VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

APLIKACE AGILNÍ METODIKY V PODNIKU

APPLICATION OF AGILE METODOLOGY IN A COMPANY

BAKALÁŘSKÁ PRÁCE

BACHELOR THESIS

AUTOR PRÁCE
AUTHOR
Bc. JAROSLAV BEDNAŘÍK

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. KAREL DOUBRAVSKÝ, Ph.D.

BRNO 2013

ABSTRAKT

Tato bakalářská práce se věnuje agilnímu projektovému řízení, konkrétně metodice Scrum. Teoretická část popisuje jednotlivé aktivní i pasivní části této metodiky a jejich přínos při řízení projektu. Praktická část prezentuje využití Scrum metody v praxi na reálném softwarovém projektu, zadaným jazykovou školou.

ABSTRACT

This bachelor thesis is aimed at agile project management, primarily at using Scrum as one of the agile practices. The theoretical part describes each and every active and passive part of Scrum during project management. The practical part describes concrete application of Scrum practices on software project for a language school, in real environment.

KLÍČOVÁ SLOVA

Projektový management, agilní vývoj software, Scrum, iterativní proces, vývoj software, plánování náročnosti práce

KEYWORDS

Project Management, Agile Software Development, Scrum, Iterative process, Software Development, Estimation

BIBLIOGRAFICKÉ CITACE

BEDNAŘÍK, J. Aplikace agilní metodiky v podniku. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2013. 62 s. Vedoucí bakalářské práce Ing. Karel Doubravský, Ph.D..

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 20. května 2013

……..

Bc. Jaroslav Bednařík

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat vedoucímu práce, panu Ing. Karlu Doubravskému, Ph.D., za ochotu spolupracovat při vzniku této bakalářské práci.

OBSAH

ÚVOD

Projektový management je nedílnou součástí každého projektu. Každý projekt vyžaduje dobrou strukturu a plán. Softwarové projekty nejsou výjimkou. Existují dva základní modely, kterými jsou tzv. Waterfall Process Model (sestupný model) a Agile Model (agilní model). Koncept sestupného modelu vyžaduje znalost kompletního seznamu požadavků na software, která musí být hotová již v přípravné fázi (nebo obecně před začátkem realizační fáze), a během vývoje by se neměla měnit – každá změna vyžaduje zpracování přípravné fáze. Agilní přístup vznikl, aby bylo možné reagovat na externí nebo interní změny podmínek nebo požadavků během realizační fáze projektu. Tento přístup se vyplácí především v prostředí, ve kterém se podmínky mění velmi často. Takovým příkladem je vývoj software, kde nejen softwaroví projektoví manažeři mohou využít nabídnuté flexibility, ale především klienti mohou specifikovat své požadavky na základě jejich aktuálních požadavků.

V posledních několika letech využívání tradičních rigorózních metodik vývoje softwaru začíná být na ústupu a upřednostňují se agilní metody vývoje a řízení projektů nejen softwarových.

Agilní přístup není pro každého. Podniky s ročním plánováním rozpočtu určitě budou patřit do této kategorie. Agilní přístup vyžaduje „inkrementální“ financování. Agilní přístup vyžaduje otevřenou kulturu v podniku, změnu přístupu od managementu a změnu vedení lidských zdrojů. V agilních praktikách neexistuje delegování práce mezi jednotlivce – v tomto smyslu se stal z jedince tým, a už není možné přesouvat jednotlivce mezi různými projekty stejně jednoduše, jako tomu bylo u klasického nebo tzv. Plan-Driven projektů.

CÍLE PRÁCE, METODY A POSTUPY ZPRACOVÁNÍ

CÍLE PRÁCE

Práce si klade za cíl prezentovat přednosti řízení softwarového projektu agilní metodou scrum, která je v této práci také podrobněji popsána. Popisuje přednosti agilních metodik a jejich úskalí, se kterými se potýkají manažeři softwarových projektů. Značná část návrhu vlastního řešení je v elektronické podobě na přiloženém CD.

Okrajově také popisuje studie organizace Standish, která publikuje statistické informace softwarových projektů. Tyto studie byly jedním z prvních impulzů ke změně v řízení softwarových projektů. Jedna z kapitol se věnuje problematice plánování a odhadování časové náročnosti. V neposlední řade je v teoretické části popsán Agilní manifest (tj. vznik zásad agilního vývoje software a postoje jeho tvůrců), včetně stanovených principů.

Praktická část byla napsána paralelně s reálným vývojem softwarového produktu. Postupy řízení vývoje a plánování zadaného projektu, které byly aplikovány během vývoje, korespondují s agilními praktikami metody scrum, které jsou popsány v teoretické části této práce.

1 TEORETICKÁ VÝCHODISKA PRÁCE

1.1 Význam projektového managementu

Projektový manažeři neměli nikdy v minulosti těžší práci, jako mají dnes. Podniky a prostředí samotné se dynamicky vyvíjí – reorganizují se, mění se jejich struktura s jedním cílem: konkurenceschopnost. Podniky se soustředí, aby dodržely konvence posledních standardů nebo požadavků na certifikace a byly konkurenceschopní. Konkurenceschopnost je jedním z klíčových cílů podniků a pouze flexibilní podniky přežijí. Tyto podmínky podnikání vyžadují účinnější a efektivnější management projektů. Základy dnešní pracovní síly se neustále mění všech společnostech. Smlouvy o pracovním poměru se zaměstnanci se již neuzavírají na dobu neurčitou – většina společností si nepřeje mít závazky na delší dobu než je nezbytné. Mnoho projektů spojuje lidi z různými zkušenostmi a zázemím. Tlak na dnešní projektové manažery je takový, že je nad jejich možnosti neustále zpracovávat obrovské množství informací, které se k nim dostávají.

S evolucí ekonomiky projektový manažer potřebuje nacházet nové možnosti získání vědomostí a využívat externích zdrojů, jako například konzultanty nebo outsourcing. Zajisté se v minulosti setkávali s podobnými problémy, ale nikdy v takovém měřítku jako v dnešní době.

Trh také komplikuje management projektů. Zákazníci nejen že chtějí kvalitní produkt – chtějí ho dříve a levněji. Tržní časová závislost nutí projektové manažery, aby byli efektivní a zdatní; jsou tlačeni do nemožných termínů nebo do rozsáhlých projektů bez kvalitní (časově náročné) přípravné fáze. Komplexnost správy projektů nikdy nebyla tak markantní jako v dnešní době. Všechny zmíněné faktory negativně ovlivňují ukazatele jako např. risk of failure nebo return on investment. Stává se naprosto kritické, aby byly dodány části projektu včas, protože mohou být prerekvizitou projektu dalšího.

Projekt management nabízí strukturovaný přístup v řízení projektů. Poslední techniky plánování a kontroly používané při plánování projektů často odkazují na Project Management Insutute (PMI) Projekt Managemnet Body of Knowledge (PMBOK) nebo na Association for Project Management (APM) Body of Knowledge (APMBOK). Od založení PMI v roce 1969 je tato organizace jedím ze základních informačních kanálů

pro kvalitní vedení projektů v každé firmě. PMIBOK, jehož tvůrcem je PMI, je často uváděn mezi metodikami, přestože metodikou v pravém slova smyslu není. Klade důraz především na mechanismy v souvislosti s řízením projektů a na popis jednotlivých znalostních oblastí.

Řízení projektu je definované v PMBOK jako:

„…aplikování vědomostí, dovedností, nástrojů a technik do projektových aktivit za účelem dodržení požadavků stakeholderů a očekávání od projektu.“ (1)

Toto jasně identifikuje, že cílem projektu je splnit očekávání stakeholderů. Projektovému manažerovi musí být jasné, kdo jsou stakeholdeři (mimo klienta), a analyzoval jejich potřeby a očekávání.

1.2 Definice projektu

Význam slova projekt se ustálil ve smyslu námět, návrh, plán a komplexní vyřešení zamýšleného úkolu i vypracování jeho náležitostí včetně grafického znázornění. Projekt je cílevědomý návrh na uskutečnění určité inovace v daných termínech zahájení a ukončení.

Přestože definicí slova projekt existuje několik od uznávaných autorů, každá trochu jinak formulovaná, z definice je patrné několik důležitých bodů:

- Projekt sleduje konkrétní cíl,
- definuje strategii vedoucí k dosažení daného cíle,
- určuje nezbytně nutné zdroje a náklady včetně očekávaných přínosů z realizace záměru,
- vymezuje jeho začátek a prostor (konec).

Periodicita je projektu značí, že se nejedná o projekt, protože projekt je vždy jedinečný a dočasný. Projekty se liší velikostí, rozsahem cenou a časem od mezinárodních projektů s rozpočtem v rámci milionů dolarů a časovým rámcem mnoho let, po malé projekty s malým rozpočtem a několikahodinovým časovým rámcem pro splnění.

(2 p. 22)

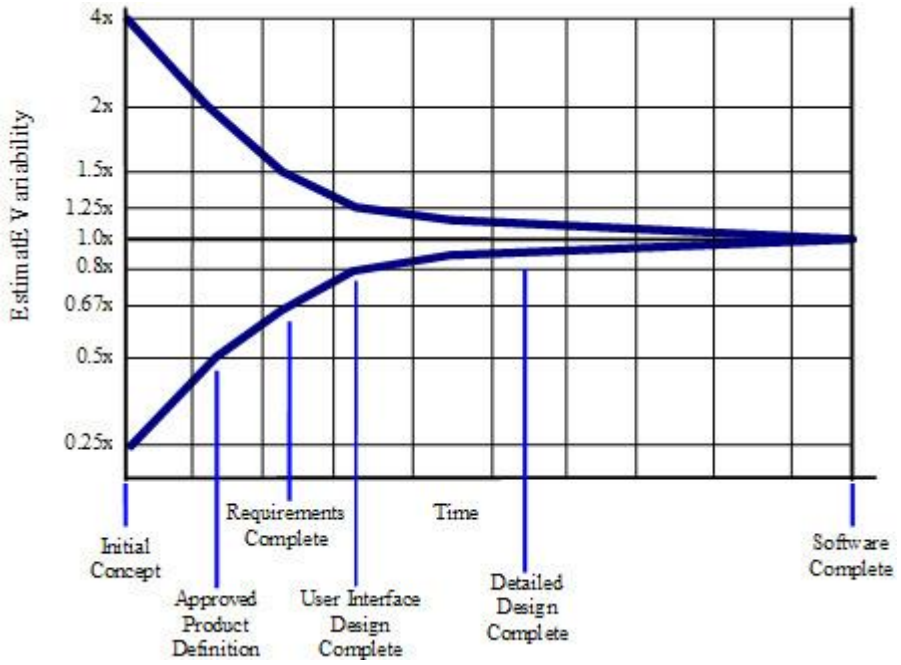
Projektu mohou být velice různorodé:

- Design a konstrukce stavby, a domu nebo jachty,
- návrh a testování nového prototypu (auta, vařiče…),
- vypuštění nového produktu na trh (reklamní a marketingový projekt),
- implementace nového počítačového systému nebo architektury (IT projekt nebo vylepšení),
- návrh a implementace nové organizační struktury v organizaci (HR projet),
- plánování a realizace auditu (quality management project),
- zlepšení produktivity během určitého časového rámce,
- světové turné hudební skupiny.

