TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: N 2612 – Elektrotechnika a informatika

Studijní obor: 1802T007– Informační technologie

Metodiky agilního řízení aplikované pro vývoj programového vybavení

Agile management methodology applied to software development

Diplomová práce

Autor: **Bc. Ondřej Kolmistr**

Vedoucí práce: **Ing. Petr Kretschmer**

V Liberci 20.5. 2011

(zde bude vloženo originální zadání)

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/200 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **souhlasím** s případným užitím své diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užití své diplomové práce či poskytnutí licence k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

Datum:

Podpis:

Abstrakt

Obsahem této diplomové práce je popis a vzájemné porovnání metodik pro řízení vývoje programového vybavení. Tyto metodiky slouží k řízení a zefektivnění práce při vývoji softwaru. Budou zde popsány jednak klasické metodiky a také moderní metodiky, které se nazývají agilními metodikami.

Mezi agilními metodikami se bude práce orientovat na metodiky: Feature Driven Development, SCRUM a Test Driven Development. Dále bude provedeno porovnání jednotlivých těchto metodik a jejich vhodnost pro použití při řízení různých projektů.

Součástí práce bude také navržení webové aplikace, která by mohla sloužit pro správu vývoje softwaru pomocí některé z metodik. V závěru budou zhodnoceny získané informace a uvedeny možnosti dalšího předpokládaného vývoje v tomto oboru.

Klíčová slova: metodiky řízení vývoje programového vybavení, agilní metodiky, Feature driven development, Scrum, Test driven development

Abstract

The content of this thesis is a description and comparison of methodologies for managing software development. These methodologies are used to control and streamline the work on software development. There are described classical and modern methodologies called agile methodologies.

Among the agile methodologies will focus on methodics: Feature Driven Development, SCRUM and Test Driven Development. There is also made a comparison of these methodologies and their suitability for using in the management of various projects.

The work will also design a web application that could be used to manage software development using some of methodology. Finally, there is an evaluation of obtained information and provided opportunities for further expected developments in this field.

Keywords: methodics for controlling software development, agile metrodologies, Feature driven development, Scrum, Test driven development

Obsah

Seznam ilustrací

1. Úvod

V dnešní době je využívání počítačů a různých dalších elektronických zařízení naprosto běžné. Tím, co však dělá tyto přístroje univerzálními pomocníky pro nejširší oblasti lidského konání je jejich programové vybavení. Tato skutečnost je zřejmá již od doby vzniku prvních počítačů. S rozšiřováním výpočetní techniky a jejím využíváním lidmi, kteří nemají žádné odborné vzdělání, je role softwaru stále důležitější.

S postupem času a růstem složitosti jednotlivých softwarových projektů, tlakem na čas vývoje a konkurenci mezi softwarovými společnosti je jasné, že metodika vývoje programového vybavení počítačů je důležitým vlivem, který ovlivňuje celý výsledný projekt. Některé tzv. tradiční metodiky se v poslední době začínají ukazovat jako nedostatečné. Nedokáží rychle reagovat na časté změny zadání projektu a upřesňování požadavků zákazníka, je jim vytýkána přílišná pomalost, těžkopádnost a velká byrokracie. Vlivem těch zkušeností s klasickými metodikami začaly vznikat nové metodiky, nazývané nejprve jako *lehké*, později pak *agilní*. Tyto metodiky vznikaly nejprve odděleně na úrovní jednotlivých firem. Postupem času se osvědčily a zároveň vznikaly některé další. V roce 2001 na setkání vývojářů softwaru vznikl Manifest agilního vývoje softwaru (Manifesto for Agile Software Development). V tomto dokumentu byly definovány vlastnosti agilních metodik, mezi které patří hlavně rychlý vývoj v různě dlouhých cyklech, častá a těsná spolupráce se zákazníky, dodávky betaverzí, silný vliv zpětné vazby a celkové omezení byrokracie.

Tyto nové agilní metodiky přinášejí nový pohled a možnosti při řízení vývoje programového vybavení. Jejich cílem je odstranění nedostatků klasických metodik a zaručení dodání softwaru v požadovaný čas, za požadovanou cenu a s funkčností odpovídající aktuálním požadavkům zákazníka. Ne všechny tyto metodiky se však hodí pro určitou společnost či typ projektu, a jejich nasazení je třeba vždy důkladně zvážit.

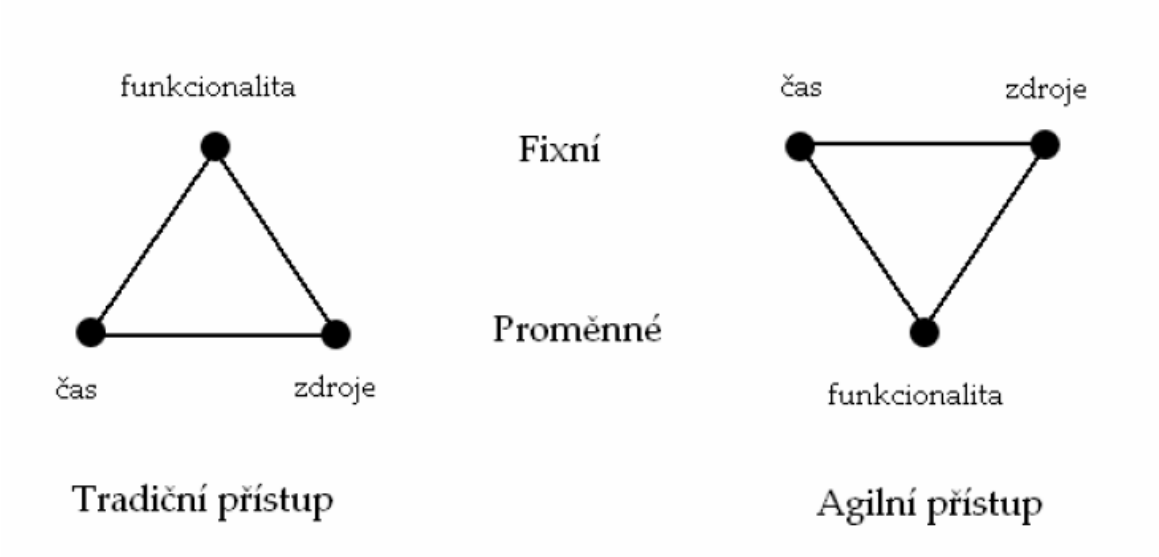
2. Klasické metodiky

2.1 Obecný popis klasických metodik

Pro porovnání s agilními metodikami je nutné popsat alespoň některé z klasických metodik vývoje softwaru. Tyto metodiky se též nazývají rigorózními. Jejich společnou vlastností je, že představují to, čemu se agilní metodiky chtějí vyhnout. Jsou to vlastnosti jako přílišná složitost, velká byrokracie a malá flexibilita, kvůli které je při jejich aplikaci obtížné reagovat na aktuální zpřesňování a úpravy požadavků zákazníka. Vyžadují striktní dokumentaci, různé revize, schvalování a podobně. Tradiční metodiky také považují za fixní element v procesu vývoje výslednou funkcionalitu, a za proměnné je považován čas a zdroje (lidské a materiální). Oproti tomu agilní metodiky považují za fixní čas a zdroje, které je ochoten zákazník do projektu investovat. Výsledná funkcionalita se tedy může postupně během vývoje měnit.

Pro tyto vlastnosti přestaly být klasické metodiky vhodné pro malé projekty a malé vývojové týmy. Ani vývoj těchto metodik je nedokázal uzpůsobit moderním požadavkům a jako reakce na potřebu nových metodik, které odstraní zmíněné nedostatky, vznikly agilní metodiky.

V této kapitole budou popsány některé metodiky reprezentující klasický přístup, aby oproti nim vynikl rozdíl, který představují agilní metodiky.



Obrázek 1 - porovnání tradičního a agilního přístupu

2.2 Vodopádový model

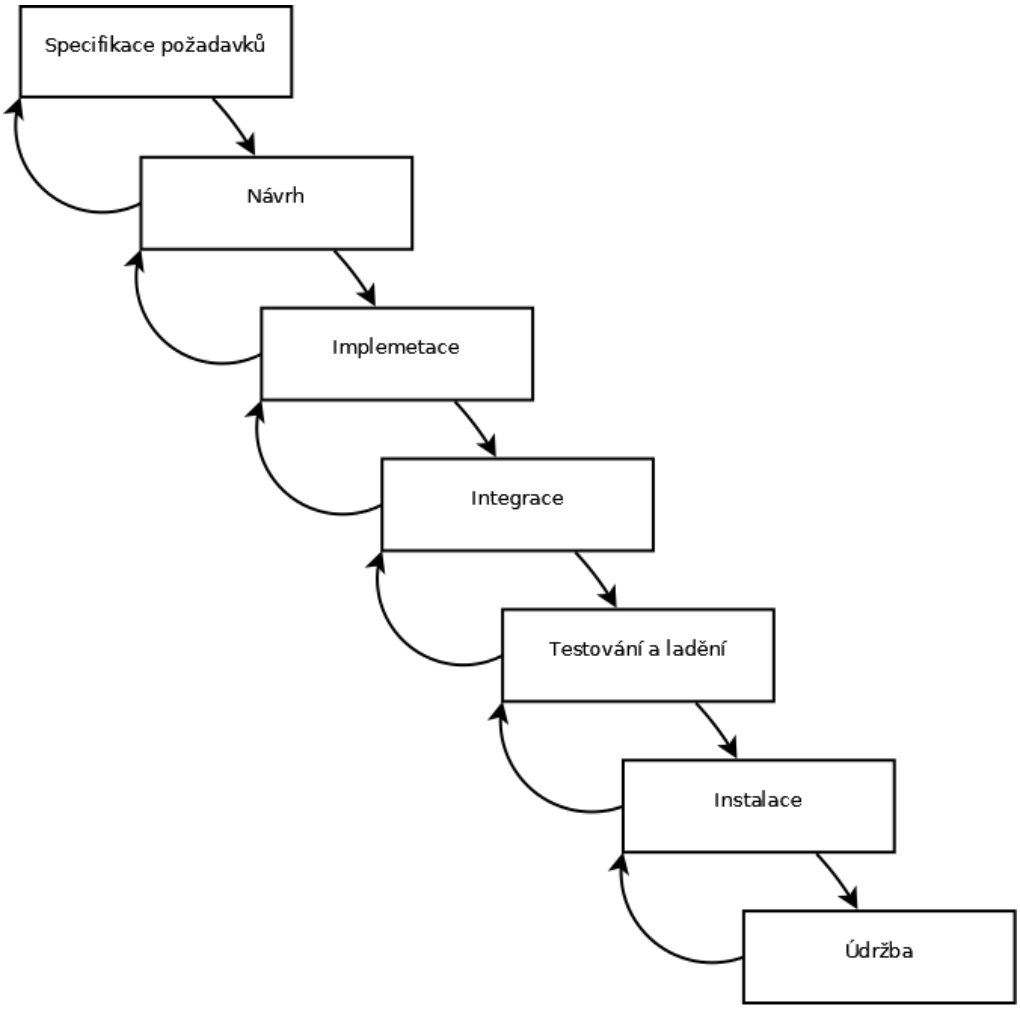
Vodopádový model patří k nejstarším a nejznámějším metodikám vývoje softwaru a všeobecně se považuje za první ucelenou metodiku. V tomto modelu se na vývoj nahlíží jako na sekvenci jednotlivých fází bez iterací. Přesun k další fázi je provázen schvalovacím procesem, kdy se oveří kompletnost a správnost předchozí fáze. Tyto fáze se mohou u různých variant vodopádového modelu mírně lišit.

Obvyklé fáze jsou následující:

1. Specifikace požadavků
2. Návrh
3. Implementace
4. Integrace
5. Testování a ladění

6. Instalace
7. Údržba

Těmito fázemi se sekvenčně postupuje, jak je znázorněno na obrázku číslo 2.



Obrázek 2 - vodopádový model

Pokud vývoj postupuje bez problémů, posunujeme se tímto modelem od jedné fáze do další. V případě, že zjistíme v průběhu vývoje nějaké nedostatky, můžeme se vrátit o jednu fázi zpět. Takto se můžeme postupně vrátit až na počátek vývoje. Pokud tedy například ve

fázi implementace zjistíme, že jsme udělali chybu v návrhu, můžeme se vrátit o jednu fázi zpět k návrhu a chybu odstranit. Poté je třeba opět schválit veškeré dokumenty provázející projekt a můžeme znovu postoupit k implementaci.

Na základě toho principu je zřejmé, že odstranění chyby, na kterou se několik fází nepřišlo, může být velmi drahé. Pro její odstranění se musíme vracet v modelu až do fáze, kdy vznikla a opět postupovat modelem dolu. V nejhorším případě můžeme na projektu pracovat v podstatě znovu od začátku. Aby se této situaci předešlo, je kladen velký důraz na kontrolu a ověřování jednotlivých fází. K tomu se ve vodopádovém modelu vypracovávají dokumenty, čímž se snižuje možnost chyby.

