiné představy. Také prostředí a otázky, které systém řeší, se nezbytně vyvíjí v čase (Sommerville, 1989, uvedeno v Richta a Sochor, 1998).

Správně a kvalitně zpracované požadavky na systém přinášejí organizaci řadu výhod. Největším přínosem je zmenšení objemu práce, kterou není nutné v pozdních fázích vývoje a během udržovacího období dělat znovu (Wiegers, 1996a, uvedeno v Wiegers, 2008). Pro dosažení správného procesu pro zpracování požadavků je nezbytné

zdůraznit kolektivní přístup k vývoji systému a partnerství všech zúčastněných stran. Dobře identifikované požadavky umožní vývojářům lépe pochopit uživatelskou komunitu či trh, což je pro zdárné dokončení projektu rozhodujícím faktorem. Sběrem požadavků jsou uživatelé systému zapojeni do vývojového procesu. Lze tak podpořit jejich „zápal“ pro nový produkt a budovat tak jejich loajalitu. Zmenší se tak i rozdíl mezi potřebami uživatele a funkcionalitami systému, který vývojář dodá. Větším důrazem na uživatelské úkoly místo povrchně atraktivní funkce systému, se lze vyhnout psaní kódu, který nebude nikdy aplikován. Jelikož se zpětné vazby od uživatelů nakonec dostane vždy, je lepší se jí snažit získat zavčasu, například za pomocí prototypů. Analýza požadavků stojí čas, ale pomáhá efektivnímu vynaložení zdrojů na systém (Wiegers, 2008).

4.2 Metodické přístupy budování IS/ICT

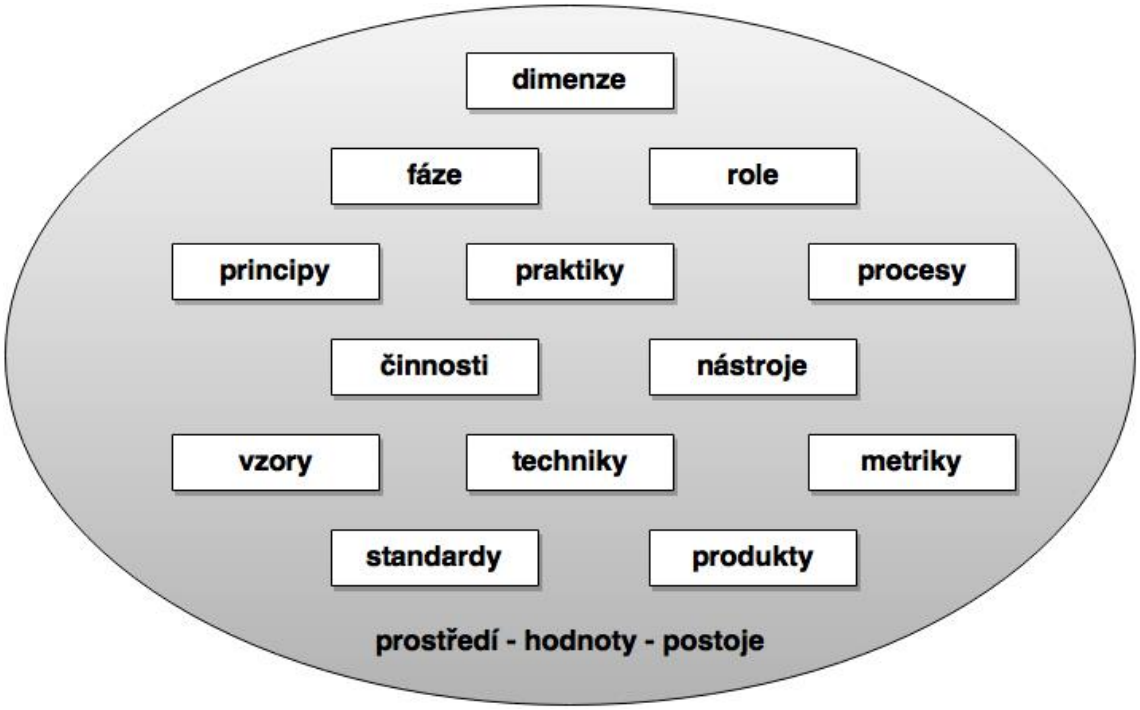
Metodika v obecném smyslu představuje soubor metod a postupů pro vykonání určitého úkolu (Buchalcevová, 2005). Tato kapitola se dále zabývá metodikami, které jsou zaměřeny na vývoj a údržbu informačních systémů.

„Metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“ (Buchalcevová, 2005, s. 13)

Kadlec (2004) definuje metodiku jako soubor komplexních postupů a návodů pro tvorbu softwaru, která pokrývá všechny etapy řešení, tedy všechny fáze životního cyklu. Chlapek, Řepa a Stanovská (2011) doplňují, že metodika tvorby IS se věnuje globálním aspektům vývojového procesu IS. Metodika tedy nemusí být detailní, ale měla by se zabývat celým životním cyklem vývoje informačního systému, to znamená, že by měla být úplná. Podle Buchalcevové (2009) je u metodiky budování IS/ICT přípustné, že se nemusí jednat o vývoj čistě nového řešení, ale i o zavedení hotových řešení nebo integraci komponent a služeb.

Metodické přístupy lze rozdělit na dva hlavní proudy, rigorózní metodiky (též označované jako tradiční metodiky) a agilní metodiky (Buchalcevová, 2005). Konkrétní metodiky se od sebe odlišují převážně tím, které prvky používají. Na obrázku č. 5 jsou

zachyceny prvky, které metodiky zpravidla obsahují. Minimální povinné prvky, které by měla metodika obsahovat, nelze jednoduše určit, jelikož se vyskytují velké rozdíly především mezi rigorózními a agilními metodikami, které budou rozebrány v podkapitolách 4.2.1 a 4.2.2. U rigorózních metodik je možné vymezit jako základní prvky metodiky role, procesy, činnosti a produkty. Minimálními prvky pro agilní metodiky jsou principy a praktiky. (Buchalcevová, 2009).



Obrázek č. 5 – Prvky metodiky (převzato z Buchalcevová, 2009, s. 55)

V následujících odstavcích budou podle Buchalcevové (2009) popsány jednotlivé prvky metodik.

- Fáze obsahuje většina metodik budování IS/ICT. Metodika je postavena na modelu životního cyklu, jehož součástí jsou jednotlivé fáze vývoje IS. Obsah, míra podrobnosti, velikost a počet fází se můžou u různých metodik lišit.
- Dimenze představuje různé úhly pohledu na vývoj softwaru (například z pohledu funkčního, datového, hardwarového, finančního a podobně).
- Role vykonávají specifické činnosti, anebo jsou odpovědné za tvorbu určitých produktů. Typické role, které se v metodikách objevují, jsou byznys analytik, architekt, vývojář, tester a podobně.

- Princip lze definovat jako myšlenkový přístup pro pochopení a analýzu problému a s ním spjatá pravidla pro řešení problému (Voříšek, 2008, uvedeno v Buchalcevová, 2009). Moderní metodiky, především agilní, se místo striktního definování procesů, činností a kroků, snaží vymezit základní principy, které je nutné dodržovat. U agilních metodik může být základním principem například: „Hlavním faktorem úspěchu je funkční software“.
- Procesy jsou základním prvkem rigorózních metodik. Příkladem procesu může být řízení projektu či ověřování.
- Praktiky se liší od procesů větším zaměřením na lidi a více reflektují realitu. Zahrnují i zkušenost, jelikož jsou formulovány lidmi, kteří určitou činnost reálně vykonávají. Praktikou je například párové programování.
- Činnosti označují jednotky práce a skládá se z nich proces.
- Technika podle Řepy (1999, uvedeno v Buchalcevová, 2009) vymezuje, jak dosáhnout požadovaného výsledku. Většinou definuje přesný postup jednotlivých kroků, způsob užití nástrojů, možnosti rozhodnutí v konkrétních situacích a co z nich plyne. Dále předepisuje obor působnosti a podobně. Jako techniku lze označit například modelování případů užití.
- Nástroje slouží jako automatizované prostředky k podpoře technik, činností a procesů. Mezi často používané se řadí nástroje pro datové modelování, objektové modelování, vývojové nástroje, ale i nástroje pro správu požadavků, testovací nástroje, nástroje pro týmovou spolupráci a podobně.
- Produkty představují artefakty, které jsou tvořeny, upravovány nebo používány za účelem vykonání procesů a činností. Role využívají produkty jako vstupy pro realizaci nějaké činnosti, popřípadě jako výsledek dané činnosti. Jedná se samozřejmě především o zdrojový kód, dokumentaci, modely a podobně.
- Metriky slouží k měření efektivnost procesů tvorby IS/ICT.
- Standardy představují publikované dokumenty vzniklé na základě dohod. Mohou zahrnovat pravidla, směrnice a podobně. Dále obsahují technické specifikace či jiná konkrétní kritéria, která je nutné pevně dodržovat. Zabezpečují, že materiály, produkty, procesy a služby dosáhnou svých cílů.

- Vzory umožňují vytvářet znovupoužitelný kód. Zapouzdřují specifické znalosti architektury, platformy či technologie. Využívají se například návrhové vzory, anebo architektonické vzory.

Chlapek, Řepa a Stanovská (2011) shledávají jako základní a nejvýznamnější přínos aplikace metodik budování informačních systémů především nárůst kvality vyvíjeného softwarového produktu a zároveň posílení konkurenceschopnosti firmy. Toho je dosaženo díky tomu, že metodika umožňuje:

- Vymezit vývojové etapy životního cyklu IS, činnosti, které se v rámci těchto etap realizují, návaznosti etap a podobně, čímž je dosaženo větší produktivity a součinnosti projektových týmů,
- vytvořit jasně definované komunikační standardy pro všechny účastníky projektu,
- zaměřit projektové týmy či jednotlivce na konkrétní etapy životního cyklu vývoje IS,
- docílit větší pružnosti vytvářených softwarových řešení vzhledem k změnám potřeb uživatelů,
- dosáhnout kvalitní a aktuální dokumentace,
- vymezit kritéria kvality budovaných IS pro jednotlivé etapy životního cyklu,
- lépe plánovat a kontrolovat práci,
- snížit rizika plýtvání finančními, technickými i personálními zdroji na nerealizovatelné IS,
- přenášet vyvíjené IS do jiných implementačních prostředí, díky oddělení obsahových charakteristik systému, od charakteristik pro použitou technologii a implementační prostředí.

4.2.1 Rigorózní metodiky

Podle Buchalcevové (2005) jsou rigorózní metodiky založeny na předpokladu, že tvorbu IS/ICT lze popsat, plánovat, řídit a měřit. Často bývají velmi objemné, jelikož usilují o striktní a detailní definování procesů, činností a vytvořených produktů. Zpravidla bývají založeny na vodopádovém vývoji a jednotlivé fáze vývoje tak probíhají

sekvenčně za sebou. Vyskytují se ovšem i rigorózní metodiky postavené na iterativním a inkrementálním vývoji, jako je například metodika Rational Unified Process (RUP).

4.2.2 Agilní metodiky

Šochová a Kunce (2014) obecně pojem agilní definují, jako výčet vlastností představujících dynamičnost, iterativnost, interaktivnost a rychlost reakce na změnu. Dodávají, že existuje celá řada synonym, které agilitu charakterizují. Z jejich pohledu se spíše jedná o filozofický náhled na způsob života, preferující jiné hodnoty, jako je například hmatatelný výsledek oproti striktním procesům nebo změna před předem plánovaným.

Potřebu změn v metodikách a zavedení nových (agilních) přístupů spatřuje Buchalcevová (2005) v odlišnostech technologií a ekonomického prostředí a požadavcích na rychlé nasazení IS/ICT do provozu. V těchto podmínkách přestávají být tradiční rigorózní metodiky vhodné. Vzniká tak prostor pro uplatnění metodik, které mohou nabídnout řešení ve velmi krátkém čase a pružně reagovat na měnící se požadavky. Agilní metodiky vznikaly od druhé poloviny 90. let minulého století a staví na myšlence, že jedinou možností jak ověřit správnost systému, je takový systém nebo jeho část co nejrychleji vyvinout, předložit k posouzení zákazníkovi a na základě jeho zpětné vazby systém upravit. Bruckner a kol. (2012) uvádějí, že podle zastánců agilních přístupů je software natolik specifický, že nelze dopředu plně popsat proces jeho vývoje. Proto je nutné jej kontinuálně sledovat a přizpůsobovat změnám. Z toho důvodu agilní přístupy nedefinují procesy při vývoji softwaru, ale vymezují jen principy a praktiky.

Šochová a Kunce (2014) jako nejčastější důvody pro přechod na agilní metody uvádějí následující faktory:

- **Flexibilita**
  V dnešní době je kladen požadavek na co nejrychlejší dodání produktu, pokud možno ihned jak je hotova některá funkční část. Nelze tak již vyvíjet a dodávat software v lepším případě v řádu několika měsíců.

- **Efektivita**

  Jelikož je týmová práce efektivnější než práce jednotlivce, agilní metody zavádějí různé postupy pro spolupráci vývojářů. Nabízí se například párové programování, kdy jednu činnost vykonávají dva lidé. Nejenže tato metodika přináší větší efektivitu, ale i vyšší kvalitu a rychlost, než samostatná práce obou pracovníků. Další možností je sestavení týmu, kde si jednotliví členové pomáhají, spolupracují a sami se organizují, za účelem dosažení společného cíle. Agilní metody mohou pomoci i z hlediska řízení funkcionality. Pomáhají se zaměřit na to, co zákazník opravdu potřebuje a co mu přinese nejvyšší hodnotu.
- **Předvídatelnost**

  Pro zlepšení předvídatelnosti rozdělují agilní metodiky projekt na menší části. Odhady jsou prováděny v relativních jednotkách a do procesu je zapojen celý tým. Lze tak docílit dokončení projektů bez časové tísně a v termínu.
- **Kvalita**

  Zapojením zákazníka, respektive uživatele, do procesu vývoje lze lépe pochopit jeho požadavky, potřeby a návyky. Průběžným prezentováním výsledků je možné řídit zákazníkovo očekávání a vyhnout se odmítnutí finálního produktu, který by nesplnil jeho očekávání. Dalším způsobem pro zvýšení kvality, který agilní metody používají, je zavedení odpovědnosti celého týmu za kvalitu produktu.
- **Zábava**

  Každý člen týmu ví, co a proč tvoří, chápe smysl produktu a rozumí zákazníkovi. Komunikace a spolupráce s dalšími členy týmu může přinášet pracovníkovi v jistém smyslu zábavu. Motivace členů týmu může zajistit i větší efektivitu.

Stoica, Mircea a Ghilic-Micu (2013) připomínají, že existuje řada agilních metodik vývoje softwaru. Přestože má každá z nich specifický přístup, sdílejí stejné hodnoty a vize, které jsou popsány v Manifestu agilního vývoje.

Manifest agilního vývoje software zní: *„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.“ (Agile Alliance, 2001a)

Manifest agilního vývoje vychází a řídí se 12 základními principy (Agile Alliance, 2001b):

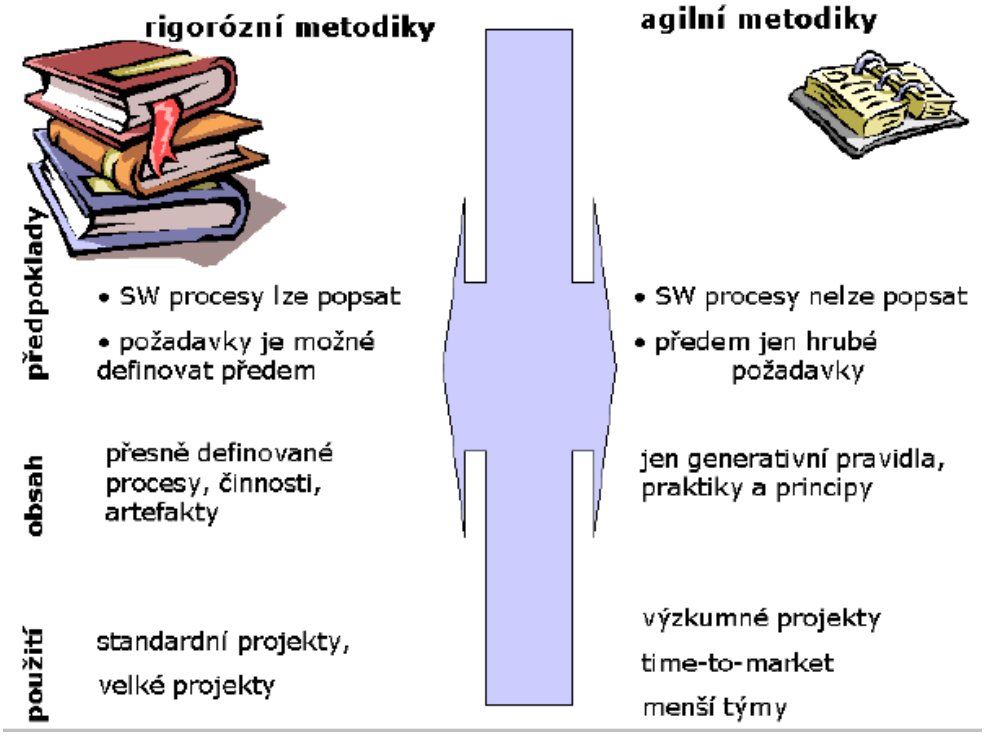
- Nejzásadnějším principem je splnit požadavky zákazníka včas, průběžným zaváděním kvalitního softwaru.
- Požadavky na software se mohou měnit v průběhu všech fází vývoje, tedy i v pozdějších fázích. Změny, které agilní procesy zavádějí, přinášejí lepší konkurenceschopnost zákazníka.
- Dodávky provozuschopného softwaru jsou v horizontu týdnů až měsíců. Snaha je i o kratší intervaly.
- Spolupráce mezi vývojovým týmem a zákazníkem probíhá denně a probíhá po celou dobu projektu.
- Na vyvíjených projektech se podílejí zapálení jednotlivci. Pro jejich práci je vytvářeno adekvátní prostředí, jsou podporovány jejich potřeby a důležitá je důvěra v jejich schopnosti, že vykonají dobrou práci.
- Pro komunikaci s vývojovým týmem zvnějšku i zevnitř je nejúčinnějším a nejefektivnějším způsobem osobní konverzace.
- Pro sledování pokroku je hlavním kritériem funkční software.
- Udržitelný rozvoj je agilními procesy podporován. Všichni účastníci projektu by měli být schopni pokračovat v projektu stálým tempem.
- Neustále se věnovat dobrému návrhu a technické specifikaci vede ke zvyšování agility.
- Jednoduchost je zásadní. Spočívá ve schopnosti maximalizovat množství nevykonané práce.
- Spontánně vzniklé týmy přinášejí nejlepší architektury, požadavky a návrhy.

- V týmu dochází k pravidelnému zamýšlení nad zvýšením efektivity a uzpůsobení chování a návyků k jejímu dosažení.

4.2.3 Porovnání rigorózních a agilních metodik

Agilní metodiky jsou založeny na adaptivních vývojových metodách, zatímco tradiční (rigorózní) metodiky vycházejí z prediktivního přístupu. Tradiční přístup poskytuje vývojovému týmu detailní plán, úplný seznam charakteristik a úkolů, které mají být splněny v nadcházejících měsících nebo během celého životního cyklu. Jelikož je tradiční přístup založen na prediktivních metodách, je zcela závislý na analýze požadavků a pečlivém plánování v počátečních fázích vývojového cyklu. Adaptivní přístup použitý u agilních metodik nevyžaduje úvodní detailní plánování a jsou navrženy pouze úkoly, které jsou jasně patrné. V agilních metodikách se tým dynamicky přizpůsobuje změnám v požadavcích na produkt. Častým testováním produktu se předchází vzniku budoucích závažných chyb. Typickou charakteristikou agilního vývojového prostředí je otevřená komunikace a minimální dokumentace. Silnou stránkou agilních metodik je potom značná interakce se zákazníkem. Spolupráce mezi vývojovými týmy je úzce spjatá a často bývá realizována ve stejném prostředí, například v jedné místnosti (Stoica, Mircea a Ghilic-Micu, 2013).

Odlišné předpoklady zavedení, rozdílný obsah a oblasti použití rigorózních a agilních metodik podle Buchalcevové (2005) vycházejí z odlišného chápání procesu vývoje softwaru u obou metodik (viz obrázek č. 6). Rigorózní metodiky nahlíží na proces vývoje softwaru jako na definovaný proces, zatímco agilní metodiky jsou založeny na předpokladu, že se jedná o proces empirický.



Obrázek č. 6 – Srovnání rigorózních a agilních metodik (převzato z Buchalcevová, 2005, s. 70)

Podle Buchalcevové (2004) je do určité míry přípustná i kombinace obou metodických přístupů. Některé agilní přístupy lze aplikovat v rámci odlehčených rigorózních metodik. Příkladem může být aplikace agilních přístupů v metodice RUP. I u agilních metodik může vzniknout potřeba větší formalizace a dokumentace, především u větších projektů. Porovnání agilních a rigorózních metodik z různých hledisek je uvedeno v tabulce č. 2.

Hledisko	Rigorózní metodiky	Agilní metodiky
Základní hypotéza	Systém je zcela specifikovatelný, předvídatelný a vyvíjený prostřednictvím rozsáhlého a detailního plánování	Software je adaptivní, vyvíjený za použití neustálého zlepšování designu a testování založeného na rychlé zpětné vazbě a změnách
Vnímání procesu vývoje softwaru	Definovaný proces, lze jej opakovaně aplikovat	Empirický proces, nelze konzistentně opakovat

Hledisko	Rigorózní metodiky	Agilní metodiky
Predikce	Vychází z předvídatelnosti, plánování a sběr požadavků předem	Nepřipouští předvídatelnost, zaměření na adaptaci na změny
Podrobnost metodiky	Velmi podrobná specifikace procesů a činností, snaha popsat vše, co se může v průběhu vývoje vyskytnout	Nízká podrobnost, zaměření na činnosti, které přinášejí hodnotu pro zákazníka, popisují minimální množinu věcí, které je třeba udělat
Změny	Minimalizace změn, změny podléhají procesu řízení změn	Změny jsou vítány, požadavky lze přehodnotit
Požadavky	Detailní, stabilní, definovány před začátkem kódování a před implementací	Nahlíženo jako na interaktivní vstup, rychle se mění
Architektura	Navržena podle současných i budoucích požadavků	Navržena podle současných požadavků
Model vývoje softwaru	Vodopádové, přírůstkové a iterativní modely s delšími iteracemi	Iterativní model s krátkými iteracemi
Modelování	Modelování hraje významnou roli, především modelování budoucnosti	Smyslem agilního modelování je komunikace, nejde o model jako takový
Velikost	Velké týmy a projekty	Malé týmy a projekty
Kvalita	Klíčová je kvalita procesů, kvalitní procesy vedou ke kvalitnímu produktu	Klíčová je vysoká kvalita produktu a hodnota, kterou přináší zákazníkovi
Komunikace	Formální, převážně písemná	Neformální, osobní
Styl řízení	Řídit a kontrolovat	Vést a spolupracovat
Rozsah řešení	Snaha o integraci budoucích požadavků	Pouze požadovaná funkcionalita

Hledisko	Rigorózní metodiky	Agilní metodiky
Vztah zákazníků a vývojářů	Smluvní, nedůvěra	Spolupráce, důvěra
Dokumentace	Obsáhlá dokumentace	Stěžejní není dokumentace, ale porozumění
Vnímání lidí	Lidé bráni jako sekundární faktor, zaměnitelní, s úzkou specializací	Lidé jako klíčový faktor úspěchu, důraz na znalosti, schopnosti a dovednosti, individualismus, sdílení znalostí v týmu
Vnímání zákazníka	Účast v počátečních a koncových fázích, řízení projektu v rukou technologických pracovníků	Na zákazníka je nahlíženo jako na řídící subjekt během celého procesu vývoje, může měnit funkcionalitu a její prioritu při každé iteraci
Prostředky	Zdroje nejsou považované za statické veličiny a zpravidla rostou	Snaha z daných zdrojů vytěžit největší hodnotu pro zákazníka, nikoliv dokonalý systém

Tabulka č. 2 – Srovnání rigorózních (tradičních) a agilních metodik (zpracováno podle Buchalcevová, 2005 a podle Stoica, Mircea a Ghilic-Micu, 2013)

V následujících kapitolách jsou popsány vybrané agilní metodiky a přístupy včetně principů a praktik, které s nimi souvisejí.

4.2.4 Metodika Scrum

Kadlec (2004) uvádí, že metodika Scrum patří mezi jednu z nejpoužívanějších agilních metodik. Název metodiky vznikl z anglického slova Scrum (skrumáž, „mlýn“) používaného ve hře ragby, kdy se seskupení několika hráčů snaží dotlačit míč na požadovanou pozici. Paralela s tvorbou softwaru je tak na první pohled patrná. Vývojový tým se společně snaží o „dotlačení míče“ respektive softwarového produktu do fáze úspěšně dokončeného projektu. Vývojový tým „zvítězí“, pokud je zákazník

spokojený. Podle Buchalcevové (2009) je metodika Scrum, podobně jako hra ragby, rychlá, adaptivní a založena na samoorganizujících se týmech.

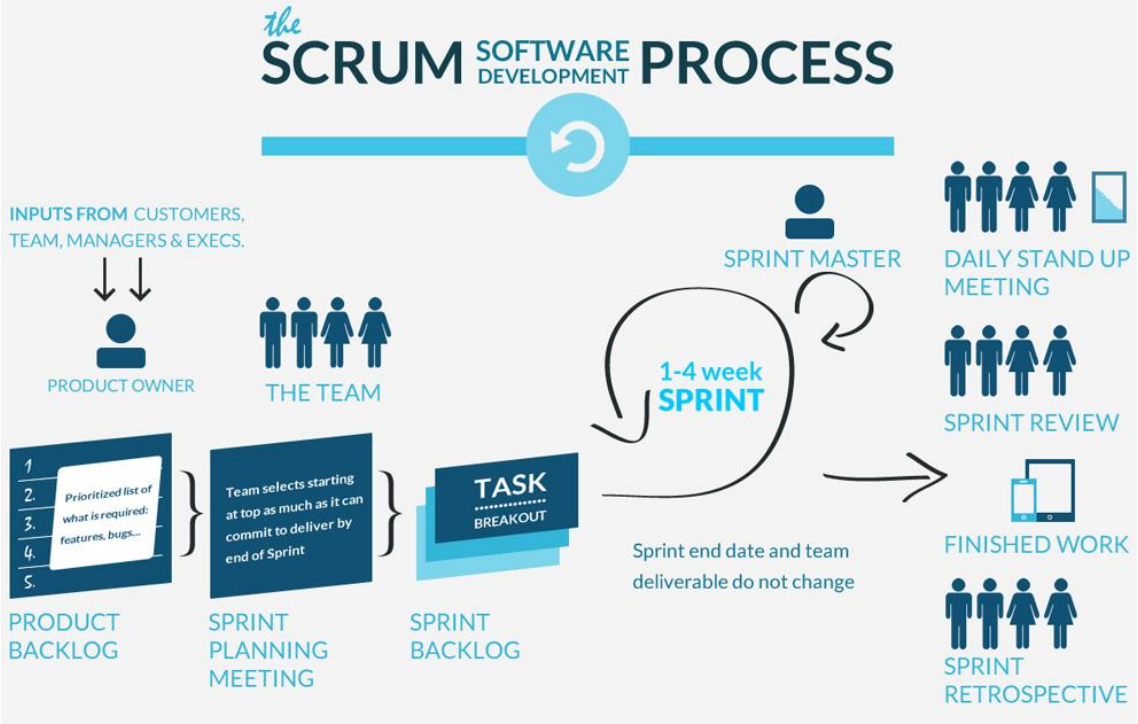
Scrum je převážně zaměřen na projektové řízení. Autoři metodiky staví na přesvědčení, že vývoj softwaru není proces definovaný, ale empirický, proto je nutný odlišný styl řízení (Buchalcevová, 2009). Dathan a Menon (2010) Scrum popisují jako iterativní a inkrementální proces vhodný pro řízení jakéhokoli projektu. Ve srovnání s tradičními sekvenčními přístupy se ve Scrumu vývojové fáze silně překrývají a celý proces vývoje software je vykonáván multifunkčním týmem, ve kterém si jednotliví členové vzájemně pomáhají a sami se organizují podle aktuálních potřeb. Tímto se stává vývoj produktu agilním a flexibilním. Šochová a Kunce (2014) připomínají, že kromě samoorganizujícího se týmu, Scrum staví na principech transparentní komunikace a otevřené kultury pro usnadnění spolupráce a sdílení informací. Kadlec (2004) doplňuje, že Scrum je založen na objektově orientovaném přístupu a přináší inovativní způsob řízení a organizace procesu.