1.3 Odhadování časové náročnosti v klasickém pojetí

Odhadování časové náročnosti je rigorózní proces. K přesnému odhadu je zapotřebí mnoho zkušeností a let praxe. Detailně popsaný proces je kvalitně popsán v knize Agile Estimating and Planning. Plánování je kritická část každého projektu. V roce 1981 Berry Boehm vytvořil první verzi grafu, který je v dnešní době znám jako *Cone of Uncertainity* (**Error! Reference source not found.**). Graf ukazuje, že během počátečních fází projektu (koncept, schválení definice požadavků) se přesnost odhadu pohybuje mezi 60% - 160% (tj. projekt, který je odhadován na 20 týdnů, se skutečné dokončení projektu pohybuje někde mezi 12 až 32 týdny. Po sepsání oficiální specifikace požadavků se odchylka pohybuje kolem +/- 15%. U projektu, který má rozpočet $100 000, to dělá v první fázi odchylku $60 000. První fáze bývá většinou pro zadavatele nebo investora rozhodující, zda bude projekt chtít uskutečnit, či nikoliv. Je důležité, aby odchylka od finální hodnoty byla tedy co nejmenší.

(3)



Obr. 1 Cone of Uncertainity (zdroj http://www.construx.com/uploadedImages/Construx/Construx_Content/Resources/Books/Cone01.jpg)

1.3.1 Metody využívané při odhadování časové náročnosti

Metody PERT a CPM vyžadují znalost detailní struktury projektu a jeho jednotlivých mezníků. Je také třeba znát, jak jsou na sebe jednotlivé menší úkoly závislé. Je také nutné znát, jaké zdroje budou dostupné v jednotlivých částech úkolu. Jedna z možností jak definovat strukturu většího úkolu je např. work breakdown structure (WBS). WBS představuje hierarchickou dekompozici projektu nebo komplikovanějšího cíle do menších, po sobě následujících úkolů, uspořádaných do úrovní. Každá úroveň je označena zvyšujícím se číslem (tj. 10, 20, 30, 40, 50). Jestliže existuje podúroveň, je zapsána oddělovačem (např. tečkou) za vlastním předchůdcem (tj. 10.1, 10.2, 10.3…).

PERT i CPM jsou reprezentovány v podobě jednosměrných sítových grafů. Dalším základním nástrojem k plánování projektu jsou Ganntovy diagramy.

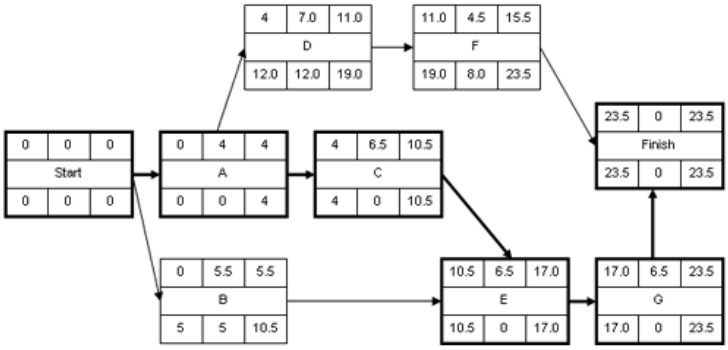
(4 pp. 6-7)

Poslední tři desetiletí softwaroví manažeři využívali, a stále využívají, tyto diagramy. Ve své praxi jsem se setkal s diagramy na plátně velkém 2m x 4m, na kterém byly vyznačeny všechny úkoly, závislosti a jejich propojení.

Je zřejmé, že Informační hodnota u metod PERT a CPM se stává méně přesná s rostoucím počtem změn, které se vyskytnou během realizační fáze projektu.

PERT

PERT představuje techniku odhadování a plánování větších projektů. Kategorizuje se jako technika pravděpodobnostní. Metoda PERT využívá statistické výpočty ke zjištění pravděpodobnosti dokončení projektových mezníků k určitému termínu. Výsledkem PERT jsou tři výsledné hodnoty: nejlepší, nominální a nejhorší hodnota. Tyto výsledky jsou proměnné, které lze využít ke zjištění standardní odchylky a očekávané doby trvání realizační fáze projektu. Podmínkou využití této metody je znalost časové náročnosti jednotlivých drobnějších úkolů a jejich možné odchylky, u kterých se dělá optimistický, nominální a pesimistický časový odhad. PERT je využíván u odvětví, ve kterých nelze s větší přesností odhadnout doba trvání úkolů.



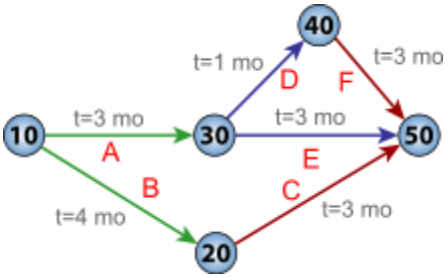
Obr. 1 CPM graf (zdroj: http://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Pert_chart_colored.svg/220px-Pert_chart_colored.svg.png)

(2) (4 p. 6)

CPM

CPM je naopak metoda deterministická. Je také jednou ze standardních metod pro zjištění časové náročnosti úkolů. Využívá se pouze jedné proměnné. Projektový manažeři v této metodice ignorují pravděpodobnosti a pracují pouze s nominálními hodnotami.

(4 p. 7)



Obr. 2 PERT graf (zdroj http://www.processma.com/resource/images/pert2.gif).

1.3.2 Analýza využití metod PERT a CPM při softwarovém vývoji

PERT i CPM jednou z entit v těchto metodách je například Ganntův diagram nebo síťové diagramy, ve kterých jsou více či méně přehledně zapsány nejen časové náročnosti nebo kritickou cestu projektu, ale také lze zde zjistit závislost jednotlivých úkolů.

Během jakéhokoliv vývoje počet paralelních úkolů nemůže být větší, než je počet lidí v týmu. To je důležitá závislost, kterou většinou v těchto diagramech nelze nalézt.

Ve vývoji software je většina závislostí vyhnutelných. Přestože se může zdát logické, že se nejprve musí vytvořit funkcionalita pro přihlášení uživatele do aplikace a teprve poté funkcionalita pro odhlašování, tyto úkoly mohou mít ve skutečnosti velmi málo společného a mohou být samostatně vytvořeny a otestovány v jakémkoli pořadí. Z vlastní skutečnosti vím, že datový model se nemusí tvořit před vytvořením dotazů do databáze z aplikace. Společně by ale měly mít stanoveny nějaké definice objektů, dle kterých by se poté mohly vyvíjet paralelně. Tato propojovací (business) vrstva může aplikovat vlastní restrikce v aplikační doméně, které mohou být jiné než restrikce, které jsou definovány v datové doméně, atd.

S tímto faktem již diagramy a jejich závislosti nejdou příliš ruku v ruce. Síťové diagramy se tak staly pouze základním nástrojem v řízení softwarových projektů. Výsledkem plánování požadavků nejen na software, ale libovolný komplexní projekt, je dlouhý

seznam úkolů, které by mezi sebou měli velice málo závislostí. Dalo by se říci, že evolucí došlo ke vzniku Project Backlogu, který budu popisovat v kapitole artefakty SCRUM.

Prostředí, ve kterém existují tisíce dodavatelů, časová náročnost úkolů se počítá na týdny nebo měsíce, úkoly nejsou paralelní a závislosti reálné je využití metody PERT velmi vhodné. PERT nebo CPM zde má možnost využít svého potenciálu. Tyto metody excelují u projektů, které mají jednotlivé fáze, což u agilního softwarového vývoje je spíše překážka.

(5)

1.4 Softwarový projektový management

Management softwarových projektů se v některých částech odlišuje od managementu projektů ve výrobě z několika důvodů; software je výrobek, který není hmatatelný, který není vyrobený ze surovin. Software je produkt založený na myšlenkách lidí. Není omezený fyzikálními zákony nebo limitovaný některými omezeními ve výrobním procesu. Je velmi obtížné nalézt a kompletně předcházet chybám a nedokonalostem. Software může být velmi komplexní. Tyto stanoviska pomáhají organizace jako PMI začlenit do PMBOK, Software Project Management.

(1)

1.4.1 Softwarový projekt

Softwarový projekt se neliší v definici od kteréhokoli jiného projektu (6);

- Který je časové omezený, s definovaným začátkem a koncem,
- neexistují dva stejné softwarové projektu (přestože mohou existovat podobné),
- projekt může být samostatný nebo součástí většího projektu,
- projekt obsahuje nejméně tři fáze – inicializační, realizační a závěrečná.

Vlastnosti softwarového projektu jsou doplněny o následující body (6);

- Výstup softwarového projektu je vždy nehmotný.
- I přes vývoj aplikací využívaných k zobrazování diagramů a procesů v softwaru, stále nedosahují takové preciznosti jako například CAD výkresy.