Mezi pozitiva tohoto modelu patří jeho jednoduchost. Je lehký na zavedení, role jednotlivých pracovníků jsou jasně rozdělené a je zřejmá odpovědnost za jednotlivé fáze projektu. Podle toho, v jaké fázi modelu se projekt nachází je také možné relativně dobře odhadovat celkovou časovou náročnost projektu a sledovat pokrok ve vývoji. Za výhodu by se také dalo považovat to, že zákazník není v průběhu vývoje rušen, i kdyz o to více může být na závěr překvapen. Vlivem ověřování a testování jednotlivých fází projektu také vzniká u tohoto modelu dokumentace, která je pak relativně snadno použitelná a například při odchodu části týmu jsou dosavadní znalosti uchované v této dokumentaci.

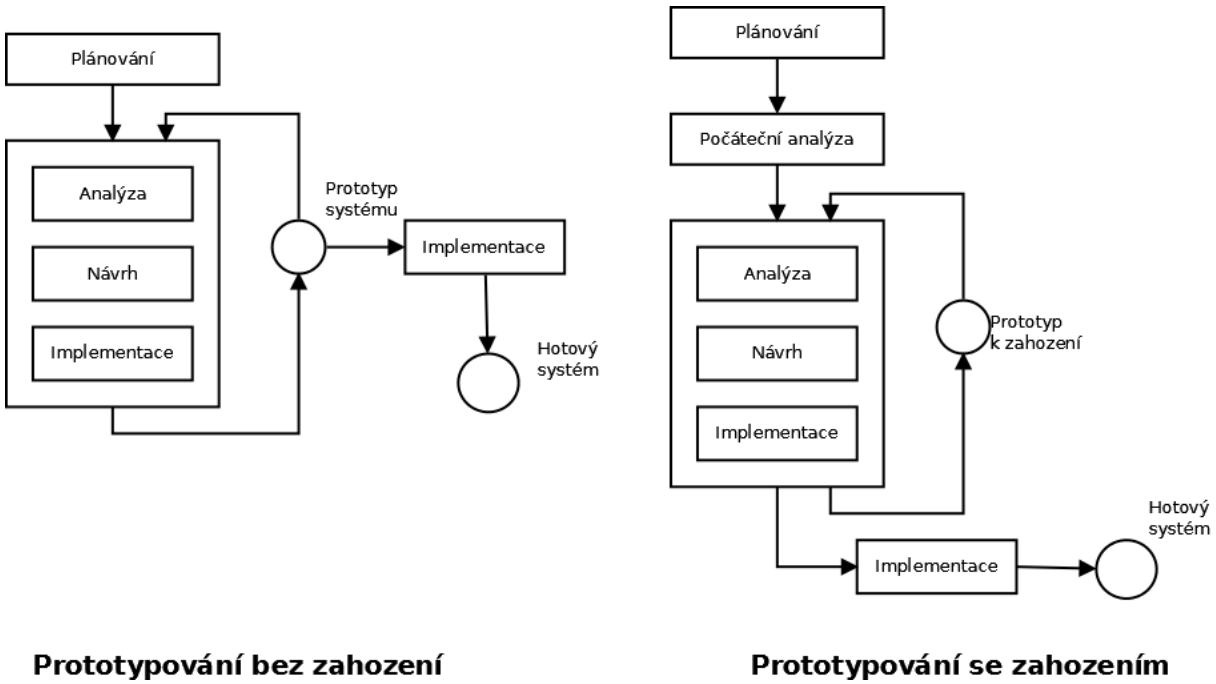
I přes zmíněné výhody má vodopádový model i silné nevýhody, které jeho nasazení velmi omezují. Je to zčásti dáno dobou vzniku a posunem v chápání vývoje softwaru. Již první fáze modelu, tedy návrh, je u rozsáhlých projektů velmi těžko realizovatelná beze zbytku. I samotný klient nemusí mít v této fázi přesné představy a nemusí domyslet různé důsledky svého zadání. Vytvoření kompletního návrhu je v dnešním dynamickém světě téměř nemožné a k posunu do další fáze by vůbec nemuselo dojít. Další nevýhodou, která souvisí s tím že zákazník není po dobu vývoje rušen, je to, že pokud nepřesně specifikuje své zadání, nebo analytik udělá chybu, dodaný software se může od jeho představ dosti lišit. V tom případě by projekt musel projít celým vývojem až do první fáze, a celý dosavadní vývoj by přišel nazmar. Model také nepočítá s úpravami a zpřesňováním zadání zákazníkem, předpokládá se, že všechny aspekty modelu byly zohledněny při návrhu.

Ve své době byl tento model revoluční a velmi používaný. Dnes však slouží spíše jako ukázka klasické metodiky, nebo ke studijním účelům. Pro malé projekty je příliš formální

a náročný na dokumentaci. U velkých projektů je problematické vytvořit návrh, který se po celou dobu vývoje již nebude měnit. Hlavním problémem modelu je jeho neflexibilila – neschopnost reagovat na změny a aktuální upřesňování. Další informace o tomto modelu vývoje můžete najít například v [1].

2.3 Prototypování

Prototypování není samostatnou kompletní metodikou, ale spíše přístupem v rámci nějaké metodiky. Je často využívané a je vhodné se o něm zmínit. Smyslem prototypování je předvést klientovi náhled na systém, byť ještě s omezenou funkcionalitou. Tento náhled je realizován tak rychle, jak je to možné, a zákazník má hned na počátku vývojového procesu šanci aktivně do něj zasahovat. Komunikace se zákazníkem probíhá ve dvou fázích vývoje, jednak při analýze a návrhu systému a pak také u hodnocení prototypu. Na základě hodnocení prototypu můžeme začít upravovat původní návrh.



Obrázek 3 – prototypování

Mezi různými variantami a aplikacemi postupu prototypování se dají rozlišit dvě skupiny – *prototypování bez zahození* a *prototypování se zahozením*.

Prototypování se zahozením

Již od počátku tvorby prototypu se počítá s tím, že nebude využit pro další vývoj. Jeho výhodou je velmi rychlá realizace. Často se jedná například pouze o nefunkční uživatelské rozhraní systému. Po rychlé a tedy levné realizaci takového prototypu se ve spolupráci se zákazníkem hodnotí a pak se upravuje zadání projektu. Případné chyby a nedostatky jsou tedy odhaleny velmi rychle a jejich odstranění přináší pouze malé náklady.

Prototypování bez zahození

Při aplikaci tohoto postupu vzniká robustní a snadno rozšiřitelný prototyp, se kterým se od počátku počítá jako se základem celého systému. Nejprve se implementují požadavky, které jsou jasně zadané. Postupně se dodává další funkcionalita na základě upřesňujících se požadavků zákazníka. Tento prototyp je v určité fázi již funkční, což demonstruje zákazníkovi jak probíhá vývoj. Je také možné nasadit již tento prototyp u zákazníka, pokud je pro něj akceptovatelné využít systém, který ještě není kompletní, avšak některé požadavky již dokáže splnit.

Výhody prototypování jsou zřejmé - zákazník v průběhu vývoje přesně definuje co vlastně od systému očekává. Tím je zajištěna úspora nákladů i času, který by se mohl strávit nad plněním nepřesného nebo chybného zadání. Spoluprací se zákazníkem se také spolehlivě vyhneme tomu, že výsledný produkt by neodpovídal jeho představám. To by se mohlo například u vodopádového modelu celkem snadno stát, a pokud by k odhalení nesrovnalostí došlo až při předání, bylo by nutné absolvovat celý proces vývoje softwaru znovu. Tento postup je také využitelný v případě, že zákazník sám nemá příliš přesnou představu co byl měl výsledný systém plnit. Postupnou spoluprací na hodnocení jednotlivých prototypů se tato představa upřesní, a šance úspěšného předání výsledného projektu zákazníkovi se zvyšuje.

Jako každá metodika a přístup k vývoji softwaru přináší i prototypování některé negativní vlastnosti, na které je nutné brát ohled a minimalizovat jejich vliv. Asi největším rizikem je nedostatečná analýza. Při navrhování prototypů systém zjednodušíme a tím

pomineme některé souvislosti, které ve finálním produktu budou hrát vliv. Je tedy vhodné již při počáteční analýze počítat se systémem jako s celkem. Dalším rizikem je to, že zákazník bude pokládat předkládané prototypy za produkt, který je již blízko k dokončení. To vede ke špatnému odhadu doby vývoje, kdy je od prototypu k výslednému projektu ještě velký, a často časově i velmi náročný, kus práce. Předkládané prototypy také mohou mást zákazníka svými odlišnostmi, kdy například ve finálním systému očekává vlastnosti a funkce, které byly pouze pracovně navrženy u některé verze prototypu. Vlivem většího počtu prototypů se také jejich správa stává méně přehlednou a celkově náročnější. Při tvorbě jednotlivých prototypů je důležité nevěnovat přílišnou pozornost nepodstatným věcem. To vede k příliš dlouhému vývoji prototypu, který řeší v aktuální chvíli i nepodstatné věci.

I přes některé zmíněné nevýhody převažují u prototypování výhody. Vlivy nevýhod se však dají při jejich znalosti velmi redukovat. Prototypování se využívá v různé míře i u ostatních metodik, a je proto vhodné mít o něm alespoň základní povědomí. Celkově by se dalo říci, že je to metodika velmi vhodná například pro projekty se složitým uživatelským rozhraním, které se v jednotlivých krocích upravují tak, aby vyhovovaly zákazníkovi. Dá se samozřejmě použít i u ostatních projektů, nicméně její výhody nebudou tak výrazné. Pro bližší studium prototypování doporučuji knihu [2].

2.4 Spirálový model

Spirálový model byl uveden na svět v roce 1988 jako reakce na na nedostačující vodopádový model. Sám autor shrnuje model v knize [3]. Oproti němu přináší dvě podstatné novinky a tím je podrobná analýza rizik a iterativní přístup k vývoji.

Při analýze rizik se hodnotí situace a vlastnosti, které by mohly projekt ohrozit. Odhaduje se jejich vliv na projekt a pravděpodobnost, s jakou mohou nastat. Na základě této analýzy se rozhodujeme o dalším vývoji projektu, jelikož nám může odhalit slabá místa v projektu, která by jinak mohla ohrozit jeho průběh.

Iterativní přístup znamená vývoj projektu v několika opakujicích se cyklech. Tím je

dosaženo toho, že zpočátku není potřeba naprosto přesné zadání, jako například u vodopádového modelu. Ze začátku stačí stanovit jen obecnější vlastnosti výsledného produktu a ty v jednotlivých iteracích postupně rozpracovávat. To byla oproti vodopádovému modelu velká změna, která činí tento model vhodnější pro rozsáhlejší projekty.

Jak již bylo zmíněno, vývoj probíhá v několika iteracích, přičemž každá se skládá z několika částí:

1. DEFINICE POŽADAVKŮ NA SYSTÉM

   Tyto požadavky by měly být co nejpřesnější, což vyžaduje komunikaci se zákazníkem a budoucími uživateli systému.