V metodice Scrum je vývoj realizován pomocí pevně daných časových intervalů, takzvaných sprintů, které představují základní rozdělení procesu vývoje softwaru (Kadlec, 2004). Během každého sprintu (iterace), který obvykle trvá 15 – 30 dní (může být i kratší, délku stanovuje tým), je týmem vytvořen přírůstek použitelného a potencionálně nasaditelného softwaru. V každém sprintu je zpracována sada funkcí z takzvaného Product Backlogu, který představuje seznam prioritizovaných funkcionalit, které chceme dodat zákazníkovi (Datham a Menon, 2010). Proces analýzy, designu a vývoje jsou ve Scrumu považovány za nepředvídatelné. Za tohoto předpokladu nelze dopředu důkladně plánovat přesný obsah konkrétních sprintů (Kadlec, 2004). Scrum místo toho zavádí denní schůzky zvané Scrum Meeting nebo Stand-up meeting. Na těchto schůzkách, které netrvají déle než 10 minut, je členy týmu konfrontován dosavadní postup vývoje, představeny dosažené výsledky, identifikovány případné problémy. Je zde prostor pro sdílení znalostí a tyto schůzky mají také vliv na soudržnost týmu (Šochová a Kunce, 2014).

Scrum definuje tři základní role (Šochová a Kunce, 2014):

- **Scrum Master**, který je koučem a moderátorem týmu, ovšem není manažer týmu či teamleader v klasickém významu slova. Jeho úkolem je vystupovat jako mezičlánek mezi týmem a jakýmikoli vnějšími rušivými elementy, odstraňovat problémy, pomoci týmu při dosažení jejich cílů a motivovat tým. Výsledkem jeho snažení by měl být spokojený, samostatný a efektivní samoorganizující se tým.
- **Product Owner** je vlastníkem produktu, jehož primárním cílem je porozumění produktu. Vymezuje vizi projektu. Má zodpovědnost za celý Product Backlog a za celkovou úspěšnost produktu. Na základě komunikace se zákazníkem určuje priority úkolů a stanovuje, co je třeba udělat a v jakém pořadí.
- **Scrum Tým**

  Tým ve Scrumu je nejen samoorganizující se, ale i role jednotlivých členů (analytik, programátor, tester a podobně) by měly být zastupitelné a multifunkční. Tým se tak stane flexibilním a efektivnějším. Například při práci nad User Story (funkcionalita popsaná řečí zákazníka) se analytik zamýšlí nad možnou funkcionalitou a současně konzultuje chování funkce s programátorem. Tester nahlíží na User Story z pohledu výskytu možných chyb a identifikuje chybové scénáře. Výhodou je že, všichni členové týmu pracují současně na stejné věci, jsou rovnoměrně vytížení a nemusí ověřovat své předpoklady a zjišťovat souvislosti, jak druhý vývojář konkrétní věc zamýšlel.



Obrázek č. 7 – Proces vývoje softwaru dle Scrum (převzato z Shalygin, 2015)

Šochová a Kunce (2014) popisují začátek procesu vývoje ve Scrumu představou Product Ownera, co se bude dělat v následujícím období. Ta by měla být schválena managementem společnosti a poté je s dalšími spolupracovníky dále rozpracována a vytvářena vize. Následuje takzvaná discovery fáze, jejímž výsledkem je za účasti členů týmu Product Backlog. V případě aktuálně probíhajícího projektu Product Owner a tým pracují na Product Backlogu kontinuálně a celá přípravná fáze je kratší. Product Owner stanoví priority pro následující sprint a v rámci takzvaného planningu vybere tým konkrétní User Stories do Sprint Backlogu. Tým se tak zaváže k dokončení vybraných úkolů v rámci sprintu a denně na Stand-up Meetingu závazek reviduje a domlouvá se na další spolupráci a postupu. Dokončené User Stories schvaluje Product Owner, případně během sprintu dále zpřesňuje funkcionalitu systému. Na konci sprintu jsou v rámci Sprint Review představeny dokončené User Stories zákazníkovi. Pomocí retrospektivy je možné členy týmu zlepšit či změnit proces vývoje, identifikovat postupy, které se jim osvědčily a oslavit společně úspěšně dodaný produkt. Schéma zachycující proces vývoje softwaru ve Scrumu je uvedeno na obrázku č. 7.

Zavedení metodiky Scrum přináší řadu výhod. Největší z nich je schopnost pružně reagovat na změny v projektu. Do úvahy jsou denně brány změny a stejně tak odhalována možná rizika. Silnou stránkou Scrumu je i svoboda vývojářů při návrhu řešení a možná změna směřování projektu podle aktuálních požadavků zákazníka (Kadlec, 2004). Další výhody již zmínili Šochová a Kunce (2014). Ty plynou především z přístupu k týmu a jeho velikosti. Nejenže se členové týmu sami řídí, aby na konci sprintu dodali funkční část systému, ale především vybudování alespoň částečně sdílené znalosti uvnitř týmu umožní členům lepší spolupráci a značně zvýší efektivitu a flexibilitu celého týmu. Buchalcevová (2009) dále mezi silné stránky a výhody doplňuje zlepšování samotného procesu pomocí retrospektivy, snížení chaosu při procesu vývoje ustálením úkolů pro sprint a podporu metodiky různými softwarovými nástroji.

Problémy u Scrumu mohou nastat při jeho zavádění, kdy se mohou vyskytnout obtíže s přijetím nového způsobu práce a odlišného pojetí vývoje softwaru. Nutné je proto pro tým najít aktivní a motivované lidi. Nevýhoda Scrumu také spočívá ve skutečnosti, že nedefinuje konkrétní kroky a technické pozadí metodiky vedoucí k vývoji softwaru (Kadlec, 2004). Ionel (2008) jako slabou stránku Scrumu uvádí fakt, že by měl být zákazník schopný a ochotný zapojit se ve značné míře do projektu. U projektů používajících Scrum má vize zákazníka značný vliv na proces vývoje. Pokud zákazník nemá jasnou představu o směřování produktu, členové vývojového týmu budou mít sklon chovat se stejným způsobem a výsledný produkt se tak může zásadně lišit od toho, co bylo očekáváno. Proto je jednou z hlavních slabin Scrumu právě jeho silná stránka – zapojení klienta v procesu vývoje.

4.2.5 Metodika Kanban

Kanban byl poprvé použit ve společnosti Toyota Company, jako implementace metody pro řízení výroby Just in Time (JIT). JIT je založen na systému tahu, kdy je materiál pro daný stroj dodán v okamžiku aktuální potřeby. Existuje tak pouze omezená fronta zásob materiálu. Snahou je docílit optimálního pohybu materiálu a co nejefektivnější výroby s vysokou kvalitou. Kanban, který lze do češtiny přeložit jako informační tabule, či vizuální signál, byl navržen k řízení plynulosti provozu výroby pomocí dobře

načasovaných vizuálních signálů. Díky tomu lze plnit požadavky zákazníků v reálném čase vysláním jasných signálů pro řízení produkce. Toyota původně používala kartičky připojené k různým kontejnerům, jež označovaly materiál, který je potřeba doplnit a který aktuálně použít. Postupem času se Kanban stal jedním z přístupů k řízení projektů a byl aplikován v dalších odvětvích, jako je softwarové inženýrství a vývoj (Ordysiński, 2013).

Kanban je podobně jako Scrum vhodný především pro projektové řízení a představuje velice volný a flexibilní agilní přístup k procesu vývoje softwaru. Kanban předepisuje pouze tři základní principy, které je nutné dodržovat (Šochová a Kunce, 2014):

- zredukovat aktuálně rozdělanou práci
- omezit čas průchodu
- vizuálně zaznamenávat postup

Šochová a Kunce (2014) zjednodušeně popisují Kanban jako přehlednou tabuli (Task Board) zobrazující proces vývoje rozdělený na jednotlivé fáze (například Backlog, in Progress, Done). Na kartičkách jsou uvedeny jednotlivé úkoly a je stanoven limit kartiček (úkolů) v jednotlivých fázích. Kanban, na rozdíl od Scrumu, není sám o sobě proces. Úspěšnou implementaci Kanbanu je nutné vždy o něco doplnit. Kanban tak lze obohatit o programovací praktiky Extreme Programming, Stand-up Meetingy, User Stories a podobně. Naopak principy Kanbanu jsou často využívány pro Scrum, například při snaze o snížení počtu rozpracovaných User Stories a jejich rychlejší dokončení. Kanban bývá implementován v situacích, kdy vzhledem k povaze práce není iterativní vývoj vhodný. Jedná se například o práci operations and maintanance týmů. Pro vývoj produktu je naopak vhodný Scrum.

Kombinování metodik Scrum a Kanban dalo za vznik hybridní agilní metodice Scrumban. Tato metodika je navržena tak, aby se přizpůsobila dynamicky se měnícím požadavkům zákazníka a častým problémům s kódováním. Scrumban zahrnuje best practices z obou metodik. Ze Scrum zavádí User Stories, Stand-up Mettingy a myšlenku samoorganizujících se týmu. Narozdíl od Scrumu již neobsahuje sprinty, které jsou vnímány jako problematické, kdy mohou nastat situace, při kterých například jeden

vývojář čeká na dokončení práce jiného vývojáře. Tento nedostatek odstraňuje best practices z Kanbanu zavedením koordinačního mechanizmu tahu pro omezení rozpracované práce (work in progress – WIP). Důležité je zde věnovat pozornost stanovení optimálních limitů WIP pro dosažení optimálního toku procesu. Zavedením limitů WIP dochází k omezení multitaskingu (provádění více procesů současně) a ke zlepšení produktivity procesu softwarového vývoje (Yilmaz a O'Connor, 2016).

4.2.6 Další přístupy a praktiky používané při vývoji softwaru

Kromě Scrumu a Kanbanu existuje řada dalších agilních metodik. Níže jsou některé z nich uvedeny a krátce charakterizovány.

- Lean Development

  Poskytuje řadu principů a pravidel, jimiž se snaží dosáhnout snížení času a rozpočtu potřebného pro vývoj softwaru na třetinu, a zmenšení četnosti výskytu chyb na třetinu (Kadlec, 2004). Šochová a Kunce (2014) dodávají, že se jedná spíše o přístup podobný agilnímu, a konkrétní metodikou aplikující myšlenky „leanu" (štíhlého procesu) je pak například Kanban.
- Extrémní programování (XP)

  Základní myšlenkou je využití běžně známých postupů, které se osvědčily při vývoji, a jejich použití dovést do extrému. Pokud se například osvědčilo testování, testuje se vše a neustále, pokud se osvědčily revize kódu, kód bude ustavičně revidován (Kadlec, 2004).
- Future Driven Developlment

  Podle Buchalcevové (2009) se tato metodika liší od ostatních agilních metodik tím, že definuje procesy a klade důraz na modelování předem. Je založena na vysoce iterativním vývoji a je řízena užitnými vlastnostmi produktu (features).
- Test Driven Development

  Tato metodika spatřuje testování jako základ vývojového procesu. Je založena na principu, že ještě před naprogramováním zdrojového kódu určité funkcionality, je nutné pro tuto funkčnost nejprve napsat testovací kód. Až po dokončení testovacího kódu je možné programovat (Kadlec, 2004).

V následujících kapitolách jsou popsány přístupy a praktiky často používané v kombinaci s agilními metodikami.

4.2.6.1 User Experience

User Experience není metodikou, ale jedná se o dnes poměrně rozšířený přístup k budování softwaru. V této kapitole je zmíněn z toho důvodu, že jeho praktické využití bude představeno v případové studii, především při sběru požadavků a návrhu uživatelského rozhraní.

Pojem User Experience (UX) představuje přístup, ale i sadu praktik a metod, k tvorbě nejen softwarových produktů. UX proces lze rozdělit do následujících pěti vrstev (Nemberg, 2014), viz obrázek č. 8:

- **Strategii (Strategy)**, která definuje smysl projektu, očekávání, uživatele a jejich potřeby.
- **Rozsah (Scope)**, který je vymezen požadavky a formuluje funkcionalitu.
- **Strukturu (Structure)**, určující celky (moduly, agendy a podobně), ze kterých je produkt složen.
- **Kostru (Skeleton),** jež předepisuje, jaké komponenty (wireframes – formuláře, stránky) lze v rámci produktu používat.
- **Vizuální design (Surface)**, který je reprezentován uživatelským rozhraním (User Interface, zkráceně UI) a pomocí kterého dochází k interakci s uživatelem. Všechny předešlé vrstvy by se v něm měly odrážet.

Obrázek č. 8 – Vrstvy v UX (převzato z Nemberg, 2014)

UX zahrnuje několik disciplín a specifických činností, které jsou v jejich rámci vykonávány (UX Asociace, 2016):

- **Uživatelský výzkum**, ve kterém probíhají rozhovory se zákazníkem, skupinové diskuze, on-line výzkumy či testování použitelnosti.
- **Interakční design** (IxD), v jehož rámci dochází k návrhu složitějších interakcí u aplikací či konkrétních stránek při návrhu webu.
- **Informační architektura** (IA), která vymezuje informační strukturu.
- **Vizuální design**, kdy dochází k návrhu vzhledu a jeho působení na emoce pomocí grafických prvků.

Pro User Experience (UX) bývá často používán český ekvivalent „uživatelský prožitek“. Vyjadřuje, jak je uživatel spokojen s produktem. UX lze tedy označit jako pocit uživatele, který v něm zůstane po interakci s produktem, lidmi či událostmi (FatDUX Group, 2013).

Smyslem a cílem UX je navržení produktů či služeb, které umožní naplnit stanovený uživatelský prožitek na straně uživatele. Hlavním znakem UX designu je jeho zaměření na potřeby, cíle, ale i omezení uživatele. Opravdové porozumění cílovým uživatelům je tak stěžejním východiskem UX. Návrh v UX by měl být funkční, smysluplný, zabývat se reálnými problémy a také působit estetickým a emotivním dojmem. UX přistupuje k návrhu komplexně, zkoumá tedy celý ekosystém související s daným produktem (Šrutka, uvedeno ve Válka, 2011).

Při návrhu uživatelského prožitku se můžeme v praxi setkat s dvěma pojetími. UX buď můžeme chápat jako samostatnou disciplínu vedle ostatních oborů, jako jsou informační architektura, informační design, návrh rozhraní či návrh interakce. Nebo lze UX chápat jako proces, do kterého další disciplíny vstupují. Potom je možné veškeré profese, které mají podíl na návrhu produktu, považovat za návrháře uživatelského prožitku (UX Asociace, 2016).

4.2.6.2 User Centered Design

Dalším pojmem, respektive přístup, který je v souvislosti s User Experience často používán, je User-Centered Design (UCD). UCD je iterativní proces jehož základním předpokladem je zapojení uživatele ve všech vývojových fázích. UCD je návrh zaměřený na hluboké porozumění uživatelům, kteří jsou pomyslně umístěni do středu aplikace. Soustředí se na potřeby uživatele a jeho schopnosti se v aplikaci orientovat. Cílem je dosáhnout takového návrhu, který poskytne funkční řešení s vysokým uživatelským prožitkem (U. S. Department of Health & Human Services, 2016).

User Centered Design je realizován v několika fázích, které jsou znázorněny na obrázku č. 9 a kterými jsou: (EasyUX Team, 2016):

- **Discover**

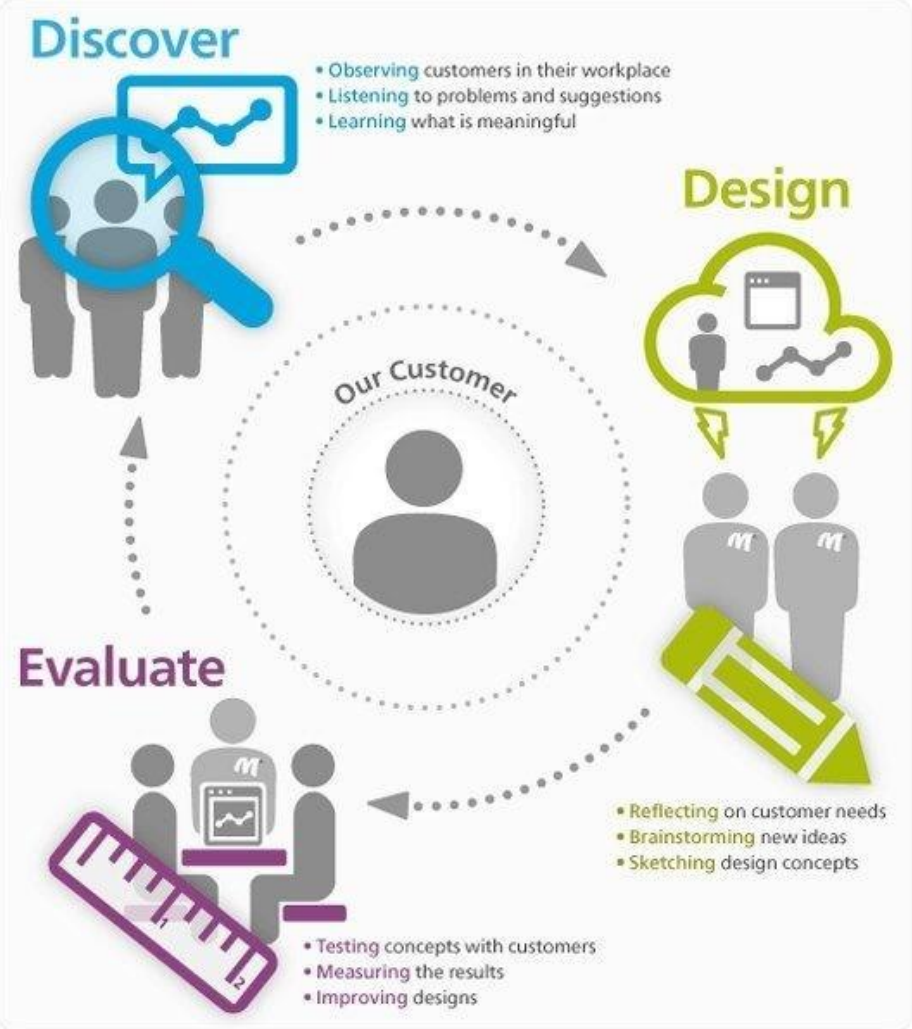
  Fáze, ve které dochází k nalezení uživatelů a identifikaci jejich potřeb. Uživatelé jsou pozorování přímo ve svém pracovním prostředí a je naslouchánо jejich problémům a návrhům. Díky tomu lze odhalit to, co je pro zákazníka podstatné a smysluplné.

- **Design**

  Fáze návrhu odráží potřeby zákazníka, vznikají zde nové skupinové nápady a na jejich základě jsou načrtnuty základní koncepty návrhu.

- **Evaluate**

  Ve fázi hodnocení jsou koncepty otestovány se zákazníkem, vyhodnoceny výsledky a podle nich návrh dále zkvalitňován.



Obrázek č. 9 – User Centered Design (převzato z EasyUX Team, 2016)

4.2.6.3 Persony (Personas)

Jedna z praktik souvisejících s UCD je použití person při procesu tvorby softwaru. Ačkoliv se jedná o praktiku hojně využívanou v oblasti marketingu, zavádění person je velice vhodné i při vývoji softwaru.

Hlavní myšlenkou tvorby person je, že pokud má být navrhovaný software efektivní, musí být navrhován pro konkrétní osobu. Persona tedy představuje archetypální, často fiktivní, osobu založenou na znalosti reálných uživatelů. Na rozdíl od aktérů, kteří budou specifikováni v kapitole o diagramu případu užití, persony nejsou rolemi uživatele systému. Persony popisují vzorovou instanci aktéra (Ambler, 2014).

Myšák (2013) personu popisuje jako abstraktní reprezentaci cílové skupiny uživatelů. Persony mají nejčastěji formu detailního profilu osoby, který obsahuje informace jako jméno, pohlaví, věk, pracovní pozice, pracovní návyky, cíle a odpovědnosti, informace o osobním životě a podobně. Je také vhodné opatřit personu fotografií. Persony jsou nástrojem k lepšímu poznání zákazníků a pochopení jejich skutečných potřeb.

Níže jsou uvedena některá doporučení při vytváření person podle Coopera (2004, uvedeno v Ambler, 2014):

- Není dobré vymýšlet zcela fiktivní persony, ale vytvářet je jako vedlejší produkt rozhovorů se zákazníkem při analýze požadavků
- Většinu uživatelů lze popsat třemi hlavními personami
- Poznáním cílů persony lze odvodit, co má systém vykonávat
- Pokud uživatelské rozhraní jednoho uživatele neuspokojuje potřeby druhého, jedná se pravděpodobně o další typ persony
- Někdy je dobré, vzhledem k povaze projektu, identifikovat „negativní“ persony, které představují osoby, pro které není produkt navrhován

4.3 Unified Modeling Language

Za vznikem každého komplexního systému stojí konkrétní představa. Je nutné, aby byla tato vize od počátku projektu dokonale pochopena vývojáři i vizionáři, kteří by ji měli mít neustále na paměti. Pro správné pochopení rozdílného vidění různých účastníků a zajištění úspěchu projektu slouží standardizovaný unifikovaný modelovací jazyk. Díky němu lze zachytit obecnou představu o systému a předat ji všem účastníkům (Schmuller, 2001). Snahy analytiků a designérů, kteří v 80. a 90. letech minulého století usilovali o vytvoření vhodné metody k popsání objektově orientované analýzy a návrhu, vedly ke vzniku modelovacího jazyka UML (Kanisová a Müller, 2006). Na konci 90. let minulého století přišli Grady Booch, Ivar Jacobson a Jim Rumbaugh s finální verzí jazyka UML, který je v současnosti považovaný za standard. Dnes je definován standardizační skupinou Object Management Group (OMG) v aktuální verzi 2.5 (Page-Jones, 2001).

Jazyk UML tvoří řada grafických prvků, jejichž vzájemnou kombinací lze podle pevně daných pravidel vytvářet diagramy (Schmuller, 2001). Tyto diagramy zachycují systém z různých úhlů pohledu a s různou mírou abstrakce. Zaměření je jen na podstatné prvky a nahlížení na komplexní systém pomocí několika zdánlivě nezávislých pohledů, vede k lepšímu porozumění problému a lepší komunikaci (Buchalcevová, Pavlíčková a Pavlíček, 2007).

Podle OMG (2007) je UML vizuální modelovací jazyk pro specifikaci, sestavování a dokumentaci prvků systému, nezávislý na aplikační doméně (například obchod, školství) a implementační platformě (například JAVA, .NET). Arlow a Neustadt (2007) uvádějí, že v jazyku UML se pohlíží na svět, jako kolekci vzájemně se ovlivňujících objektů, které zde vystupují jako pevné seskupení dat a funkcí, což koresponduje se základním předpokladem objektově orientovaného přístupu. Podle jednoho z duchovních otců UML Gradyho Booche (uvedeno v Arlow a Neustadt, 2007) se jazyk UML skládá ze tří stavebních bloků, kterými jsou předměty (things), vztahy (relationships) a diagramy (diagrams).

4.3.1 Součásti jazyka UML

Součásti jazyka UML zahrnují:

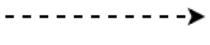
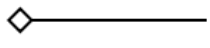
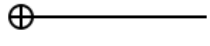
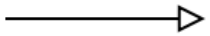
Předměty (things)

Předměty, někdy označované jako abstrakce, představují samotné prvky systému. Lze je dělit na (Arlow a Neustadt, 2007):

- Strukturní abstrakce (structural things)
  V UML modelech představují například třídy, případy užití, komponenty, uzly, rozhraní a podobně.
- Chování (behavioural things)
  Představují například stav, interakce a podobně.
- Seskupení (grouping things)
  Umožňují seskupit prvky do soudržných, významově souvisejících jednotek.
- Poznámky (annotational things)
  K modelu lze připojit takzvané anotace, pro zachycení dalších informací.

Relace (ralationships)

Pomocí relací lze vyjádřit sémantický vztah mezi dvěma předměty. Určují vztahy mezi strukturními abstrakcemi a seskupováním. Přehled vybraných hlavních relací je zachycen na obrázku č. 10 (Arlow a Neustadt, 2007).

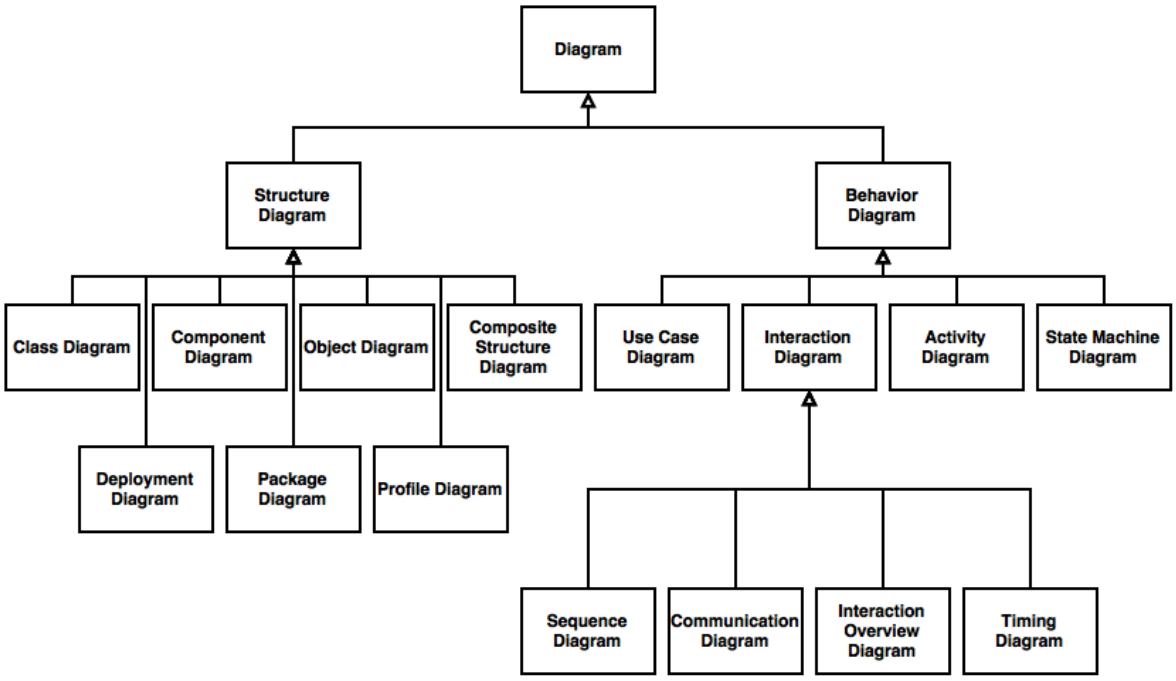
Typ relace	Syntaxe UML zdroj cíl	Stručný popis
Závislost (Dependency)		Změna jednoho prvku se promítá do druhého.
Asociace (Association)		Popis množiny spojení mezi objekty.
Agregace (Aggregation)		Vyjadřuje vztah celek / součást. Jeden objekt((celek), využívá služeb jiného objektu (části).
Kompozice (Composition)		Silnější forma agregace. Funguje za většího omezení.
Ochranná nádoba (Containment)		Cílový prvek je vnořen ve zdrojovém prvku.
Zobecnění (Generalization)		Jeden prvek je specializací prvku druhého.Specifický prvek je konzistentní s obecným prvkem, ale obsahuje více informací.
Realizace (Realization)		Znázorňuje spolupráci objektů, kdy jeden klasifikátor definuje dohodu, jejíž uskutečnění zaručuje druhý klasifikátor.