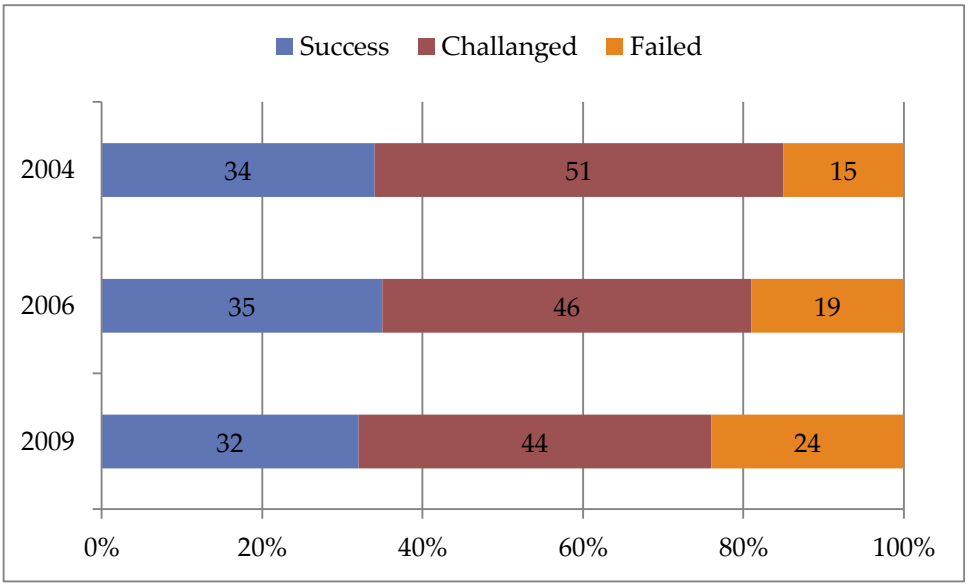
- Pro vývoj softwaru stále nejsou definovány normy v takové míře, jako je tomu u jiných inženýrských disciplín. Přestože Mezinárodní organizace pro normalizaci (ISO) a Institut pro elektrotechnické a elektronické inženýrství (IEEE) již definovaly několik norem, nejsou příliš propracované.
- Přestože vývoj v metodikách pro vývoj software je stále velmi aktivní jejich aplikace, produktivita a kvalita jsou velmi závislé na lidském faktoru a rozhodování. Existují nástroje pro testování, a automatizaci testování softwaru, avšak stále nejsou na takové úrovni, jako je tomu u ostatních inženýrských oborů. U ostatních inženýrských oborů existují nástroje, které umožnují využít lidský úsudek v kombinaci s nástroji a procesy (jinak řečeno; nástroje a procesy zvyšují množství informací které člověku umožnují dělat lepší rozhodnutí).
- Plánování je důležitý krok v každém projektu a pomáhá udržovat jeho cíl. Zatímco v ostatních inženýrských oborech mohou být metody PERT nebo CPM dostatečné, u projektů softwarových je vysoce důležité zvýšit přesnost plánování a dokumentů k plánování potřebných, za podmínky, že budou stále velmi přehledné.

1.5 Standish Group

Standish Group je společnost složená z profesionálů v oborech risk managementu, zhodnocení, řízení projektu a investic v oboru informačních technologií. Byla založená s cílem hodnotit úspěšnost softwarových projektů. Jejich zaměření je především na projekty neúspěšné, u kterých se snaží zjistit hlavní příčinu.

Studie Standish Group kategorizují projekty do tří výsledných stavů:

- Successful - projekt je dokončený ve stanovený termín a ve stanoveném rozpočtu včetně funkčnosti, která byla specifikována.
- Challenged - projekt je dokončený a provozuschopný, ale přesáhl rozpočet, časový rámec a/nebo chybí funkčnost, která byla specifikována. Označení projektu za challenged neznamená, že je projekt neúspěšný.
- Failed - projekt byl zrušen před dokončením.



Graf 1 Konečný stav softwarových projektů dle The Standish Chaos Report v letech 2004, 2006, 2009 (zdroj: vlastní zpracování)

Průměrné překročení rozpočtu projektu	45 %
Přesáhnutí časového rámce	63 %
Průměrná funkčnost dodaná k termínu projektu oproti funkčnosti požadované	67 %

Tabulka 1 Výsledky studie The Standish Report z roku 2009. Challanged projekty (zdroj: vlastní zpracování)

"Výsledky studie v tomto roce ukazují pokles úspěšných projektů. 32% úspěšných projektů, které byly dokončeny ve stanovený termín a ve stanoveném rozpočtu včetně funkčnosti, která byla dopředu specifikovaná, bylo 44 % v kategorii challeneged, (tzn. projekt byl dokončený a provozuschopný, ale přesáhl rozpočet, časový rámec anebo chybí funkčnost, která byla specifikována), a 24 % projektů byl zrušeno před dokončením a nikdy nebyly vydány.

Tyto čísla ukazují klesající tendenci úspěšnosti softwarových projektů a také nárůst projektů, které byly označeny neúspěšné (v kategorii failure). Úspěšnost je nejnižší za posledních pět let. Letošní rok (2009) dokonce reprezentuje nejvyšší výskyt neúspěšných projektů za poslední desetiletí."

(7)

The Standish Chaos studie z roku 1995 je pravděpodobně nejvíce citovaná práce na téma úspěšných a neúspěšných softwarových projektů. Studie z roku 2009 přinesla na světlo skutečnost, že neúspěšných projektů v roce 2009 bylo více v poměru k předchozím rokům.

(8)

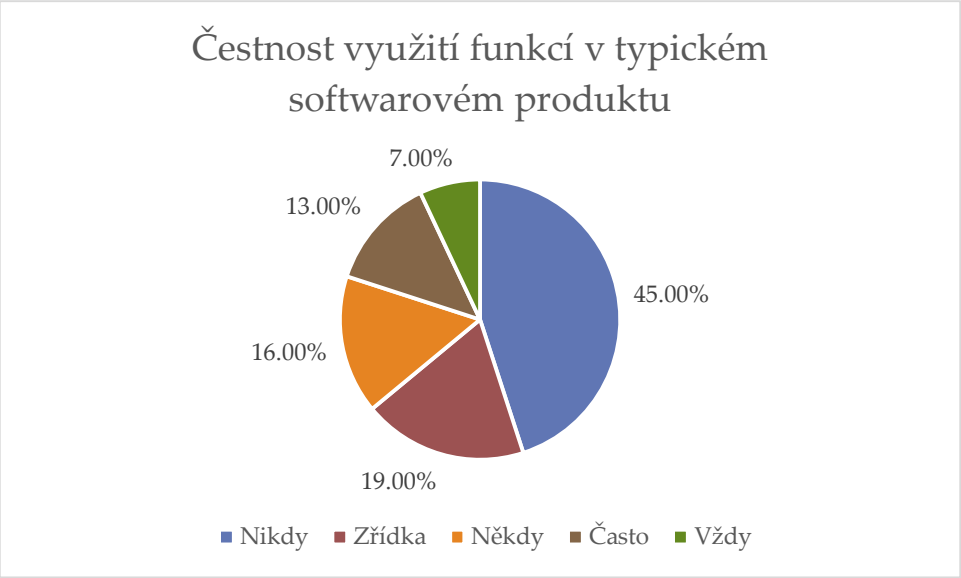
Přestože reporty a studie Standish Group jsou cennými ukazateli, bylo zjištěno, že projekty, které byly označeny za úspěšné v ukazatelích Standish Group, ještě stále nemusí být úspěšné v reálném světě – stále mohou selhat například díky statečnosti, že nebudou pro klienty nebo uživatele atraktivní, nebo nepřinesou cennou hodnotu do společnosti. Jako příklad tohoto paradoxu uvedl R. Ryan Nelson vývoj CRM systému založeného na Lotus Notes pro realitní kancelář. Strávili dva roky vývojem, aby software přidal strategickou výhodu při pronájmu volných ubytovacích zařízení. Projekt splňoval všechny potřeby společnosti. Dle Standish Group by byl označený za success, ale jeho integrace do procesů zadavatele naprosto selhala. Využitelnost aplikace nakonec bylo nulová.

(9)

Podobně tak může nastat situace opačná, kdy projekt nesplnil podmínky Standish Group, aby byl označený za *failed* (např. několikanásobně přesáhl rozpočet, časový rámec…), ale jeho užitná hodnota je obrovská.

1.6 Paradigma plánování u softwarového projektu

Běžně se u projektu vytvoří seznam požadavků, který je schválen a následně se plánuje. Projektový manažer v této chvíli předpokládá, že ze seznam požadavků se stal neměnným, a může pro začít plánovat. Fundamentální nedostatek při tomto přístupu je předpoklad, že se požadavky nezmění. Studie The Standish Group ale ukázaly, že se požadavky vždy změní. Navíc tato organizace zjistila, že 45 % z požadavků na software se nikdy nevyužije, často se využívá 13 % a pouhých 7 % z celkových požadavků na software je využito vždy.



Graf 2 Čestnost využití funkcí v typickém softwarovém produktu (zdroj: vlastní zpracování)

Tradiční přístup řízení softwarového projektu již není příliš vhodný. Důvodem je rozdíl vlastností empirických projektů, ke kterým patří vývoj software, od projektů, u kterých je proces jasně definovaný.

(10)

1.7 Agile software management

Software lze vyvíjet celou řadou způsobů. Ať už se jedná o tzv. Waterfall Process Model (klasický přístup, od analýzy přes návrh, alfa a beta verzi k finálnímu produktu) nebo o agilní model, je nutné, aby manažer týmu analytiků, vývojářů a testerů koordinoval, bral v potaz zájem klienta a zároveň kladl důraz na kvalitu a produktivitu samotných pracovníků.

Pojem *agilní* definoval Jim Heightsmit jako:

"Agilní je schopnost vytvořit a reagovat na změny pro dosažení profitu v turbuletním prostředí.", "Agilní je schopnost vyvážit flexibilitu a stabitlitu."

(11)

Agilní vývoj je populární. Všichni velcí hráči na trhu ho provozují: Google, Yahoo, Symantec, Microsoft, atd.[1]

[1] Zdoj: konference věnované agilnímu řízení softwarových projektů.

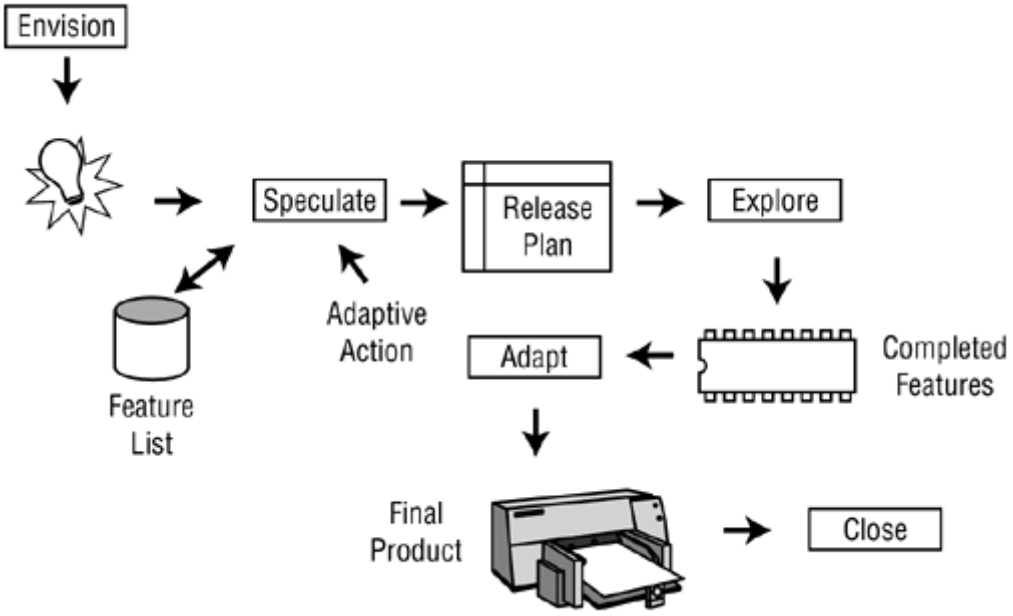
V nejistém a nestálém prostředí patří úspěch velkým společnostem, kteří mají prostředky, aby vytvořili změnu, která může vytvořit chaos v ostatních společnostech. Tvorba změny je zbraň proti rivalům a celého ekosystému obchodního prostředí. Vytvoření změny potřebuje inovaci: vývoj nového produktu, vytvoření nových prodejních kanálů, snížení času potřebného pro vývoj. Společnosti musí být schopné na změny očekávané i neočekávané reagovat rychle a efektivně.