2. PŘEDBĚŽNÝ NÁVRH SYSTÉMU

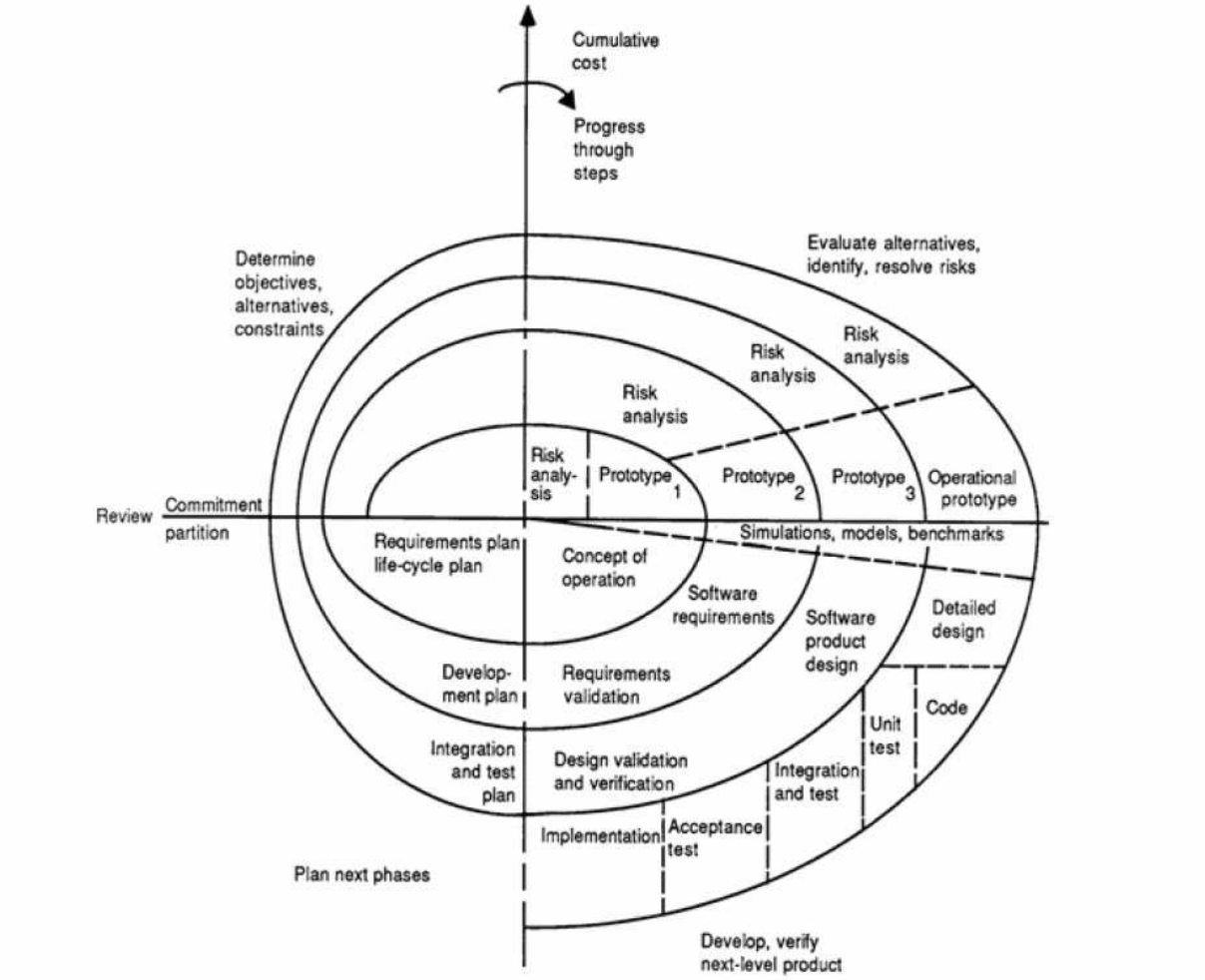
   Tato fáze je nejdůležitější ze všech. Vyhodnocují se různé alternativy a možnosti podle jejich efektivity a přínosu k celkovému řešení projektu. Dále se identifikují a hodnotí všechna možná rizika projektu, což má velký vliv pro další úspěšný vývoj.

3. VYTVOŘENÍ PROTOTYPU Z PŘEDBĚŽNÉHO NÁVRHU

   V této fázi se vytvoří obvykle zmenšená verze cílového produktu, která se, podle úrovně iterace, postupně přibližuje jeho vlastnostem.

4. TVORBA DALŠÍHO PROTOTYPU

   - Hodnocení předchozí verze protypu, jeho rizik a silných a slabých stránek
   - Definice požadavků na další verzi prototypu
   - Návrh nového prototypu
   - Konstrukce a testování nového prototopu



Obrázek 4 - spirálový model

Schéma vývoje podle spirálového modelu najdeme na obrázku 4, který pochází ze [4].

Výhody spirálového modelu jsou jeho komplexnost a nezávislost na konkrétní metodice. Díky komplexnosti se hodí i pro velké projekty, u kterých probíhá vývoj postupně v iteracích a není snaha o vývoj celého projektu najednou (v určitých krocích) jako u vodopádového modelu. Díky této komplexnosti je na druhou stranu příliš složitý pro malé projekty, kdy by tento projekt vznikl hned v první iteraci. Zde by takový model absolutně postrádal smysl. Jeho další výhodou je, že se důsledně věnuje analýze rizik. Tím se zvyšuje šance na odhalení potenciálních chyb a nedostatků. Vlastnost, kterou tento model neodstranil, a která byla vytýkána již vodopádovému modelu, je způsob předání celého projektu

zákazníkovi. Ten probíhá až po dokončení posledního cyklu

I přesto, že tento model byl použit pro vývoj řady především velkých projektů, se od něj v současné době ustupuje. Zčásti proto, že jsou k dispozici celkově propracovanější metodiky a také pro jeho nedostatečnou pružnost, která je pro mnoho projektů velmi podstatná.

2.5 RUP

Zkratka RUP znamená *Rational Unified Process*. Tato metodika byla vyvinuta společností Rational, později byla tato společnost koupena firmou IBM, která dále vyvíjí i RUP. Jedná se o velmi rozsáhlou a komplexní metodiku, která je vhodná hlavně pro větší projekty. Vzhledem k této složitosti je nutné metodiku vždy předem přizpůsobit pro konkrétní projekt. Vývoj projektu probíhá v cyklech, v úvodním cyklu je již zhotoven funkční software, u kterého se v dalších iteracích rozvíjí funkcionalita.

Metodika RUP využívá 6 obecných praktik, které zajišťují efektivní vývoj softwaru:

- Iterativní vývoj – podobně jako ve spirálovém modelu je projekt vyvíjen postupně, čímž se mimo jiné snáze odhalují skryté chyby.
- Správa a řízení požadavků – bere ohled na to, že požadavky zákazníka se v čase vyvíjejí a je třeba je nějakým způsobem zpracovávat.
- Komponentová architektura – jednotlivé části systému se mohou vyvíjet odděleně, mohou být použity opakovaně a i jejich změny jsou jednodušší.
- Vizuální modelování – pro větší přehlednost projektu a usnadnění komunikace v rámci týmu se využívá jazyk UML.
- Ověřování kvality – v průběhu vývoje se průběžně ověřuje kvalita produktu. Kontrolovaná kritéria kvality jsou: funkcionalita, spolehlivost a výkon.

- Řízení změn – jednolivé změny projektu přináší nutnost jejich správy a dokumentace tak, aby byl stav projektu v průběhu celého vývoje přehledný.

Metodika RUP je výhodná pro svoji rubustnost. Je velmi obecná a dá se přizpůsobit na míru jednotlivým projektům. Proto se v úvodní fázi upravuje metodika pro konkrétní projekt, který bude pomocí ní vyvíjen. Pro nutnost úprav a celkovou obsáhlost a složitost však je třeba mít zkušený tým, který se bude v metodice dobře orientovat. Tato metodika je tedy vhodná spíše pro větší projekty, kde je kladen velký důraz na kvalitu.

Popis celé metodiky by však svým rozsahem přesahoval rozsah této práce a pro další studium doporučuji: [5].

3. Agilní metodiky

3.1 Důvod vzniku agilních metodik

Po úvodním popisu rigorózních metodik, které byly uvedené pro srovnání, je tato část práce věnována obecnému popisu agilních metodik a detailnějšímu pohledu na některé z nich.

Pro pochopení smyslu výrazu *Agilní metodiky* se můžeme podívat do slovníku na překlad slova „agile“. Zjistíme, že do češtiny se dá přeložit jako „hbitý, čilý, svižný nebo bystrý“. Tyto výrazy by tedy měly popisovat takzvané agilní metodiky. Ty se začaly objevovat jako reakce na změny trhu se softwarem. V dnešní době, kdy napsat jednoduchý program, nebo pomocí volně dostupných nástrojů vytvořit internetovou prezentaci zvládne stále více lidí, začíná být rychlost vývoje a čas dodání projektu velmi důležitým parametrem. Vytvořit zadaný projekt tak není problém, zásadní je čas a náklady, které tento projekt spotřebuje. Klienty totiž často nezajímá ani tak dokonalý návrh systému a obsáhlá dokumentace, ale fungující aplikace, která jim přinese výhodu v konkurenčním prostředí. Je tedy výhodnější vytvořit projekt, který nebude naprosto dokonalý ve všech ohledech, ale musí být dodán včas a za rozumnou cenu.

Na základě myšlenek, která zohledňují tato fakta, začaly postupně vznikat agilní metodiky. Ty umožňují vyvíjet software rychle a efektivně, a také počítají s tím, že v průběhu projektu se bude měnit a upřesňovat původní zadání. Na rozdíl od klasických metodik nepovažují výslednou funkcionalitu jako fixní hodnotu, která je stanovena na počátku projektu a musí se splnit, na což jsou potřeba určité zdroje a čas. To vede u klasických metodik k častému problému, kterým je překročení stanoveného času vývoje a stanovených nákladů ve snaze splnit původní zadání. Oproti tomuto přístupu považují agilní metodiky za fixní veličiny zdroje a čas potřebný k realizaci projektu, zatímco funkcionalitu jako proměnnou v čase. V praxi to znamená, že zákazníkovi může být dodán produkt, který poskytuje jen některé funkce. Ty mohou být postupně, podle jejich priority, dodělávány. Pro zákazníka bývá totiž mnohdy výhodnější využívat nekompletní software, než čekat na dodání

výsledného projektu. Po tuto dobu pro něj projekt představuje investici, která se ještě nemá šanci začít splácet.

Agilní metodiky, které zohledňují tyto skutečnosti, obsahují několik společných prvků:

- **Iterativní a inkrementální vývoj**

  Vývoj probíhá v krátkých iteracích, při nichž je postupně dodávána celková funkcionalita. Zákazník se tak může již v průběhu vývoje seznamovat s produktem, upřesňovat zadání a připravovat se na využívání finálního produktu. Zároveň se jednodušeji a levněji odhalují a odstraňují případné chyby. Výhody iterativního vývoje se využívají i v některých tradičních metodikách, u agilních metodik se však většinou využívá kratších iterací.

- **Komunikace mezi vývojovým týmem a zákazníkem**

  Zákazník úzce spolupracuje s vývojovým týmem nejen při návrhu, ale rozhoduje například i o prioritách jednotlivých částí projektu a poskytuje zpětnou vazbu vývojovému týmu. Při této spolupráci, kdy se zákazník může stát i členem vývojového týmu, se nestane, že by se vyvíjený produkt začal odlišovat od jeho představ.

- **Průběžné testování**

  Vzhledem ke krátkým iteracím a častým změnám softwaru je nutné pravidelně v průběhu vývoje testovat jeho funkčnost. Je tedy záhodno, aby byl celý produkt snadno testovatelný, a aby ho bylo možno testovat pomocí automatizovaných testů.

Tyto využívané prvky vývoje byly popsány v Manifestu agilního vývoje softwaru, kterému se věnuje následující kapitola.

3.2 Manifest agilního vývoje softwaru

Přelomem ve vývoji agilních metodik, který do té doby probíhal víceméně nahodile a chaoticky, bylo sestavení *Manifestu agilního vývoje softwaru.* To se stalo v únoru 2001, kdy se v lyžařském středisku Snowbird v Utahu sešlo sedmnáct softwarových vývojářů a tento manifest na základě debaty sestavili. Jednalo se v podstatě o sepsání vlastností jednotlivých nově se vyvíjejících metodik, které se vzájemně velmi podobaly do ucelené podoby.

Základní teze manifestu jsou:

- Přijmout a umožnit změnu je efektivnější než ji bránit
- Požadavky na změny se určitě objeví.

Na základě těchto tezí došli autoři manifestu k závěru, že metodika by měla dávat větší význam:

- Individualitě a komunikaci před procesy a nástroji
- Fungujícímu softwaru před obsáhlou dokumentací
- Spolupráci se zákazníkem před vyjednáváním smluv
- Reakci na změny před plněním plánu

Je nutné podotknout, že jistou váhu mají i položky na pravé straně, větší důraz se však klade na položky nalevo.

Manifest agilního vývoje softwaru dále obsahuje také dvanáct principů, které jsou

uplatňovány u agilních metodik. Jsou to:

- **Uspokojování požadavků zákazníka rychlým dodáním užitečného softwaru**

  Zákazník potřebuje využívat fungující program, rozsáhlá dokumentace mu nepřináší žádnou výhodu a není proto pro něj zajímavá.

- **Vítání změn v projektu, a to i v pozdějších fázích**

  Počítáme se změnami zadání projektu, které vznikají jako důsledek vyvíjejícího se prostředí, ve kterém zákazník operuje. V průběhu vývoje tedy mohou být zadávány nové požadavky, které reflektují aktuální situaci. Agilní metodiky se tedy nesnaží o získání kompletního zadání projektu hned v počátku vývoje, ale počítají s tím, že zadání se může měnit a upravovat.

- **Časté dodávky fungujícího softwaru – spíše v řádu týdnů než měsíců**

  Vývoj projektu v krátkých cyklech usnadňuje zpřesňování zadání zákazníkem. Pomocí této zpětné vazby se projekt ubírá správným směrem. Pozitivem pro zákazníka je také to, že již fungující software může nasadit do provozu, kde mu přináší užitek, a nemusí čekat na konečnou verzi.

- **Funkční software je hlavním měřítkem pokroku ve vývoji projektu**

- **Trvale udržitelné tempo vývoje**

  Při dlouhodobém přetěžování zaměstnanců klesá jejich výkonnost. Je proto nutné zajistit, aby vývoj odpovídající rychlostí byl zvládnutelný bez dlouhodobého překračování pracovní doby a vystavování zaměstnanců přílišnému stresu.