Obrázek č. 10 – Hlavní typy relací používaných v jazyce UML (zpracováno podle Arlow a Neustadt, 2007)

Diagramy

Při tvorbě diagramů je důležité si uvědomit, že diagram není modelem, ale představuje pohled na model z určitého hlediska. Ve všech CASE nástrojích založených na jazyku UML jsou nově zadávané elementy v diagramu ukládány do repositáře modelu. Diagramy tak slouží i jako základní mechanismus pro zadávání nových informací do modelu. Odstranění předmětů a relací z diagramu ještě nevede k jejich odstranění z modelu (Arlow a Neustadt, 2007).

Z obrázku č. 11 jsou patrné dva hlavní typy diagramů v jazyku UML.



Obrázek č. 11 – Kategorizace diagramů v jazyce UML 2.5 (zpracováno podle OMG, 2015)

Statickou strukturu objektů v systému zachycují diagramy struktury (Structure Diagrams), které zobrazují prvky systému bez ohledu na čas. Prvky v diagramu struktury reprezentují smysluplné pojetí aplikace a představují reálné, abstraktní a implementační koncepty. Nezabývají se detaily dynamického chování, které jsou znázorněny diagramy chování (OMG, 2015).

Diagramy chování (Behavior Diagrams) zobrazují dynamické chování objektů v systému, včetně jejich spolupráce, aktivit, stavů a postupů. Dynamické chování systému lze popsat jako řadu změn v systému v průběhu času. Několik diagramů chování lze ještě dále klasifikovat do samostatné skupiny digramů interakce (Interaction Diagrams), které popisují interakci mezi jednotlivými částmi systému (OMG, 2015).

Tato taxonomie poskytuje logické uspořádání hlavních druhů diagramů. Nicméně se nevylučuje kombinování různých typů diagramů, kdy jeden diagram kombinuje strukturální prvky a prvky chování (OMG, 2015). Arlow a Neustadt (2007) také uvádějí, že není pevně stanovené přesné pořadí, podle kterého by měly být UML diagramy vytvářeny. Při modelování se většinou začíná diagramem případu užití (Use Case Diagram), který poskytuje dobrou představu o rozsahu platnosti navrhovaného

systému. Kromě diagramu případů užití je často jako jeden z prvních diagramů modelován diagram tříd (Class Diagram), na kterém lze identifikovat prvky systému, jejich vztahy a dobře ilustruje strukturu systému. Výjimkou není ani práce na několika diagramech najednou, kdy jsou v průběhu práce postupně odhalovány nové detaily systému.

V následující tabulce č. 3 jsou uvedeny diagramy dle UML 2.5 a jejich krátká specifikace.

Diagram	Specifikace
Diagram tříd (Class Diagram)	Třída předepisuje vlastnosti a chování určité množiny objektů. Diagram tříd zachycuje všechny třídy modelovaného systému a vztahy mezi nimi.
Diagram komponent (Component Diagram)	Komponenta je modulární část systému, která zapouzdřuje svůj obsah. Diagram zachycuje komponenty, ze kterých se skládá aplikace, systém nebo podnik.
Objektový diagram (Object Diagram)	Zachycuje objekty a jejich vztahy v určitém časovém okamžiku. Označován také jako diagram instancí.
Diagram vnitřní struktury (Composite Structure Diagram)	Zobrazuje interní strukturu prvku (například třídu, případ užití a podobně) a jeho spolupráci s ostatními prvky systému.
Diagram nasazení (Deployment Diagram)	Popisuje fyzickou architekturu počítačového systému (hardwarové zdroje) a rozložení softwarových komponent na těchto zdrojích a jejich spolupráci.
Diagram balíčků (Package Diagram)	Umožňuje elementy modelu (například třídy) uspořádat do skupin (balíčků) a zachytit závislosti mezi nimi.
Diagram profilu (Profile Diagram)	Slouží pro znázornění rozšíření jazyka UML pomocí stereotypů.

Diagram	Specifikace
Diagram případů užití (Use Case Diagram)	Zachycuje chování systému z pohledu uživatele. Popisuje, jaké činnosti jsou v rámci systému vykonávány a jaké typy uživatelů (aktérů) se v systému vyskytují.
Diagram aktivit (Activity Diagram)	Umožňuje zachytit posloupnost činností. Slouží k modelování byznys procesů, logiky scénářů a podobně. Jsou objektově orientovanými vývojovými diagramy, jejichž sémantika je založena na Petriho sítích.
Stavový diagram (State Machine Diagram)	Slouží k vyjádření jednotlivých stavů objektů a přechody mezi těmito stavy.
Sekvenční diagram (Sequence Diagram)	Používá se k zachycení interakce mezi objekty za pomocí časové posloupnosti zasílání zpráv mezi těmito objekty.
Diagram komunikace (Communication Diagram)	Zachycuje vztahy a toky zpráv mezi instancemi. Sémanticky podobný sekvenčnímu diagramu, ale důraz je více kladen na strukturální aspekty interakce.
Diagram přehledu interakcí (Interaction Overview Diagram)	Představuje variantu diagramu aktivit s fragmenty sekvenčních diagramů. Dává přehled o toku řízení v systému nebo procesu.
Diagram časování (Timing Diagram)	Dává přehled o změně stavu nebo podmínky objektu v čase.

Tabulka č. 3 – Přehled diagramů UML 2.5 (zpracováno podle Bruckner a kol., 2012 a Arlow a Neustadt, 2007)

V praxi je možné setkat se i s dalšími typy diagramů, kterými jsou například Model Diagram, Internal Structure Diagram, Collaboration Use Diagram, Manifestation Diagram, Network Architecture Diagram, Information Flow Diagram, Behavioral State Machine Diagram a Protocol State Machine Diagram. Tyto diagramy však nejsou

součástí oficiální UML 2.5 taxonomie diagramů a v zásadě slouží k upřesnění způsobu použití vybraných původních diagramů pro specifické oblasti návrhu.

V následujících kapitolách jsou blíže popsány nejčastěji používané diagramy jazyka UML. Diagramy byly vybrány i z důvodu vhodnosti jejich použití v praktické části.

4.3.2 Diagram případů užití (Use Case Diagram)

Buchalcevová, Pavlíčková a Pavlíček (2007) uvádějí, že diagram případů užití popisuje chování systému z pohledu uživatele. Diagram odhaluje, jaké typy uživatelů se systémem pracují a které činnosti jsou vykonávány. Use Case Diagram tedy slouží k modelování funkcionality systému, kdy jednu funkčnost systému reprezentuje právě jeden případ užití (Use Case, zkráceně UC). Arlow a Neustadt (2007) modelování případů užití přirovnávají ke specifickému způsobu inženýrství požadavků a slouží tak jako doplněk k získávání a dokumentování požadavků. Modelování požadavků provázejí následující aktivity:

- Rozpoznání hranic systému
- Identifikace aktérů (actors)
- Vyhledání případů užití a jejich specifikace
- Vytváření scénářů
- Opakování výše uvedených akcí, dokud nejsou hranice systému, aktéři a případy užití konzistentní

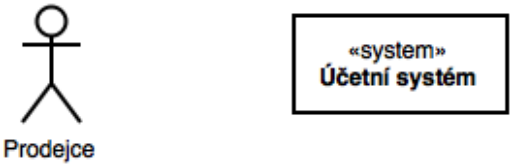
Po dokončení zmíněných aktivit je výsledkem model případů užití, jež se skládá ze čtyř komponent (Arlow a Neustadt, 2007):

- **Aktéři (actors)**, kteří představují role osob nebo předmětů v systému.
- **Případy užití (use cases)** jsou činnosti (funkcionality), které aktéři mohou v rámci systému vykonávat.
- **Relace (relationships)** označují vztahy mezi aktéry a případy užití.
- **Hranice systému (system boundary)**, která označuje území nebo hranice systému kolem případů užití.

Aktér (Actor)

Pod pojmem aktér se rozumí externí entita, u které dochází k výměně informací se systémem. V rolích, které aktér reprezentuje, nemusí vystupovat pouze lidé, ale i hardwarové zařízení, externí systém nebo čas. Jeden fyzický uživatel může vystupovat ve více rolích a naopak jednu roli může vykonávat vícero uživatelů (Buchalcevová, Pavlíčková a Pavlíček, 2007). Kanisová a Müller (2006) dále uvádějí, že případy užití jsou vykonávány právě aktéry, přičemž platí, že jeden aktér může vykonávat více případů užití a jeden případ užití může být vykonáván více aktéry. Buchalcevová, Pavlíčková a Pavlíček (2007) doplňují, že identifikací všech aktérů dochází k vymezení okolí systému a určení jeho hranic.

Na obrázku č. 12 jsou znázorněny symboly používané v UML, které reprezentují aktéry, přičemž obě dvě podoby notace jsou možné. Symbol postavy je nejčastěji používán u rolí, které jsou přiděleny osobám. Obdélník, který může být opatřen ještě stereotypem, značí role, které představují další systémy, hardwarové komponenty, čas a podobně (Arlow a Neustadt, 2007).



Obrázek č. 12 – Příklady používaných symbolů aktérů v UML (vlastní zpracování, 2016)

Případ užití (Use Case)

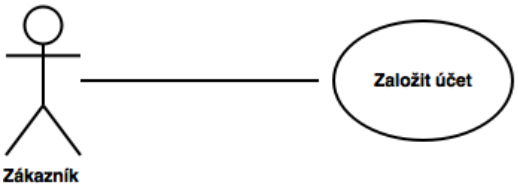
Arlow a Neustadt (2007) definují případ užití jako specifikaci určité funkcionality systému. Případ užití je vždy napsán z pohledu účastníka systému (aktéra) a účastníkem je i vždy iniciován. Podle Brucknera (2012) je funkcionalita, kterou případ užití vyjadřuje, realizována scénářem případu užití. Scénář představuje slovní popis sekvence interakcí mezi aktérem a systémem. Je tak vyjádřením jednoho konkrétního průchodu případem užití. Případ užití většinou obsahuje jeden hlavní scénář (ideální průchod případem užití) a několik alternativních scénářů. Případ užití je tak kolekcí možných scénářů. Případ užití je v notaci UML znázorněn elipsou s názvem případu užití, viz obrázek č. 13.

Relace (relationships)

V diagramu případu užití se vyskytují tři druhy relací, neboli vztahů (Buchalcevová, Pavlíčková a Pavlíček, 2007):

- **Vztah mezi aktérem a případem užití**

  Vztah představuje tok informací mezi aktérem a případem užití. Je znázorněn asociací - rovnou čarou - mezi aktérem a případem užití, někdy též označován jako komunikační asociace, viz obrázek č. 13.



Obrázek č. 13 – Příklad relace mezi aktérem a případem užití v UML (Vlastní zpracování)

- **Vztah mezi případy užití**

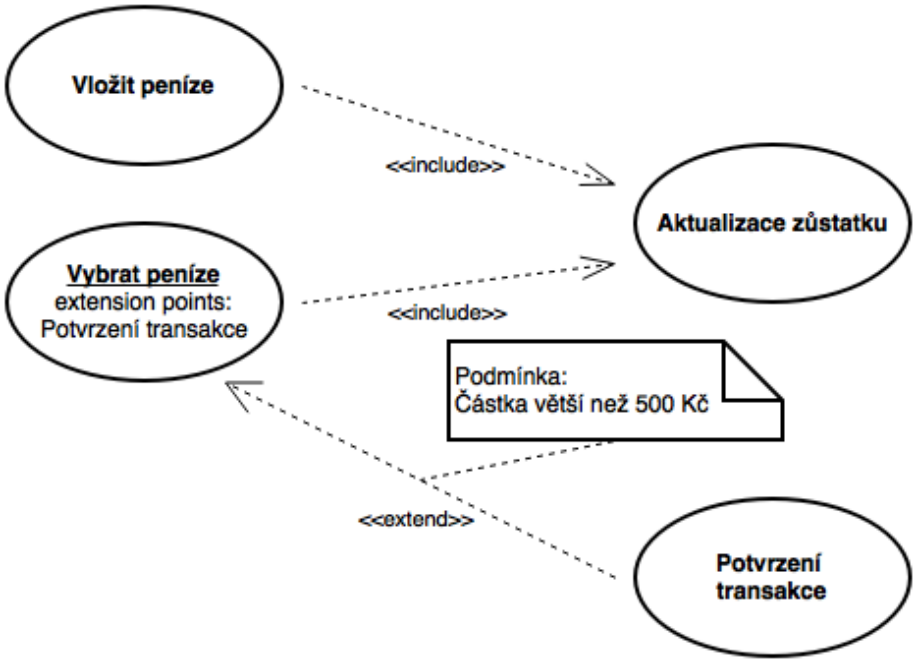
  Mezi případy užití se mohou vyskytovat tři typy vztahů – include, extend a generalization.

  - Include

    Pomocí relace include, viz obrázek č. 14, je do případu užití zahrnut jiný případ užití. Používá se v situacích, kdy se opakující se činnost v různých případech užití vyjme do samostatného případu užití a na ten je poté odkazováno. Vložený případ užití je nedílnou součástí základního případu užití, který nemůže existovat sám o sobě.

  - Extend

    Relace typu extend, viz obrázek č. 14, rozšiřuje případ užití v určitém místě takzvaném bodu rozšíření (extension point). K rozšíření základního případu užití dochází pouze v případě splnění podmínky. Původní případ užití tak může existovat sám o sobě, na rozdíl od relace typu include.



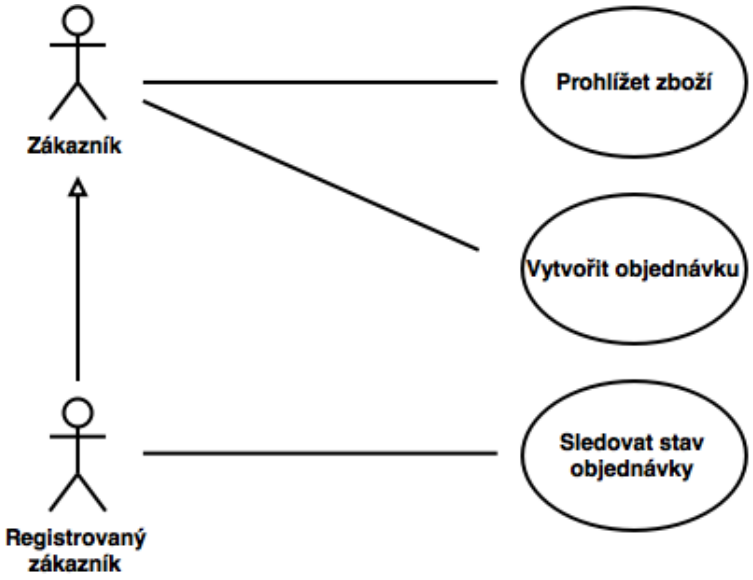
Obrázek č. 14 – Příklad relace typu include a extend v UML (vlastní zpracování, 2016)

- Generalization

  Méně často modelovaný typ relace je zmíněn jen pro úplnost. Umožňuje zachytit obecné chování, které platí pro několik specializovaných případů užití, do mateřského případu užití. Tato technika zvyšuje nepřehlednost a zhoršuje srozumitelnost modelu a také přináší úskalí při psaní scénářů.

- **Vztah mezi aktéry**

  Mezi aktéry je možné zavést relaci typu generalizace, kdy potomek dědí všechny případy užití předka a zároveň může přidat další případy užití specifické pro roli, ve které vystupuje. Příklad generalizace mezi aktéry je uveden na obrázku č. 15.



Obrázek č. 15 – Příklad relace generalizace aktérů v UML (vlastní zpracování, 2016)

Hranice systému (system boundary)

Systém od zbytku světa oddělují hranice systému. Hranice systému stanovují aktéři, kteří vymezují okolí systému, a funkcionalita (případy užití), již systém definuje. V UML notaci je hranice systému znázorněna rámečkem s názvem systému. Aktéři stojí mimo rámeček, kdežto případy užití jsou vyznačeny uvnitř (Arlow a Neustadt, 2007).

4.3.3 Diagram tříd (Class Diagram)

Diagram tříd zobrazuje strukturu tříd modelovaného systému. Zachycuje typy objektů, které v systému vystupují a jejich vztahy. Pohled na systém je statický, nevyjadřuje tedy interakce mezi třídami, jež nastávají v čase. Třída zde vystupuje jako základní element diagramu, jenž představuje abstrakci objektů se stejnými vlastnosti, chováním a vztahy k dalším objektům. Na třídu lze nahlížet jako na kolekci objektů stejného typu na úrovni analýzy, nebo jako na šablonu, podle které jsou vytvářeny objekty (instance) na úrovni návrhu a implementace (Bruckner a kol., 2012).

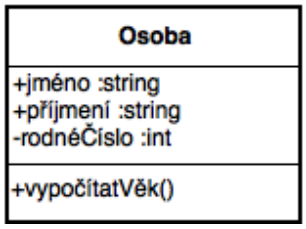
Buchalcevová, Pavlíčková a Pavlíček (2007) dále uvádějí, že je nutné si uvědomit, že class diagram se zpravidla začíná vytvářet již ve fázi analýzy a v dalších fázích je postupně doplňován a zpřesňován. Mimo jiné slouží i jako základ pro implementaci a je součástí dokumentace. Proto je nutné si uvědomit, jaký je jeho konkrétní účel

a co je jeho prostřednictvím třeba zachytit. Podle výše uvedených autorů, tak lze v modelu rozlišovat tři úrovně abstrakce:

- Konceptuální úroveň (doménový, analytický model)
  Model tvoří takzvané Business classes, které označují třídy z problémové oblasti. Uvedeny jsou většinou pouze názvy atributů, stěžejní metody a jsou naznačeny vztahy mezi třídami.
- Designová úroveň (model návrhu)
  Model je rozšířen o další třídy (třídy pro systémové události, třídy uživatelského rozhraní) a jsou zpřesňovány a doplňovány popisy tříd (viditelnost atributů a metod, datové typy, návratové hodnoty a podobně). Pozornost je také věnována vazbám mezi třídami.
- Implementační úroveň (implementační model)
  Zaměřuje se na zobrazení implementovaného kódu do grafické podoby.

Arlow a Neustadt (2007) k výše uvedenému dodávají, že podle účelu diagramu, jsou použity jen některé, či všechny oddíly tříd s různou úrovní podrobnosti (viz syntaxe třídy v jazyce UML dále v textu). Pokud je například potřeba zachytit pouze relace mezi jednotlivými třídami, stačí využít pouze oddíl s názvem třídy. Jestliže diagram slouží pro mapování tříd do tabulek relační databáze, je poté kromě oddílu s názvem třídy nutné připojit oddíl s atributy a jejich datovými typy a případně dalšími informacemi.

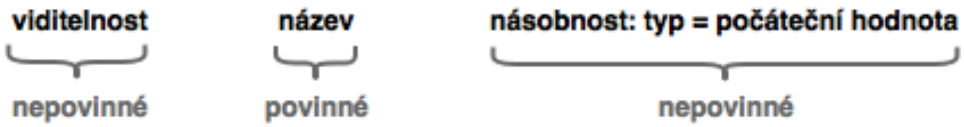
Grafickou syntaxi třídy v jazyce UML představuje obdélník složený ze tří částí. Příklad třídy je uveden na obrázku č. 16.



Obrázek č. 16 – Příklad třídy v modelu tříd (vlastní zpracování, 2016)

V horním oddílu se nachází název třídy, psaný jako řetězec slov, z nich každé začíná velkým písmenem, je psané tučně a zarovnané na střed. Prostřední oddíl slouží pro zápis atributů, které nesou informace o objektu. Názvy atributů jsou psány malým

písmenem se zarovnáním vlevo. Pro zápis atributu se používá syntaxe uvedená na obrázku č. 17, přičemž povinnou částí je jen název (Arlow a Neustadt, 2007).



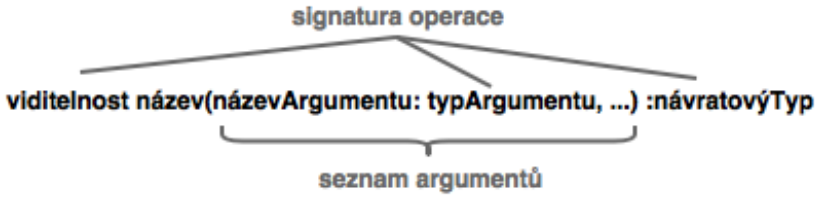
Obrázek č. 17 – Syntaxe pro zápis atributu v diagramu tříd (převzato z Arlow a Neustadt, 2007)

Pomocí viditelnosti lze definovat, zda bude atribut viditelný i mimo třídu. Přípustné typy viditelnosti v UML jsou definované v tabulce č. 4. Násobnost určuje počet prvků v atributu (například „adress (3): string“ říká, že adresa se skládá z pole tří řetězců). Kromě datového typu je také možné stanovit počáteční hodnotu, které atribut nabyde ihned po vytvoření dané instance. (Arlow a Neustadt, 2007).

Symbol	Typ viditelnosti	Přístup
+	Veřejný (public)	Všem objektům s přístupem k dané třídě
-	Soukromý (private)	Pouze instance dané třídy
#	Chráněný (protected)	Pouze instance dané třídy a její potomci
~	Balíček (package)	Objekty, které jsou uvnitř stejného balíčku

Tabulka č. 4 – Typy viditelnosti v diagramu tříd (zpracováno podle Arlow a Neustadt, 2007)

Do spodního oddílu třídy se zapisují metody, též označované jako operace, které představují funkce náležící k dané třídě. Jsou psány malým písmenem se zarovnáním vlevo. Pro zápis metody se používá syntaxe, viz obrázek č. 18. Identita operace je dána jedinečnou signaturou operace, která je tvořena kombinací názvu operace, typu předávaných argumentů a návratové hodnoty (Arlow a Neustadt, 2007).



Obrázek č. 18 – Syntaxe pro zápis metody v diagramu tříd (převzato z Arlow a Neustadt, 2007)

Vztah mezi třídami lze vyjádřit pomocí tří typů relací – asociace, agregace, kompozice, generalizace.

Asociace (association)

Asociace představuje relaci mezi spolupracujícími třídami. Díky asociaci získává jeden objekt dočasně odkaz na jiný objekt, který mu je předán formou parametru metody (Bruckner a kol., 2012).

Syntaxe asociace v UML obsahuje (Arlow a Neustadt, 2007):

- Název asociace
  Název představuje činnost, která je vykonána zdrojovým objektem pomocí objektu cílového. Měl by tedy být vyjádřen slovesem.
- Názvy rolí
  Třídám lze přiřadit názvy rolí na obou koncích asociace. Vyjadřují, v jakých rolích k sobě objekty spojené asociací vystupují. U asociace se uvádí buď název asociace, nebo názvy rolí. Použití obou názvů je nadbytečné.
- Násobnost (multiplicity)
  Představuje způsob omezení, které určuje, kolik instancí dané třídy se účastní příslušné relace. Násobnost se značí na koncích asociace. Typy násobností jsou uvedeny v tabulce č. 5.

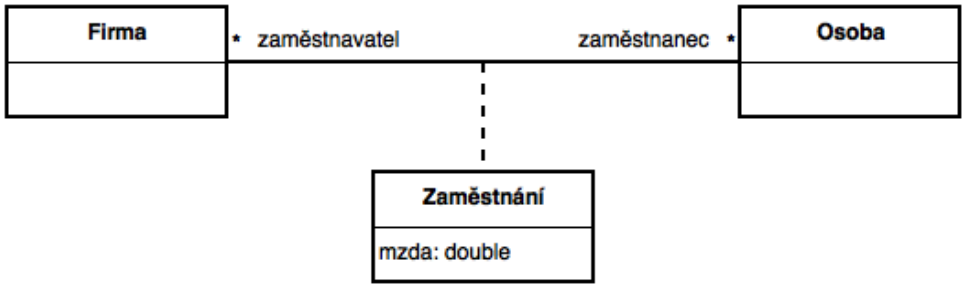
Symbol	Popis
1	Právě jedna
1..*	Jedna nebo více
0..1	Nula nebo jedna
0..*	Nula nebo více
*	Nula nebo více
m..n	V určeném intervalu (například 2..5)

Tabulka č. 5 – Typy násobnosti v UML (vlastní zpracování, 2016)

- Řiditelnost

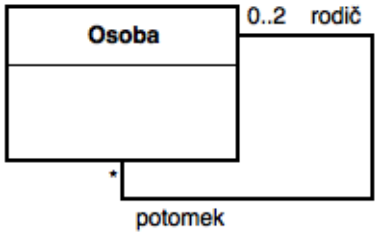
  Pomocí řiditelnosti lze vyjádřit, že zprávy lze zasílat pouze ve směru šipky asociace. Asociace, které nejsou zakončeny šipkou, jsou považované za obousměrné. Řiditelnost tak říká, že objekty jedné třídy řídí objekty třídy druhé v závislosti na multiplicitě, ale ne naopak.

S asociací se dále váže pojem asociační třída. Asociační třída se používá v situacích, kdy samotná relace obsahuje nějakou informaci, kterou nelze sémanticky přiřadit jako atribut ani jedné třídě relace (Buchalcevová, Pavlíčková a Pavlíček, 2007). Arlow a Neustadt (2007) uvádějí příklad takové asociace mezi třídami, viz obrázek č. 19. Pokud je například zavedeno další pravidlo, že osoba dostává mzdu, nelze tuto skutečnost jednoduše přiřadit jako atribut ke třídě Osoba nebo Firma, protože je vlastností samotné asociace.



Obrázek č. 19 – Příklad asociační třídy (převzato z Arlow a Neustadt, 2007)

Asociace nemusí být pouze mezi dvěma třídami, ale vztah může existovat mezi objekty téže třídy. Takováto asociace je označována jako reflexní asociací (Arlow a Neustadt, 2007). Na obrázku č. 20 je uveden příklad reflexní asociace, kdy je sledován biologický vztah mezi objekty třídy Osoba.

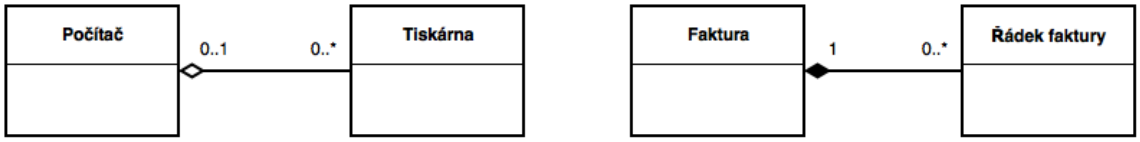


Obrázek č. 20 – Příklad reflexní asociace (vlastní zpracování, 2016)

Asociaci lze dále zpřesnit do formy agregace (aggregation) nebo kompozice (composition), které jsou popsány v následujících odstavcích.

Agregace (aggregation)

Agregace, viz obrázek č. 21, vyjadřuje vztah mezi objekty typu celek a součást. Celek řídí relaci a využívá služeb části, která pouze reaguje na požadavky. Vlastností agregace je volný vztah celku a součásti, kdy součásti mohou existovat nezávisle na celku. Celek může existovat také nezávisle na součástech, jindy je na nich naopak závislý. Součást může být sdílena i více celky. (Arlow a Neustadt, 2007). Buchalcevová, Pavlíčková a Pavlíček (2007) připomínají, že agregace je tranzitivní a asymetrická. Tranzitivnost vyjadřuje, že je-li třída C součástí třídy B a třída B součástí třídy A, potom třída A obsahuje i třídu C. Asymetričnost následně vyjadřuje stav, kdy třída A obsahuje třídu B, ale třída B nemůže obsahovat třídu A.