(11)

Agilní proces může vytvářet nejasnou strukturu a absence struktury nebo stability vytváří chaos. Naopak příliš pevná struktura vytváří nepoddajnost. Agilní manažeři si kladou otázku: "Jak málo struktury mi stačí?". Být agilní tedy znamená být hbitý a flexibilní v určitém kontextu. Problém s přístupem k tradičnímu softwarovému vývoji a projektovému managementu je, že je vše definováno příliš sevřeně a do velkých detailů, což ponechává jen velmi malý prostor, aby se dalo flexibilně reagovat na změnu bez zvýšeného výdaje prostředků.

1.7.1 Agilní proces

Každý proces musí být svázaný s určitým cílem. Struktura modelu APM (agile proces management) se skládá z pěti hlavních procesů: představa, uvažovat, zkoumat, adaptovat, ukončit. V modelu lze vidět podobnost v klasickém vývojové procesu: začít, plánovat, spravovat, kontrolovat. Představa zde nahrazuje klasice započetí procesu. Zde se tvoří myšlenka projektu. Druhá část uvažovat nahrazuje plánování. Slova sdělují určité významy, a ty významy vyplývají ze systematického používání v průběhu času. Slovo "plánování" je spojeno s predikací a relativní nejistotou. Slovo zkoumat nebo také spekulovat indikuje nejistou budoucnost. Je známo, že projekty s velkým faktorem výzkumu jsou ruku v ruce s velkým faktorem nejistoty. I přesto se snažíme plánovat a odhadovat tuto nejistotu. V agilním prostředí se spíše spekuluje a adaptuje, než plánuje a tvoří. Samotný výrobní proces pod pojmem spravovat byl v agilním projektovém managementu zaměněn výrazem zkoumat.



Obr. 3 Struktura modelu APM (zdroj: Agile Management, interní dokumentace firmy Micronic Mydata AB)

- představa
- spekulace
- zkoumat
- adaptovat
- ukončit

1.8 Agilní přístup

V dnešní rychlém prostředí plném změn a zvratů musí společnosti dodávat projekty v co nejkratších časových intervalech s lepším výsledným ROI a kvalitněji. Agilní přístup je definovaný hodnotami, vysvětlen principy a manifestován praktikami.

1.8.1 Reakce na změnu

S měnícím se prostředím je zapotřebí být velmi flexibilní a společnosti se snaží přijít na trh s novými produkty a obsadit trhy, na kterých ještě nepůsobí. Schopnost adaptovat se a novému neustále se měnícímu prostředí zde tvoří pomyslnou dělící linii mezi zdarem a nezdarem. Při vývoji software je třeba být zaměřený na to opravdu podstatné.

1.8.2 Agilní hodnoty, Manifesto for Agile Software Developement

V roce 2001 skupina autorů, venujícím se oboru IT, sepsala dokument nazvaný Manifesto for Agile Software Development s cílem určit hodnototvorné cíle, ze kterého by nejvíce získal proces softwarového vývoje.

Jejich práce nastavila agilnímu světu managementu tyto hodnoty:

- Individualita a interakce jsou nad procesy a nástroji,
- fungující software je nad komprehensivní dokumentací,
- spolupráce se zákazníkem je nad vyjednáváním smluvních podmínek,
- reakce na změnu je nad plánem.

(12)

1.8.3 Agilní principy

Dokument The Manifesto for Agile Software Development definuje sadu 12 principů, které reprezentují podstatu agilního procesu vývoje. (12)

1. ***Naší nejvyšší prioritou je spokojenost zákazníka od začátku vývoje až po jeho ukončení.***

Přestože je tento princip na nejvyšších příčkách ve většině společností, v případě softwarového vývoje bývá velmi často nedodržován. Je důležité si uvědomit, že zákazník požaduje fungující software s určitou hodnotou, která pro něj bude přínosná. Nechce sadu návrhů, dokumentů nebo prototypů, nebo software s příliš mnoho funkcemi, ve kterých se klient pak bude ztrácet. Na druhé straně zákazník ale také často ani neví, jako funkce by v softwaru požadoval a časem bude přicházet na nove. Agilní prostředí umožňuje flexibilně reagovat na tyto změny.

2. ***Uvítání změn, dokonce i pokročilé fázi vývoje.***

Během vývoje software jsou zákazníci zdrojem změn. Také oni jsou pod tlakem nejen klientů, ale také pod tlakem konkurence a měnícího se trhu.

3. ***Dodat fungující software často, od par týdnů po několik měsíců, s preferencí kratších termínů.***

Velmi důležitým faktorem v jakémkoli řízení projektů je fáze kontroly - přesněji řečeno fáze zpětné vazby. Pouze při časné dodávce software je možné feedback získat dříve a této výhody využít.

4. ***Manažeři a vývojáři musí pracovat společně během celého vývoje. Manažeři ale nejsou v tomto kontextu zákazníci.***

Ve většině případů by bylo nepraktické pořádat setkání se zákazníkem každý den. Většinou ale existuje nějaký prostředník mezi zákazníkem a vývojovým týmem. Tito prostředníci mají větší znalosti o manažerském prostředí a dovednostech než vývojový tým. Mohou jim být analytici, produktoví manažeři, nebo programoví manažeři. Klíčem tohoto principu je udržovat neustálou komunikaci mezi vývojovým týmem a manažery, aby bylo zajištěno, že projekt je na správné cestě k cíli.

5. ***Kolem projektu musí být motivovaní lidé***

Je důležité umožnit lidem, kteří pracují na projektu prostředí a podporu, kterou vyžadují. Lidé jsou zdroje s potřebami. Vývoj software je odlišný od výroby. Vývoj software je druh umění. Projektový tým musí být motivovaný a důvěryhodný.

6. ***Nejlepší způsob předávání informací je osobní komunikace.***

Mnoho informací je ztraceno při jiném druhu komunikace. Osobní komunikace umožňuje oběma stranám vnímat nejen verbální komunikaci, ale také nonverbální.

7. ***Funkční software je jedinou proměnnou při měřený výkonu.***

Zákazníka nezajímá software, který je nefunkční. V klasickém pojetí projektového řízení manažer řekne, že je například 30 % hotovo, při splnění všech požadavků. U plan-driven projektů je toto tvrzení správné, ale u value-driven projektů, kde hodnotu představuje software, představuje 30 % právě 30 % již integrované, otestované a implementované funkčnosti. Toto je základním rozdílem mezi plan-driven a value-driven projektech.

8. ***Agilní proces prosazuje soustavný vývoj.***

V klasickém procesu vývoje software se velmi často stává, že čím blíž je termín dodání, tím více vývojový tým pracuje a je více pod tlakem. K tomu dochází převážně z důvodu

rozvržení aktivit již během plánovacího procesu; ke konci dojde ke škrtům anebo k doplnění funkčnosti software. V agilním vývoji software je produkt dodáván ve funkční verzi v předem stanovených cyklech (řekněme, že každých 14 dní). Tento proces dodání se nazývá iterační. V každé iteraci přibude přidané hodnota k projektu. Při soustavném vývoji je eliminován stav zvýšeného tlaku na vývojový tým ke konci termínu.

9. ***Neustálá pozornost k technické dokonalosti a dobrý design zvyšuje agilitu.***

Dobrá architektura software umožňuje její vývoj v pokročilých fázích vývoje. Opět je to druh umění abstrakce. Například využití SOLID programovacích praktik podporují dokonalou architekturu.

10. ***Jednoduchost - je esenciální tvořit věci jednoduše a pokud možno vytvořit co nejméně.***

Žádný programový kód znamená žádný bug[2]. Čím více programového kódu je vytvořeno, tím větší je šance, že software bude chybový. Často je tvořen kód s výhledem do budoucna - s funkcemi, které mohou být v budoucnu využity. V agilním prostředí je tento postup špatný. Tvoří se něco, co není aktuálně požadováno - plýtvá se tedy prostředky.

11. ***Nejlepší architektura, prerekvizity a design vzniká ze samostatně-organizovaných týmů.***

V tradičním rozložený vývojových týmů existují týmy software architektů, kteří navrhnou architekturu, designérů, která vytvoří návrhy, a analistů, kteří sepíší prerekvizity. Vznikne dokumentace, podle které vývojáři napíšou programový kód. V samostatně-organizovaných týmech by měli být profesionálové ze všech potřebných oborů pro daný projekt (nebo jeho část, která je týmu) přiřazena. Jedině tak lze využít předností takového týmu.

12. ***V předem stanovených intervalech tým dostává feedback, aby zjistil jak být více efektivní - tento feedback ihned implementuje do procesu***

V agilním prostředí je tento bod jedním z nejdůležitějších. Častá zpětná vazba a její neodkladná implementace zvyšuje produktivnost týmu. Bez zpětné vazby je tým na status

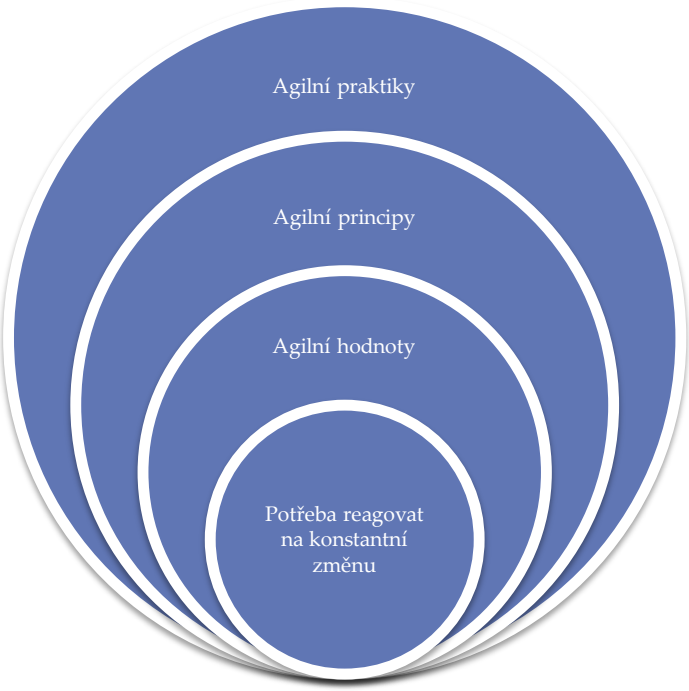
[2] chyba v programu

quo. Je potřeba zjistit, co tým dělá dobře a to ponechat, a je třeba identifikovat chyby, které tým děla a tyto chyby odstranit.