- **Častá, klidně i denní, spolupráce se zákazníkem**

  Vzhledem k tomu, že agilní metodiky počítají s tím, že zadání projektu je zpočátku nepřesné a upřesňuje se až v průběhu vývoje, vyžadují častou komunikaci se zákazníkem. Ten může být i součástí vývojového týmu

a průběžně upřesňuje svoje požadavky. To je možné i díky možnosti testování projektu v průběhu vývoje.

- **Osobní komunikace je nejrychlejším a nejefektivnějším způsobem předávání informací**

  Pokud je to možné, je vhodné používat komunikace tváří v tvář. Její efektivita je vyšší než komunikace pomocí psaného textu. Týká se to, jak komunikace se zákazníkem, tak i komunikace v rámci vývojového týmu.

- **Motivovaní vývojáři, kterým je dána dostatečná důvěra**

  Klíčem k úspěchu projektu jsou lidé, musí být motivovaní, mít dostatek důvěry a přiměřené kompetence k provádění vlastních rozhodnutí.

- **Neustálé zaměření na technickou dokonalost a dobrý návrh**

  Změny ovlivňují projekt v jeho průběhu vyžadují kvalitní návrh systému, aby je bylo možné snadno zpracovávat

- **Jednoduchost**

  Projekt se skládá z jednodušších částí, které se snáze testují i upravují. Je důležité postupovat tak, aby se s minimálními náklady vytvořilo co nejvíce užitečné práce.

- **Samoorganizující se týmy**

  Nejefektivnější organizací je ta, které se sama organizuje podle svých potřeb. K tomu je však třeba zkušený a dobře pracující tým.

- **Pravidelné adaptace na měnící se okolnost**

  Jelikož v průběhu vývoje dochází ke změnám jak okolního prostředí, tak samotných požadavků zákazníka, je nutné se těmto změnám neustále přizpůsobovat.

Těmito vlastnostmi tedy v různé míře disponují všechny agilní metodiky. Pomocí nich se metodiky snaží dovést vývojový tým k včasnému dodání funkčního programu v rozumný čas. Jedině to přináší zákazníkovi konkurenční výhodu.

Agilní metodiky mají také další společnou vlastnost – nejedná se o přesný postup jak projekt vyvíjet. Je ponechán poměrně velký prostor pro úpravy a adaptaci metodiky pro potřeby konkrétního projektu. To vychází i z výše uvedených základních principů agilního přístupu k vývoji. Agilní metodiky jsou tedy poměrně jednoduché, flexibilní a zároveň upravitelné dle konkrétních potřeb.

3.3 Feature Driven Development

Pojem „Feature Driven Development“ znamená v češtině vývoj řízený funkcemi, nebo užitnými vlastnostmi softwaru. Často se také používá pouze zkratka FDD. Feature Driven Development je jedním z důležitých představitelů agilních metodik vývoje softwaru. Jeho základy, kterými bylo propojení iterativního přístupu a praktik, které se osvědčily v průmyslu, položil Peter Coad. Později pak ve spolupráci s Jeffem de Lucou a Stephenem Palmerem FDD zdokonalil a rozšířil. V literatuře jsou tito tři uváděni jako spoluautoři FDD. Jejich snahou bylo, aby byl vývoj softwaru pomocí FDD jednodušší, čitelnější a efektivnější. Metodika byla poprvé prezentována v [6].

3.3.1 Popis metodiky FDD

Metodika vývoje softwaru FDD se z fází životního cyklu softwaru zaměřuje na návrh a implementaci. Její výhody jsou v neustálém monitoringu stavu projektu, dohlížením na kvalitu v průběhu vývoje a častých dodávkách fungujících dodávek zákazníkovi. Metodika je složená z pěti sekvenčních procesů, přičemž dva poslední se iterativně opakují. Jeden z autorů metodiky, Stephen Palmer ve své knize dokonce říká, že metodika je vhodná i k vývoji kritických aplikací, pro které se většina dalších agilních metodik nedá příliš doporučit [7].

Cílem FDD je odstranění chaosu a nejistot, čímž je umožněna efektivní práce, přinášející maximální zisk. Podle metodiky je fungující výsledný produkt to jediné, k čemu směřuje naše snaha.

Základní myšlenkou je rozložit celý budoucí systém na množinu vlastností, které se poté postupně implementují. Na začátku vývoje je vytvořen celkový model systému. Tento model je tvořen z velkého nadhledu, je celkem abstraktní a neobsahuje jednotlivé detaily. Na základě tohoto modelu se určí hlavní vývojová linie, po které se bude tým při vývoji ubírat. V následujících iterativních fázích se pak implementují jednotlivé vlastnosti systému. Délka těchto iterací je různá, určení podle konkrétního projektu. Obvykle jsou však relativně krátké, obvykle dva týdny. V každé iteraci se paralelně implementují určené vlastnosti. Výhodou tohoto iterativního přístupu je, stejně jako u dalších agilních metodik, možnost dodávat produkt postupně, klidně i po každé iteraci. Zákazník má tedy možnost ovlivňovat vlastnosti budoucího produktu již ve fázi vývoje, a také přehled o postupu vývoje. Metodika je tedy dostatečně flexibilní, a je připravena na upřesňování zadání ze strany zákazníka v průběhu vývoje.

3.3.2 Pojmy

Jedním ze základních pojmů, se kterými metodika FDD operuje je *vlastnost*. Je ostatně uveden i v jejím názvu. Vlastnost je v podstatě malý kousek funkcionality, který přináší nějaký konkrétní užitek pro zákazníka. Tato vlastnosti mají několik parametrů, které je charakterizují.

- Realizovatelnost - každá jednotlivá vlastnost musí být realizovatelná v rámci jedné iterace. Pokud není realizovatelná, je nutno ji rozdělit na více menších vlastností.
- Srozumitelnost – vlastnost musí být srozumitelně popsána, musí být definován její účel a výsledek.
- Měřitelnost – implementovaná vlastnost musí být ta, kterou zákazník požaduje.

Kvůli přehlednosti zavádí metodika jednotný formát zápisu vlastnosti. Definice vypadá takto:

<akce> <předmět > <podobnosti>

Akce označuje činnost, která se provádí v rámci dané vlastnosti, *předmět* je element, nad kterým je konkrétní akce prováděna a *podrobnosti* zpřesňují zadání, omezují jeho platnost a tak podobně. Příklady zadání jednotlivých vlastnosti mohou být například takovéto:

- Spočítání příjmů za aktuální měsíc
- Tisk přehledu na tiskárně
- Poslání údajů o platbě zákazníkovi

Dalším důležitým elementem v rámci metodiky FDD je modelování. To se provádí prakticky ve všech fázích FDD. Nejdříve se tvoří globální model a poté se v jednotlivých iteracích vytvářejí podrobnější modely, které detailně popisují konkrétní vlastnosti. Díky velkému využití modelování se metodika FDD také někdy nazývá *metodikou řízenou modelem.*

3.3.3 Role v rámci FDD

Metodika FDD určuje role jednotlivých členů vývojového týmu. Tyto role mají určeno, co je jejich úkolem v jednotlivých fázích vývoje. Platí, že v jedné roli může účinkovat více osob, stejně tak může jedna osoba plnit několik rolí. Každá role má svůj smysl a úkoly, které by měla plnit. Z hlediska důležitosti se tyto role dají rozdělit do třech skupin: *klíčové, podpůrné* a *další.* Nyní následuje popis těchto rolí.

Klíčové role

Projektový manažer (Project Manager) je hlavním vedoucím projektu. Zodpovídá za jeho administrativní i finanční stránku. Mezi jeho úlohy patří zajištění odpovídajících pracovních podmínek pro tým tak, aby práce byla efektivní. V jeho kompetenci je také

rozsah projektu, časový harmonogram i personální obsazení vývojového týmu.

Hlavní architekt (Chief Architekt) je zodpovědný za globální návrh systému a má rozhodující slovo co se týká návrhu na všech úrovních. Není ovšem jediným autorem návrhu, spíše koordinátorem ostatních členů během procesu návrhu. Role hlavního architekta vyžaduje velké dovednosti v oblasti modelování. Pro složité projekty, které vyžadují složité technické řešení, se může jeho role rozdělit na dvě – technický architekt (Technical Architekt) a doménový architekt (Domain Architekt).

Vývojový manažer (Development manager) má za úkol řešit každodenní problémy vznikající při vývoji. Jde hlavně o problémy se zdroji a o problémy v rámci týmu. Jeho funkce se částečně kryjí s funkcemi projektového manažera nebo hlavního architekta, takže může být tato role obsazena člověkem vykonávajícím zároveň jednu z výše uvedených rolí.

Hlavní programátoři (Chief Programmer) jsou členové týmu, kteří vedou jednotlivé menší (obvykle 3 až 5 členů) týmy programátorů, které v iteracích provádějí analýzu, návrh a vývoj jednotlivých vlastností. Jedná se zkušené vývojáře, mající zkušenosti z předchozích projektů. Tito programátoři se účastní také prvotní analýzy, stanovení požadavků na systém a jeho návrhu. Jejich dalším úkolem je komunikace s ostatními hlavními programátory. Jejich úkolem je také podávat zprávy o pokroku v projektu v rámci jejich týmu.

Vlastník třídy (Vlase Owner) je člen menšího týmu, vedeného Hlavním programátorem, a jeho úkolem je návrh, testování a dokumentace jednotlivých konkrétních vlastností. Je tedy zodpovědný za třídu, jejíž je vlastníkem, a to po celou dobu vývoje. Z vlastníků tříd se vytvářejí *týmy pro vlastnosti*, jejichž složení se může v každé iteraci měnit.

Doménový expert (Domain Expert) je člověk, který dokonale rozumí oblasti, pro kterou je systém vyvíjen. Může to být přímo koncový uživatel, klient, analytik, nebo nějaká kombinace těchto postavení. Jeho úkolem je předávat vývojářům své znalosti prostředí tak, aby mu daný projekt co nejlépe odpovídal. Poskytuje také zpětnou vazbu a kontrolu, zda se daný projekt neodchyluje od zadání klienta. Tato role vyžaduje

velmi dobré komunikační, vyjadřovací a prezentační schopnosti tak, aby mohl doménový expert rychle a přesně předávat své poznatky.

Podpůrné role

Manažer pro dodávky (Release Manager) je osoba, mezi jejíž úkoly patří kontrola postupu vývoje, shromažďování zpráv o tom co je již hotové a v jaké fázi vývoje se projekt nachází. Tyto informace poskytuje projektovému manažerovi.

Jazykový specialista (Language Guru) je odborník dokonale ovládající používanou technologii, který své znalosti poskytuje ostatním členům vývojového týmu. Pod používanou technologií se rozumí například daný programovací jazyk. Role jazykové specialisty nabývá na důležitosti při přechodu na novou technologii, kdy se může jednat i externího konzultanta.

Konstruktér (Build Engineer) je zodpovědný za nastavení provedení procesu sestavení výsledné aplikace. Jeho dalšími úkoly jsou správa systému pro kontrolu verzí a zveřejňování dokumentace.

Nástrojář (Toolsmith) vytváří malé podpůrné programy, které pomáhají ostatním členům týmu ve vývoji. Pod tuto funkci spadá například správa databáze a tvorba webových stránek týkajících se projektu.

Administrátor (Systém Administrator) se stará o hardwarové a softwarové prostředky používané vývojovým týmem.

Ostatní role

Tester (Tester) testuje vyvíjený projekt a kontroluje, zda odpovídá požadavkům zákazníka. Může to být i externí pracovník pracující odděleně od týmu.

Správce nasazení nových verzí (Deployer) zajišťuje nasazování nových verzí u zákazníka a s tím spojené případné změny v používaných datech nebo v databázi

Písař (Writer) vytváří technickou dokumentaci a manuály pro používání.