Obrázek č. 21 - Příklad agregace (vlevo) a kompozice (vpravo), (vlastní zpracování, 2016)

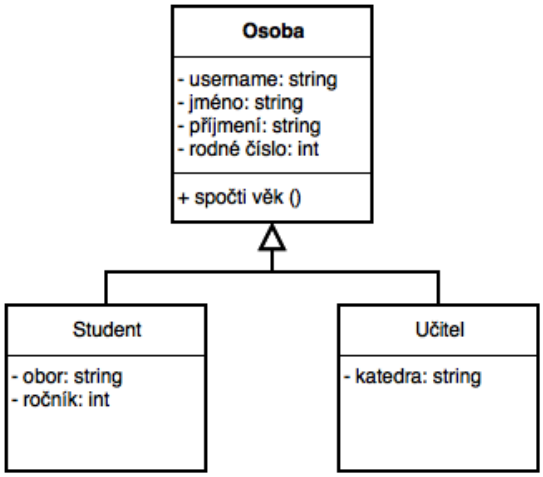
Kompozice (composition)

Kompozice, viz obrázek č. 21, vyjadřuje silnější vztah celek a součást. Kompozice je také tranzitivní a přechodná s podobnou sémantikou jako agregace. Rozdíl oproti agregaci spočívá v tom, že součásti nemohou existovat bez celku. Pokud tedy zanikne objekt celku, zanikají i objekty části. Součásti vždy patří pouze jednomu celku, který je jejich vlastníkem. Kompozice tedy vyjadřuje vztah existenční závislosti. (Arlow a Neustadt, 2007).

Generalizace

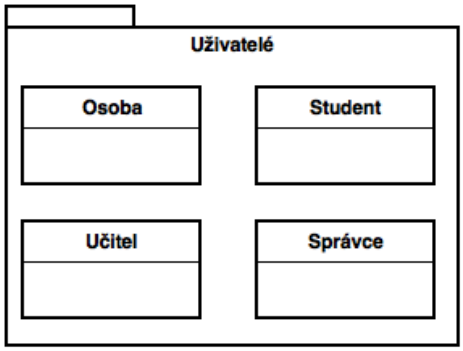
Generalizace neboli zobecnění umožňuje vyjádřit pomocí abstrakce vztah mezi mateřskou třídou a speciálním případem této mateřské třídy. Specializovaná třída dědí vlastnosti a chování mateřské třídy, které je pro všechny třídy v tomto vztahu společné,

případně přidává vlastnosti a chování typické jen pro ni. (Buchalcevová, Pavlíčková a Pavlíček, 2007). Příklad generalizace tříd je uveden na obrázku č. 22.



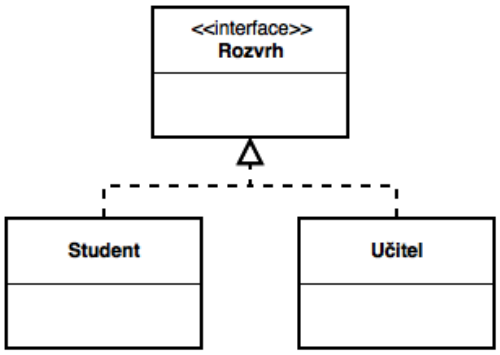
Obrázek č. 22 – Příklad generalizace tříd (vlastní zpracování, 2016)

V diagramu tříd se dále objevují elementy jako balíček (package), viz obrázek č. 23, který slouží k uspořádání tříd do sémanticky souvisejících celků.



Obrázek č. 23 – Příklad balíčku v diagramu tříd (vlastní zpracování, 2016)

Dalším elementem je rozhraní (interface), které specifikuje používání tříd, viz obrázek č. 24. Jinými slovy, z pohledu UML, třída realizuje rozhraní. Z pohledu programování lze také říci, že třída implementuje rozhraní (Buchalcevová, Pavlíčková a Pavlíček, 2007).



Obrázek č. 24 – Příklad rozhraní (interface) v diagramu tříd (vlastní zpracování, 2016)

4.3.4 Diagram aktivit (Activity Diagram)

Diagram aktivit slouží k modelování procesů jako posloupností aktivit a přechodů mezi nimi, probíhajících jak sekvenčně, tak i paralelně (Kanisová a Müller, 2006). Umožňuje modelovat logiku scénářů, strukturu obchodních procesů či pracovních postupů. Pomocí diagramu aktivit lze zachytit chování libovolného elementu (nejčastěji například případu užití, třídy, metody, rozhraní, uzlu), ke kterému ho lze připojit (Arlow a Neustadt, 2003).

V notaci UML se diagramy aktivit skládají z následujících prvků (Arlow a Neustadt, 2003):

- Akce

  Akce vyjadřuje stav nějaké činnosti. Základem diagramu aktivit jsou stavy akcí (action states) neboli aktivity a stavy dílčích aktivit (subactivity states). Stavy akcí (aktivity) představují nejmenší stavební jednotku diagramu aktivit, které nelze dále rozdělit na menší části. Aktivitu nelze přerušit, je tedy vždy vykonána celá najednou a okamžitě. Naopak dílčí aktivity lze dále dělit na jednotlivé aktivity nebo akce a mohou být přerušeny, viz obrázek č. 25.

Obrázek č. 25 – Příklad aktivity (vlevo) a dílčí aktivity (vpravo), (vlastní zpracování 2016)

V každém diagramu aktivit se vyskytují i dva speciální stavy – počáteční (initial) a koncový (final), viz obrázek č. 26. Počátečním stavem diagram aktivit začíná a koncovým stavem je ukončen.

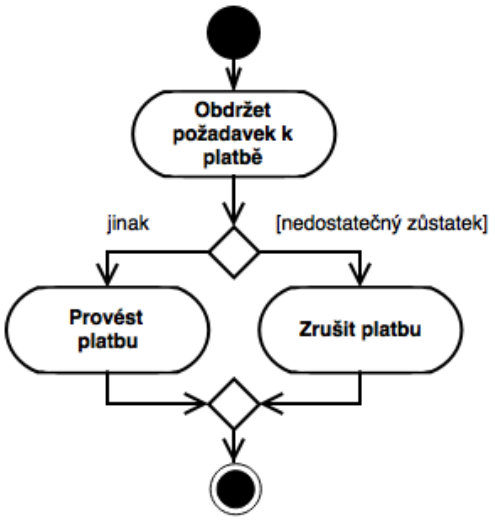
Obrázek č. 26 – UML notace počátečního (vlevo) a koncového (vpravo) stavu (vlastní zpracování, 2016)

- Přechody

  Přechody mezi stavy nastávají automaticky po ukončení příslušné akce. V UML notaci jsou přechody mezi stavy znázorněny šipkou mezi jednotlivými stavy.

- Hodnocení přechodů

  Hodnocení přechodů představuje logickou podmínku (decision) podmiňující přechod. Přechod ze stavu může být realizován pouze jednou cestou. Splněna může být tedy vždy jen jedna podmínka. Symbol pro hodnocení přechodů je kosočtverec, který zároveň slouží jako symbol pro sloučení (merge), při sloučení větví alternativních cest. (viz obrázek č. 27, kde je podmínka na obrázku uvedena v hranatých závorkách).



Obrázek č. 27 – Příklad hodnocení přechodů (vlastní zpracování, 2016)

- Větvení a spojení

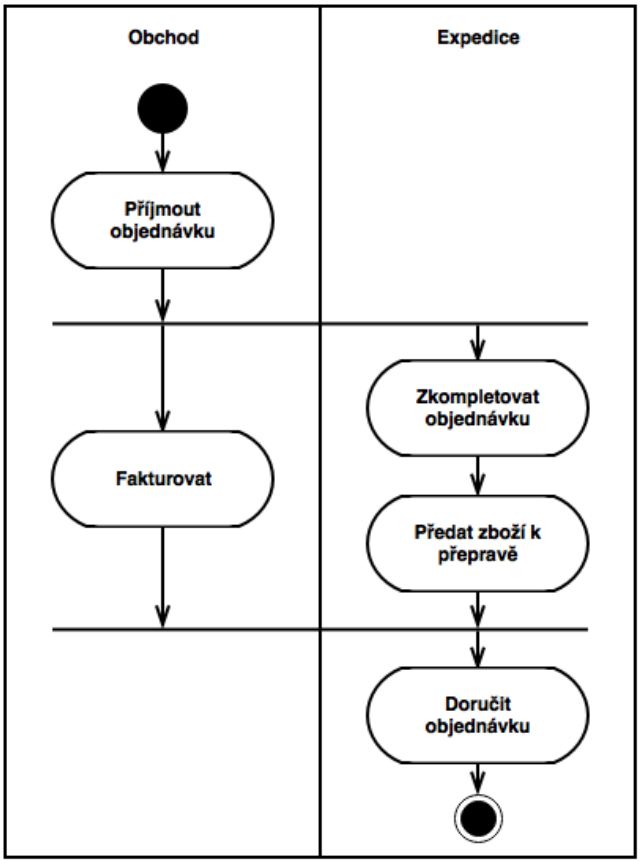
  V diagramech aktivit mohou být zachyceny i toky činností, které probíhají paralelně. Rozvětvení (forks) definuje jeden vstupní a několik výstupních přechodů, přičemž jsou všechny výstupní přechody realizovány zároveň.

Spojení (joins) slouží k synchronizaci toků, má tedy několik vstupních přechodů a pouze jeden výstupní, viz obrázek č. 28.

Obrázek č. 28 – UML notace rozvětvení (vlevo) a spojení (vpravo), (vlastní zpracování, 2016)

- Zóny (swimlanes)

  Zóny neboli „plavecké dráhy“ umožňují rozdělit diagram do významově souvisejících celků. Lze tak například specifikovat, kdo vykonává příslušné činnosti, viz obrázek č. 29.



Obrázek č. 29 – Příklad zón (vlastní zpracování, 2016)

Pro úplnost, lze diagram aktivit dále doplnit o toky objektů. Uvedeny bývají pouze důležité objekty (reprezentovány symbolem obdélníku s podtrženým názvem), který při vykonávání aktivit mění svůj stav. Další méně používanou technikou je zavedení signálů, které představují soubor informací, jež jsou asynchronně předávány mezi objekty (Kasinová a Müller, 2006).

5 Praktická část

V praktické části této práce jsou některé principy a postupy popsané v předchozích kapitolách demonstrovány v reálném prostředí. Teoretické poznatky jsou ověřovány na procesu vývoje softwaru ve společnosti Ders, s.r.o. (dále jen Ders).

V první části je představeno vývojové prostředí společnosti Ders. Nejprve je obecně popsána strategie a cíle společnosti, hlavní oblasti působnosti firmy a hodnoty, na kterých společnosti staví. Následně jsou představeny metodické přístupy používané při vývoji softwaru a popsána jejich aplikace v rámci společnosti. V návaznosti na tuto kapitolu jsou uvedeny identifikované nedostatky a rozpory s těmito metodikami a popsány možné příčiny. Stejně tak jsou navržena řešení těchto nedostatků.

V druhé části je popsána případová studie, která sleduje proces analýzy a návrhu softwaru, který je zaváděn formou modulu do již nasazeného a používaného podnikového informačního systému. Objednatelem je společnost GfK Czech, s.r.o. (dále jen GfK), která jako stávající zákazník softwarové společnosti Ders, požaduje novou funkcionalitu do svého podnikového informačního systému.

Následně je stručně charakterizována společnost GfK, nastíněna oblast jejího podnikání a představen stávající podnikový informační systém. Navazující kapitoly odhalují požadavky a představy zákazníka, vymezují procesy, které bude systém podporovat, a poskytují prvotní představu o budoucím softwarovém řešení. Identifikované procesy a struktura systému jsou modelovány s pomocí jazyka UML a zachyceny na odpovídajících diagramech.

5.1 Charakteristika vývojového prostředí

Kromě stručného představení společnosti Ders, jsou v této části zkoumány procesy související s vývojem softwaru. Konkrétně je popsána aplikace metodiky Scrum a Kanban. Proces vývoje nových produktů se řídí podle Scrumu, zatímco Kanban je nasazen v prostředí údržby a servisu. V závěru této kapitoly jsou popsány

identifikované nedostatky, chyby, jejich možné příčiny a řešení v jednom z týmu společnosti.

5.1.1 Profil firmy

Společnost Ders působí na poli poskytování softwarových řešení, systémové integrace a poradenské činnosti. Základní strategií společnosti je poskytovat produkty a služby, které evidentně přinášejí zákazníkovi přidanou hodnotu. Společnost se dále zaměřuje na spolehlivost a otevřený přístup při komunikaci se zákazníkem i uvnitř společnosti (Ders, 2014).

Hlavními oblastmi, pro které společnost Ders vyvíjí softwarová řešení, jsou (Ders, 2016b):

- **Oblast VaVaI** – Software pro zajištění podpory projektu, záměru, inovace, objevu či nápadu z oblasti výzkumu, vývoje a inovací (VaVaI).
- **Oblast Koloběhů** – V oblasti koloběhů je dodáván software pro podporu ekonomických, provozních a legislativních agend při společné tvorbě obsahu dokumentů, jejich schvalování a oběhu v rámci organizace.
- **Zakázkové aplikace** – Software na zakázku je vyvíjen při řešení problémů, které alespoň okrajově souvisí s oblastí VaVaI, oblastí koloběhů či jako nadstavba stávajících informačních systémů Akademie věd nebo veřejných vysokých škol.
- **Sociální služby** – S tvorbou softwarových řešení v oblasti sociálních služeb má společnost bohaté zkušenosti, ať už se jedná o znalosti terminologie, procesu benchmarkingu, řízení sítě včetně podpory jejího financování nebo tvorby elektronického katalogu.

Hodnoty, na kterých společnost Ders staví a které jsou vytvářeny a sdíleny všemi zaměstnanci, jsou (Ders, 2015):

- Informace, které rozhodují

  Propojení moderních technologií, selského rozumu a zkušeností z různých oborů do nástrojů, které slouží k poskytování relevantních informací, lze podpořit správné rozhodování a zároveň docílit větší konkurenceschopnosti.
- Technologie, které slouží

  Aby výsledný produkt lidem pomáhal a dobře sloužil, je nutné věnovat značnou pozornost poznání smyslu práce uživatele systému. I díky osobnímu kontaktu jsou technologie přibližovány lidem a stávají se přirozenou součástí jejich života.
- Odlišnosti, které spojují

  Spolupracující skupina je silnější než jednotlivec. Přesto je důležitá samostatnost, svoboda tvorby a myšlení jedince. Znalosti, dovednosti a názory každého člena mohou skupinu obohacovat.
- Složitost, která zjednodušuje

  Význam má vytvářet pouze takový software, který má jasný smysl a přínos. Pro předejití informačního chaosu je před uživatelem skryta vnitřní složitost systému a navenek vystupuje použitelnost a jednoduchost.

5.1.2 Metodické přístupy k budování software v Ders

Společnost Ders při tvorbě softwaru využívá několika moderních praktik a přístupů. Firemní kultura staví na agilních principech, otevřené komunikaci a úzké spolupráci se zákazníkem, který je vnímám jako rovnocenný partner. Nejen při odhalování jeho představ a návrhu uživatelského rozhraní, jsou aktivně používány přístupy User Experience (UX), ale i programátoři a ostatní techničtí pracovníci si uvědomují, že software by měl lidem sloužit a bavit je. Při samotném procesu tvorby softwaru je postupováno podle metodik Scrum a Kanban, jejichž procesy a činnosti jsou podporovány nástrojem JIRA Software.

JIRA Software je produktem australské společnosti Atlassian, která se specializuje na produkty z oblasti podpory softwarového vývoje a projektového řízení. JIRA Software je univerzální nástroj navržený pro zajištění činností všech členů týmu od plánování, sledování postupu až po implementaci softwaru. Je také

nenahraditelným nástrojem při komunikaci a podpoře zákazníků společnosti Ders. JIRA Software při podpoře řízení projektů adoptuje principy metodik Scrum, Kanban či hybridní kombinaci obou přístupů.

5.1.3 Aplikace metodiky Scrum při vývoji softwaru v Ders

Vývoj nových produktů, respektive zpracování nových požadavků na funkcionalitu, se ve společnosti Ders řídí podle metodiky Scrum a u většiny projektů probíhá následovně. Vývojový cyklus je rozdělen do dvou sprintů, z nichž každý trvá dva týdny. Na konci cyklu následuje nasazení do testovacího prostředí u klienta a jeho akceptace. Po akceptaci klienta následuje nasazení do produkce, to znamená nasazení finální verze produktu do reálného prostředí klienta.

Začátku každého cyklu předchází fáze plánování, ve které pracovník oddělení „Dodávka“ zastupující klienta (Product Owner) s klientem konzultuje podobu nových požadavků a to, co bude obsahem nadcházejícího cyklu. Klient požadavky zadává ve svém vlastním elektronickém prostředí, takzvaném agilním boardu pro správu požadavků (Product Backlog). Klient následně přiřadí požadavkům prioritu a zvolí požadavky, které chce v následujícím cyklu odbavit. Product Owner ke zvoleným požadavkům přidá přibližné odhady časové náročnosti pro vytvoření základní představy o rozsahu projektu. Každý klient má předem určenou produkční kapacitu, která je v rámci dvou sprintů spotřebovávána. Ta je vyjádřena v časových jednotkách (hodinách).

Na plánovací schůzce tým vytváří přibližnou analýzu zadaných požadavků. Pro každý požadavek je vytvořen příběh, který je provázán s ostatními příběhy. V případě, že Product Owner dokáže zodpovědět veškeré dotazy týmu, vznikne detailní zadání příběhu a je možné poskytnout přesnější odhad časové i cenové náročnosti projektu. Odpovědi na nezodpovězené otázky musí Product Owner získat u klienta. Na základě detailních odhadů klient potvrdí, které požadavky budou zpracovány. Tyto požadavky jsou následně zařazeny do Sprint Backlogu a v rámci sprintů odbavovány.

Na konci prvního sprintu je představena a předána dosavadní odpracovaná práce Product Ownerovi, který výsledky prezentuje klientovi. Klient může prezentaci

dosavadních výsledků připomínkovat. Připomínky klienta jsou podle jejich povahy zapracovány v dalším sprintu, nebo jsou zařazeny do Product Backlogu, pokud se jedná například o novou funkcionalitu. Na konci cyklu by měly být odpracovány všechny požadavky, které se tým zavázal v rámci sprintů splnit.

Následně je celý obsah cyklu nasazen do testovacího prostředí, kde vybraná skupina uživatelů klienta dostane dostatek času k odzkoušení nového systému či modulu a případnému nahlášení připomínek. Následuje fáze akceptace, ve které dochází k odbavení odhalených chyb a akceptačních připomínek z testovacího prostředí. K podpoře fáze akceptace slouží servisní board, který podporuje principy metodiky Kanban. Poté již dochází k nasazení do reálného produkčního prostředí klienta.

Aby byl uvedený proces vývoje softwaru kompletní, je nutné také zmínit, že po dokončení každého třetího cyklu následuje týden, který slouží pro zlepšení interních procesů týmu. Tento čas by měl být vyhrazen pro snížení pracovního vypětí týmu, vzdělávání, zlepšení dosavadních pracovních postupů, zefektivnění a usnadnění vykonávané práce.

K výše uvedenému procesu vývoje softwaru je třeba doplnit, že fáze akceptace není součástí vývojového procesu dle metodiky Scrum. V agilních metodikách tento proces obecně nahrazuje demonstrace produktu, získání zpětné vazby a její rychlé zapracování bez toho, aniž by probíhala formalizovaná akceptace produktu. Nicméně ve společnosti Ders je vývojový proces o tuto fázi doplněn a jistě má své opodstatnění, proto nebude tento nesoulad v kapitole 5.1.5 pojednávající o odchylkách od teoretického popisu dále rozebírán.

5.1.4 Aplikace metodiky Kanban při vývoji softwaru v Ders

Jak již bylo uvedeno v teoretické části, nasazení Kanbanu je vhodné především v prostředí maintanance týmů (týmů pro údržbu). V Dersu Kanban slouží pro odbavování chyb a jiných provozních požadavků pomocí servisního boardu. Zde se zobrazují různé incidenty, které představují události se stanovenou prioritou, jež nejsou součástí běžného fungování dané služby. Zobrazují se zde například nahlášené chyby či omezení, které vznikají v důsledku používání aplikace.

Incidenty s nejvyšší prioritou jsou označovány jako blockery, které představují závažnou chybu ohrožující fungování aplikace a jsou řešeny přednostně. Dalším typem incidentů s vysokou prioritou, které přímo souvisí s vývojem produktu, jsou akceptační připomínky. Vyřešení akceptačních připomínek je klíčové, aby mohl být produkt fakturován a klient souhlasil s jeho převzetím. V případě, že se jedná o úpravu některé funkcionality, je taková akceptační připomínka přesunuta do boardu pro správu požadavků a řešena v některém z příštích cyklů. Chyby, které se vyskytují v průběhu užívání aplikace, jsou vyřizovány na základě servisních smluv, které mimo jiné stanovují sankce za pozdní odstranění chyby.

5.1.5 Odchylky od teorie při vývoji softwaru v reálném prostředí

Při vývoji softwaru se i přes aplikování nejrůznějších metodik mohou objevovat méně či více závažné problémy, které ohrožují dodání objednaného softwaru, jeho kvalitu či přinejmenším ztěžují práci vývojářů. Proto v následujících odstavcích budou popsány identifikované odchylky, nedostatky a chyby proti ideálnímu teoretickému popisu, včetně jejich důsledků ve výše popsaném procesu vývoje softwaru za konkrétní tým společnosti Ders. Tyto odchylky v procesu vývoje se v různé skladbě a míře vyskytují v každém větším softwarovém týmu.

Za konkrétní projekt popsaný v praktické části věnované případové studii je odpovědný poměrně malý tým, který se snadno řídí (2-3 lidé) a projekt je svým rozsahem spíše menší – jedná se o přírůstkový rozvoj stabilní a dlouhodobě používané aplikace. Proto i odchylek od teoretického ideálu je logicky poměrně málo.

Nicméně pro objektivní porovnání teoretického popisu a praktické aplikace metodiky Scrum bylo provedeno šetření i v dalším týmu společnosti a zjištěné praktické poznatky jsou uvedeny v následujícím textu.

Nerespektování produkční kapacity týmu

První problém, na který lze narazit již v počátcích vývoje, je nerespektování produkční kapacity týmu. Jedním z případů je situace, kdy Product Owner nasmlouvá množství práce, které je větší než produkční kapacita týmu. Do produkční kapacity také není

zahrnutý čas, který je potřebný po nasazení produktu do produkce a který tak není ani klientovi fakturován. Produkční kapacitu týmu mohou ponížit i okolnosti, které nelze předvídat, jako je například nemoc členů týmu. V tomto případě se však už nejedná o nerespektování produkční kapacity, ale o možné ohrožení.

Následkem výše popsaných nedostatků dochází k tlaku a přetěžování členů týmu, zpoždění dodávek a následné nespokojenosti klienta. Překračování produkční kapacity ze strany Product Ownera či její špatné stanovení vede k porušení jedné ze zásad Scrumu, kdy vzniká vůči produktu technický dluh.

Vznik technického dluhu

Podle Scrumu, vybráním konkrétních User Stories (příběhů) týmem do Sprint Backlogu, vznikne závazek týmu na dokončení těchto úkolů v rámci příslušného sprintu. Jelikož se jedná o prioritizovaný seznam funkcionalit, měly by se požadavky odbavovat všechny, a to postupně od shora seznamu (Šochová a Kunce, 2014).

Výše popsané problémy s produkční kapacitou týmu mohou být jednou z příčin zpoždění dodávek produktů a vzniku tlaku na jejich rapidní vývoj. Proto je prioritou dodat to, co klient nutně požaduje a co očekávají jeho uživatelé, tedy řádně otestovanou aplikaci s požadovanou funkcionalitou. Méně prioritní úkoly, jako je například analytická dokumentace k projektu, jsou odbavovány se zpožděním. Nastávají i situace, kdy je upřednostňováno zahájení vývoje nového produktu před odbavením méně prioritních úkolů z předchozích cyklů. Důvodem může být nedostatek času na odbavení těchto méně prioritních úloh a hrozící sankce za zpoždění dodávky, ale i nezvládnutí time managementu (efektivního řízení času) a komunikace.

Průběh plánovací schůzky (Planning Meeting)

Průběh plánovací schůzky (Planning Meeting) se také vzdaluje od ideálního postupu popsaného ve Scrumu. Na plánovací schůzce by podle Šochové a Kunce (2014) měly být, za účasti celého týmu, vybrány a analyzovány konkrétní požadavky, které budou zařazeny do Sprint Backlogu a jež se tým zaváže v následujícím sprintu dokončit.

Místo toho Scrum Master přiřadí část předvybraných požadavků konkrétnímu analytikovi, který je bude zpracovávat, a je naplánována schůzka. Plánovací schůzky se účastní pouze Scrum Master, Product Owner, analytik a případně programátor pro zpřesnění odhadů. Plánování obsahu sprintů tak probíhá v rámci několika schůzek v průběhu celého cyklu, namísto jedné plánovací schůzky v úvodu prvního sprintu, kdy by měl být výsledkem úplný obsah Sprint Backlogu pro nadcházející cyklus.

Důvodem silnější role Scrum Mastera může být i další nedostatek, a to tým, který není dostatečně vyspělý na to, aby si dokázal práci sám organizovat.

Tým není samoorganizující se (self-organized)

Vzhledem k tomu, že Scrum Master je nucen vystupovat nad rámec své role a přiřazovat požadavky k odbavení konkrétním členům, ztrácí tým statut samoorganizujícího se týmu. Aby byl tým self-organized, je nutná jistá vyspělost týmu. Členové týmu by si měli uvědomovat priority, znát nastavené mantinely a procesy, které nelze překračovat či měnit.

Vzhledem k tomu, že tým postrádá jistou úroveň vyspělosti, nedokáže si sám práci organizovat. Dochází k nadměrnému pracovnímu vytížení Scrum Mastera, který se dostává spíše do role klasického teamleadera. Snížením zodpovědnosti a možnosti o věcech rozhodovat přichází tým o výhody plynoucí ze samoorganizujícího se týmu, jako je vyšší efektivita, samostatnost a pocit zodpovědnosti za svěřené úkoly.

Absence denních Stand-up meetingů

Z procesu vývoje byly odstraněny i denní Stand-up meetingy, které mají sloužit ke shrnutí dosavadních výsledků a odhalování případných problémů. Rozhodnutím Scrum Mastera zrušit Stand-up meetingy členové týmu přichází o možnost svůj závazek denně revidovat před celým týmem. Během těchto schůzek si mohou členové týmů vyměňovat informace o práci na projektu a sdílet získané znalosti. Šochová a Kunce (2014) upozorňují, že sdílení znalostí není cílem těchto meetingů, ale pouze prostředkem k odpovědi, zda budou dokončeny všechny položky Sprint

Backlogu včas a v požadované kvalitě, případně je identifikováno, co je pro to potřeba udělat. Vliv mají také na soudržnost a spolupráci v týmu.

Snižování významu retrospektivy

Retrospektiva představuje jednu z nejvýznamnějších agilních praktik a slouží k zapojení členů týmu do rozhodování o samotném procesu vývoje. Kritickým hodnocením lze odhalit, s čím jsou členové týmu nespokojení, co se jim naopak líbí a případně co je nutné zlepšit.

Čas, vyhrazený pro zlepšení interních procesů, bývá ovšem často využit pro dokončení méně prioritních úkolů, jako je například dokumentace či odstranění méně závažných chyb, které jsou odbavovány se zpožděním. Tým se tak ochuzuje o jedinečnou možnost nastavit vývojový proces tak, aby co nejlépe vyhovoval povaze jejich práce a práci jim usnadnil.