(13)

1.8.4 Agilní praktiky

Poslední vrstu (**Error! Reference source not found.**) v agilním přístupu softwarového vývoje tvoří agilní praktiky. Tyto aktivity pomáhají manifestovat principy a hodnoty agilního myšlení. Agilních praktiky se mohou implementovat do procesů nebo metodik jako například TDD, FDD (Feature driven development), BDD (Behavior driven development), Pair progrmming, XP programming atd…



Obr. 4 Schéma agilních praktik (zdroj: http://i.msdn.microsoft.com/dynimg/IC511953.jpg)

V následující části se budu věnovat konkrétně jedné z agilní praktik nazvané Scrum.

1.9 SCRUM

Scrum [3]je jedna z metodik v agilním řízení projektů.

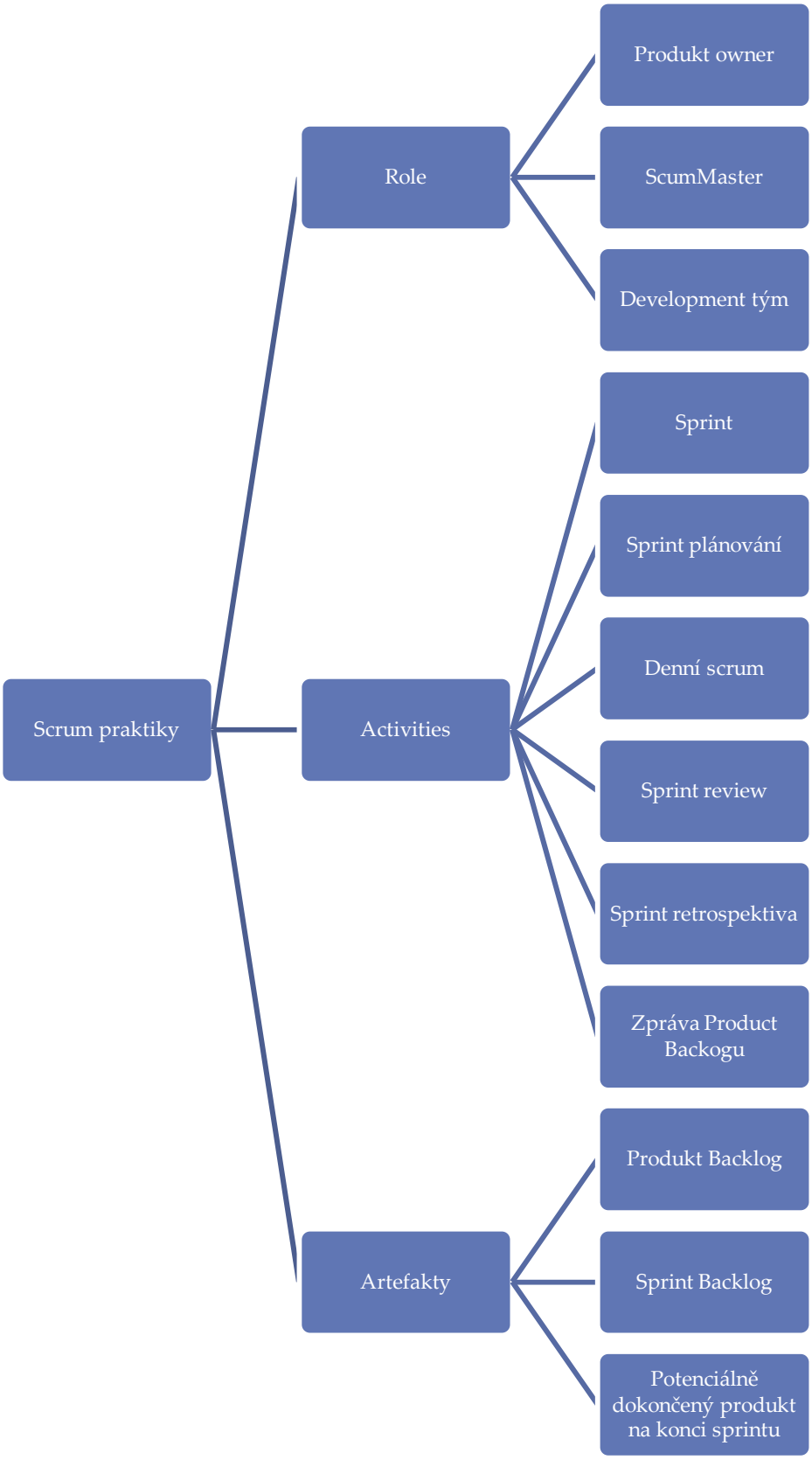
[3] Pojem scrum je používán v ragby. Je to fáze, kdy jsou hráči v kruhu a snaží se vykopnout míč, který je uprostřed. V této fázi tým posouvá míč tam a zpět, dokud nedostane příležitost míč vykopnout. (15 str. 3)

Kořeny této metodiky sahají do ve článku The New New Product Development Game autorů Takeuchi a Nonaka z roku 1986. Tento článek popisuje jakým způsobem společnosti jako např. Honda, Cannon a Fuji-Xerox využívali škálovatelného, tým-centrického přístupu k vývoji produktů. Autoři analogicky přirovnali ragby ke kompetitivnímu prostředí, kdy je zapotřebí velké flexibility a maximální rychlosti.

(14 stránky 137–146) (15 str. 3)

Scrum je jednoduchý framework zaměřený na lidi, čestnost, zodpovědnost, otevřenost, respekt a spolupráci.

(16)



Obr. 5 Scrum praktiky (15 str. 14)

1.9.1 Artefakty

Scrum obsahuje několik jednoduchých pomůcek pro řízení a grafickou reprezentaci stavu projektu. Mezi základní patří Produkt Backlog, Sprint Backlog, Taskboard, Burndown grafy.

User Story

Představují (většinou) požadavek, který má určitou užitnou hodnotu pro projekt. Ron Jeffries definuje User Story jako tři C: Card (karta), Conversation (konverzace), Confirmation (potvrzení).

(17)

Karta představuje formu, jakou je User Story interpretována. Omezující je velikost – používají se tzv. lepící bločky, nebo kartičky o velikosti 3“ x 5“. Relativně malá velikost kartiček nutí psát informace strukturované a informativně. Běla by obsahovat, kdo bude funkci vykonávat, požadovanou funkci, a důvod, proč je funkce požadovaná. Detaily jsou předmětem diskuze. Konverzace je logický přechod od komprehensivní dokumentace k agilním principům, které jsou zaměřeny na člověka, a komunikaci. Dokumentace slouží jako doplňující informace. Při diskuzi se používají skeče, zjišťují se překážky, atp.

(15 str. 83)

Pro kompletní definici User Story je nutné definovat, za jakých podmínek je User Story hotová. Tato definice představuje vstup pro tzv. Acceptance Tests (Akceptační testy). Testování není tématem této práce. Pro informace o testovacích praktik bych doporučil knihu Agile Testing: A Practical Guide for Testers and Agile Teams (ISBN-10: 0321534468, ISBN-13: 978-0321534460) z roku 2009, od autoru Lisa Crispin a Janet Gregory.

Product Backlog

Ve Scrumu se pracuje na požadavcích, které mají největší přidanou hodnotu pro produkt. Je povinností PO komunikovat se stakeholdery a vytvořit seřazený seznam požadovaných User Stories. S vytvořeným seznamem se setká s týmem, který ohodnotí náročnost implementace požadovaných funkcí. V klasickém projektovém managementu lze nalézt paralelu, kdy náročnost implementace odpovídá finančnímu zatížení společnosti na vývoj dané User Story. Je možné, že PO změní prioritu funkcí, u kterých je náročnost implementace vysoka/nízká v poměru s hodnotou User Story pro projekt.

(18 stránky 19-23) (19 str. 167)

Sprint Backlog

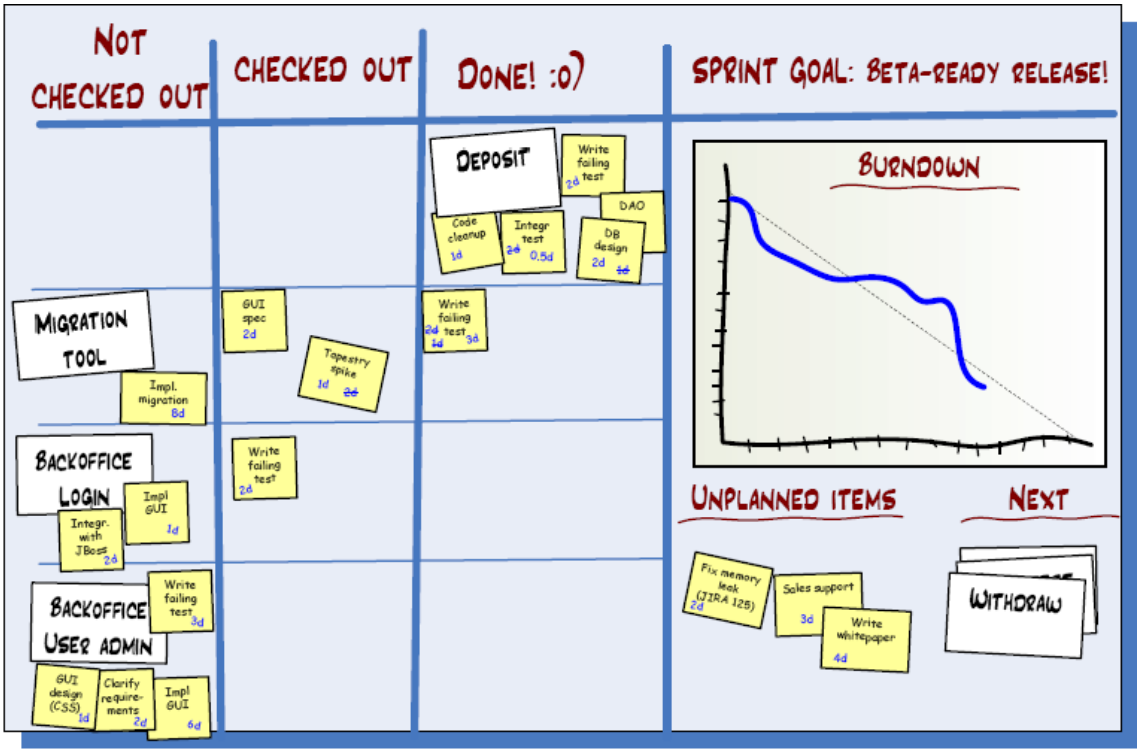
Představuje vybrané položky z Product Backlogu pro daný Sprint/Iteraci. Jednotlivé položky se mohou dle uvážení týmu rozložit na drobné pod úkoly. Je možné, že tým zjistí, že je náročnost User Story vetší či menší než původně odhadovali. Je tedy na PO aby upravil a seřadil Produkt Backlog, v případě, že je třeba.