3.3.4 Procesy

Metodika FDD definuje v rámci vývoje produktu několik procesů, které na sebe navazují. Pro jejich definici zavádí jednotný formát obsahující:

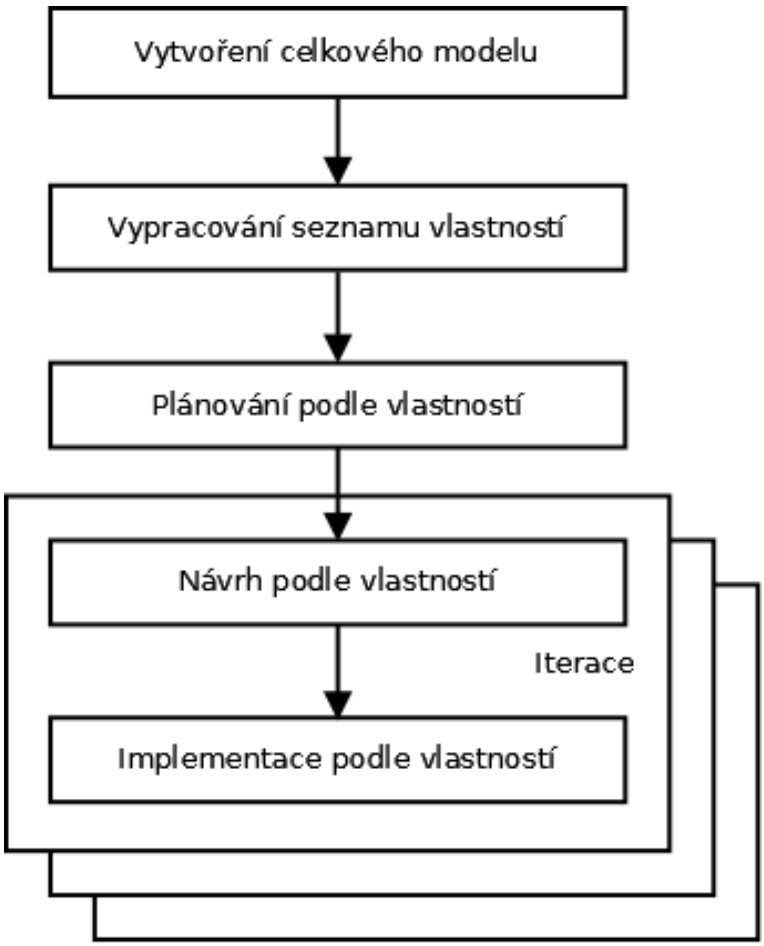
- Popis procesu
- Vstupní kritéria
- Úkoly
- Verifikaci
- Výstupní kritéria

Avšak narozdíl například od metodiky RUP, která patří k nejpropracovanějším, zavádí FDD tyto procesy jen relativně volně. Metodika RUP tyto procesy podrobně rozebírá a definuje konkrétní metody, šablony a vzorové příklady. Tím provádí vývojový tým celým procesem s malou možností improvizace. Oproti tomu FDD představuje jen obecný rámec, kterého by se měl tým držet a poskytuje řadu možností k vlastnímu řešení. Pokud tedy FDD předepisuje provést návrh systému, pak již striktně neurčuje jak a pomocí jakých nástrojů by tento krok měl být proveden. Metodika říká pouze, co se má dělat, ale neurčuje jak. Toto rozhodnutí je v rukou vedoucích členů týmu, kteří mohou vybrat optimální způsoby a nástroje, jak daný krok nejlépe provést. To samozřejmě ale klade větší nároky na zkušenosti hlavních členů týmu. Touto svobodností se zamezí pasivnímu následování jednotlivých předepsaných kroků, bez zamyšlení nad jejich smyslem a optimalizací. Metodika FDD je tedy v souladu s pravidly agilního vývoje postavená volněji, je připravená na změny a úpravy podle aktuálních podmínek.

Těchto pět procesů, definovaných metodikou FDD jsou:

1. vytvoření celkového modelu (develop an overall model)
2. vypracování seznamu vlastností (build a features list)
3. plánování podle vlastností (plan by Feature)

4. návrh podle vlastností (design by Feature) -opakující se
5. implementace podle vlastností (build by Feature) -opakující se



Obrázek 5 - FDD

Vytvoření celkového modelu

Popis procesu: V této fázi hrají hlavní roli doménoví experti, hlavní programátoři a hlavní architekt. Úkoly pro první fázi vývoje jsou studium domény a vytvoření globálního modelu. Studium domény, tedy prostředí, ve kterém bude plánovaná aplikace nasazena, představuje studium dokumentů. Doménoví experti musí seznámit celý tým se zaměřením projektu, požadavky a očekáváními, které nad projektem jsou. Ve fázi tvorby modelu jsou informace předávány v přirozeném jazyce, a dále formou

náčrtků a komentářů. Přílišná snaha o detailnost a formalizace požadavků je v tomto kroku spíše na škodu.

Vstupní kritéria: jsou vybrání doménoví experti, hlavní programátoři a hlavní architekt

Úkoly: sestavit tým pro tvorbu modelu, prozkoumat podrobně doménu a s ní související dokumenty, vytvoření globálního modelu, jeho prověření a vyladění

Verifikace: posudky – interní v rámci týmu a externí se zákazníkem

Výstupní kritéria: vytvořený model; diagram tříd s atributy a metodami, sekvenční diagramy, další detaily uvedeny v poznámkách

Vypracování seznamu vlastností

Popis procesu: Jak již název procesu napovídá, tak v této fázi dochází ke tvorbě seznamu vlastností. Je samozřejmé, že seznam vlastností vytvořený v této fázi nebude kompletní a dále neměnný. Je však snaha o co největší rozsah vlastností, kryjící co největší oblast navrhovaného systému. V tomto seznamu se vlastnosti seskupují na základě své podobnosti do souvisejících skupin. Ke tvorbě seznamu se využívá model z fáze 1, ale také ostatní dokumenty a také zkušenosti a schopnosti doménových expertů

Vstupní kritéria: je vytvořen celkový model

Úkoly: určení týmu pro tuto fázi, a realizace seznamu vlastností podle uvedeného popisu

Verifikace: interní i externí posouzení vytvořeného seznamu

Výstupní kriteria: je hotový seznam vlastností, který je schválený k použití

Plánování podle vlastností

Popis: Výstupem této třetí fáze je plán, jak postupně implementovat vlastnosti. Vychází se ze seznamu vlastností, který je seřazen podle priorit a závistí. Každá skupina vlastností je přidělena jednomu hlavnímu programátorovi, a jednotlivé třídy

přiděleny jejich majitelům. Tito za ně pak budou zodpovídat po celou dobu vývoje. Pro skupiny vlastností je dále vhodné definovat určité body, jejichž splnění bude indikovat patřičný posun ve vývoji.

Vstupní kritéria: musí být k dispozici sestavený seznam vlastností

Úkoly: vytvoření týmu pro plánování, určení pořadí implementace vlastností, přiřazení skupiny vlastností hlavním programátorům a třídy vlastníkům tříd

Verifikace: průběžná interní hodnocení

Výstupní kritéria: hotový plán vývoje s přibližnými daty dokončení, hlavní programátoři jsou přiděleni ke skupinám vlastností, třídy mají své vlastníky

Návrh podle vlastností

Popis: Následující dvě fáze se iteračně opakují. V jejich průběhu se vyvíjejí jednotlivé vlastnosti. Hlavní programátor, na základě priorit a závislostí, vybere vlastnosti, které se budou v určité jednotlivé fázi implementovat. Poté kontaktuje vlastníky tříd, jichž se tato vlastnosti týká, a ti vytvoří dočasný tým. Tento tým vypracuje podrobný návrh určité vlastnosti včetně časového rámce.

Vstupní kritéria: proces plánování je dokončen

Úkoly: sestavit podle vlastnictví tříd potřebné dočasné týmy, provést detailní návrh dané vlastnosti, může dojít i k detailnímu zkoumání určité části cílové domény či doplnění celkového modelu

Verifikace: inspekce návrhu

Výstupní kritéria: vytvořen *balíček návrhu* obsahující obecné informace o vlastnosti, je vytvořen objektový model dané vlastnosti, hlavičky tříd a metod, existuje harmonogram prací

Implementace podle vlastností

Popis: v poslední fázi zbývá implementovat vybrané vlastnosti. To provádějí vlastnící

tříd, kterých se daná funkcionalita týká, a tito také zodpovídají za testování. Pro implementaci vlastností tedy vznikají dočasné týmy z vlastníků tříd. Tyto týmy dynamicky podle potřeby vznikají a zanikají. Jejich sestavování má za úkol hlavní programátor, který také posléze integruje dokončenou vlastnost do systému. Jelikož vývoj probíhá do určité míry paralelně, může být jeden člověk členem více týmů

Vstupní kritéria: předchozí fáze je dokončena – balíček návrhu prošel inspekcí

Úkoly: implementovat třídy a metody pro danou vlastnost, provést testování, zajistit integraci do systému

Verifikace: inspekce kódu, testování jednotek

Výstupní kritéria: implementované metody a třídy absolvovaly inspekci a tvorba vlastnosti byla dokončena

3.3.5 Praktiky

Vývoj s využitím metodiky FDD přináší také několik ověřených praktik. Jejich dodržování vede tým k efektivnosti a včasnému dokončení projektu podle zadání. Podle autorů se nemusí zavádět všechny praktiky najednou, ale podle aktuální situace, hlavně zkušenosti týmu, je postupně aplikovat.

Objektové modelování domény

Jelikož objektový model je nejbližší normálnímu lidskému myšlení, je vhodné cílovou doménu modelovat objektově. Cílovou doménou se rozumí oblast ve které bude aplikace nasazena. V objektovém modelu můžeme využívat vztahy mezi třídami, dědičnosti a interakce mezi třídami pomocí metod. V fázi návrhu modelu je nutné, aby analytici nezabíhali do zbytečných detailů. Tento model má být abstraktní, sloužící k seznámení s doménou. Detailní architektura vzniká postupně v dalším vývoji.

Vývoj podle vlastností

V metodice FDD jsou stěžejním pojmem *vlastnosti*. Zaměřením na ně se věnujeme

těm požadavkům, které na nás má zákazník. Tím se minimalizuje možnosti odchýlení vývojového týmu od těchto požadavků.

Vlastnictví tříd

Tato praktika se nevyužívá jen v FDD ale je součástí i dalších agilních metodik. Vlastník třídy má na starosti její implementaci, testování i celkovou konceptuální integritu. To je podstatné, pokud s již hotovou třídou má spolupracovat nová třída. Vlastník dané třídy musí zajistit, aby jeho třída byla schopna komunikovat s novou třídou. Další výhoda je ta, že vlastník třídy o ní má dokonalý přehled, je schopen rychle dodávat novou funkcionalitu a nemusí daný kód studovat jako někdo další.

Dynamické týmy

Týmy se sestavují pro implantaci určité vlastnosti, protože tato většinou zasahuje přes více tříd. Jelikož vlastnosti se vyvíjí paralelně, jeden člověk může být členem více týmů. Vlastníky podstatných tříd si do týmu vybírá vedoucí týmu pro implementaci vlastnosti. Tyto týmy po dokončení vlastnosti zanikají.

Inspekce

Inspekce se netýkají jen samotného kódu, ale také modelů, vznikajících při návrhu. Slouží k odhalování chyb, kterým se v průběhu vývoje prakticky nedá úplně vyhnout.

Pravidelné dodávky nových verzí a správa konfigurací

Třídy musejí být integrovány tak, aby bylo možné sestavit z nich funkční aplikaci v každém okamžiku. Ta nemusí být kompletní, a slouží k získání zpětné vazby od zákazníka, ale také jako ukázka postupu vývoje. Správa konfiguracím za funkci jednoznačné identifikování souborů s kódy a uchovávání historie změn.

Zprávy o stavu projektu

V metodice FDD je na přehledné a podrobné sledování stavu vývoje kladen důraz. Jsou stanoveny tvary tabulek a šablony, které slouží k zachycení aktuálního stavu vývoje, stupně zapracovanosti a plánů do další iterace. Výstupy z těchto informací jsou dodávány celému týmu. V současnosti se pro tyto účely používají speciální aplikace.

3.3.6 Shrnutí FDD

V metodice FDD je ústředním pojmem *vlastnost*. Tyto jednotlivé části vyžadované funkcionality se v průběhu vývoje realizuji. Tím, že je projekt zaměřen na tyto menší části, nedochází v průběhu vývoje k odklonění od původního cíle. FDD zajímavým způsobem slučuje prvky jak tradičních, tak agilních metodik. Agilní přístup představují iterace ve vývoji, komunikace se zákazníkem v průběhu vývoje a tedy schopnost reagovat pružně na změny. Z tradičních metodik využívá FDD plánování vývoje a hlavně důraz na modelování a návrh. Spojením těchto vlastností vznikla agilní metodika, která je zároveň použitelná pro projekty s vysokými nároky na kvalitu a je vhodná pro širokou škálu projektů.

3.4 Scrum

Slovo *scrum* přeložené do češtiny znamená *skrumáž*, nebo také *mlýn* – formaci hráčů v rugby. V této sevřené formaci hráči postupují vpřed a vzájemně si nahrávají. Takto by měl podle této mediky fungovat i vývoj softwaru. Vzhledem k tomu že slovo *scrum* není zkratka, není také žádný rozumný důvod ho psát velkými písmeny, i když je to často k vidění. Metodika Scrum vznikla v roce 2002, patří tedy mezi relativně mladé metodiky, kdy ji její autoři Ken Schwaber a Mike Beedle zveřejnili v [8].

3.4.1 Popis metodiky

Scrum slouží k podpoře vývoje, nejedná se tedy o metodiku, které by řešila konkrétní nástroje a postupy používané při vývoji, ale řeší způsob komunikace a spolupráce v rámci týmu. Scrum vychází z faktu – podobně jako ostatní agilní metodiky - že vývoj přináší nepředvídatelné události, na které je třeba adekvátně reagovat. Scrum tedy vylepšuje

dosavadní postupy vývoje tím, že se specializuje na hledání a překonávání překážek ve vývoji produktu. Vlastnosti metodiky scrum odpovídají dalším agilním metodikám, využívá se iterativní vývoj, komunikace se zákazníkem, dodávání částí aplikace a metodika je připravena reagovat na požadavky měnící se v průběhu vývoje.