5.1.6 Možné příčiny nedostatků v konkrétním týmu a jejich řešení

Nedostatky a pochybení při aplikaci metodiky Scrum v jednom z týmů společnosti Ders uvedené v předchozí kapitole, byly analyzovány a níže jsou uvedeny možná řešení nastalých situací:

- **Nerespektování produkční kapacity týmu**
  Aby nedocházelo k překračování produkční kapacity týmu, Product Owner by měl být dobře obeznámen s aktuálním využitím produkční kapacity týmu a respektovat ji. Jelikož tým zpracovává požadavky několika různých klientů, je důležité správně rozdělit produkční kapacitu týmu mezi jednotlivé klienty a pečlivě plánovat obsah cyklů. Product Owner by měl být také schopný nahlížet na produkt a požadavky klientů abstraktněji a volit vhodné kompromisy k tomu, aby bylo možné vytvářet konsolidované moduly, které lze nasadit u více klientů.

  V případě, že nasazení produktu vyžaduje složitou integraci, například napojení na další systémy, měl by být čas nezbytný pro nasazení produktu do produkce počítán jako normální produkční doba a měl by tedy být zahrnut do produkční

kapacity týmu. V ostatních případech by měl být čas potřebný pro nasazení produktu do produkce zkrácený pomocí zjednodušení a automatizace procesu nasazení, ke kterému se i společnost snaží dlouhodobě směřovat.

Při procesu stanovování produkční kapacity týmu na nadcházející období je potřeba počítat s výskytem možných chyb a tedy i časem na jejich odstranění, plánováním dovolených, onemocněním členů týmu a podobně. Tato rizika a ohrožení lze částečně eliminovat vytvořením kapacitní rezervy.

- **Vznik technického dluhu**

  Odstraněním problémů s produkční kapacitou může dojít k vyřešení otázky vzniku technického dluhu. Tým tak může získat dostatek času na vyřešení veškerých závazků, které má v rámci sprintu dokončit. Po vyřešení nedokončených úkolů, které jsou po termínu plánovaného odbavení, může nastat v týmu ideální situace, kdy jsou v rámci cyklu odbaveny veškeré požadavky, a vůči produktu nevznikl žádný technický dluh.

  Problémy se vznikem technického dluhu může vyřešit i stanovení Minimum Viable Product (MVP), neboli minimálního životaschopného produktu. Ve fázi plánování obsahu sprintů by měly být vybrány pouze takové User Stories, které umožní vytvořit produkt s nejmenší možnou funkcionalitou a na které je potřeba vynaložit minimum času a sil (produkční kapacity), ale zároveň sloužil významné skupině uživatelů. Tým by se poté měl zavázat a být s to dokončit včas veškeré úlohy z příslušného Sprint Backlogu.

- **Průběh plánovací schůzky (Planning Meeting)**

  Jedním z důvodů, proč neprobíhají plánovací schůzky za účasti celého týmu, může být obsah práce jednotlivých členů týmu, kteří mohou pracovat na požadavcích pro různé klienty, čím účast celého týmu na plánovací schůzce pozbývá významu.

  Přestože jsou členové týmů alokováni na různých projektech, lze principy Scrumu aplikovat na úrovni menších subtýmů. V průběhu jedné plánovací schůzky tak lze stanovit obsah Sprint Backlogu pro daný subtým, místo

jeho operativního vytváření v průběhu celého cyklu přes různé kombinace členů týmu.

- **Tým není samoorganizující se (self-organized)**
  Získání statutu samoorganizujícího týmu by mohlo být dosaženo připojením zkušenějších pracovníků z jiných týmů společnosti k problémovému týmu. Další možností je přeřazení některých členů problémového týmu do již zaběhlých týmu pro získání potřebných zkušeností.
- **Absence denních Stand-up meetingů**
  Příčinou zrušení Stand-up meetingů je, že celý tým nepracuje na jednom modulu, ale po menších skupinkách (většinou dvojicích) na různých modulech pro různé klienty. Aktuální stav prezentovaný jedním členem týmu tak nemusí být relevantní pro ostatní členy týmu. Dalším důvodem zrušení Stand-up meetingů byla i jejich časová náročnost.

  Nicméně podobně jako u plánovacích schůzek, lze provádět Stand-up meetingy po menších skupinkách, kde jsou sdílené informace relevantní všem účastníkům meetingu. Potřeba zrušit Stand-up meetingy a jejich časová náročnost může pramenit i z nepochopení účelu samotného meetingu. Na Stand-up meetingu by měly být podle Šochové a Kunce (2014) řešeny pouze otázky „co jsem dokončil, co dokončím“ a identifikované problémy, nikoliv reporting toho, „co jsem dělal, co budu dělat a jaké mám problémy“. Stand-up meeting by neměly být kontrolou práce členů týmu Scrum Masterem, ale pouze triviálním meetingem týmu o tom, jak tým stíhá odbavovat svou práci a zda stihne dokončit to, k čemu se zavázal.
- **Snižování významu retrospektivy**
  Nedostatečné věnování pozornosti významu retrospektivy je opět dáno časovou tísní, ve které se tým nachází. Přesto by si měl tým najít čas pro přehodnocení nastavených procesů, jejichž zdokonalení může přinést vyšší efektivitu práce a spokojenější tým.

Nedostatky a rozpory s metodikou Scrum v tomto konkrétním týmu mohou plynout i z následujících příčin:

- **Nedostatky plynoucí z povahy projektu**

  Klienty společnosti Ders z velké části tvoří organizace z veřejné sféry, pro které platí zvláštní pravidla a nařízení. O finanční prostředky na software tak bývá často žádáno skrze výzvy v rámci dotačních programů různých státních i mezinárodních orgánů. Na základě těchto výzev musí být již předem stanovena cena produktu, jeho obsah, termíny předání a podobně. Stanoveny jsou zpravidla také sankce za nedodržení těchto termínů. Samotný proces vývoje je proto těmito nařízeními značně svázán, což vede k rozporům s agilními metodikami.

  Funkcionality, které klient požaduje a které se firma zavázala dodat, musí finální produkt obsahovat všechny. Je tedy nutné naplánovat celý obsah Product Backlogu do cyklů, které musí být dokončeny v předem stanovených termínech. Tímto se proces vývoje dostává do rozporu s metodikou Scrum, kdy je pouze část Product Backlogu detailně analyzována na jasně definované User Stories a odbavena v aktuálním cyklu. Zbytek Product Backlogu zůstává v „surové formě“, jelikož se předpokládá změna priorit a požadavků zákazníka.

  Kvůli tomu, že produkt musí být dodán včas, je obsah cyklů, potažmo i sprintů fixní. Přesunutí nedokončené úlohy do dalšího cyklu vede ke zpoždění dodání produktu a nesplnění smluvních podmínek. Splnění těchto podmínek vyžaduje relativně přesnou a detailní analýzu na začátku projektu pro získání odhadů časové i cenové náročnosti.

- **Důsledky plynoucí z přirozené evoluce procesu vývoje**

  V minulosti byli klienti zvyklí na zakázkovou výrobu. Takto vyvíjený produkt byl vytvářen přímo na serveru u klienta, čímž byl klient zvyklý na rychlost dodání. Dnes jsou pro zvýšení efektivity moduly konsolidovány tak, aby mohl být jeden modul nasazen u více klientů. To sebou přináší náročnější vývoj, což vede ke zdražování produktů, které zákonitě vede k nespokojenosti klientů.

  Konsolidace, kromě náročnějšího vývoje, přináší i další problémy. Jelikož má každý klient jistá specifika, je nutné u takovéhoto konsolidovaného modulu nastavovat různé parametry u každého klienta zvlášť. Nasazení nových

modulů také ztěžují různé verze frameworků (aplikačních rámeců) u klientů. Důsledkem toho s každým nasazením nového modulu vzniká řada chyb. Na začátku projektu je tak těžké vytvořit odhady na takovéto moduly, jelikož nelze přesně předvídat komplikace při zavádění těchto modulů u jednotlivých klientů. Proto je tak snaha o sjednocení verzí a přechod na novější technologii, což je někdy těžké obhájit u klientů, kteří těmto požadavkům ze strany vývojářů nerozumí.

5.2 Případová studie

Případová studie se zabývá zadáním nového softwarového projektu společností GfK, která je dlouholetým klientem společnosti Ders. Požadavkem bylo dodání nové funkcionality do stávajícího podnikového systému. Nový modul měl mít podobu projektového „úkolovníku“ pro správu a sledování plnění jednotlivých úkolů na různých projektech.

Cílem této kapitoly je z analytického pohledu identifikovat a popsat podnikové procesy a navrhnout odpovídající řešení k podpoře těchto procesů tak, aby byly uspokojeny požadavky zákazníka. Toho bylo dosaženo využitím UX principů při sběru a analýze požadavků, které byly následně modelovány a aplikace navržena pomocí modelovacího jazyka UML.

5.2.1 Charakteristika zákazníka

GfK je mezinárodní společností, které se zaměřuje na výzkum trhu. GfK v České republice působí od roku 1991 a řadí se mezi nejvýznamnější poskytovatele kompletních služeb v oblasti výzkumu trhu, marketingových analýz a konzultačních služeb v oblastech průmyslu, obchodu, médií a poskytování služeb. Společnost z velkého objemu sesbíraných dat o spotřebitelích, obchodu, výrobě, službách či médiích vyvozuje za použití inovativních technologií a analýz užitečné informace. Díky těmto smysluplným informacím se klientům dostává lepší informovanosti při rozhodování, čímž zvyšují svou konkurenční výhodu (GFK, 2016).

5.2.2 Vymezení stávajícího systému

Společnost GfK při své činnosti shromažďuje v pravidelných intervalech velká množství dat o prodejích, která jsou následně odesílána a zpracovávána v centrále společnosti v Německu. Pro zachycení těchto činností byla společností Ders dodána aplikace Evidence, jejímž smyslem je především podpora řízení vztahů jak s dodavateli dat, tak i odběrateli výsledných analýz (Ders, 2016a). Tato aplikace je dále vyvíjena a provozována.

Základním účelem aplikace Evidence je podpora hlavního hodnototvorného procesu společnosti GfK Czech s.r.o., tedy podpora při sběru dat o prodejích od různých dodavatelů dat. Aplikace Evidence slouží k podpoře klíčových procesů probíhajících ve společnosti GfK, kterými jsou (Ders, 2016a):

- Sjednávání a uzavírání smluv s dodavateli dat
  Cílem je získání nového obchodního partnera, který se zaváže posílat za určitých podmínek a v předem stanovené periodicitě data o prodaném zboží ve svých prodejnách.
- Notifikace dodavatelů dat
  Účelem je rozesílání a evidence zpráv dodavatelům dat, od nichž jsou očekávána data.
- Evidence při sběru dat
  Smyslem je získat přehled o stavu dodávky dat od jednotlivých zdrojů.
- Přehled o nákladovosti při získávání dat
  Sleduje vyúčtování auditorům a dodavatelům dat.
- Plánování výnosů
  Zabývá se plánováním odhadovaných výnosů z prodeje reportů a statistik o prodejích
- Výpočet extrapolačních faktorů pro statistiky
  Na základě reálných vzorků dat vypočítává extrapolační faktor a sleduje stupeň pokrytí v jednotlivých oblastech výzkumu.

5.2.2.1 Moduly aplikace Evidence

Osoby

V aplikaci Evidence se vyskytují dva druhy osob. Jedná se buď o interní osoby GfK, které vystupují jako uživatelé aplikace, nebo osoby, které jsou spojeny s firmami, dodavatelé dat. K zaznamenávání těchto skutečností slouží modul Osoby.

Firmy

Základ aplikace tvoří modul Firmy, na jehož objekty se vážou veškeré další činnosti. Evidovány jsou zde všechny společnosti, se kterými GfK spolupracuje. Především se jedná o dodavatele dat, ale i zákazníky GfK a IT firmy spravující software na prodejnách dodavatelů dat.

Evidovány jsou dvě základní entity Firma a Prodejny, které jsou neoddělitelnou součástí firem. Kromě obchodních atributů (název, adresa, IČO a podobně) jsou zaznamenávány atributy potřebné pro business společnosti (oblast podnikání, velikost prodejní plochy, obrat a podobně).

V této části systému se také nachází agenda Komunikace, která slouží k zaznamenávání komunikace mezi pracovníkem GfK a dodavatelem dat. Lze tak sledovat historii kontaktů s konkrétní prodejnou, respektive konkrétní osobou pro zlepšení a zvýšení efektivity při dalších jednáních.

Dále je k dispozici agenda Census, která umožňuje export a import dat z Evidence a slouží k aktualizaci stávajících dodavatelů dat.

Evidence dat

Modul Evidence dat slouží pro zaznamenávání informací o stavu dodávky konkrétní sady dat (například data Přišla, Nepřišla) pro jednotlivé zdroje za určitou periodu.

Sady dat jsou definovány v agendě Zdroje, která předepisuje seskupení jednotlivých prodejen do množiny dat. Data se zpracovávají za jednotlivé zdroje a dále jsou z nich vytvářeny konkrétní statistiky.

Perioda značí časový úsek s různou frekvencí (týdenní, měsíční a podobně), za který jsou data zpracovávána. Evidovány jsou v číselníku Periody.

Avizace

Účelem modulu Avizace je rozesílání a evidence zpráv dodavatelům dat, při dodávkách dat. Podporován je editor pro tvorbu šablon avizačních emailů a také lze předepisovat pravidla pro rozesílání avizací.

Vyúčtování

Modul Vyúčtování slouží k fakturaci dodavatelů dat a exportu podkladů pro účetní procesy. Nutné je přitom rozlišovat dva druhy agend pro vyúčtování s dodavateli dat. Prvním způsobem je přímé vyúčtování s dodavateli dat, kteří posílají data přímo do GfK. Druhým způsobem je vyúčtování přes prostředníky takzvané Auditory, kteří získávají data z prodejen a následně je předávají do GfK. Zde je nutné navíc u každého auditora evidovat prodejny, které spravuje, odměnu, kterou jim za data vyplatil, náklady na sběr dat a podobně.

Podporováno je i reportování předpokládaných nákladů, přehledů a podkladů pro účetnictví.

Forecasty

Modul Forecasty zajišťuje podporu při sestavování předpovědí odhadovaných výnosů za konkrétní klienty v určitých sektorech trhu. Forecast tedy umožňuje odhad výnosů z prodeje přidané hodnoty koncovým zákazníkům.

Extrapolace

Tento modul slouží pro výpočet extrapolačního faktoru a sledování stupně pokrytí vzorkem prodejen. Modul extrapolace tak dává představu o tom, jaké množství dat je potřebné k tomu, aby měly analýzy vypovídací hodnotu za celý segment trhu.

5.2.3 Zadání projektu

Společnost GfK, jako dlouholetý klient, pro kterého je vyvíjen zakázkový software, vyslovila požadavek na rozšíření podnikového systému. Nová funkce by měla sloužit k zaznamenání výstupu z porad vedení, jejichž obsahem je příprava na rozšíření služeb o nový segment trhu. Výstupem porady je seznam aktivit, které je nutné splnit v rámci přípravy na nový projekt. Kromě porad vedení společnosti, na kterých vznikají úkoly pro jednotlivá oddělení, by měla aplikace podporovat i další typy porad s projekty spojené. Jedná se například o porady se členy týmů, na kterých jsou úkoly delegovány na další pracovníky, případně vznikají úkoly zcela nové.

5.2.4 Strategie

Smyslem nového modulu Projekty je především:

- Pohodlně přiřadit úkoly z porad odpovědným osobám
- Podpořit odbavování úkolů
- Usnadnit vykonávanou práci pomocí chytrých funkcí
- Účelně třídit úkoly podle zadaných parametrů
- Sledovat a kontrolovat aktuální stav plnění úkolů

5.2.5 Analýza požadavků

Klient své požadavky, respektive prvotní vizi realizace nové funkcionality, komunikoval prostřednictvím agilního boardu pro správu požadavků. Zde byl popsán současný proces a nastíněny hrubé požadavky na podporu těchto činností. Komentáři, upřesňujícími dotazy a odpověďmi na ně postupně vznikala podrobnější podoba zadání požadavků pro realizaci nového modulu. Po dokončení této před-analýzy následovala analytická schůzka. Zde byly podle zásad User Experience zodpovězeny zbývající dotazy a bylo ověřeno, zda je představa vývojářů v souladu s představou zákazníka.

Analytická schůzka probíhala formou semistruktorovaného rozhovoru s kontaktní osobou společnosti GfK, Pavlem Rosenbaumem. Rozhovor byl veden v souladu se zásadami User Experience a za využití základních praktik pro UX rozhovor, které jsou best practices využívané společností Ders:

- 4x proč
  Podle této zásady se nesmíme spokojit s vágní odpovědí na otázku, ale ptát se, dokud nebude otázka řádně zodpovězena.
- Otevřené dotazy
  Klást otevřené dotazy tak, aby se klient rozpovídal a odhalil pro nás skryté souvislosti.

- Mluvit co nejméně, hlavně poslouchat
  Nechat klienta mluvit co nejvíce, rozhovor pouze usměrňovat, případně se na odpovědi doptat nebo klienta doplnit.
- Bumerang
  Pokud je odpověď klienta na otázku nedostatečná nebo vágní, zopakováním poslední obdržené informace od klienta, jej lze znovu podnítit k tomu, aby odpověď dále rozvedl.

Výsledkem analytické schůzky je níže uvedený zápis, ve kterém lze identifikovat nejenom stávající proces, požadavky na funkcionalitu a technické parametry, ale i osoby, takzvané persony, které budou software reálně využívat. Identifikací person lze lépe pochopit chování každého jednotlivého uživatele softwaru, což je klíčové pro dosažení co nejvyššího uživatelského prožitku.

5.2.5.1 Persony

Cílem první části rozhovoru bylo identifikovat persony a úlohy, které typicky vykonávají. Jednou z person, které budou aplikaci reálně využívat, je právě Pavel Rosenbaum. V GfK zastává pozici vedoucího pracovníka jednoho z oddělení společnosti. Jeho pracovní rolí je i mimo jiné spolupráce se společností Ders. Zajišťuje sběr a správu nových požadavků na software, které dále předává kontaktní osobě v Dersu. GfK zastupuje i při jednáních o ceně, termínech realizace a podobně. Zároveň je první osobou, která aplikaci testuje, než je předána dalším uživatelům.

Profil této persony je uveden v tabulce č. 6.

Persona: Vedoucí produkce	
Fotografie	
Jméno	Pavel Rosenbaum
Pracovní pozice	Vedoucí oddělení produkce
Osobní údaje	• Muž • Vzdělání VŠE Praha
Cíle a úkoly, pracovní náplň	• Vedoucí oddělení Produkce zabývající se sběrem dat od dodavatelů dat, jejich evidováním v podnikovém systému Evidence a odesláním do centrály v Německu • Předávání informací podřízeným o nových úkolech z porad vedení, delegace pracovních úkolů, monitoring, kontrola a pomoc zaměstnancům při plnění zadaných úkolů • Sběr požadavků na nový software, komunikace s Ders, testování nových funkcionalit
Spolupráce, odpovědnost	• Zdeněk Bárta Přímý nadřízený Pavla Rosenbauma, mimo jiné schvaluje rozpočet na softwarová řešení • Pavel Buzek Vedoucí oddělení Terén odpovídající za získávání nových dodavatelů dat a vyúčtování dodavatelům, je jedním ze zdrojů požadavků na software

Persona: Vedoucí produkce	
	• Dále zodpovídá za cca 20 zaměstnanců
Pracovní prostředí	Práci vykonává v prostředí své kanceláře, účastní se porad vedení, schůzek s teamleadry, meetingů s týmem
Osobní motto	Každou aplikaci dokáže průměrně zkušený programátor naprogramovat v autě cestou domů

Tabulka č. 6 – Profil persony Vedoucí produkce (vlastní zpracování, 2016)

Výše uvedené persona zastupuje jednu z hlavních uživatelských skupin, a to vedoucí pracovníky oddělení. Od ostatních uživatelů se odlišují především tím, že zadávají, popřípadě přímo delegují podřízeným pracovníkům úkoly pro dosažení stanovených cílů určitého projektu.

V tabulce č. 7 je tentokrát uvedena fiktivní persona, která zastupuje Ředitele společnosti. Tato persona zakládá nové projekty a reviduje plnění zadaných cílů.

Persona: Ředitel společnosti	
Fotografie	
Jméno	Tomáš Boss
Pracovní pozice	Výkonný ředitel
Osobní údaje	• Muž • Vzdělání ČVUT Praha

Persona: Ředitel společnosti	
Cíle a úkoly, pracovní náplň	• Výkonný ředitel společnosti. Udává směr, na co se bude společnost v nadcházejícím období zaměřovat • Moderuje porady vedení, představuje vize a cíle projektů, sleduje a reviduje plnění zadaných cílů • Odpovídá za úspěšnost projektu
Spolupráce, odpovědnost	Vedoucí oddělení
Pracovní prostředí	Práci vykonává především v prostředí své kanceláře, případně zasedacích místností
Osobní motto	Dnes udělám to, co jiní neudělají, takže zítra dokážu to, co jiní nemůžou

Tabulka č. 7 – Profil persony Ředitel společnosti (vlastní zpracování, 2016)

Persony usnadňují práci při modelování požadavků, jelikož odhalují vzorové uživatele systému a procesy, které bude systém podporovat. Realistickým znázorněním klíčových uživatelů lze vyjádřit a zobecnit jejich potřeby a nastínit představu, jak budou aplikaci pravděpodobně používat.

5.2.5.2 Zápis z analytické schůzky

Druhá část rozhovoru se již konkrétně zabývala budoucím modulem Projekty. Smyslem modulu je podpořit proces přípravy na rozšíření služeb o nový segment trhu. Tento proces se interně nazývá „příprava nového projektu“ a je plánován na poradách vedení společnosti.

- **Současný průběh porady vedení**
  Obsahem porad vedení společnosti je stanovit cíle pro rozšíření služeb o nové tržní segmenty. Jsou vymezeny aktivity, které je nutné vykonat v jednotlivých odděleních, pro dosažení těchto cílů. Úkoly vzniklé na těchto poradách jsou revidovány, kontrolovány a případně podle potřeby doplňovány.

Současný stav procesu je nastaven tak, že záznam z porad, kterých se účastní pouze vedoucí pracovníci jednotlivých oddělení, ředitel a zapisovatelka, je zapisován do textového dokumentu Microsoft Word. Sekretářka následně zápis přepíše do tabulkového editoru Microsoft Excel v podobě jednotlivých úkolů a nahraje na sdílené uložiště. Následně pomocí emailu rozešle notifikaci o nahrání nové verze úkolů. Každý zaměstnanec si poté vyhledá a třídí své úkoly mezi ostatními. Sdílený dokument s úkoly má podobu tabulky, ve které jsou evidovány po sloupcích datum meetingu, typ meetingu, název úkolu, komentář k úkolu, odpovědný pracovník za úkol, původní a konečný termín a dále jsou uvedeny různé typy statusů plnění úkolu, viz obrázek č. 30.

date of the meeting	kind of the meeting	TASK	my comments	RESPONSIBLE	ORIGINAL DEADLINE	status in %	DP status in %	HL status in %	RETAIL status in %	Production status in %	NEW DEADLINE	current status	finished date (sent to Zdeněk)
31.10.2012	TL´s whole day				23.11.2012	0%	0%	100%	0%	0%			
16.07.2012	Výjezd Kyjov				31.7.2012	100%	100%	100%	100%	100%			
17.07.2012	Výjezd Kyjov				31.7.2012	0%	0%	0%	0%	0%			
17.07.2012	Výjezd Kyjov				20.7.2012	0%	0%	0%	0%	0%			
17.07.2012	Výjezd Kyjov				31.7.2012	0%	0%	0%	0%	0%			
17.07.2012	Výjezd Kyjov				31.7.2012	100%	0%	0%	0%	0%			
09.05.2012	Personal				31.5.2012	100%	0%	0%	0%	0%		done	
03.05.2012	Marketing				30.05.2012	100%	0%	0%	0%	0%			
03.05.2012	Marketing				14.06.2012	0%	0%	0%	0%	0%			
19.04.2012	TL´s whole day				30.4.2012	100%	0%	0%	0%	0%		sent	May 3
19.04.2012	TL´s whole day				asap	30%	0%	0%	0%	0%			
19.04.2012	TL´s whole day				15.05.2012	0%	0%	0%	0%	0%			
16.04.2012	TL´s				19.4.2012	0%	100%	0%	0%	0%			
07.02.2012	Marketing				30.04.2012	100%	0%	100%	0%	0%		client not	#######
28.11.2011	TL´s				asap	100%	0%	0%	0%	0%		send an e	again to e
14.11.2011	Marketing				25.11.2011	100%	0%	0%	0%	0%		done	
31.10.2011	Marketing				15.11.2011	100%	0%	0%	0%	0%			
31.10.2011	Marketing				15.11.2011	100%	0%	0%	0%	0%			
31.10.2011	Marketing				07.11.2011	100%	0%	0%	0%	0%		not interested	
19.09.2011	Marketing				21.09.2011	100%	0%	0%	0%	0%		done last year	

Obrázek č. 30 – Současná podoba evidence projektových úkolů (vlastní zpracování, 2016)

Takto nastavený proces s sebou nese řadu obtíží. Dokument s úkoly je nahraný na firemní síti, a proto k němu uživatelé mohou přistupovat pouze z této sítě. V souboru se také zobrazuje řada starých úkolů, nebo úkoly, které nejsou pro daného uživatele relevantní. Úkoly je také nutné filtrovat, či složitě vyhledávat, pokud je u úkolu uvedeno více řešitelů. Při použití filtrů se také uloží poslední nastavení filtru. Uživatelé tak musí pokaždé filtry nastavovat znovu podle svých potřeb. Záznamy vedené v buňkách zhoršují přehlednost, jelikož se nemusejí do buněk vejít nebo zasahují do buňky vedlejší. Složité je také evidování úkolů, které byly delegovány nebo přiřazeny podřízeným

pracovníkům vedoucích oddělení. Takové úkoly jsou evidovány v jiných souborech a tím se celá evidence úkolů stává nepřehlednou.

- **Využití agendy**

  Agenda Projekty má podporovat evidenci úkolů zadaných nejen na poradách vedení společnosti, ale i úkoly s daným projektem spojené, které vedoucí oddělení zadávají svým podřízeným nebo které na ně delegují. Smyslem je, aby všichni pracovníci, kteří se podílejí na projektu, měli přehled o svých úkolech.

- **Co je potřeba evidovat**

  V nové agendě budou evidovány:

  - Projekty
  - Úkoly

  Konkrétním případem projektu může být například rozšíření o nový segment trhu, kde jsou sledována data o prodejích chytrých hodinek. Plánování a stanovení aktivit, které je potřeba udělat v rámci přípravy nového projektu probíhá na poradách vedení společnosti.

  Vedoucí oddělení dostávají úkoly pro dosažení stanoveného cíle, za které zodpovídají. Jejich plnění mohou rozdělit na dílčí úkoly a delegovat některé z nich svým podřízeným. Úkolem oddělení obchodu tak může být získání kontaktů na nové dodavatele dat, oddělení produkce má naopak za úkol vyřešení otázky množství potřebných dat, aby měly analýzy vypovídající hodnotu.

- **Jaké atributy se budou zaznamenávat**

  U projektů se bude evidovat:

  - Název
  - Popis projektu
  - Vedoucí
  - Zástupce (vedoucího)
  - Vytvořeno (datum vytvoření projektu)
  - Aktivní (status projektu)
  - Uživatelé

Povinnými poli budou Vedoucí a Název. Vedoucího projektu bude možné vybrat ze seznamu osob, stejně jako Zástupce. Vedoucí může být právě jeden a Zástupce může být u projektu uveden maximálně jeden. Atribut Vytvořeno bude automaticky přidán při založení projektu. Atribut Aktivní bude typu boolean, který vyjadřuje logický datový typ (reprezentovaný hodnotami pravda, nepravda). Při vytváření projektu se také bude zobrazovat pole Uživatelé, kde bude možné navolit N uživatelů, kteří budou mít přístup k projektu. Pokud nebude žádný vybrán, viditelnost projektu nebude omezena.