(15 stránky 17-20) (18 stránky 45-47)

Taskboard/Dashboard

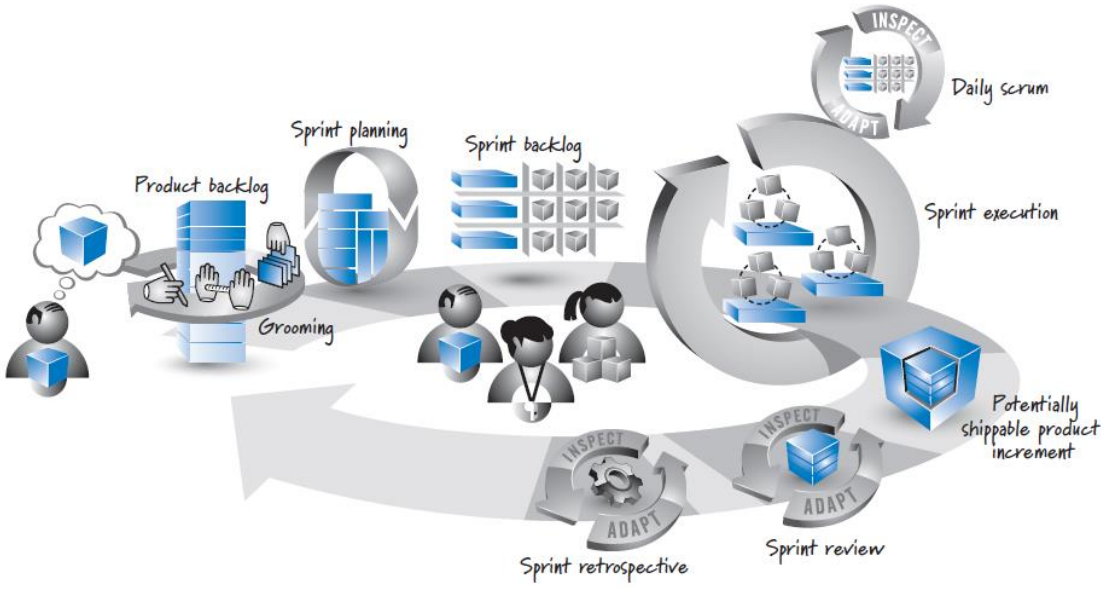
Taskboard je plocha, na které je zobrazen aktuální stav Sprintu. Většinou bývá na místě, které bylo stanoveno pro Denní Sprint Meetingy. Měl by obsahovat stav pro jednotlivé User Stories (Základními stavy jsou: Čeká na zpracování, Zpracovává se, Done. Některé společnosti využívají stavy další: Testuje se, Revize kódu, atp…). Je doporučováno, aby byl také přítomen graf, který zobrazuje aktuální stav Sprint. Tento graf se nazývá Sprint Burdown graf.

(18 str. 48)



Obr. 6 Taskboard (zdroj: http://www.infoq.com/minibooks/scrum-xp-from-the-trenches)

1.9.2 Události



Obr. 7 Scrum aktivity (zdroj: http://glenndejaeger.files.wordpress.com/2011/01/scrum_large.png)

Pro lepší orientaci ve scrum aktivitách, byl přiložen obr. 8. Na diagramu je zobrazen začátek vlevo. Vize Product Ownera se transformuje do Product Backlogu, což představuje neustále „žijící“ (upravovaný) dokument. Process pokračuje Sprint Planning Meeting, na kterém se naplánuje práce na následující sprint/iteraci. Po dobu iterace jsou denní Sprint Daily Meeting, kde se aktualizuje Sprint Backlog. Výsledkem každé iterace je potenciálně vydatelný produkt. Další aktivity nejsou povinné a je na zvážení ScrumMastera, popř. týmu, aby je provedly; Sprint Review a Sprint Retrospective. Tyto aktivity mohou vést ke změnám v procesu nebo projektu.

(15 stránky 17-18)

Ohodnocení User Stories

Jednotlivé iterace se skládají z ohodnocených User Stories. Tým se domluví na hodnotě, kterou bude jeden Story Point představovat. Jednomu týmu vyhovuje, když jeden Story Point představuje jeden ideální [4] pracovní den, jednu ideální pracovní hodinu, komplexnost, nebo čista abstraktní jednotku. Pro začínající týmy s přístupem

[4] Ideální znamená nerušená práce – žádné emaily, meetingy, telefony…

se doporučuje využít časovou referenci a stanovit, že jedna User Story představuje jeden pracovní den (hodinu).

(19 str. 88) (3 stránky 35-36)

Odhadování probíhá za účasti celého týmu, Product Ownera a ScumMastera. Během odhadování se diskutuje o obsahu, rozsahu a komplexnosti User Story. Až mají všichni konkrétní představu o obsahu User Story, napíšou na kousek papíru číslo, kterým by User Story ohodnotili. Vznikne sada čísel, která bude pravděpodobně velice rozmanitá, což je dobře. Ti, kteří dali nejvyšší a nejnižší ohodnocení se vyjádří, proč tak učinili. Je důležité, aby diskuse nebyla útočná. Po diskusi se User Story ohodnotí číslem, ke kterému většina inklinuje. Může také dojít k vytvoření nových User Stories.

Je pravidlo, že čím delší nebo komplikovanější User Story je, tím méně přesné je ohodnocení. Z toho důvodu se využívá se upravená Fibonacciho posloupnost - ½, 1, 2, 3, 5, 8, 13, 20, 40, 100.

(3 str. 52) (19 str. 93)

Ve stejných jednotkách se musí stanovit také rychlost týmu (viz. rychlost týmu).

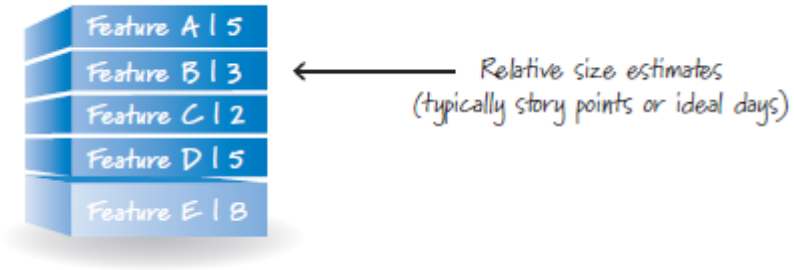
Sprint planning meeting

Samotný Product Backlog může obsahovat stovky položek (nebo měsíce práce), což je pro několikadenní sprint příliš mnoho. Sprint Planning Meeting je fáze projektu, určená k vybrání určitého počtu User Stories, které je tým schopen zvládnout zpracovat během stanovené délky pro Sprint. Během tohoto plánování se může stanovit obecný cíl sprintu.

(18) (15 stránky 21-22)

Tým znovu ohodnotí vybrané položky z Product Backlogu relativní náročností k ostatním položkám. Položky, které se dají rozložit na menší části, nebo které trvají dlouho (v poměru ke stanovené délce Sprintu), se rozloží na menší části a časově znovu ohodnotí (je-li potřeba). Tým vybere pouze takové množství práce, které je schopen ve sprintu zvládnout (viz. Rychlost týmu). Nikdy by nemělo dojít k tomu, že tým dokončí naplánovanou práci před koncem. Doporučuje se si vzít více než méně.

(18) (15 stránky 21-22) (19 str. 44)



Obr. 8 Položky z Product Backogu ohodnocené relativní velikostí. (4, str. 20)

Denní meeting

Scrum je první z agilních metodik, která přinesla to procesu denní týmové meetingy (beedle 1999). Před začátkem prvního sprintu se musí stanovit místo a čas pro denní meetingy. Tyto meetingy jsou časově omezené (15 min.) a konají se většinou v dopoledních hodinách. Každý má několik minut, aby sdělil odpovědi na tyto otázky:

- Co jsi udělal předcházející pracovní den?
- Co máš v plánu dělat dnes?
- Překáží ti něco v práci?

(18)

Je povinností ScrumMastera, aby odstranil (nebo pomohl odstranit) překážky, které by bránili ve vykonávání práce.

V této části často dochází k tomu, že SM nebo PO *delegují práci* mezi tým. V agilních praktikách již není povinností manažera práci delegovat. Každý člen týmu by měl být sám od sebe aktivní a podle toho také jednal (když ukončí práci, automaticky si bere práci, která je ve sprintu naplánovaná). Závazky členu týmu nejsou k manažerovi, firmě, PO nebo k SM – jsou k týmu a jeho členům. Delegování práce manažerem může způsobit agresi v týmu, případě ze tým zná agilní praktiky lépe než manažer. ScumMaster by měl v takovém případě být neutrální a vysvětlit jednotlivé role a jejich povinnosti.

(18)

Retrospektiva

Sprint Retrospektiva je jedna z událostí, která se může konat po ukončení sprintu. Retrospektivy se povinně účastní všichni členové týmu, SM a PO. Cílem je produktivní diskuze, která by měla objevit překážky, chyby nebo nedostatky, které nastaly během

sprintu. Druhým cílem je stanovení omezeného množství zjištěných chyb a soustředit, aby se v následujícím Sprintu neopakovaly. (15 str. 375)

Sprint review

Sprint review je další událost, která se může konat na konci sprintu. Cílem této aktivity je aktivní konverzace mezi PO, stakeholdery, týmem, SM, popřípadě i dalšími osobami. Úspěšná konverzace by měla obsahovat obousměrný informační tok, při kterém budou všechny zúčastněné strany informovány o stavu produktu a plánovaných krocích.

(15 str. 363)

Demo

Na konci každého sprintu by mely být zákazníkovi předvedeny v aplikaci nové funkce v nebo změny funkcí stávajících. Pomocí toho je zákazníkovi umožněno reagovat na nedokonalosti v implementaci, ke kterým mohlo dojít během iterace. Z vlastní zkušenosti mohu dodat, že díky prezentacím se zlepšila kvalita aplikace. Přestože by se na tyto prezentace nemělo speciálně připravovat (vše by mělo být inherentně funkční – dle metodiky Scrum nefunkční nebo nedokončené User Stories nejsou hotové – neměly by být tak ani reportované, tzn. označené v Project Backlogu, popř. Sprint Backlogu).