3.4.2 Pojmy

Scrum využívá několik poněkud speciálních pojmů, které budou v následující kapitole osvětleny.

Sprint

Sprint je první typ iterace. Je poměrně dlouhý, trvá přibližně měsíc. Jejich počet se u různých projektu může lišit, ale obvykle jich je tři až osm. Před začátkem sprintu se koná schůzka, kde se určuje, co se bude implementovat. Po ukončeni sprintu se opět koná schůze, na které se hodnotí, co se povedlo implementovat, co se nestihlo, a jaké nové požadavky se v průběhu vývoje objevily.

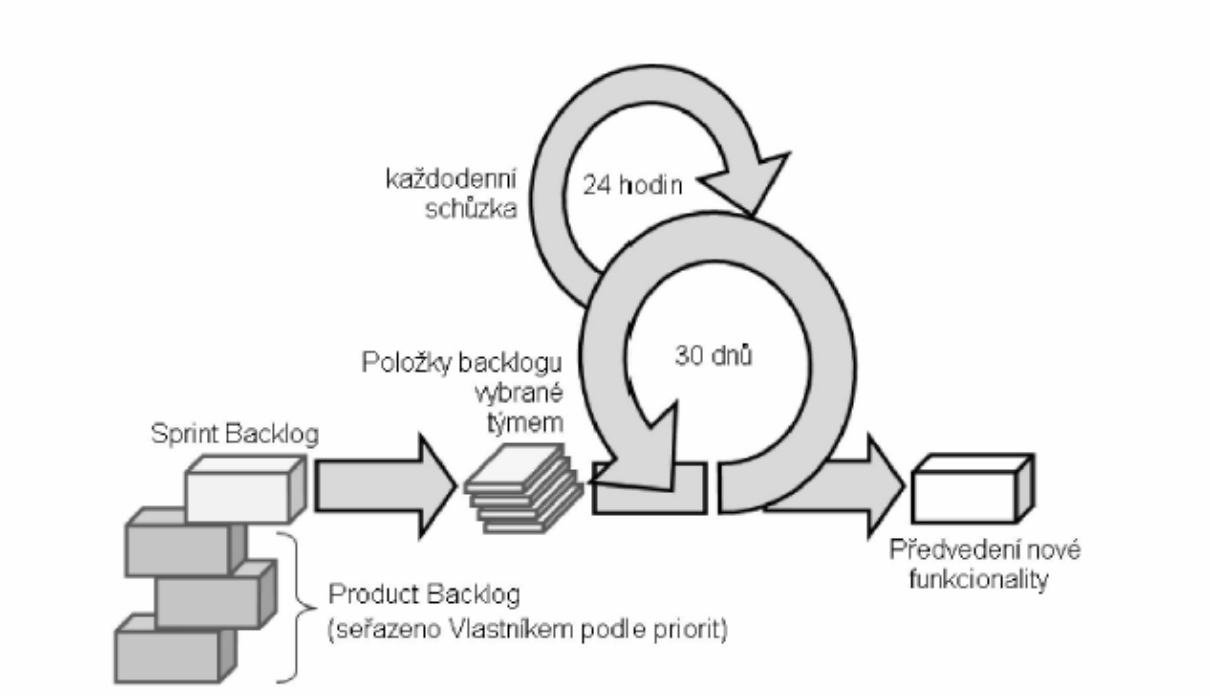
Scrum

Scrum představuje další typ iterace. Tento je mnohem kratší oproti sprintu a trvá pouze jeden den. Na počátku dne se uspořádá schůzka, která se nazývá *Scrum Meeting.* Její smysl je popsán níže. Diky takto krátkým iteracím je vývoj podrobně sledován, a vývojový tým je schopen operativně řešit problémy vznikající při vývoji.

Backlog

Metodika FDD zavádí dva druhy backlogů. Jedná se o *Produkt Backlog* a *Sprint Backlog.* Produkt Backlog je tvořen seznamem požadované funkcionality. Vlastník produktu tento seznam uspořádá. Bere přitom ohled na prioritu logické souvislosti jednotlivých částí. Poté tým část ze seznamu vybere a tu přemístí do Sprint Backlogu.

Filozofii metodiky Scrum znázorňuje obrázek, který pochází z [8].



Obrázek 6 - Schéma metodiky Scrum

3.4.3 Role

I v metodice Scrum, podobně jako například v FDD jsou určeny role jednotlivých členů týmu, podle kterých je patrné co má kdo za úkoly a za co nese zodpovědnost.

Scrum Master

-zodpovídá za dodržování zásad metodiky. Je spojovacím článkem mezi vývojáři, managementem a zákazníkem. Jeho dalším úkolem je odstraňování překážek, které brání plynulému a efektivnímu vývoji. V případě, že určitý problém je nad jeho schopnosti, musí najít v týmu člověka, který potřebnými znalostmi a schopnostmi disponuje.

Vlastník produktu

-vybírají ho Scrum Master, management a zákazník. Tato role zodpovídá za chod projektu, řídí vývoj a stará se Produkt Backlog. Ohledně Produkt Backlogu má také rozhodující slovo. Upravuje obsah Backlogu, ale ten se smí měnit pouze v období mezi

sprinty.

Scrum tým

-je tým, který se sám organizuje a jeho úkolem je splnit cíle daného sprintu. Mezi jeho úkoly také patří odhad potřebného času k implementaci určité vlastnosti a spolurozhodování o položkách Sprint Backlogu. Podílí se také na vyhledávání a odstraňování překážek, které by mohly projekt ohrozit.

Zákazník

-se na základě svých požadavků podílí na sestavování Produkt Backlogu.

Management

-rozhoduje o projektu z hlediska uzavírání smluv a podobně. Podílí se na určování požadavků a cílů projektu i jeho etap.

3.4.4 Schůzky

Metodika také zavádí několik typů schůzek, které se navzájem liší četností, délkou i obsahem.

Schůzka plánování sprintu (Sprint Planning Meeting) se koná před každým sprintem. Pro tuto schůzku je potřeba mít vypracovaný backlog, obsahující hodnoceni jednotlivých vlastností, jak z hlediska obtížnosti implementace, tak z hlediska užitku. Schůzka se může konat, jakmile backlog obsahuje alespoň tolik vlastností, aby se využil celý sprint. Pokud je jich více, vybereme ty, které jsou pro nás nejdůležitější.

Schůzka začíná tím, že vlastník produktu představí své vize, plány a aktuální stav produkt backlogu. Po kontrole údajů přebírá iniciativu tým. Musí se rozhodnout, kolik práce je reálné zvládnout. Potom se z produkt backlogu vybere objem vlastností odpovídající schopnostem týmu. Vybráním vlastností přechází schůzka do další fáze. V té scrum master rozdělí vlastnosti na jednotlivé úlohy nutné pro jejich implementaci. Z tohoto seznamu úloh vzniká sprint backlog, který je hlavním produktem schůzky.

Každodenní schůzka (Daily Scrum Meeting) Vzhledem k tomu, že se koná každý den, je časově omezena, měla by trvat zhruba 20 minut. Každý člen týmu by měl zhodnotit, na čem pracoval včera, na čem bude pracovat dnes, a jaké problémy se v jeho části práce objevily. Cílem těchto schůzek je udržovat pojem o aktuálním dění, odhalovat nové problémy a závislosti a přizpůsobovat plán aktuální situaci.

Schůzka hodnocení sprintu (Sprint Review Meeting) probíhá vždy na konci sprintu. V první fázi organizuje vlastník a jejím cílem je zhodnocení výsledků sprintu, tedy kódu a jeho funkčnosti. Vlastník produktu na základě pokroku upraví produkt backlog. V diskuzi se může upravit priorita dalších vlastností. Dále se stanoví cíle dalšího sprintu. Ve druhé fázi schůzky scrum manager s celým týmem hodnotí podmínky pro práci, a navrhují možná zlepšení. Pokud sprint, na který schůzka navazovala není poslední, následuje brzy schůzka pro plánování dalšího sprintu.

3.4.5 Procesy

Jednotlivé hlavní elementy, ze kterých se metodika Scrum skládá, byly již popsány. Můžeme tedy stručně popsat vývoj podle této metodiky. Na počátku specifikuje vlastník produktu výsledný produkt. To provede buď na základě požadavků konkrétního zákazníka, nebo podle potřeb trhu. Výstupem z této fáze je téměř kompletní produkt backlog, do kterého se doplní odhadovaná náročnost jednotlivých vlastností. Poté se provádějí sprinty, ve kterých se vlastnosti z produktu backlogu postupně implementují. Po implementování všech vlastností je produkt hotový.

3.4.6 Shrnutí

Jak již bylo řečeno ve stručném popisu metodiky, věnuje se Scrum manažerskému řízení projektu. Nehledí již na konkrétní postupy při implementaci a podobně, ale na spolupráci a komunikaci v týmu a na organizaci práce. Je vhodný spíše pro menší týmy,

typická velikost týmu je 5-10 lidi, ovšem dá se samozřejmě použít i pro větší týmy. Hlavní výhodou metodiky Scrum je pružná reakce na změny, na čemž se podílejí každodenní schůzky. Tím je zaručeno i rychlé odhalení potenciálních problémů, které mohou nastat. Jako nevýhoda by se dala vidět ta vlastnost, že Scrum nepředepisuje konkrétní nástroje pro implementaci. Je však možné a vhodné aplikovat Scrum tam, kde je již nějaká metodika zavedena. Pak jsou převzaty implementační postupy a nástroje z používané metodiky a Scrum slouží ke zlepšení a zefektivnění vedení vývoje projektu.

3.5 Test Driven Development

Vývoj každého produktu, a to nejen softwaru, vyžaduje testování. To je pro kvalitní výsledek v podstatě nezbytné. Autoři metodiky *Test Driven Development* ovšem považují testování nikoli jako jednu z fází vývoje, ale přiřazují mu největší důležitost. V pojetí vývoje podle FDD je klasický postup v podstatě obrácený, když je nejprve funkce testována a až poté implementována. Možná se to může jevit jako nesmysl, ale po bližším seznámení s metodikou musíme uznat, že její autor Kent Beck dokázal poskládat své myšlenky tak, že dávají smysl. Metodiku TDD poprvé představil v [9].

3.5.1 Charakteristika

Základní myšlenou metodiky TDD je, že před implementováním sebemenší funkcionality, je potřeba napsat test, který ji spolehlivě otestuje. Po vytvoření testu programujeme samotnou funkcionalitu, a to tak, že se snažíme splnit test a nepsat nic zbytečného navíc. Poté hotový kód testu i funkce optimalizujeme a odstraníme duplicity. Tím je hotova jedna iterace. Vzhledem ke své jednoduchosti trvají tyto iterace klidně i pouze několik minut.

Správným postupem je napsat test, který po spuštění nejprve selže. To je dáno tím, že testovaná funkcionalita není ještě implementována. Poté vytváříme funkcionalitu, která je

hotova v okamžiku, kdy projde testem. Abychom mohli jednotlivé části funkcionality kvalitně otestovat, měl by testovací kód splňovat několik podmínek:

- rychlost – jelikož nespouštíme při testování pouze jeden test, ale celou sadu, musí testy probíhat jednoduše, nemůžeme při iteracích trvajících několik minut čekat na výsledek testování dlouhou dobu
- jednoduchost – test se musí zaměřovat na konkrétní testovanou funkcionalitu, pokud by tato byla příliš složitá, je vhodné rozdělit ji do několika menších částí
- kvalita – kódu testu musíme věnovat stejnou pozornost jako samotnému funkčnímu kódu, není to něco, co napíšeme narychlo a provizorně, pro kvalitní testování je zapotřebí kvalitních testů, proto také v poslední fázi iterace zefektivňujeme a zpřehledňujeme nejen kód funkce, ale také testovací část kódu
- automatizace – jelikož test se nespouští samostatně, ale jako součást balíku testů, musí probíhat plně samostatně, není možné, aby test v průběhu běhu vyžadoval interakci od uživatele
- nezávislost – jednotlivé testy musí být navzájem nezávislé, nezávislé na prostředí a nezávislé na pořadí, ve kterém jsou spouštěny

3.5.2 Procesy

Jednotlivé iterace, jak je chápe metodika TDD se skládají z následujících kroků:

1. tvorba testu, který však po spuštění selže vlivem neimplementované funkcionality
2. spuštění ostatních testů, tím se přesvědčíme, že testy navzájem nekolidují a nově vytvořený test z bodu č.1 opravdu selže
3. tvorba funkcionality, ta musí po dokončení projít testem z bodu č. 1
4. spuštění všech testů a případná úprava kódu tak, aby všechny uspěly
5. úprava testovacího a funkčního kódu tak, abychom odstranili duplicity a přesun

testovací části do sady automaticky spouštěných testů

Z těchto fází se tedy každá, relativně krátká (často i několikaminutová), iterace skládá. Je vidět, že mezi těmito fázemi není žádná, která by se věnovala tvorbě seznamu požadovaných funkcionalit. Metodika TDD se věnuje pouze tvorbě samotného kódu.