U úkolů se budou zaznamenávat:

- Zadavatel
- Shrnutí
- Řešitelé
- Původní termín
- Náhradní termín
- Zadáno
- Vyřešeno
- Odesláno
- Stav v %
- Dokončeno
- Popis (Úkol)
- Komentář stavu
- Komentář řešitele
- Vytvořil
- Upraveno
- Upravil

Atributy, které musí být povinně vyplněny, jsou Shrnutí, Původní termín a Stav v %, který je defaultně (ve výchozím nastavení) nastaven na 0. Zadavatel může být i jiná osoba, než ta, která úkol vytváří. To nastává například v situacích, kdy úkol zadává sekretářka (Vytvořil) za ředitele společnosti (Zadavatel). Pole Vytvořil bude vyplněno automaticky podle přihlášeného uživatele, který úkol

zakládá, podobně jako pole Upravil. Atribut Dokončeno bude typu boolean. Pole Upraveno doplní systém podle aktuálního data.

- **Omezení rozsahu zadání systému**

  Důležité je vyspecifikovat i omezení rozsahu systému, aby výsledný software skutečně odrážel potřeby klienta, místo zavedení povrchně atraktivních funkcí, které klient nepotřebuje. Nutné je mít na paměti i rozpočet projektu.

  Podle klienta není třeba sledovat vazbu mezi původními mateřskými úkoly a úkoly, které byly delegovány na další osoby nebo rozděleny na dílčí úkoly.

  Klient dále nevyžaduje notifikace, například formou emailu o nově přiřazených úkolech. Každý pracovník se o přiděleném úkolu dozví osobně a sám si hlídá plnění úkolů.

- **Uživatelské rozhraní aplikace**

  Nový modul pro správu projektových úkolů bude zasazen jako nová agenda do stávajícího menu aplikace. Rozdělena bude na dvě části, kde první bude sloužit pro správu projektů, a ve druhé budou evidovány úkoly. Klient vyslovil požadavek na jednoduché a rychlé filtrování úkolů. Zobrazovat se budou pouze úkoly, ke kterým má daný uživatel právo přístupu. Pomocí přednastavených filtrů může uživatel filtrovat pouze úkoly, které zadal nebo které aktuálně řeší. Bude moci skrývat vyřešené úkoly nebo úkoly, které jsou součástí již neaktivních projektů. Samozřejmostí bude i filtrování po sloupcích podle zadaných parametrů.

- **Smart funkce**

  Aplikace bude také obsahovat chytré funkce, které „zkomfortní“ práci uživatelům a zvýší jejich uživatelský prožitek. Jedna z nich bude zajišťovat zadávání úkolů v řadě. Poté co uživatel vytvoří první úkol, systém zobrazí nové dialogové okno s předvyplněnými poli Zadavatel, Projekt, Řešitelé a Původní termín. Uživatel tak může zadat několik různých úkolů jednomu či více řešitelům v rámci určitého projektu rychle za sebou. Funkce bude také dostupná u již vytvořených úkolů.

Druhá funkce bude usnadňovat práci uživateli při delegování úkolu. Uživatel vybere úkol a zvolí funkci Delegovat. Zobrazí se dialogové okno, ve kterém navolí osoby, kterým úkol deleguje, případně přidá komentář nebo změní datum zadání úkolu. Původní úkol se zkopíruje a vytvoří N nových úkolů. Následně se zobrazí detail původního úkolu, ve kterém může uživatel vyplnit poznámku, že úkol delegoval, případně změní další údaje a uloží. Uživatel se tak vyhne zdlouhavému ručnímu vyplňování a zadávání úkolů.

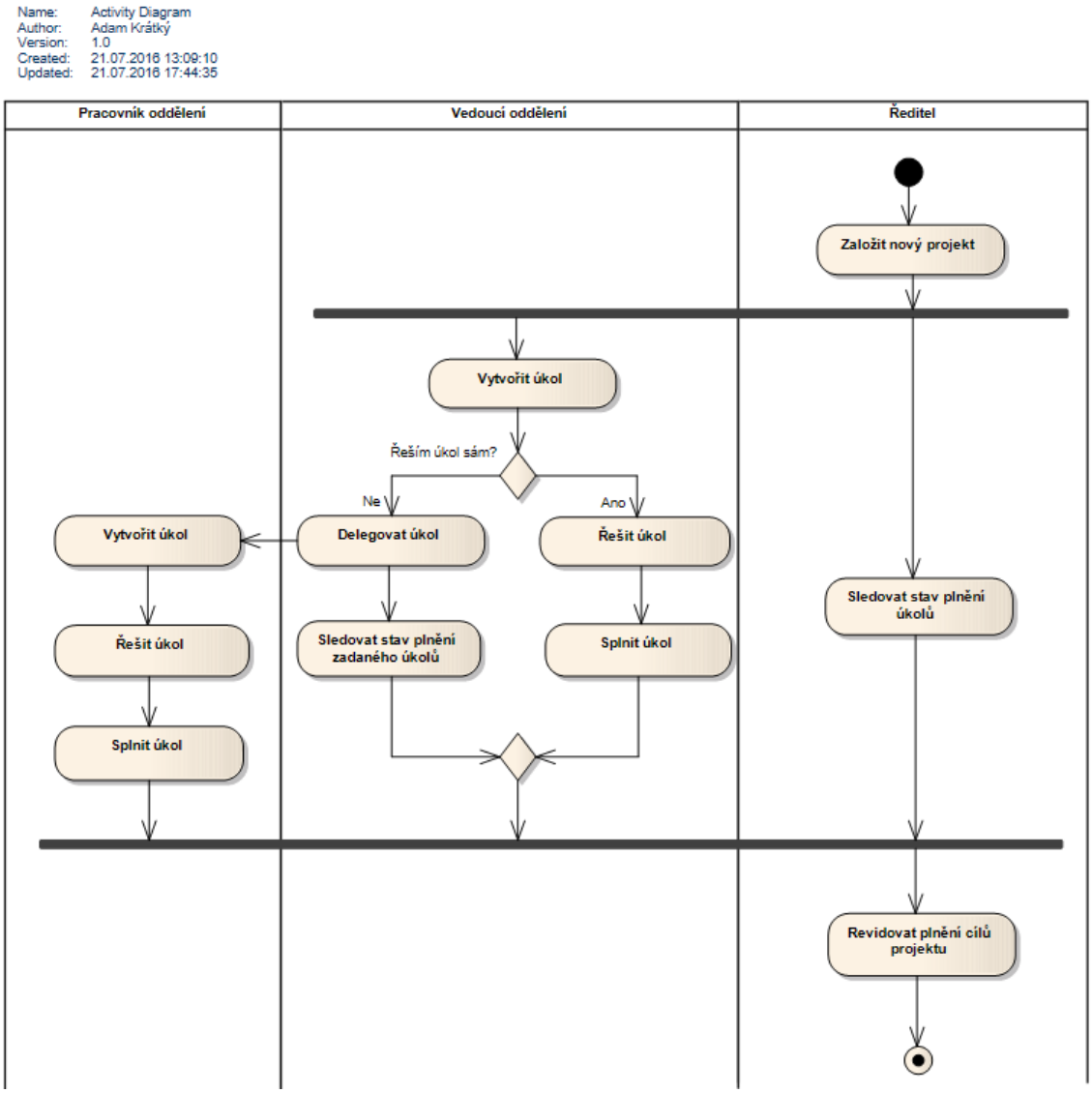
5.2.6 Modelování a tvorba designu

Na základě výše popsané analýzy požadavků je v následujících kapitolách pomocí modelovacího CASE nástroje Enterprise Architect navrženo řešení budoucí aplikace. Enterprise Architect je komerční nástroj, a proto byla využita starší verze programu 7.1 (aktuální verze je 12.1 ke dni 26.7.2016) se školní licencí k použití softwaru, která ovšem poskytuje veškeré potřebné nástroje k modelování s podporou UML specifikace ve verzi 2. Enterprise Architect byl vyvinut australskou společností Sparx Systems, která se zaměřuje na poskytování komplexních nástrojů založených na standardu UML a souvisejících specifikacích vydávaných společnosti OMG.

S přihlédnutím k tomu, že navrhovaná aplikace představuje relativně jednoduchý systém pro evidenci úkolů, byly k zachycení a návrhu systému vybrány níže uvedené modely, jež obsahují veškeré relevantní informace. Nejprve je podnikový proces, který je předmětem řešení, zachycen pomocí diagramu aktivit. Z uživatelského pohledu jsou následně zachyceny základní funkcionality systému pomocí diagramu případu užití. Pohled na strukturu systému vykresluje analytický a návrhový model tříd. Závěr této kapitoly je věnován návrhu uživatelského rozhraní.

5.2.6.1 Modelování podnikatelských procesů

Před zahájením modelování požadavků na systém je pro lepší přehlednost zachycen hlavní tok procesu popsaný v analýze požadavků pomocí diagramu aktivit, viz obrázek č. 31.



Obrázek č. 31 – Podnikatelské procesy zachycené pomocí diagramu aktivit (vlastní zpracování, 2006)

Modelovaný pracovní postup byl vhodně rozdělen do zón, kdy každá reprezentuje určitou roli vyskytující se v obchodní doméně. Rozvětvením je naznačeno souběžné vykonávání aktivit, kdy po založení nového projektu ředitelem vznikne nový úkol pro vedoucí oddělení. Po vytvoření úkolu následuje logická podmínka podmiňující další přechod. Pokud se jedná o úkol, jehož vykonání náleží přímo vedoucímu oddělení, úkol řeší a následně splní sám. Jinak úkol deleguje na své podřízené. Pracovníkovi oddělení tak vznikne úkol, který musí řešit a splnit. Odpovědnost za dokončení delegovaného úkolu stále nese vedoucí oddělení, který po delegování úkolu sleduje jeho plnění.

V průběhu vykonávání úkolů ředitel sleduje stav plnění úkolů na projektech. Po vykonání zadaných úkolů dochází k synchronizaci činností a je revidováno plnění stanovených cílů projektů.

Diagram nezachycuje veškeré aktivity, které účastníci daných rolí vykonávají a ani nevypovídá nic o tom, kdy přesně musí dané aktivity vznikat. Jeho účelem je pouze zobrazit jeden konkrétní tok aktivit a poskytnout základní představu o průběhu podnikatelského procesu, který je předmětem budoucího softwarového řešení.

5.2.6.2 Diagram případů užití

Modelování případů užití je součástí inženýrství požadavků a jednou z prvotních aktivit při návrhu systému. Na základě předchozí analýzy požadavků je na následujícím diagramu zachyceno chování systému z pohledu uživatelů. Jednotlivé případy užití charakterizují typové použití systému určitým uživatelem, respektive představují veškeré funkce systému. Dále jsou identifikováni všichni aktéři, kteří interagují se systémem, vztahy mezi aktéry a případy užití. Naznačeny jsou také hranice systému a vytvořeny scénáře případů užití.

Níže jsou uvedeni aktéři komunikující se systémem a činnosti, které vykonávají.

- Zadavatel úkolu
  Pokud uživatel vystupuje v systému v roli Zadavatele úkolu, může vytvářet nové úkoly a má dostupnou funkci pro zadávání úkolů v řadě. Úkoly, které má dostupné, může zobrazovat, editovat, mazat a filtrovat.
- Řešitel úkolu
  Jestliže je uživatel uveden u úkolu i jako Řešitel úkolu, má kromě výše vypsaných funkcí pro Zadavatele úkolu navíc k dispozici funkci pro delegování úkolu. Dědičnost funkcionalit je na obrázku č. 32 naznačena vztahem generalizace mezi těmito aktéry.
- Zadavatel projektu
  Zadavatel projektu může v systému vykonávat stejné činnosti jako Zadavatel úkolu. Dědičnost je opět naznačena vztahem generalizace mezi těmito aktéry.

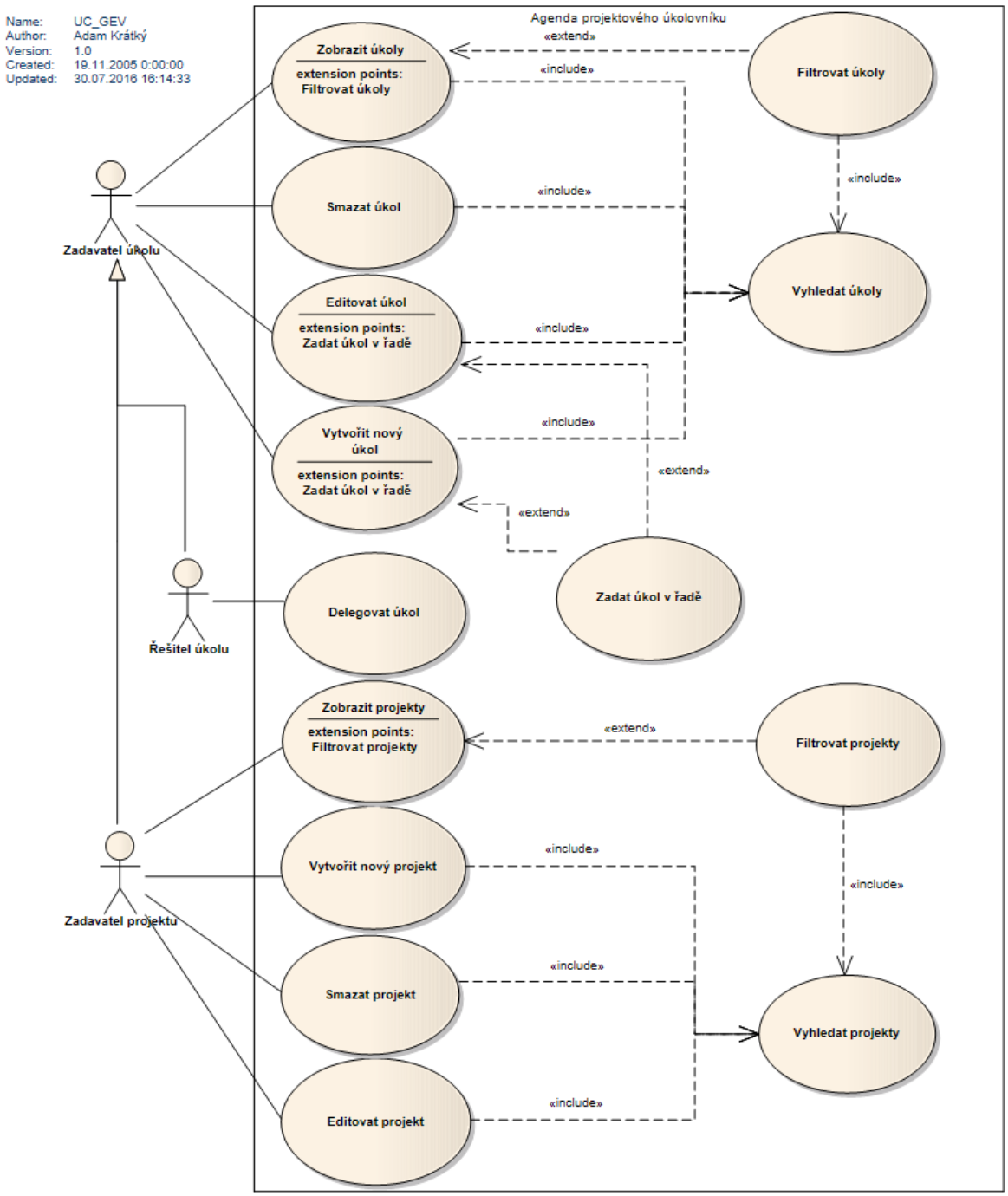
Navíc může jako jediný vytvářet nové projekty, zobrazovat přehled projektů, projekty editovat, mazat a filtrovat.

Pomocí person identifikovaných v rámci analýzy požadavků lze mimo jiné definovat i business role (podnikové role) a role (aktéry), ve kterých budou vystupovat v systému jako uživatelé. Tento vztah je zachycen v tabulce č. 8.

Business role	Role v systému
Ředitel	Vystupuje jako Zadavatel projektu a je Zadavatelem úkolů vedoucím oddělení
Vedoucí oddělení	Vystupuje v roli Řešitele úkolu, které má od Ředitele, je Zadavatelem úkolu pro podřízené a může být i Zadavatelem projektu
Pracovník oddělení	V systému vystupuje pouze jako Řešitel úkolu

Tabulka č. 8 – Business role a jejich role v systému (vlastní zpracování, 2016)

Na obrázku č. 32 je zachycen diagram případů užití. Přestože se jedná o poměrně jednoduchý systém pro evidenci a správu úkolů, ve kterém jsou prováděny především CRUD úlohy (Create - vytvoření, Read - čtení, Update - editace, Delete - smazání), jsou i tyto úlohy pro přehlednost zobrazeny pomocí use case diagramu. Již v této fázi modelování je patrné, že v systému budou evidovány dva hlavní objekty Ukol a Projekt. Proto lze předpokládat, že výsledný systém bude rozdělen do dvou agend. První bude sloužit pro správu projektů. Druhá agenda bude přístupná všem pracovníkům účastnících se projektů a budou v ní evidovány úkoly.



Obrázek č. 32 – Diagram případů užití systému pro správu úkolů Projekty (vlastní zpracování, 2016)

Vybrané případy užití zobrazené na výše uvedeném diagramu jsou blíže specifikovány v níže uvedených tabulkách č. 9 – 14. Do obsahové části práce byla vybrána textová specifikace případu užití Vytvořit úkol, společně se zahrnutým případem užití Vyhledat úkoly a rozšiřujícím případem užitím Zadat úkol v řadě. Specifikace zbylých případů užití jsou uvedeny v příloze č. 1. U každého případu užití jsou identifikováni:

- Aktéři, kteří vykonávají danou činnosti (případ užití)
- Vstupní podmínky definují stav systému před spuštěním případu užití a musí být splněny
- Hlavní scénář popisuje interakci aktéra se systémem, respektive tok událostí při vykonávání případu užití
- Výstupní podmínky definují stav systému po ukončení případu užití a tento stav musí nastat
- Alternativní scénář představuje zástupný tok událostí, případně chybové scénáře

Vytvořit nový úkol	
ID	UC01
Stručný popis	Uživatel zakládá nový úkol
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v agendě Úkoly a požadavky
Hlavní scénář	1) Uživatel iniciuje vytvoření nového úkolu 2) Systém zobrazí dialogové okno s detailem úkolu 3) Systém automaticky vyplní pole Zadavatel a Zadáno 4) Uživatel vyplní alespoň povinné údaje 4.1) Pokud uživatel vyplňuje pole Projekt: 4.2) Systém načte projekty a zobrazí dialogové okno se seznamem projektů 4.3) Uživatel vybere projekt 4.4) Systém zavře dialogové okno a vyplní pole projekt Extension Point: Zadat úkol v řadě 5) Uživatel iniciuje uložení dat 6) Systém uloží data a zavře dialogové okno 7) Include Vyhledat úkoly

Vytvořit nový úkol	
Výstupní podmínky	Systém vytvořil nový úkol
Alternativní scénář	Změnit projekt / zadavatele Storno

Tabulka č. 9 – Specifikace případu užití Vytvořit nový úkol (vlastní zpracování, 2016)

Případ užití Vytvořit nový úkol má ve svém scénáři připravený extension point, díky kterému je možné spustit rozšiřující případ užití Zadat úkol v řadě.

UC: Zadat úkol v řadě	
ID	UC02
Stručný popis	Uživatel zadává několik úkolů za sebou
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v dialogovém okně detailu úkolu
Hlavní scénář	1) Uživatel iniciuje zadání úkolu v řadě 2) Systém uloží data a zobrazí nové dialogové okno detailu úkolu 3) Systém automaticky vyplní příslušná pole 4) Uživatel vyplní alespoň povinné údaje 5) Pokud uživatel zadává další úkol: 5.1) Uživatel opakuje krok 1 6) Jinak: 6.1) Scénář pokračuje následujícím krokem hlavního scénáře bázového případu užití
Výstupní podmínky	Systém vytvořil N nových úkolů
Alternativní scénář	Změnit projekt / zadavatele Storno

Tabulka č. 10 – Specifikace případu užití Zadat úkol v řadě (vlastní zpracování, 2016)

Případ užití Vyhledat úkoly je zahrnut v řadě dalších scénářů případů užití (viz obrázek č. 32) a je jejich neoddělitelnou součástí.

UC: Vyhledat úkoly	
ID	UC14
Stručný popis	Systém vyhledá a vypíše záznamy
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Žádné
Hlavní scénář	1) Systém podle aktuálně nastavených filtrů vyhledá úkoly 2) Systém načte úkoly 3) Systém zobrazí přehled s úkoly
Výstupní podmínky	Systém zobrazil přehled s úkoly
Alternativní scénář	Úkoly nenalezeny

Tabulka č. 11 – Specifikace případu užití Vyhledat úkoly (vlastní zpracování, 2016)

Dále jsou uvedeny alternativní scénáře výše uvedených případů užití.

Alternativní scénář: Úkoly nenalezeny	
Stručný popis	Systém informuje o výsledcích hledání
Vstupní podmínky	Systém vyhledává záznamy
Scénář	1) Alternativní scénář začíná krokem 1 hlavního scénáře 2) Pokud systém nenalezne žádné úkoly 3) Systém vypíše, že nebyly nalezeny žádné úkoly
Výstupní podmínky	Systém informoval o nenalezení žádných úkolů
Alternativní scénář	Žádný

Tabulka č. 12 – Alternativní scénář Úkoly nenalezeny (vlastní zpracování, 2016)

Ke spuštění alternativního scénáře Změnit projekt / zadavatele dojde v případě, pokud uživatel mění již vyplněné pole Projekt nebo Zadavatel.

Alternativní scénář: Změnit projekt / zadavatele	
Stručný popis	Uživatel mění hodnoty polí Projekt / Zadavatel
Vstupní podmínky	Uživatel se nachází v dialogovém okně detailu úkolu
Scénář	1) Alternativní scénář začíná krokem 4 hlavního scénáře 1.1) Pokud uživatel mění hodnotu pole Zadavatel: 1.1.1) Systém zobrazí dialogové okno se seznamem osob 1.1.2) Uživatel vybere osobu 1.1.3.) Systém zavře dialogové okno a vyplní pole Zadavatel 1.2) Pokud uživatel mění hodnotu pole Projekt: 1.2.1) Systém zobrazí dialogové okno se seznamem projektů 1.2.2) Uživatel vybere projekt 1.2.3) Systém zavře dialogové okno a doplní pole Projekt 2) Scénář pokračuje krokem 5 hlavního scénáře
Výstupní podmínky	Systém změnil pole Projekt / Zadavatel
Alternativní scénář	Žádný

Tabulka č. 13 – Alternativní scénář Změnit projekt / zadavatele (vlastní zpracování, 2016)

Alternativní scénář Storno slouží k zavírání dialogových oken bez uložení vyplněných hodnot.

Alternativní scénář: Storno	
Stručný popis	Systém zavře dialogové okno bez uložení
Vstupní podmínky	Uživatel se nachází v dialogovém okně

Alternativní scénář: Storno	
Hlavní scénář	1) Alternativní scénář může začít kdykoli 2) Uživatel iniciuje zavření dialogového okna 3) Systém zavře dialogové okno a zobrazí předchozí stav obrazovky
Výstupní podmínky	Systém zobrazil předchozí stav obrazovky
Alternativní scénář	Žádný

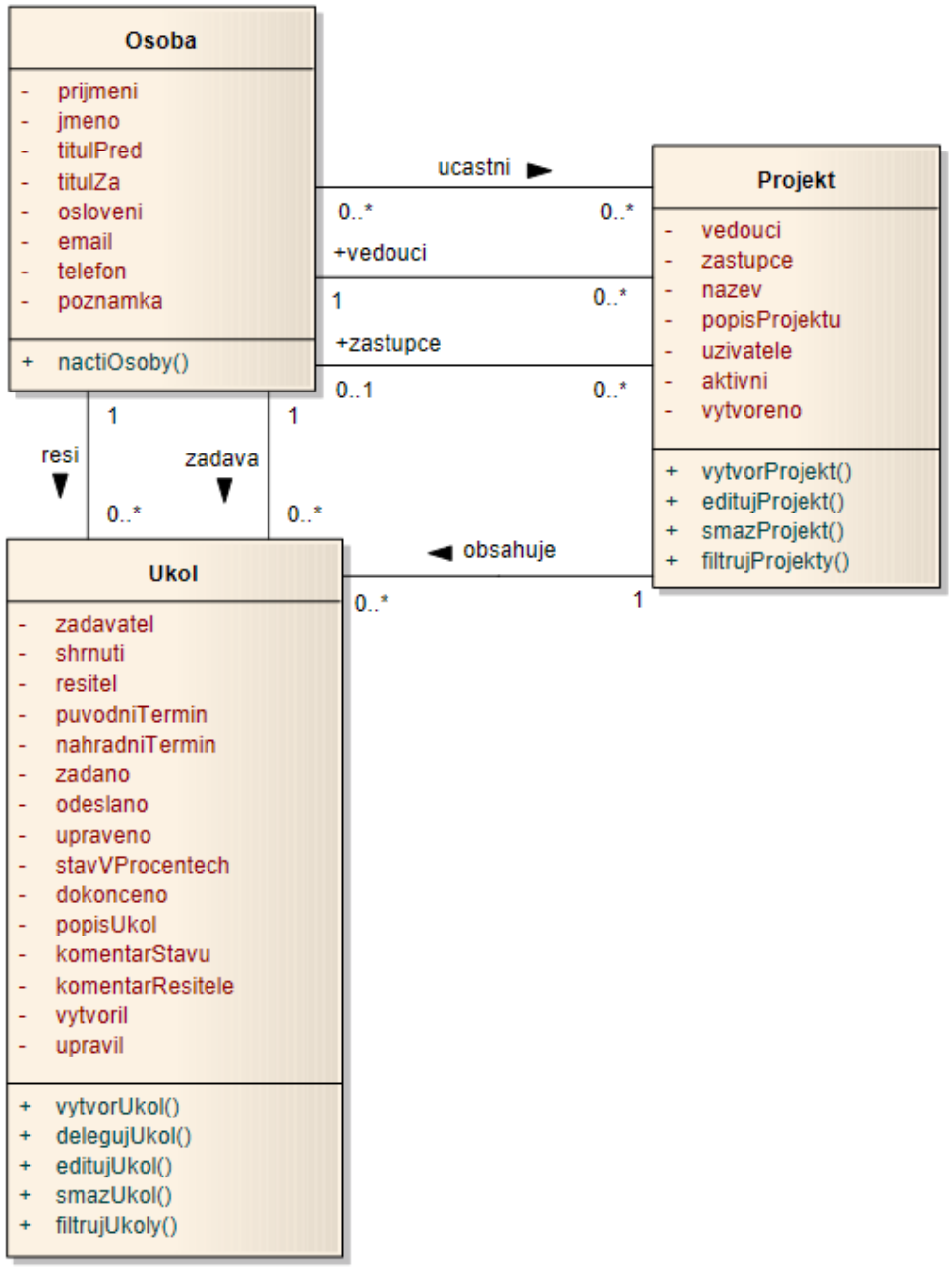
Tabulka č. 14 – Alternativní scénář Storno (vlastní zpracování, 2016)

Modelování případů užití bylo podloženo poznatky z analýzy požadavků zákazníka. Výsledný diagram případů užití dává jasnou představu o rozsahu systému a základních funkcionalitách, které budou uživatelé při své práci používat. Identifikovaní aktéři reprezentují hlavní skupiny uživatelů. Scénáře případů užití popisující interakci uživatele se systémem slouží jako základní východisko při identifikaci objektů systému a jejich budoucích operací.

5.2.6.3 Analytický model tříd

Dalším krokem při analýze systému bylo vytvoření analytického modelu tříd znázorněného na obrázku č. 33. Cílem bylo nalezení hlavních typů objektů a vztahů mezi nimi. Identifikací klíčových slov ze scénářů případů užití byly stanoveny třídy systému a jejich základní operace. Mezi třídami byly naznačeny vazby pomocí asociací a byla stanovena multiplicita. Pro lepší čitelnost diagramu byly asociace pojmenovány a opatřeny šipkou určující směr čtení asociace. Díky detailně zpracované analýze požadavků byly doplněny atributy, které bude aplikace obsahovat.