(17)

1.9.3 Role

Ve Scrumu existují tři hlavní role: Produkt Owner, ScrumMaster, a vývojový tým. Metodika scrum neomezuje počet rolí. Zmíněné role jsou ale pro Scrum nepostradatelné.

(18)

Product Owner

Produkt Owner (PO), neboli majitel produktu, většinou představuje zadavatele. Je hlavní autoritou, která rozhoduje o budoucím vývoji projektu. Rozhoduje o prioritách funkcí a o jejich přidané hodnotě pro projekt. Snaží se definovat jasnou vizi v kontextu celého projektového týmu. Bývá zodpovědný za funkční úspěšnost produktu. O aktivně komunikuje se ScrumMasterem a vývojovým týmem a musí být k zastižení v případě, že během vývoje vyvstanou neočekávané situace/otázku. Z povinností Product Ownera je inherentně jasné, že nemůže být Scrum Master. Product Owner nebývá ani jeden z členů vývojářů. (18) (15 stránky 14-16)

Scrum Master

ScrumMaster (SM) pomáhá všem zúčastněným stranám pochopit a dodržovat hodnoty, principy a praktiky Scrumu. Představuje lídra týmu. Je nedílnou součástí týmu pro adaptaci Scrumu do podnikových, vývojových procesů. Tyto přechody bývají velmi složité a Scrum se díky nedisciplinovanosti nepodaří adaptovat nebo v nedokonalé míře, která vede k nezdaru projektu. SM pomáhá řešit problému, na které tým narazil během vývoje, je také zodpovědný aby tým a artefakty Sprintu nebyly narušeny z vnějšího prostředí (např. není dovoleno, aby se měnil tým během sprintu nebo aby měnil stav Product Backlogu někdo jiný než PO). Z povinností ScrumMastera je inherentně jasné, že nemůže být Product Owner. Naopak není vyloučeno, aby ScrumMasterem byl jeden z vývojářů.

(15 stránky 14-16)

Tým

Tým ve Scrumu bývá samostatný (self-managed), cross-funkční s ideální velikostí o sedmi jedincích. Členové týmu bývají s různých oborů – vývojáři, tester, databázový administrátor, designer…

(15 str. 15) (18)

Rychlost týmu

Aby bylo možné odhadnout kolik User Stories je tým schopný vykonat během jednoho Sprintu, je nutné vědět rychlost týmu. Rychlost týmu musí být ve stejných jednotkách, jako jsou ohodnoceny jednotlivé User Stories.

Rychlost týmu lze zjistit třemi způsoby:

- Historické hodnoty.
- Provést první iteraci a využít rychlost, která byla dosažená.
- Odhad.

Využití historických hodnot je nejpřesnější metoda. Aby mohla být využita, je nutné, aby byl tým složen ze stejných lidí.

(19 str. 104)

Využití hodnot z první iterace je další způsob, kterým týmy projekt začínají. V praxi ale dochází k tomu, že týmu není umožněno začít na projektu pracovat, aniž by řekl, jak dlouho bude projekt trvat. Když i tato metoda není možná, je nutné hádat. Aby byl odhad relativně přesný, je třeba mnoho zkušeností.

(19 str. 104)

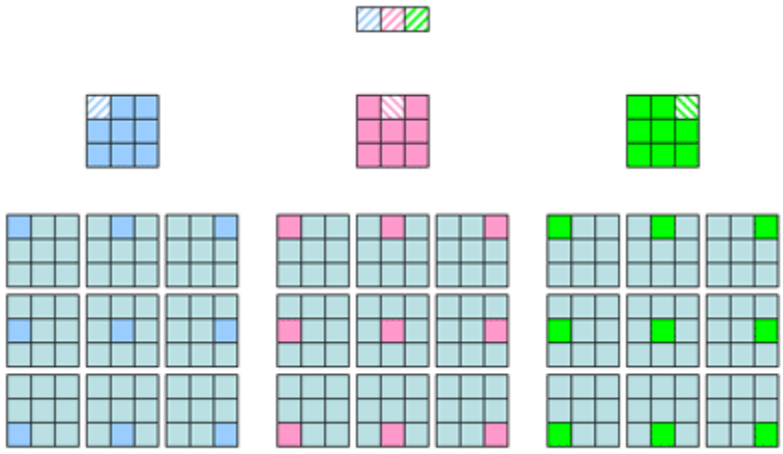
Po každé iteraci tým zjistí, kolik jednotek práce vykonal. Aritmetickým průměrem lze zjistit přesnější rychlost týmu.

(19 str. 104)

1.9.4 Škálovatelnost SCRUM

Scum tým se typicky skládá z 5-9 lidí. U velkých projektů se praktikuje tzv. Scrum of Scrums. Je vytvořen větší počet Scrum týmu (5 – 9 lidí), vybere se jeden zástupce v každém týmu. Zástupci z týmu vytvořili vlastní scrum tým, u kterého aplikují Scrum praktiky. Hierarchie je zobrazena na obrázku Obr. 9.

(20)



Obr. 9 Scrum of Scums (zdroj: http://www.mountaingoatsoftware.com/scrum/team/)

2 ANALÝZA PROBLÉMU

V klasickém řízení softwarových projektů dochází k problémům, kterým se lze vyhnout při využití agilních metodik. V mé praxi jsem se setkal s projektem, u který začal zadáním. Po naplánování a schválení rozpočtu na celý projekt začala jeho implementace. Celý projekt byl naplánován na rok. Při předání projektu a následné diskuzi se zákazníkem jsem zjistil, že zákazník ke své práci potřeboval asi 30% funkcí, a zbylých 70% mohlo být dodáno později. Komunikace se zákazníkem byla (při porovnání s agilními praktikami) minimální.

Při praktikování metodiky Scrum dochází k neustálé výměně informací a se změnami v projektu se naopak počítá. Zákazník může začít software využívat mnohem dříve s funkcemi, které jsou pro něj v té době nejdůležitější/nejpotřebnější. S každou další iterací může zákazník definovat nové funkce, upravit jejich pořadí nebo důležitost. Má svobodnou volbu mezi funkcí, která má pro něj velkou užitnou hodnotu, ale může být časově náročná na implantaci vývojovým týmem, nebo více menších funkcí, časově méně náročných. Obdobně, když se zjistí, že není možné již vývoj dále financovat, existuje verze software po posledním sprintu, která má užitnou hodnotu.

V následující kapitole Vlastní návrhy řešení se věnuji aplikování agilní metodiky Scrum v praxi na reálném projektu.

3 VLASTNÍ NÁVRHY ŘEŠENÍ

Praktická část této práce popisuje využití a přednosti metody Scrum v praxi na vývoji na míru vytvořeného softwaru. Zadavatelem byla Ing. Majka Truong Thi Thanh, spoluzakladatelka jazykové školy Aviette a branch manažerka pobočky v Brně.

3.1 Stanovení požadavků a cíle projektu

Hlavním požadavkem na software bylo zajištění přehledu studentů a kurzů, které navštěvují. Při úvodním setkání byl všem zúčastněným stranám vysvětlena metodika scrum.

Zadavatelka byla požádána o vytvoření seznamu požadavků, které na software aktuálně má. Požadavky jsou zobrazeny v tab. Tabulka 1

ID	*Název*
1	Tabulkový editor pro přidávání/editaci/odebírání studentů.
2	Zobrazení editačního formuláře pro vybraného studenta.

Tabulka 1 Seznam požadavků (zdroj: vlastní zpracování)

Před začátkem plánování bylo stanoveno, že délka iterace bude jeden týden, začátek první iterace bude 1/3/2013.

Celkový seznam požadavků na software je k dispozici na přiloženém CD.

3.2 Využité nástroje

Za šest let praxe jsem se setkal s několika softwarovými nástroji, které se používají pro správu a plánování softwarových projektů. Jmenovitě to byli například Microsoft Excel, TargetProcess[5], OnTime Suite[6] nebo Jira s GreenHopper[7] pluginem od společnosti Atlassian.

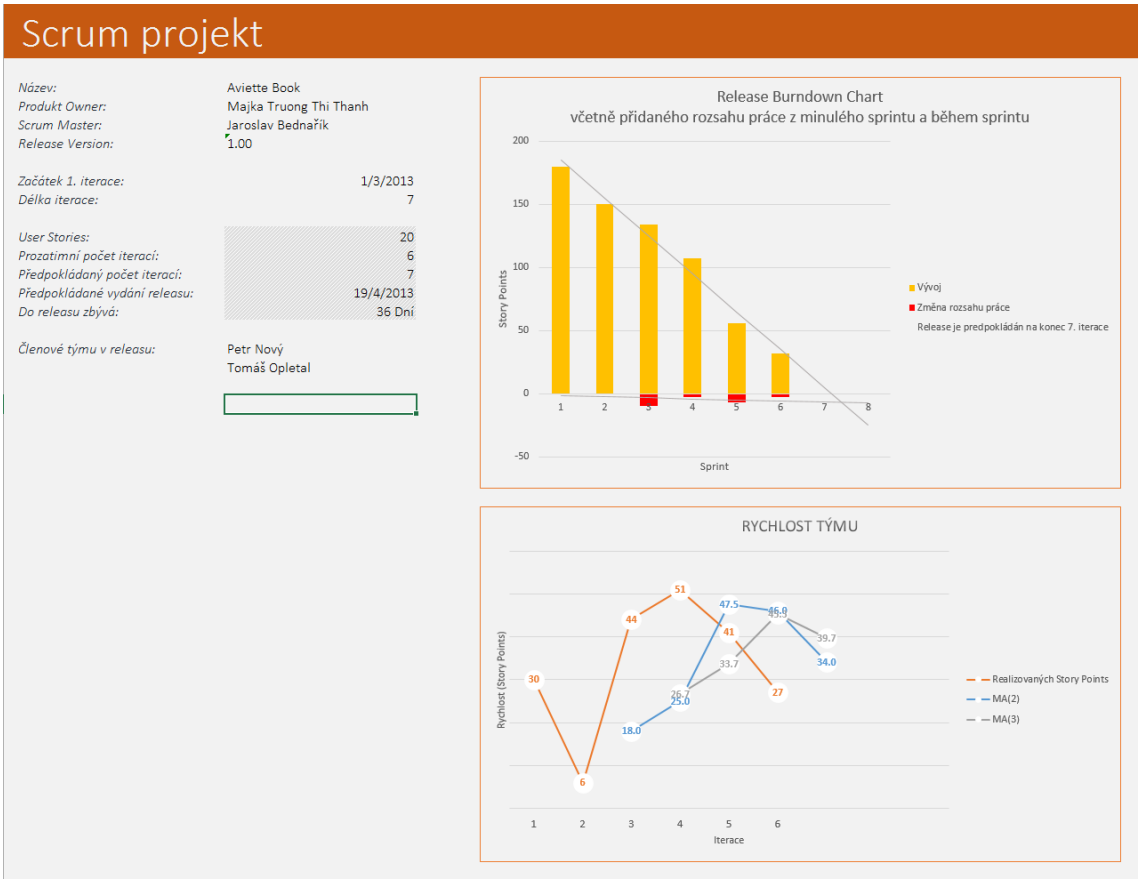
[5] http://www.targetprocess.com/
[6] http://www.ontimenow.com/
[7] http://www.atlassian.com/software/greenhopper/overview

V této práci jsem se rozhodl využít produktu Microsoft Excel, ve kterém lze transparentněji popsat výpočty a propojení mezi jednotlivými částmi. Microsoft Excel se nepoužívá u rozsáhlejších projektů kvůli rychlému znepřehlednění při vyšší počtu User Stories nebo Sprintů. Komerční nástroje jsou lépe uzpůsobeny například pro plánování nebo nastavení priorit.