Sám autor metodiky stanovil dvě pravidla využívaná v metodice TDD:

1. nový produkční kód píšeme pouze po selhání testu
2. všechny vzniklé duplicity musíme odstraňovat

Dále definoval pro využití metodiky několik dalších zásad. Tyto zásady nám mají pomoci při vývoji podle TDD:

- Dostatečná rychlost vývojového prostředí, tak aby poskytovalo patřičnou odezvu na změny. Při krátkých iteracích není možné čekat neúměrně dlouho na provedení všech testů, což by vývoj prodlužovalo.
- Testovací kód píší sami autoři funkčního kódu. Ti sami nejlépe vědí, jak danou funkcionalitu kvalitně otestovat. Zároveň by také časté čekání na to, než někdo další vytvoří testovací kód, nevyužívalo efektivně čas programátora.
- Je vhodné mít kvalitně navrženou architekturu softwaru, ideálně jednotlivé nezávislé komponenty, propojené přes vhodný middleware. To usnadní nejen testování, ale také údržbu a rozšiřitelnost výsledné aplikace.

Testy jsou stěžejní součástí této metodiky. Pro jejich maximální kvalitu a efektivitu bylo navrženo několik pravidel. Správný test dle metodiky FDD měl být:

- rychlý, bez nutnosti složitého konfigurování
- vstup i výstup testu by měl snadno ověřitelný
- používat pokud možno reálná data

- musí běžet i samostatně, popřípadě běžet bez ohledu na další testy a jejich pořadí
- mělo by z něj být patrné, jakou funkcionalitu testuje a co jeho splnění projektu přinese

3.5.3 Praktiky

Důležitou praktikou je udržování tzv. t*o-do seznamu.* V něm jsou uvedeny testy, které čekají na napsání a také úpravy které je třeba v rámci refaktorizace udělat. Tento seznam nám poskytuje informace o postupu, plánech a tom, co je již splněno. Je tedy nutné ho udržovat pokud možno aktuální. To-do seznam nám zabraňuje vzniku totálního chaosu a dává vývoji řád.

V případě nasazeni TDD je nutné vyvarovat se opuštění principů TDD ve prospěch tradičních zvyklostí. To může být, zvláště v počátcích nasazení metodiky, obtížné. Programátor pracující podle této metodiky nesmí například psát více testů najednou, zapomenout odstraňovat duplicity nebo ponechat neotestovaný kód v aplikaci.

Testování, které je jádrem celé metodiky, by mělo probíhat automatizovaně. Při větších projektech by byla ruční správa testů složitá až téměř nemožná, proto existují aplikace pro podporu automatizovaného testování . Mezi nejznámější skupinu těchto aplikací patří nástroje xUnit. Existuje jich několik desítek pro různé programovací jazyky.

Metodika TDD na první neobsahuje explicitně definovanou fázi návrhu. To však není tak zcela pravda, jelikož podle jejích autorů je návrh přítomen v každé iterace právě díky testům. Před psaním testu si totiž musí autor rozmyslet, jak bude daný modul pracovat a jak ho tedy bude možné testovat. V této fázi tedy probíhá návrh jednotlivých částí systému

3.5.4 Závěr

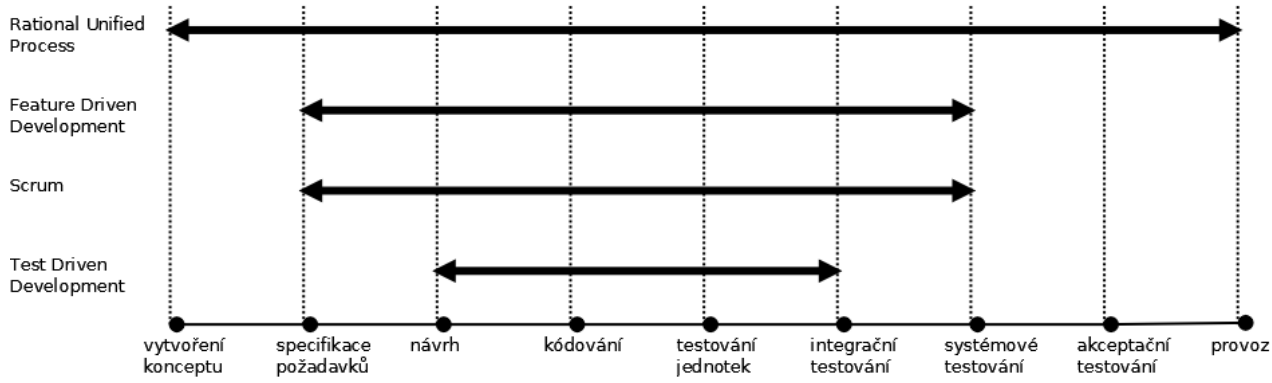
Po stručném prostudování metodiky vidíme, že to, co se mohlo jevit jako podivné až absurdní, tedy psaní testů dříve než kódu, dává smysl. Metodika TDD nepřidá k projektu

jediný kus kódu, který by nebyl otestován. Z toho plyne také to, že metodika neobsahuje fázi odstraňování chyb. Ty by totiž při správném postupu vůbec neměly vzniknout.

Metodika TDD se zaměřuje pouze na tvorbu kódu, a neřeší administrativní a manažerské otázky. Z toho důvodu si myslím, že její použití samostatně může být poněkud problematické. Jako vhodné se mi jeví použití TDD v rámci nějaké další metodiky, která ji poskytne procesní řízení a manažerské zázemí.

4. Porovnání metodik a jejich vhodnost pro určité druhy projektů

Všechny zmíněné metodiky je samozřejmě možné využít k řízení vývoje softwarového produktu. Každá z nich se však z hlediska komplexnosti přístupu poněkud liší. Metodiky se každá zaměřují na určitou část vývoje a přinášejí na něj svůj vlastní pohled. Metodika *Feature Driven Development* považuje za zásadní *vlastnosti* produktu, které postupně implementuje. S*crum* se více zaměřuje na otázky řízení a managementu projektu a *Test Driven Development* se věnuje pouze zdrojovému kódu a nezabývá se příliš širšími souvislostmi. Tyto vlastnosti plynou mimo jiné z toho, že autoři vytvářeli tyto metodiky hlavně pro sebe, resp. pro společnosti, kde pracovali. Odrážejí tedy do určité míry to, kde sami autoři cítili slabiny své společnosti a kde bylo vhodné zavést určitá zlepšení, která realizovali v podobě těchto metodik. Jelikož se tedy metodiky nevěnují kompletnímu vývojovému procesu, je jejich objektivní porovnání obtížné. Na obrázku je graficky znázorněno, kterým částem životního cyklu projektu se která metodika věnuje. Pro srovnání je kromě tří popsaných agilních metodik i jedna z oblasti tradičních a to *RUP*. Je vidět že metodika RUP je opravdu robustní a komplexně pokrývá všechny fáze vývojového cyklu.



Obrázek 7 - porovnání metodik z hlediska pokrytí fází vývoje

Oproti tomu popsané agilní metodiky očekávají, že koncepce projektu je již stanovená

a nikterak se touto fází nezabývají. První, fáze kterým se metodiky FDD a Scrum věnují, jsou specifikace požadavků a návrh systému. Metodika FDD předepisuje v této fázi schůzku se zákazníkem, kde se vypracuje celkový model systému a stanoví se počáteční seznam vlastností. Podle metodiky Scrum se v této fázi vytváří Produkt Backlog a model systému, který je ve vysokém stupni abstrakce. Metodika TDD se návrhem v podstatě nezabývá, i když podle určitého pohledu je návrh obsažen v každé iteraci. Osobně si však myslím, že fáze návrhu systému, alespoň v obecnějších rysech, by u složitého systému neměla chybět.

Následující fázi vývoje, tedy samotnému kódování se již věnují všechny metodiky, byť každá s poněkud odlišným přístupem. FDD využívá seznam požadovaných vlastností, seřazených podle priority, které se s iteracích postupně implementují, za což zodpovídají hlavní programátoři. Metodika však neurčuje jakými konkrétními programátorskými postupy a za využití jakých nástrojů má implementace probíhat. To nechá na libovůli vývojového týmu. Testováni je v metodice FDD obsaženo v poslední fázi: Implementaci podle vlastností. Metodika Scrum má nad projektem jen organizační kontrolu, konkrétní postupy a nástroje také nechává volně v rukou vývojového týmu. Po kódování probíhá i testování jednotek, to se děje v rámci každého sprintu. Zpětná vazba se získává na každodenních schůzkách, a také na hodnotící schůzce celého sprintu. U metodiky TDD probíhá testování průběžně s psaním kódu, nebo přesněji ještě před jeho psaním. Vývoj probíhá v krátkých iterací po velice malých částech. Dalšími fázemi životního cyklu produktu, akceptačními testy a provozem, se již tyto metodiky příliš nezabývají. Určují v jaké fázi vývoje je produkt připraven ke spuštění, ale konkrétní kroky a nástroje neřeší.

Pro přehlednější porovnání některých parametrů a vlastností jsem vytvořil následující tabulku:

	FDD	Scrum	TDD
Nároky na zkušenosti vývojového týmu	střední	velké	velké
Vlastnictví kódu	Vlastníci tříd	Nespecifikováno	Autoři jednotlivých funkcionalit
Délka iterace	Cca 2 týdny	2-4 týdny	Několik minut
Počet členů týmu	Od několika členů po desítky	Obvykle 5-10	Podle rozsahu projektu

Z ní je vidět, že tyto metodiky jsou povětšinou poměrně náročné na zkušenosti, jelikož přinášejí oproti tradičním metodikách poněkud odlišný přístup k vývoji. Jejich zavedení tedy může být u týmu, který s danou metodikou nemá mnoho zkušeností, komplikované. Například u metodiky TDD je třeba důsledně dodržovat výše popsaná pravidla, mimo jiné psát testy před samotným psaním kódu. To je natolik neobvyklé, že je snadné sklouznout k neplnění podmínek metodiky, nebo tyto podmínky plnit jen formálně, například dodělávat testy zpětně a podobně.

Parametr délky iterace nám v podstatě mnoho o metodikách neřekne. U TDD je oproti dalším porovnávaným metodikám extrémně krátký, je to však dáno tím, že v iteraci se implementuje jen malá část kódu a metodika neřeší iterace vývoje z hlediska řízení.

Podle doporučeného počtu členů týmu je vidět, že metodika FDD je z tohoto hlediska poměrně flexibilní a hodí se tedy na různé velikosti projektů. Pro malé projekty, kterých se nezúčastní tolik vývojářů, se pak definované role spojují a jedna osoba plní několik funkcí. Metodika Scrum je svým rozsahem vhodná spíše pro menší projekty, i když ani to nemusí být při vhodné implementaci pravidlo. Porovnání metodiky TDD z tohoto hlediska je poněkud obtížné. Metodika popisuje jen samotnou tvorbu kódu, a komunikaci a řízení vývojového

týmu neřeší. Pokud by byla nasazena samostatně, muselo by se jednat o malé projekty. V případě větších a komplikovanějších si neumím představit její samostatné nasazení právě z důvodu toho že se věnuje pouze kódu. Je však použitelná v kombinaci s jinými metodikami.

Mezi další vlastnosti, podle kterých by se daly metodiky hodnotit, může být například jejich vhodnosti pro obtížné a rizikové projekty. Zde si subjektivně myslím, že metodika Scrum je o něco vhodnější než FDD, a to hlavně proto, že se z více věnuje řízení vývoje a častým schůzkám, kde se mohou odhalit různé chyby a nesrovnalosti. FDD má však výhodu v tom, že je přesně určena role a funkce každého člena a odpovědnost za kód. Hodnotit z tohoto hlediska metodiku TDD je obtížné. Testování je v ní implicitně obsaženo a je mu věnována velká pozornost, avšak to probíhá na úrovní jednotlivých částí funkcionality, a metodika neřeší komplexní přístup k vývoji systému.

Všechny tyto metodiky mají své výhody a nevýhody. Při jejich zavedení do praxe je třeba zvážit několik faktorů – například typ projektu, velikost týmu nebo jeho zkušenosti. Nejkomplexnější z nich je FDD, v podstatě stejné části vývojového procesu pokrývá také metodika Scrum, avšak z poněkud jiného pohledu. Srovnání s metodikou TDD je obtížnější, ta postihuje jen malou část vývojového procesu, a bylo by vhodné využít ji v rámci jiné metodiky, která by zajišťovala řízení projektu. To by bylo možné i se zde popsanou metodikou Scrum, nebo s některou jinou metodikou, která nechává při výběru nástrojů a postupů pro tvorbu kódu volnost výběru.