Name: Analytický model tříd
Author: Adam Krátký
Version: 1.0
Created: 19.07.2016 14:33:04
Updated: 10.08.2016 16:02:13



Obrázek č. 33 – Analytický model tříd systému pro správu úkolů Projekty (vlastní zpracování, 2016)

5.2.6.4 Návrhový model tříd

Účelem návrhového modelu tříd je zachytit strukturu systému na takové úrovni, aby ji již bylo možné implementovat, a to konkrétně pro objektově orientovaný programovací jazyk JAVA. Pro tyto potřeby byl analytický model tříd rozšířen o některé implementační detaily.

Atributy byly doplněny o datové typy a byly zadány některé počáteční hodnoty. Operace charakterizující chování třídy byly dále specifikovány do podoby sady metod a byly vygenerovány přístupové metody takzvané gettery a settery. U třídy Osoba byly generovány pouze gettery, jelikož instance této třídy boudou sloužit pouze pro čtení a získání informací o objektech dané třídy. Třídám byly přidány i konstruktory pro inicializaci objektu a jeho obsahu.

Následně byly zpřesněny vztahy mezi třídami převedením analytických relací do podoby návrhových relací. U asociací byla po rozhodnutí, na kterém konci asociace se nachází celek a na kterém součást, zavedena kompozice nebo agregace a stanovena řiditelnost.

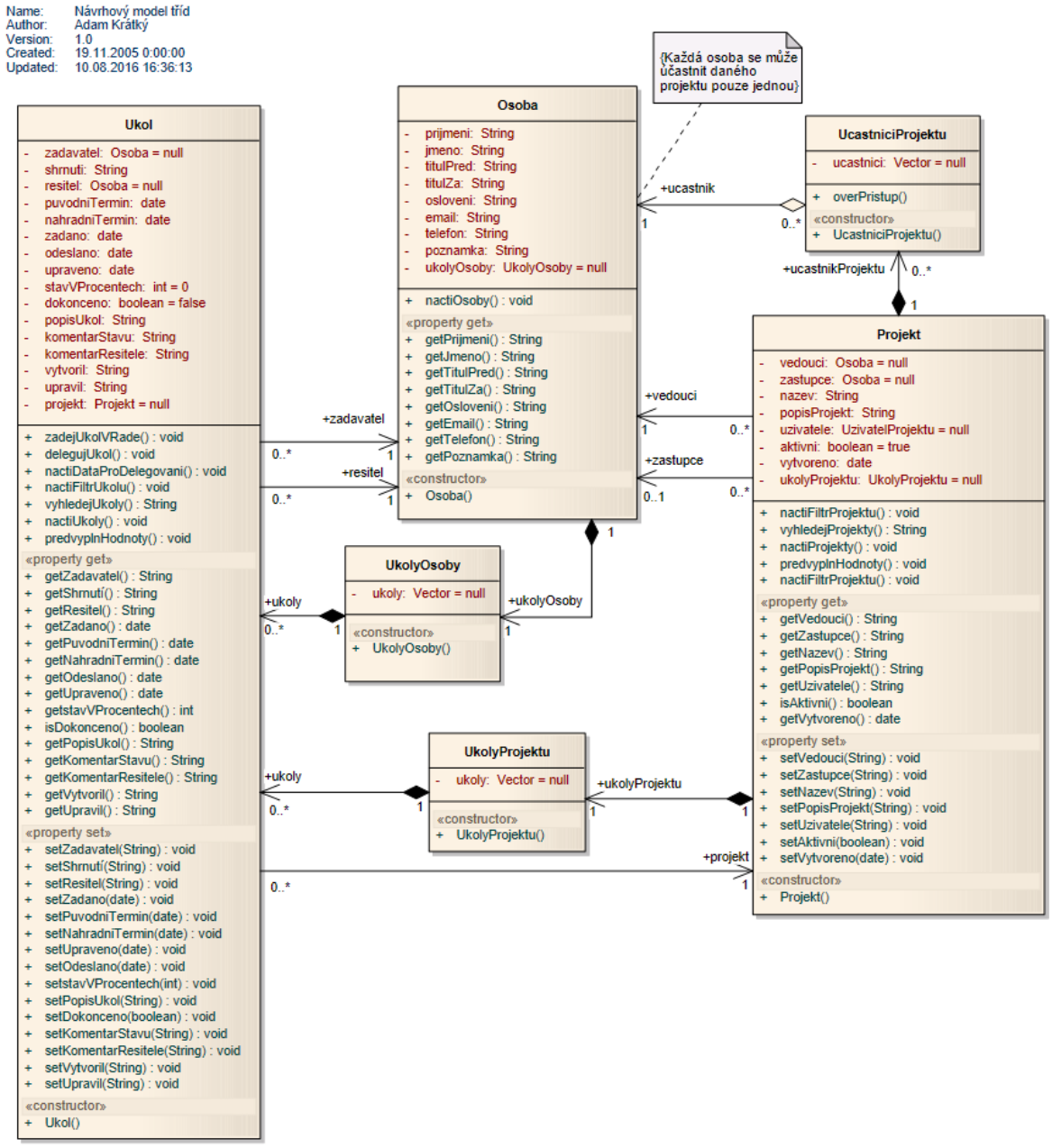
Asociace typu 1:N mezi třídami Projekt a Ukol byla implementována pomocí speciální třídy (UkolyProjektu), takzvané kolekci, jejíž instance slouží ke správě kolekcí jiných objektů. Mezi třídami UkolyProjektu a Projekt byla zavedena kompozice, protože třída UkolyProjektu je pouze součástí implementace celku. Jelikož objekty ze třídy Ukol nemohou samy existovat bez třídy Projekt a smazáním objektů této třídy dochází také k zániku příslušných objektů třídy Ukol, byl mezi třídami UkolyProjektu a Ukol zvolen typ relace kompozice. Podobně bylo postupováno při zpřesnění vazby mezi třídami Osoba a Úkol.

V modelu byly zavedeny i jednostranné asociace vyjadřující zpětnou vazbu mezi třídami, které slouží pro získání informací uložených v referenčních atributech.

Přímo nebylo možné implementovat ani asociaci typu M:N mezi třídami Osoba a Projekt. Proto byla v návrhovém modelu definována třída vymezující tuto asociaci

UcastniciProjektu. Aby původní asociace převedením do návrhového modelu neztratila svůj význam, byla připojena poznámka vyjadřující omezení asociace.

Specifikováním modelu do podoby uvedené na obrázku č. 34, bylo již možné použít tento model jako podklad při implementaci systému, případně využít integrované funkce modelovacího nástroje Enterprise Architect pro automatické generování kódu.



Obrázek č. 34 – Návrhový model tříd systému pro správu úkolů Projekty (vlastní zpracování, 2016)

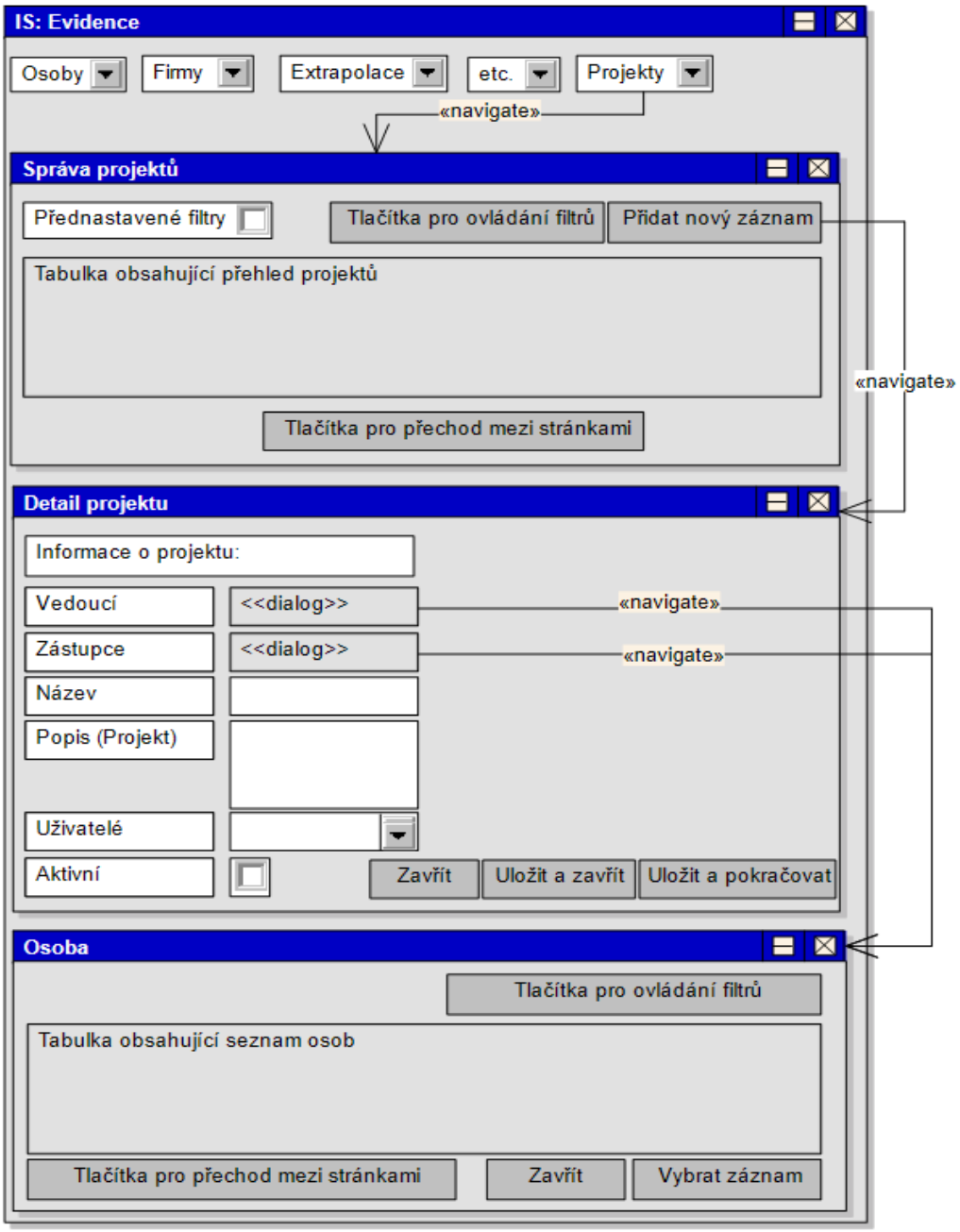
5.2.6.5 Návrh uživatelského rozhraní (User Interface)

Poslední kapitola zabývající se modelováním a návrhem aplikace Projekty, je věnována návrhu uživatelského rozhraní. Zde byly při modelování plně využity agilní principy. Původní návrhy byly fyzicky zaznamenávány na papír a podle zásad User Experience a User Centered Design byl vizuální design aplikace postupně zpřesňován rozhovory s klientem tak, aby bylo dosaženo co největšího uživatelského prožitku. Návrhy tedy primárně sloužily ke komunikaci s klientem, získání zpětné vazby a jejímu rychlému zapracování. Díky těmto jednoduchým návrhům mohl v poměrně krátkém čase vzniknout design uživatelského rozhraní, včetně specifikovaných grafických prvků, který přesně vyhovuje uživatelům budoucí aplikace.

Na níže uvedených obrázcích jsou zobrazeny návrhy uživatelského rozhraní vytvořené v programu Enterprise Architect, které znázorňují především rozložení ovládacích prvků a dialogových oken při práci s programem. Tyto návrhy jsou již zjednodušenou implementací předchozích ručně psaných návrhů, ovšem bez grafických prvků.

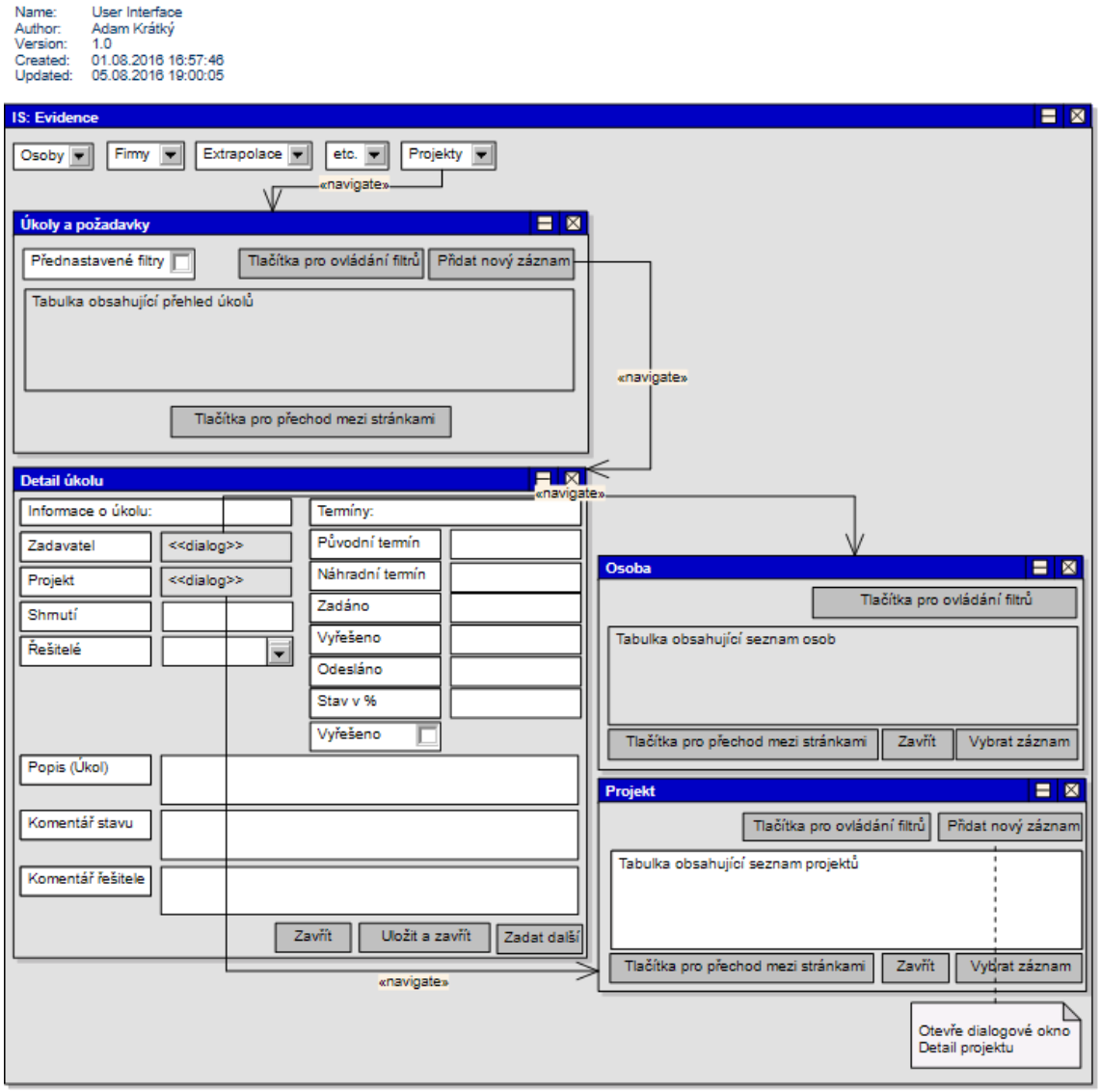
Na obrázku č. 35 je uveden návrh uživatelského rozhraní s rozložením dialogových oken a naznačením jejich návazností pro agendu Správy projektů.

Name: User Interface
Author: Adam Krátký
Version: 1.0
Created: 05.08.2016 18:27:37
Updated: 05.08.2016 19:49:57



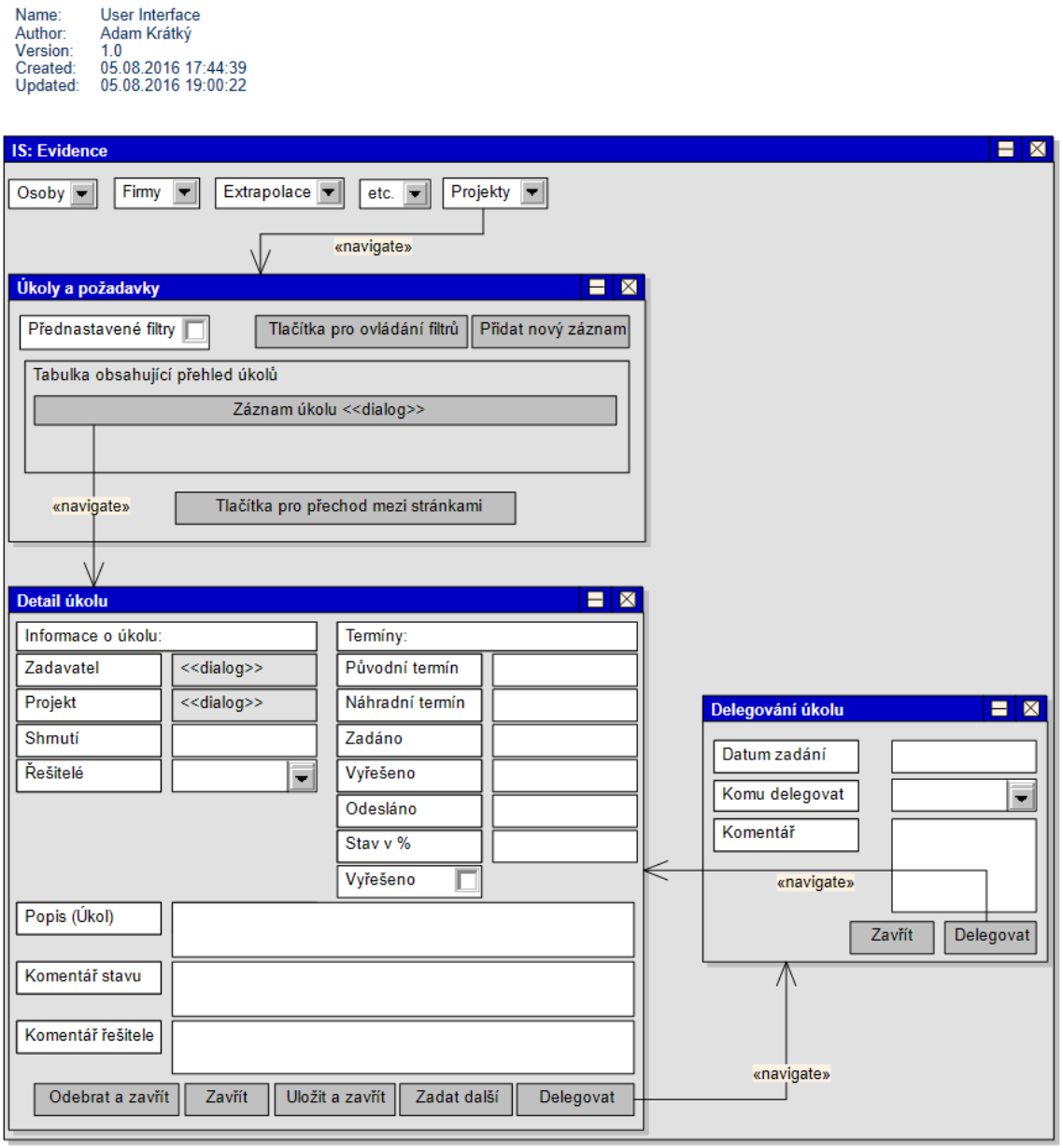
Obrázek č. 35 – Návrh uživatelského rozhraní agendy Správa projektů (vlastní zpracování, 2016)

Obrázek č. 36 ilustruje uživatelské rozhraní a rozložení obrazovek agendy Úkoly a požadavky.



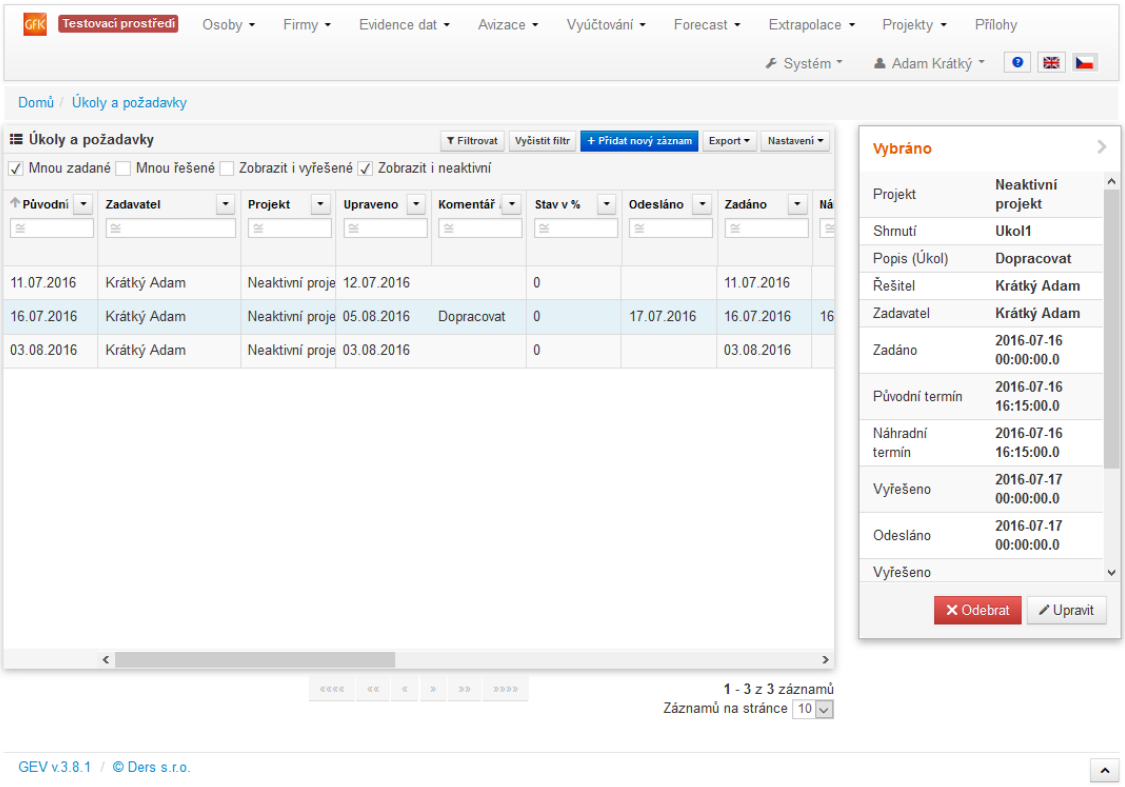
Obrázek č. 36 – Návrh uživatelského rozhraní agendy Úkoly a požadavky (vlastní zpracování, 2016)

Na obrázku č. 37 je znázorněno rozložení obrazovek při delegování již existujícího úkolu.



Obrázek č. 37 – Rozložení obrazovek při delegování úkolu (vlastní zpracování, 2016)

Konečná verze uživatelského rozhraní již s grafickými prvky, konkrétně agendy Úkoly a požadavky, je uvedena na obrázku č. 38.



Obrázek č. 38 – Uživatelské rozhraní agendy Úkoly a požadavky (vlastní zpracování, 2016)

6 Shrnutí výsledků

Při analyzování vývojového procesu softwaru ve společnosti Ders byly jako hlavní metodické přístupy identifikovány agilní metodika Scrum využívaná pro vývoj a rozvoj softwarových produktů a agilní metodika Kanban pro údržbu a servis softwaru. Dalšími agilními přístupy adoptovanými společností Ders jsou především User Experience a s ním související User Centered Design. Použitím těchto agilních metodik a přístupů může společnost Ders rychle a efektivně reagovat na změny požadavků zákazníka, dodávat software v požadované kvalitě, vytvořit vhodné pracovní prostředí pro své zaměstnance a zajistit spokojenost svých klientů.

Rozhovory se zaměstnanci společnosti Ders bylo popsáno reálné nasazení metodiky Scrum při vývoji softwaru. Provedením šetření skutečného stavu průběhu vývojového procesu v jednom z týmů společnosti a jeho porovnáním s teoretickým popisem metodiky Scrum, byly odhaleny odchylky, nedostatky a pochybení proti teorii. Zkoumáním možných příčin těchto selhání byla navržena potenciální řešení problémů pro optimalizaci vývojového procesu. Přijetím navržených opatření a doporučení může dojít ke zlepšení procesu vývoje ve zkoumaném týmu společnosti. Popsané nedostatky se mohou vyskytovat a také se v různé míře vyskytují i v dalších softwarových týmech, pro které může být tento text varováním a upozorněním na záležitosti, kterým je třeba věnovat pozornost ve snaze eliminovat zdroje potřebné k nápravě podobných nedostatků. Zároveň jsou uvedena doporučení při odstraňování překážek ve vývoji softwaru.

V další části práce byly předvedeny konkrétní aktivity vývojového procesu demonstrované na reálném projektu. Po vymezení stávajícího systému a nastínění zadání projektu, byla provedena podle zásad User Experience a User Centered Design analýza požadavků na systém. Jejím hlavním výstupem byl zápis z analytické schůzky, v němž byl popsán současný stav firemního procesu, který je předmětem softwarového řešení. Dále byly definovány představy klienta o tom, k čemu by měl nový modul podnikového systému sloužit a co vše bude třeba evidovat. Zároveň byla stanovena omezení rozsahu zadání systému vzhledem k potřebám klienta i vzhledem k rozpočtu projektu. V tomto zápisu byly upřesněny dříve předeslané požadavky a sloužil

jako jeden z hlavních zdrojů informací při modelování systému. Vedením rozhovorů s klientem při analýze požadavků byly dále identifikovány persony, které zastupují hlavní uživatelské skupiny budoucího systému. Persony byly popsány z důvodu lepšího pochopení chování cílových uživatelů a pro dosažení co nejvyššího uživatelského prožitku.

Při vlastním modelování a návrhu systému byl nejdříve pomocí diagramu aktivit zachycen podnikatelský proces, pro který byl software navrhován. Účelem tohoto diagramu bylo pro lepší představu ilustrovat slovně popsaný pracovní postup v analýze požadavků. V návaznosti na analýzu požadavků bylo díky diagramu případů užití zachyceno chování systému z pohledu uživatele. Takto byly vymezeny všechny funkce systému, pomocí aktérů role v systému a tvorbou scénářů případů užití byla naznačena interakce uživatelů a systému.

Struktura systému byla zachycena nejprve pomocí analytického modelu tříd, jehož účelem bylo odhalit hlavní typy objektů v systému a jejich vztahy. Typové objekty byly zaznamenány pomocí tříd a asociacemi mezi nimi byly naznačeny jejich vztahy. Třídám byly přiřazeny atributy a základní operace. Analytický model tříd byl následně zpřesněn do podoby návrhového modelu tříd. Atributy tříd byly doplněny o datové typy, operace byly zpřesněny do podoby metod a asociační vztahy mezi třídami zpracovány do podoby návrhových relací. Smyslem tvorby návrhového modelu tříd bylo zachytit strukturu systému na takové úrovni, aby ji již bylo možné implementovat programátorem, případně model použít pro automatické generování kódu.

Závěr práce byl věnován návrhu uživatelského rozhraní. Návrhy byly postupně zpřesňovány rozhovory vedenými s klientem tak, aby bylo při používání aplikace dosaženo co největšího uživatelského prožitku podle zásad User Experience.

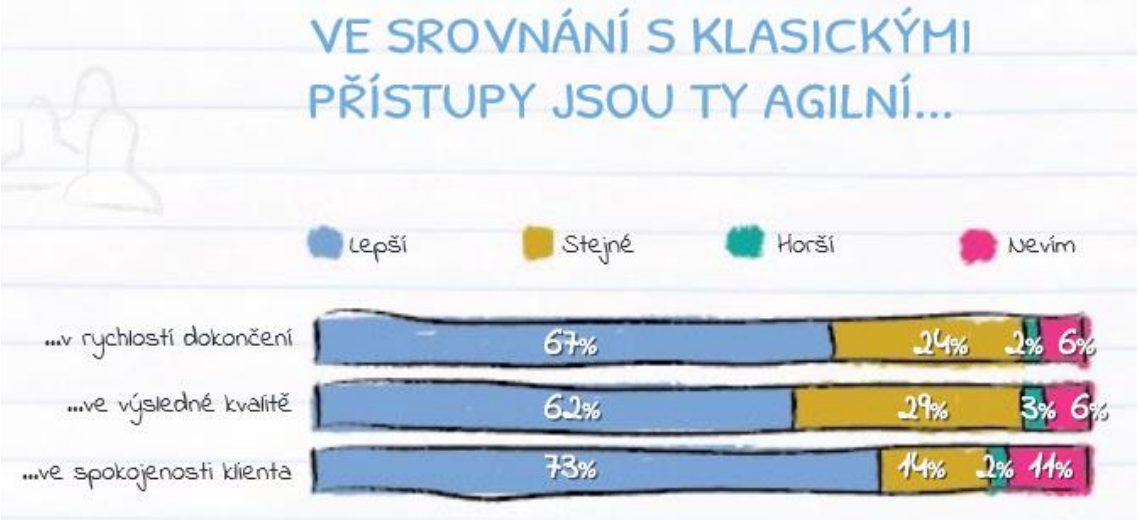
Díky výše uvedeným modelům a použitým metodickým přístupům bylo možné navrhnout software, který byl nejen akceptován a přijat klientem, ale především uspokojil požadavky jeho uživatelů, kteří aplikaci využívají k jejich spokojenosti.

7 Závěry a doporučení

Jak již bylo v úvodu práce předesláno, vývoj informačních systémů a softwaru obecně, je specifickou činností. I v dnešní době ho lze alespoň částečně považovat spíše za umění, než inženýrskou disciplínu. V dnešním prostředí, kdy zákazník požaduje co nejrychlejší dodání alespoň části systému a své požadavky na software často mění, přestávají být tradiční (rigorózní) metodiky vhodné, jelikož se snaží proces vývoje softwaru jasně definovat a vycházejí z prediktivního přístupu. Naopak agilní metodiky jsou vysoce adaptivní a flexibilní. Snižují riziko neúspěchu a zvyšují předvídatelnost rozdělením projektu na menší přírůstky, které lze prezentovat zákazníkovi. Vzhledem k současným trendům adoptovaly i některé tradiční metodiky agilní přístupy. Příkladem je metodika založená na iterativním vývoji Rational Unified Process, která je neustále vyvíjena a díky tomu přejímá řadu best practices, kterými jsou i agilní koncepty jako test-driven development (vývoj řízený testy), agile modeling (agilní modelování), agile change management (agilní řízení změn) a další.

Agilní přístupy přinášejí do procesu vývoje celou řadu výhod, nicméně jak upozorňují Šochová a Kunce (2014), je velice důležité si uvědomit, že po zavedení agilních metod veškeré problémy nezmizí, ba naopak budou ještě více viditelné. Nestačí pouze zavést pár agilních technik a fragmentů Scrumu či Extrémního programování. Aby byl vývojový proces opravdu agilní, je nutné zavést vše, co k dané agilní metodice patří, jelikož i každá nepatrná technika, která je zdánlivě nepotřebná, má své opodstatnění. Propojením jednotlivých artefaktů metodiky se jejich efekt násobí a teprve poté funguje proces jako celek. Pokud chce být společnost agilní, znamená to především změnit přístup k produktu, zákazníkovi a týmu.

Přínosy agilních přístupů nejlépe podtrhuje na obrázku č. 39 jejich srovnání s klasickými přístupy v průzkumu agilního řízení v ČR 2013 provedeného pomocí dotazníkového šetření společností Etnetera a.s. a Agilní Asociace.



Obrázek č. 39 – Srovnání agilních a klasických přístupů (převzato z Etnetera, 2013)

Předložená práce se z hlediska vývojového procesu, kromě aplikace agilních metodik, věnuje analýze a návrhu informačního systému. Při sběru a analýze požadavků se velice osvědčily agilní přístupy User Experience a User Centered Design. Díky nim bylo možné navrhnout software s vysokou přidanou hodnotou pro klienta a dopřát co největší uživatelský prožitek uživatelům systému.

V oblasti analýzy a návrhu softwaru je standardně využíván modelovací jazyk Unified Modeling Language. S jeho pomocí v této práci vznikaly základní modely analyzující problémovou oblast a návrhy zaměřené na implementaci systému. Díky diagramům jazyka UML lze zachytit systém s různou mírou abstrakce a z různých pohledů a tím usnadnit komunikaci jak uvnitř týmu, tak vývojářů a uživatele systému.

8 Seznam použité literatury

1. AGILE ALLIANCE. *Manifest Agilního vývoje software*. 2001a, [online]. Dostupné z http://agilemanifesto.org/iso/cs/, [cit. 2016-01-20]
2. AGILE ALLIANCE. *Principy stojící za Agilním Manifestem*, 2001b, [online]. Dostupné z http://agilemanifesto.org/iso/cs/principles.html, [cit. 2016-01-20]
3. AMBLER, S., W. *Personas: An Agile Introduction.* 2014, [online]. Dostupné z http://www.agilemodeling.com/artifacts/personas.htm, [cit. 2016-07-25]
4. ARLOW, J. a NEUSTADT, I. *UML a unifikovaný proces vývoje aplikací: průvodce analýzou a návrhem objektově orientovaného softwaru.* 1. vyd. Brno: Computer Press, a.s., 2003, 387 s. ISBN 80-7226-947-X
5. ARLOW, J. a NEUSTADT, I. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* 2. vyd. Brno: Computer Press, a.s., 2007, 567 s. ISBN 978-80-251-1503-9
6. BRUCKNER, T. a kolektiv. *Tvorba informačních systémů: principy, metodiky, architektury.* 1. vyd. Praha: Grada Publishing, a.s., 2012, 357 s. ISBN 978-80-247-4153-6
7. BUCHALCEVOVÁ, A. *Metodiky budování informačních systémů.* 1. vyd. Praha: Oeconomica, 2009, 205 s. ISBN 978-80-245-1540-3
8. BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky.* 1. vyd. Praha: Grada Publishing, a.s., 2005, 163 s. ISBN 80-247-1075-7
9. BUCHALCEVOVÁ, A. a PAVLÍČKOVÁ, J. *Základy softwarového inženýrství – objektově orientovaný přístup.* 1. vyd. Praha: Vysoká škola ekonomická v Praze, 2002, 216 s. ISBN 80-245-0268-2
10. BUCHALCEVOVÁ, A., PAVLÍČKOVÁ, J. a PAVLÍČEK L. *Základy softwarového inženýrství: materiály ke cvičení.* 1. vyd. Praha: Oeconomia, 2007, 221 s. ISBN 978-80-245-1270-9
11. BUREŠ, V. *Systémové myšlení pro manažery.* 1. vyd. Praha: Professional Publishing, 2011, 264 s. ISBN 978-80-7431-037-9

12. CASTELLS, M. *Manuel Castells Interview: The Network Society and Organizational Change.* 2001, [online]. Dostupné z http://globetrotter.berkeley.edu/people/Castells/castells-con4.html, [cit. 2016-04-08]
13. DATHAN, R. a MENOM, R. Managing Testing Projects -- the SCRUM way. *Journal of the Quality Assurance Institute.* 2010, roč. 24, č. 2, s. 18 - 19
14. DERS. *Profil firmy.* 2014, [online]. Dostupné z http://old.ders.cz/display/WEBP/Profil+firmy, [cit. 2016-02-22]
15. DERS. *Naše hodnoty.* 2015, [online]. Dostupné z https://wiki.ders.cz/download/attachments/2523138/ders-vize-1.04-final_tisk.pdf?version=1&modificationDate=1393842758000, [cit. 2016-02-22]
16. DERS. *GEV – Uživatelská příručka (GfK Evidence v3.0).* 2016a, [online]. Dostupné z https://wiki.ders.cz/pages/viewpage.action?pageId=174359221, [cit. 2016-02-23]
17. DERS. *Produkty.* 2016b, [online]. Dostupné z http://www.ders.cz/products.html#, [cit. 2016-03-21]
18. DOUCEK, P. *Řízení projektů informačních systémů.* 2. vyd. Praha: Professional Publishing, 2006, 180 s. ISBN 80-86946-17-7
19. EASYUX TEAM. *Don't Stand Between Your Users & Usability! Choose UCD Today.* 2016, [online]. Dostupné z http://easyux.net/2016/03/dont-stand-between-your-users-usability-choose-ucd-today/, [cit. 2016-04-19]
20. ETNETERA. Průzkum Agilního Řízení v ČR 2013, [online]. Dostupné z http://www.etnetera.cz/public/1b/43/e5/52571_103079_agilni_dotaznik_report_2013_5.pdf?download, [cit. 2016-08-09]
21. FATDUX GROUP APS. *Co je UX?* 2013, [online]. Dostupné z http://www.fatdux.com/cs/What/What-is-UX, [cit. 2016-04-06]
22. FULBRIGHT, R. Incorporating Innovation into Iterative Software Development Using the Inventive Problem Solving Methodology. *International Journal of Innovation Science.* 2013, roč. 5, č. 4, s. 203 - 212
23. GFK. *GfK v České republice.* 2016, [online]. Dostupné z http://www.gfk.com/cz/o-nas/gfk-in-your-country/, [cit. 2016-02-23]

24. GUNTAMUKKALA, V., WEN, H. J. a TARN, J. M. An empirical study of selecting software development life cycle models. *Human Systems Management*. 2006, roč. 25, č. 4, s. 265 - 278
25. CHLAPEK, D., ŘEPA, V. a STANOVSKÁ, I. *Analýza a návrh informačních systémů.* 1. vyd. Praha: Oeconomica, 2011, 157 s. ISBN 978-80-245-1782-7
26. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). *IEEE Standard Glossary of Software Engineering Terminology.* New York: Institute of Electrical and Electronics Engineers, 1990, 83 s. ISBN 1-55937-067-X
27. IONEL, N. Critical Analysys of the Scrum Project Management Methodology. *Annals of the University of Oradea, Economic Science Series.* 2008, roč. 17, č. 4, s. 435 - 441
28. ISO/IEC, IEEE*. Systems and software engineering – Vocabulary, International Standard ISO/IEC/IEEE 24765.* 1. vyd. Switzerland: ISO/IEC, 2010, 410 s. ISBN 978-0-7381-6205-8
29. KADLEC, V. *Agilní programování: metodiky efektivního vývoje softwaru.* 1. vyd. Brno: Computer Press a.s., 2004, 278 s. ISBN80-251-0342-0
30. KANISOVÁ, H. a MÜLLER, M. *UML srozumitelně.* 2. vyd. Brno: Computer Press, a.s., 2006, 176 s. ISBN 80-251-1083-4
31. MYŠÁK, P. *Persony: aneb víte, pro koho podnikáte?* 2013, [online]. Dostupné z http://www.tvorba-internetovych-stranek.cz/blog/persony-aneb-vite-pro-koho-podnikate, [cit. 2016-07-25]
32. NEMBERG, M. *8 Effective ways of measuring UX.* 2014, [online]. Dostupné z http://conversionxl.com/8-effective-ways-of-measuring-ux/, [cit. 2016-04-11]
33. OBJECT MANAGEMENT GROUP (OMG). *Unified Modeling Language: Infrastructure.* Version 2.1.1, 2007, [online]. Dostupné z http://doc.omg.org/formal/2007-02-06.pdf, [cit. 2016-03-17]
34. OBJECT MANAGEMENT GROUP (OMG). *OMG Unified Modeling Language™ (OMG UML).* Version 2.5, 2015, [online]. Dostupné z http://www.omg.org/spec/UML/2.5/PDF/, [cit. 2016-03-10]
35. ORDYSIŃSKI, T. Kanban Based Information Management in Organization. *Studia i Materialy Polskiego Stowarzyszenia Zarzadzania Wiedza / Studies &*

Proceedings Polish Association for Knowledge Management. 2013, č. 63, s. 76 – 85

36. PAGE-JONES, M. *Základy objektově orientovaného návrhu v UML.* 1. vyd. Praha: Grada Publishing, s.r.o., 2001, 367 s. ISBN 80-247-0210-X
37. POLÁK, J., MERUNKA, V. a CARDA, A. *Umění systémového návrhu: objektově orientovaná tvorba informačních systémů pomocí původní metody BORM.* 1. vyd. Praha: Grada Publishing, a.s., 2003, 195 s. ISBN 80-247-0424-2
38. POUR, J. *Informační systémy a technologie.* 1. vyd. Praha: Vysoká škola ekonomie a managementu, 2006, 492 s. ISBN 80-86730-03-4
39. PROCHÁZKA, J. a KLIMEŠ, C. *Provozujte IT jinak: agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb.* 1. vyd. Praha: Grada Publishing, a.s., 2011, 288 s. ISBN 978-80-247-4137-6
40. RICHTA, K. a SOCHOR, J. *Softwarové inženýrství I.* 1. vyd. Praha: Vydavatelství ČVUT, 1998, 229 s. ISBN 80-01-01428-2
41. ŘEPA, V. *Analýza a návrh informačních systémů.* 1.vyd. Praha: Ekopress, 1999, 403 s. ISBN 80-86119-13-0
42. SHALYGIN, O. *Working Agile-like with SCRUM and Extreme Programming.* 2015, [online]. Dostupné z http://techiejs.com/Blog/Post/Working-Agile-like-with-SCRUM-and-Extreme-Programming, [cit. 2016-02-25]
43. SCHMULLER, J. *Myslíme v jazyku UML: knihovna programátora.* 1. vyd. Praha: Grada Publishing, s.r.o., 2001, 359 s. ISBN 80-247-0029-8
44. STOICA, M., MIRCEA, M. a GHILIC-MICU, B. Software Development: Agile vs. Traditional. *Infromatica Economica*. 2013, roč. 17, č. 4, s. 64 – 73
45. ŠOCHOVÁ, Z. a KUNCE, E. *Agilní metody řízení projektů.* 1. vyd. Brno: Computer Press, 2014, 175 s. ISBN 978-80-251-4194-6
46. U.S. DEPARTMENT OF HEALTH & HUMAN SERVICES. *User-Centered Design Basics.* 2016, [online]. Dostupné z http://www.usability.gov/what-and-why/user-centered-design.html, [cit. 2016-04-07]
47. UX ASOCIACE. *Začínáte s User Experience?* 2016, [online]. Dostupné z http://www.asociaceux.cz/zacinate-s-user-experience, [cit. 2016-04-06]

48. VÁLKA, O. *Co je UX design.* 2011, [online]. Dostupné z http://valka.info/notes/2011/04/co-je-ux-design/, [cit. 2016-04-06]
49. VOŘÍŠEK, J. a kolektiv. *Principy a modely řízení podnikové informatiky.* 1. vyd. Praha: Oeconomica, 2008, 446 s. ISBN 978-80-245-1440-6
50. WIEGERS, K. E. *Požadavky na software: od zadání k architektuře aplikace.* 1. vyd. Brno: Computer Press, a.s., 2008, 448 s. ISBN 978-80-251-1877-1
51. YILMAZ, M. a O'CONNOR, R.,V. A Scrumban Integrated Gamification Approach to Guide Software Process Improvement: A Turkish Case Study. *Tehnicki vjesnik / Technical Gazette.* 2016, roč. 23, č. 1, s. 237-245

9 Příloha

Specifikace případu užití Zobrazit úkoly (vlastní zpracování, 2016)

UC: Zobrazit úkoly	
ID	UC03
Stručný popis	Uživatel zobrazuje přehled s úkoly
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel je přihlášen v systému
Hlavní scénář	1) Uživatel iniciuje zobrazení přehledu s úkoly 2) Include Vyhledat úkoly Extension point: Filtrovat úkoly
Výstupní podmínky	Systém zobrazil přehled s úkoly
Alternativní scénář	Žádný

Specifikace případu užití Editovat úkol (vlastní zpracování, 2016)

UC: Editovat úkol	
ID	UC04
Stručný popis	Uživatel provádí změny dat úkolu
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v agendě Úkoly a požadavky
Hlavní scénář	1) Uživatel iniciuje editaci úkolu 2) Systém zobrazí dialogové okno s detailem úkolu 3) Pokud uživatel provádí změny hodnot: 3.1) Uživatel provede změny hodnot 4) Jinak: 4.1) Uživatel provádí zadání úkolu v řadě z již existujícího úkolu

UC: Editovat úkol	
	Extension point: Zadat úkol v řadě 4) Systém uloží data a zavře dialogové okno 5) Include Vyhledat úkoly
Výstupní podmínky	Údaje o úkolu byly změněny
Alternativní scénář	Změnit projekt / zadavatele Storno

Specifikace případu užití Smazat úkol (vlastní zpracování, 2016)

UC: Smazat úkol	
ID	UC05
Stručný popis	Uživatel vymaže úkol
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v agendě Úkoly a požadavky
Hlavní scénář	1) Uživatel iniciuje smazání úkolu 2) Systém zobrazí dialogové okno s detailem úkolu 3) Uživatel vybere volbu smazat úkol 4) Systém vymaže data a zavře dialogové okno 5) Include Vyhledat úkoly
Výstupní podmínky	Údaje o úkolu byly smazány
Alternativní scénář	Storno

Specifikace případu užití Vytvořit nový projekt (vlastní zpracování, 2016)

UC: Vytvořit nový projekt	
ID	UC07
Stručný popis	Uživatel zakládá nový projekt

Aktéři	Zadavatel projektu
Vstupní podmínky	Zadavatel projektu
Hlavní scénář	1) Uživatel iniciuje vytvoření nového projektu 2) Systém zobrazí dialogové okno s detailem projektu 3) Uživatel vyplní alespoň povinné údaje 3.1) Pokud uživatel vyplňuje pole Vedoucí nebo pole Zástupce: 3.2) Systém načte osoby a zobrazí dialogové okno se seznamem osob 3.3) Uživatel vybere osobu 3.4) Systém zavře dialogové okno a vyplní pole Vedoucí, resp. Zástupce 4) Uživatel iniciuje uložení dat 5) Systém uloží data a zavře dialogové okno 6) Include Vyhledat projekty
Výstupní podmínky	Systém vytvořil nový projekt
Alternativní scénář	Storno

Specifikace případu užití Zobrazit projekty (vlastní zpracování, 2016)

UC: Zobrazit projekty	
ID	UC08
Stručný popis	Uživatel zobrazuje přehled s projekty
Aktéři	Zadavatel projektu
Vstupní podmínky	Uživatel je přihlášen v systému
Hlavní scénář	1) Uživatel iniciuje zobrazení přehledu s projekty 2) Include Vyhledat projekty Extension point: Filtrovat projekty

UC: Zobrazit projekty	
Výstupní podmínky	Systém zobrazil přehled s projekty
Alternativní scénář	Žádný

Specifikace případu užití Editovat projekt (vlastní zpracování, 2016)

UC: Editovat projekt	
ID	UC09
Stručný popis	Uživatel provádí změny dat projekt
Aktéři	Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v agendě Správa projektů
Hlavní scénář	1) Uživatel iniciuje editaci projektu 2) Systém zobrazí dialogové okno s detailem projektu 3) Uživatel provede změny 3.1) Pokud uživatel iniciuje změnu polí Vedoucí nebo Zástupce: 3.2) Systém načte osoby a zobrazí dialogové okno se seznamem osob 3.3) Uživatel vybere osobu 3.4) Systém zavře dialogové okno a vyplní pole Vedoucí, resp. Zástupce 4) Uživatel potvrdí změny 5) Systém uloží data a zavře dialogové okno 6) Include Vyhledat projekty
Výstupní podmínky	Údaje o projektu byly změněny
Alternativní scénář	Storno

Specifikace případu užití Smazat projekt (vlastní zpracování, 2016)

UC: Smazat projekt	
ID	UC10
Stručný popis	Uživatel vymaže projekt
Aktéři	Zadavatel projektu
Vstupní podmínky	Uživatel se nachází v agendě Správa projektů
Hlavní scénář	1) Uživatel iniciuje smazání projektu 2) Systém zobrazí dialogové okno s detailem projektu 3) Uživatel vybere volbu smazat projekt 4) Systém vymaže data a zavře dialogové okno 5) Include Vyhledat projekty
Výstupní podmínky	Údaje o projektu byly smazány
Alternativní scénář	Storno

Specifikace případu užití Delegovat úkol (vlastní zpracování, 2016)

UC: Delegovat úkol	
ID	UC11
Stručný popis	Uživatel deleguje úkol na jiné pracovníky
Aktéři	Řešitel úkolu
Vstupní podmínky	Uživatel se nachází v agendě Úkoly a požadavky
Hlavní scénář	1) Uživatel iniciuje delegování úkolu 2) Systém zobrazí dialogové okno s detailem úkolu 3) Uživatel může provést změny 4) Uživatel vybere možnost delegovat úkol 5) Systém zobrazí dialogové okno pro delegování úkolu

UC: Delegovat úkol	
	6) Uživatel vybere osoby, kterým úkol deleguje, případně zadá další údaje a potvrdí 7) Systém zkopíruje informace o původním úkolu + nové z předchozího kroku a vytvoří N nových úkolů 8) Systém zavře dialogové okno a zobrazí původní dialogové okno s detailem úkolu 9) Uživatel může vyplnit poznámku o delegování nebo změnit jiné údaje 10) Uživatel iniciuje uložení původního úkolu 11) Systém uloží data a zavře dialogové okno 12) Include Vyhledat záznamy
Výstupní podmínky	Systém vytvořil N nových úkolů Systém přepsal původní úkol
Alternativní scénář	Změnit projekt / zadavatele Storno

Specifikace případu užití Filtrovat úkoly (vlastní zpracování, 2016)

UC: Filtrovat úkoly	
ID	UC12
Stručný popis	Uživatel filtruje přehled úkolů
Aktéři	Zadavatel úkolu, Řešitel úkolu, Zadavatel projektu
Vstupní podmínky	Uživatel se nachází na přehledu úkolů
Hlavní scénář	1) Uživatel iniciuje filtrování úkolů 1.1) Pokud uživatel filtruje podle parametrů: 1.1.1) Uživatel vybere parametr a zadá hodnotu 1.1.2) Uživatel aktivuje filtr

UC: Filtrovat úkoly	
	1.2) Pokud uživatel filtruje podle přednastaveného filtru: 1.2.1) Uživatel aktivuje přednastavený filtr 2) Systém uloží nastavení filtru úkolů 3) Include Vyhledat úkoly
Výstupní podmínky	Systém uložil filtr pro přehled úkolů
Alternativní scénář	Žádný

Specifikace případu užití Filtrovat projekty (vlastní zpracování, 2016)

UC: Filtrovat projekty	
ID	UC13
Stručný popis	Uživatel filtruje přehled projektů
Aktéři	Zadavatel projektu
Vstupní podmínky	Uživatel se nachází na přehledu projektů
Hlavní scénář	1) Uživatel iniciuje filtrování projektů 1.1) Pokud uživatel filtruje podle parametrů: 1.1.1) Uživatel vybere parametr a zadá hodnotu 1.1.2) Uživatel aktivuje filtr 1.2) Pokud uživatel filtruje podle přednastaveného filtru: 1.2.1) Uživatel aktivuje přednastavený filtr 2) Systém uloží nastavení filtru projektů 3) Include Vyhledat projekty
Výstupní podmínky	Systém uložil filtr pro přehled úkolů
Alternativní scénář	Žádný

Specifikace případu užití Vyhledat projekty (vlastní zpracování, 2016)

UC: Vyhledat projekty	
ID	UC15
Stručný popis	Systém vyhledá a vypíše záznamy
Aktéři	Zadavatel projektu
Vstupní podmínky	Žádné
Hlavní scénář	1) Systém podle aktuálně nastavených filtrů vyhledá projekty 2) Systém načte projekty 3) Systém zobrazí přehled s projekty
Výstupní podmínky	Systém zobrazil přehled s projekty
Alternativní scénář	Projekty nenalezeny

Specifikace alternativního scénáře Projekty nenalezeny (vlastní zpracování, 2016)

Alternativní scénář: Projekty nenalezeny	
Stručný popis	Systém informuje o výsledcích hledání
Vstupní podmínky	Systém vyhledává záznamy
Scénář	1) Alternativní scénář začíná krokem 1 hlavního scénáře 2) Pokud systém nenalezne žádné projekty 3) Systém vypíše, že nebyly nalezeny žádné projekty
Výstupní podmínky	Systém informoval o nenalezení žádných projektů
Alternativní scénář	Žádný

Specifikace alternativního scénáře Změnit projekt / zadavatele (vlastní zpracování, 2016)

Alternativní scénář: Změnit projekt / zadavatele	
Stručný popis	Uživatel mění hodnoty polí projekt / zadavatel Pozn.: Začátek a konec alternativního scénáře se může u jednotlivých případů užití měnit v závislosti na toku událostí hlavního scénáře
Vstupní podmínky	Uživatel se nachází v dialogovém okně detailu úkolu
Scénář	1) Alternativní scénář začíná krokem X hlavního scénáře 1.1) Pokud uživatel mění hodnotu pole Zadavatel: 1.1.1) Systém zobrazí dialogové okno se seznamem osob 1.1.2) Uživatel vybere osobu 1.1.3.) Systém zavře dialogové okno a vyplní pole Zadavatel 1.2) Pokud uživatel mění hodnotu pole Projekt: 1.2.1) Systém zobrazí dialogové okno se seznamem projektů 1.2.2) Uživatel vybere projekt 1.2.3) Systém zavře dialogové okno a doplní pole Projekt 2) Scénář pokračuje krokem X hlavního scénáře
Výstupní podmínky	Systém změnil pole Projekt / Zadavatel
Alternativní scénář	Žádný

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.. Informační management (im5-p

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ.	ADRESA	OSOBNÍ ČÍSLO
Krátký Adam	Krňovice 8, Třebechovice pod Orebem Krňovice	I1000944

TÉMA ČESKY·

Analýza a návrh informačního systému s využitím agilních přístupů k vývoji softwaru

TÉMA ANGLICKY·

Analysis and Design of Information System using Agile Approaches to Software Development

VEDOUCÍ PRÁCE:

Ing. Tereza Otčenášková, B.A. KIT

ZÁSADY PRO VYPRACOVÁNÍ.

Cíl práce:
Cílem diplomové práce je představení principů tvorby informačních systémů, hlavních metodických přístupů a modelovacích technik a notací používaných při analýzách a návrzích informačních systémů. V praktické části jsou získané poznatky aplikovány na konkrétní projekt v reálném vývojovém prostředí.

Osnova:
1 Úvod
2.Cíl práce a metodologie zpracování
3 Teoretická východiska
3.1 Vymezení základních pojmů a principů tvorby IS/ICT
3.2.Architektura IS/ICT
3.3.Metodické přístupy k tvorbě informačních systémů
3.4.Modelovací techniky a notace
4.Praktická část
4.1.Popis vývojového prostředí konkrétní firmy
4.2.Charakteristika konkrétního projektu
4.3 Volba vhodné metodiky pro konkrétní projekt
4.4.Výběr vhodné modelovací techniky a její implementace
5.Shrnutí výsledků
6.Závěry a doporučení
7.Seznam použité literatury
8.Přílohy

SEZNAM DOPORUČENÉ LITERATURY·

A. Buchalcevová; Metodiky vývoje a údržby informačních systémů
T Bruckner J Voříšek, A. Buchalcevová, I. Stanovská, D. Chlapek, V Řepa; Tvorba Informačních systémů principy metodiky architektury
A. Buchalcevová, J Pavlíčková; Základy softwarového inženýrství objektově orientovaný přístup
V Řepa; Analýza a návrh informačních systémů
P Doucek; Řízení projektů informačních systémů
J Polák, V Merunka, A. Carda; Umění systémového návrhu
J Procházka, C. Klimeš; Provozujte IT jinak; Agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb
J Arlow· UML 2 a unifikovaný proces vývoje aplikací

Podpis studenta: ..... Datum: 3 8 2016

Podpis vedoucího práce: ..... Datum: 3 8 2016