5. Návrh aplikace realizující zvolenou metodiku

V následující kapitole bude navržen informační systém realizující vývoj podle zvolené metodiky. Vybral jsem si metodiku Test driven development, hlavně z důvodu její přehlednosti, ale zároveň dostatečného rozsahu.

Celý systém by se realizoval jako webová aplikace. To zajišťuje použitelnost z jakéhokoliv místa s dostupným připojením k internetu. Data budou uchovávána v databázi na serveru, čímž bude zajištěna jejich aktuálnost. V systému se budou evidovat jednotlivé projekty a jejich detaily. U každého projektu budou uvedeny kromě dalších údajů také vlastnosti, tedy části funkcionality, které je třeba implementovat. Zaměstnanci firmy budou tvořit týmy, které budou spolupracovat na jednotlivých projektech.

Systém bude vývojový tým provádět jednotlivými fázemi metodiky FDD a kontrolovat zda všechny předepsané kroky, nutné k dokončení dané fáze, jsou splněny. Nabídne také stručný popis jednotlivých fází vývoje.

5.1 Základní objekty

Lidé

Sem patří všichni lidé, kteří nějakým způsobem vystupují v rámci vývojového procesu. Jedná se tedy například o všechny členy firmy, kteří spolupracují na dané zakázce. U každého člověka se budou do databáze ukládat:

- informace o dané osobě (ID, jméno, příjmení)
- informace umožňující kontaktovat danou osobu (telefon, e-mail)
- informace pro přihlašování do systému (login a heslo).

Podle atributu *role* se určuje, jaká práva má konkrétní uživatel v systému.

Projekty

To jsou všechny jednotlivé projekty, každá zakázka je obvykle tvořena jedním projektem. U každého projetu se evidují:

- základní informace o projektu(ID, ID projekt manažera, datum vytvoření, datum dokončení)
- milníky (datumy kontrolních bodů vývoje)
- týmy, které na daném projektu spolupracují

Týmy

Pomocí entity *Tým* jsou spojeny *Lidé* a *Projekty*. Jednotlivé týmy se skládají z členů, a věnují se určitému projektu. Týmu jsou pak přidělovány k implementaci jednotlivé vlastnosti, které daný systém bude obsahovat. Jeden člověk pak může být ve více týmech současně. U týmu jsou uležené následující informace:

- ID projektu, ke kterému je tým vytvořen
- ID vedoucího týmu
- ID osob, které jsou členy týmu

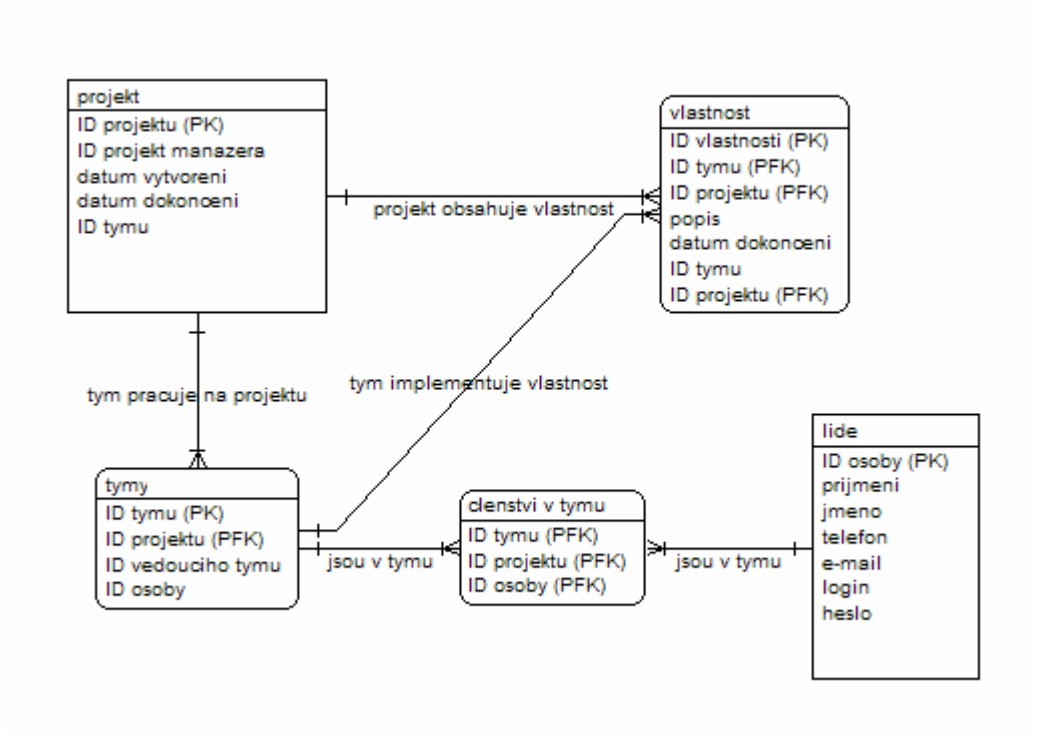
Vlastnosti (features)

Vlastnosti jsou v pojetí metodiky FDD jednotlivé části funkcionality, které budou v rámci daného projektu implementovány. Každá vlastnost obsahuje:

- ID vlastnosti
- ID projektu, ke kterému vlastnost patří
- Popis vlastnosti
- Stupeň dokončení
- Datum ukončení

- ID týmu, který danou vlastnost bude implementovat

Pro lepší přehlednost je propojení jednotných entit zobrazeno na obrázku číslo 10.



Obrázek 8 - ER diagram

5.2 Seznam vlastností

Zde bude uveden základní seznam vlastností, kterými by měl systém disponovat, tak aby mohl sloužit svému účelu:

- Uživatel s rolí *Administrátor* může zakládat a upravovat projekty, dále spravuje databázi zaměstnanců
- Uživatel s rolí *Projekt manažera* sestavuje týmy, zadává do systému vlastnosti určené k implementování, přiděluje jednotlivé vlastnosti menším týmům

programátorů, může editovat vlastnosti projektu

- Uživatel s rolí *Hlavního programátora* vyplňuje stupeň dokončení jednotlivé vlastnosti přidělené k implementaci
- Všichni uživatelé zaregistrovaní v systému mohou měnit své osobní informace (e-mail, telefon…)

Dále jsou na systém kladeny nároky v podobě výstupů. Ty slouží k informovaní managementu, tvorbě statistik a plánování dalšího rozvoje firmy:

- Přehled projektů – vyvíjených i dokončených
- Vlastnosti projektů – jaké týmy se jich účastní, jaké jsou důležité datumy související s projektem, stav jeho dokončení
- Přehledy o vytíženosti zaměstnanců jejich zapojením do týmů
- Statistiky produktivity vývoje

5.3 Možnosti rozšíření aplikace

Zde navržená aplikace by mohla po realizaci poskytovat podporu vývojovému týmu, postupujícímu podle metodiky FDD. Je však samozřejmé, že by se dala v mnoha směrech rozšířit. Bylo by například vhodné specifikovat, co bude pro konkrétního uživatele po přihlášení do systému nejdůležitější a co bude vidět na úvodní stránce webové aplikace. Dále by bylo vhodné implementovat například automatické posílání emailu, pokud by došlo pro určitou osobu k relevantní změně. Pro případ zaměstnanců s omezeným časovým fondem (poloviční úvazek) by bylo vhodné tento fakt zohlednit při sestavování týmů.

6. Závěr

Hlavním cílem práce bylo popsat obecně agilní metodiky řízení vývoje softwaru a dále se věnovat několika z nich podrobněji. To bylo splněno v kapitole 3. V té jsem se, kromě popisu důvodu vzniku a obecných vlastností agilních metodik, věnoval metodikám *Feature driven development*, *Scrum* a *Test driven development.* Pro porovnání a pochopení rozdílu mezi agilními a tradičními metodikami byly v kapitole 2 popsány některé tradiční metodiky. Z nich jsem zvolil *Vodopádový model*, *Spirálový model* a *RUP*. Je tedy možno porovnat rozdíl v přístupu k vývoji softwaru mezi těmito metodikami.

V kapitole 4 jsem porovnával výše popsané agilní metodiky. Zjistil jsem však, že jelikož jsou agilní metodiky víceméně pouze souborem pravidel, postupů a doporučení, bude jejich objektivní porovnání obtížné. Navíc se každá z nich věnuje jen části vývojového procesu, a to ještě povětšinou z vlastního úhlu pohledu. To je velmi dobře vidět u metodiky *Test driven development*, která se s dalšími těžko porovnává. I přesto jsem je podle několika vlastností porovnal a jako základní srovnání by tato práce sloužit mohla. Zkoumání vhodnosti dané metodiky pro konkrétní účel by však vyžadovalo detailnější pohled, ve kterém by se zohlednily například i vlastnosti vývojového týmu, který by měl o metodiku zájem. Z důvodu rozdílného pohledu na vývoj by i srovnání metodik na ukázkovém projektu, které nebylo v zadání, bylo poněkud obtížné.

Problémem, se kterým jsem se musel potýkat, byl nedostatek dostupné literatury. To je dané jednak tím že tyto metodiky jsou poměrně nové a také je to celkem specifický obor softwarového inženýrství. Dostupné jsou však články na internetu, různé blogy a podobně. Autoři jednotlivých metodik zveřejnili povětšinou svoji metodiku v knižní publikaci, ale tyto knihy jsou poněkud obtížně dostupné, a většinou se věnují jen jedné metodice a chybí jim vzájemné srovnání. Pozitivní je však fakt, že autoři svůj přínos netají a své myšlenky volně zpřístupňují. Publikují na internetu v článcích a poměrně rozsáhlé informace lze nalézt i na encyklopedii Wikipedia, která uvádí i zdroje informací.

Vývoj agilních metodik je podle mého názoru velmi perspektivní. Během relativně krátké doby své existence ukázaly, že mohou být velmi prospěšné. Je pravděpodobné, že

budou dále vznikat další agilní metodiky, které budou odstraňovat negativní vlastnosti, které s sebou některé přinášejí. To se z části děje a bude dít i kombinováním již existujících metodik. Je však nutné podotknout, že agilní metodiky nejsou všemocné. Jejich nasazení je nutné pečlivě zvážit tak, aby se maximalizoval jejich užitek.

Seznam použité literatury

[1] RICHTA, K; SOCHOR, J. *Softwarové inženýrství I.*. Praha : ČVUT, 1996.

[2] WARFEL, T. Z. *Prototyping: A Practitioner*. New York, USA : Rosenfeld Media, 2009. 197 s. ISBN 978-1933820217.

[3] BOEHM, B. *Spiral development: Experience, principles and refinements*, Carnegie Mellon University, Software Engineering Institute, 2000.

[4] KRÁL, J. *Informační systémy.* Science. 1998.

[5] KRUCHTEN, P. *The Rational Unified Process: An Introduction*. Boston, USA : Addison Wesley, 2000. 320 s. ISBN 978-0201707106.

[6] COAD, P.; LEFEBVRE, E.; DE LUCA, J. *Java Modeling In Color With UML: Enterprise Components and Process* . Upper Saddle River : Prentice Hall, 1999. 221 s.

[7] PALMER, S.R.; FELSING, J.M. *A Practical Guide to Feature-Driven Development*. Upper Saddle River : Prentice Hall, 2002. 304 s. ISBN 978-0130676153.

[8] SCHWABER, K.; BEEDLE, M. *Agile Software Development with Scrum*. Upper Saddle River : Prentice Hall, 2002. 158 s. ISBN 978-0130676344.

[9] BECK, K. *Test Driven Development: By Example*. Boston, USA : Addison-Wesley, 2002. 240 s. ISBN 978-0321146533.

[10] BECK, Kent , et al. *Manifesto for Agile Software Development* [online]. 2001 [cit. 2011-03-18]. Dostupné z WWW: <http://agilemanifesto.org/>.

[11] *Agile Alliance* [online]. 2011 [cit. 2011-03-11]. Dostupné z WWW: <http://www.agilealliance.org/>.

[12] *Scrum Alliance* [online]. 2011 [cit. 2011-04-03]. Dostupné z WWW: <http://www.scrumalliance.org>.

[13] WINSTON, Dr; ROYCE, W. *Managing the development of large software systems* [online]. 2003 [cit. 2011-01-19]. Dostupné z WWW: <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>.

[14] *Interscience Publishers* [online]. 2004 [cit. 2011-03-25]. Dostupné z WWW: <http://www.inderscience.com/>.

[15] *IBM Rational Software* [online]. 2011 [cit. 2011-05-03]. Dostupné z WWW: <http://www-01.ibm.com/software/rational/#>.

[16] KADLEC, V. *Agilní programování - Metodiky efektivního vývoje softwaru*. Brno : Computer Press, 2004. 280 s. ISBN 80-251-0342-0.

[17] *Feature Driven Development - The portal for all things FDD* [online]. 2011 [cit. 2011-03-08]. Dostupné z WWW: <http://www.featuredrivendevelopment.com/>.