3.2.1 Správa projektu

Rozhodl jsem se udržet všechny informace o projektu v jednom souboru věnovanému pro release 1.0. Jednotlivé části jsou rozděleny do záložek.

Project Dashboard



Obr. 10 Project Dashboard (zdroj: vlastní zpracování)

Záložka Project Dashboard obsahuje základní informace o stavu projektu. Obsahuje datum první iterace, celkový počet User Stories, předpokládaný počet iterací pro dokončení aplikace, předpokládané datum vydání (bez beta testování) a seznam vývojářů.

V pravé části jsou informativní grafy o zbývající práci se zobrazením přesouvání práce z jednoho sprintu do druhého (v případě, že není user story v dané iteraci hotová). Druhý graf interpretuje informace o rychlosti týmu v jednotlivých iteracích.

Project Backlog

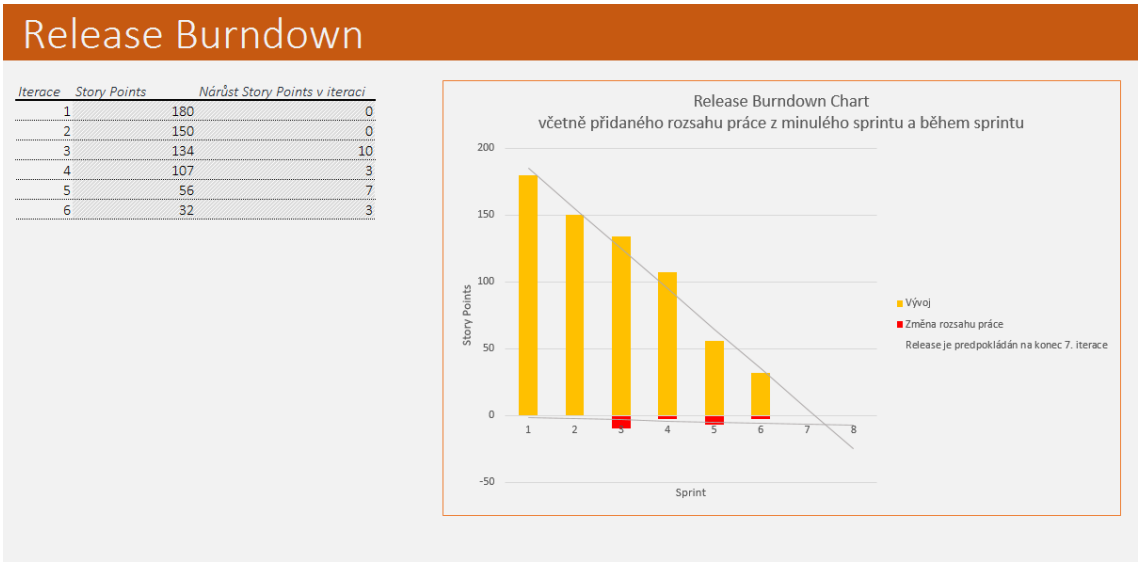
Project Backlog

ID (20)	Název	Story points	Priorita	Status	Iterace	Popis	Komentář
1	Tabulkový editor pro přidávání/editaci/odebírání studentů.	16	8	Hotovo	2		
2	Zobrazení editačního formuláře pro vybraného studenta.	12	7	Hotovo	3	Možnost přepínání mezi jednotlivými obrazovkami.	
3	Grafický koncept aplikace.	30	5	Hotovo	1	Vytvoření grafického návrhu, včetně workflow a navigace mezi jednotlivými obrazovkami ap.	
4	Mít možnost využívání aplikace/dat na různých zařízeních.	90		V plánu		Zobrazní v IPodu, Macu, PC	
5	Změna barev applikace, aby korespondovala s firemními barvami.	18	3	Hotovo	4		
6	Přidat poznámku ke studentovi.	8	5	Hotovo	5		
7	Pridat databázi poznámek v kontextu celé aplikace.	16	4	Hotovo	6		
8	Uložení barevného profilu při změně.	10	1	V plánu			
9	Vybrat grafická témata pro aplikaci	1	5	Hotovo	5		
10	Zobrazení editoru pro zařazení studenta do kurzu.	12	7	Hotovo	3	Tapulkové zobrazení s možností CRUD operací	
11	Zobrazení seznamu všech kurzů.	12	7	Hotovo	4		
12	Zobrazení detailu vybraneho kurzu se seznamem studentu.	12	7	Hotovo	4		
13	Přidat kontextové menu na řádek kurzu s prepnutím do jeho detailu.	9	4	Hotovo	5		
14	Všechny tabulkové seznamy by měly mít možnost řazení dle kliku na hvlavičku sloupce.	6	6	Hotovo	5		
15	Implementovat automatický update aplikace při startu.	3	8	Hotovo	3	Spouštění .application souborem.	ClickOne apporach.
16	Přidat možnost tisku seznamu studentů v kurzu.	16		Ve vývoji	6		
17	Přidat možnost zálohy dat.	12		V plánu			
18	Mít více barev v nastavení.	12	1	V plánu			
19	Přidat kontextové menu na řádek studenta s možností přepnutí na přehled jeho kurzů.	9	4	Hotovo	4		

Obr. 11 Project Backlog (zdroj: vlastní zpracování)

Záložka Project Backlog obsahuje informace o požadavcích zákaznice. Každý požadavek představuje zároveň i ohodnocenou User Story s užitnou hodnotou a iteraci, ve které se na ní začalo pracovat. Každá položka může mít právě jeden z těchto stavů: *Hotovo, V Plánu, Ve Vývoji.*

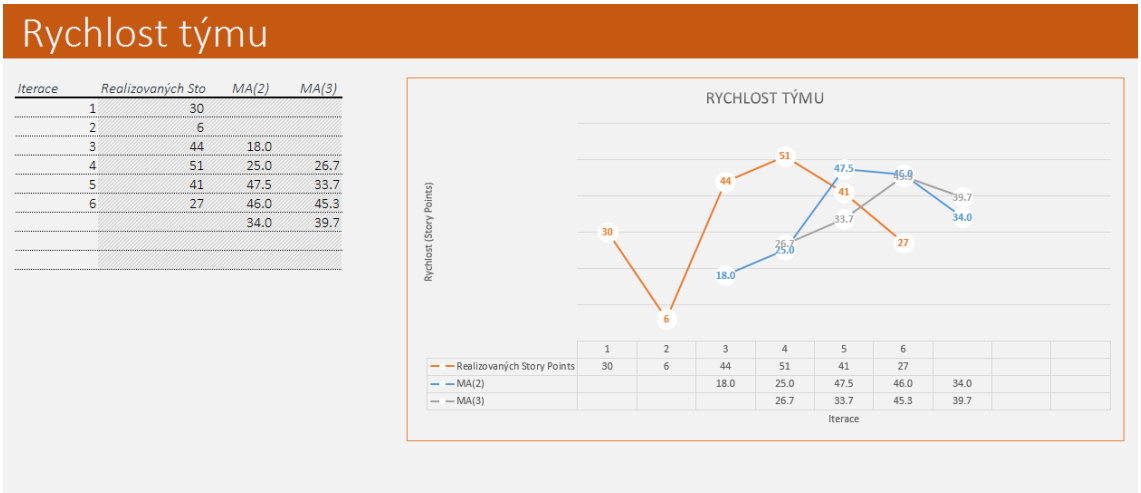
Release Burndown



Obr. 12 Release Burndown (zdroj: vlastní zpracování)

Tato záložka interpretuje data z jednotlivých iterací. Rozhodl jsem se zobrazit i nárůst Story Points, když nebyla User Story dokončena v iteraci, ve které začala. V Grafu se taková data zobrazují červeně v negativních číslech na ose y. S těmito informacemi jsem schopen vypočítat předpokládaný konec protnutím dvou přímek využitím interní funkce LINEST v MS Excelu (vstupní data jsou v záložce Misc, která je schovaná).

Rychlost týmu



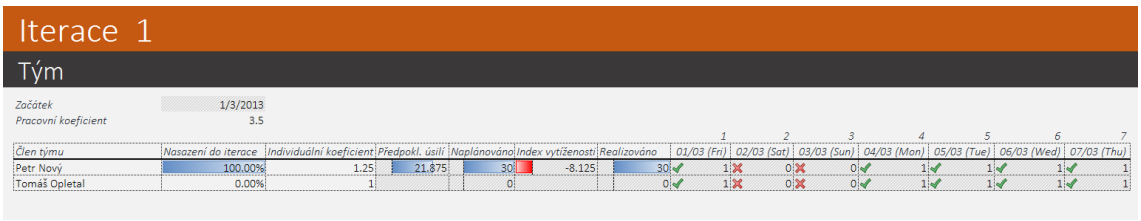
Obr. 13 Rychlost týmu (zdroj: vlastní zpracování)

Záložka Rychlost týmu interpretuje, kolik práce byl tým schopen v iteraci vykonat. Zároveň zobrazuje průměr dvou a tří posledních sprintů (tzv. moving average) pro lepší predikci rychlost týmu.

Iterace

Každá iterace má dvě záložky v tomto formátu: *Sprint [0-9] ["Tým"|"Backlog"]*. Těmto záložkám se blíže věnuji v kapitole 3.3.

Iterace – Tým



Obr. 14 Iterace – tým (zdroj: vlastní zpracování)

Tým se skládal ze dvou developerů, ScumMastera a Project Ownera.

Člen týmu	Nasazení do iterace	Individuální koeficient	Předpokl. úsilí	Naplánováno	Index vytíženosti	Realizováno
Petr Nový	100.00%	1.25	21.875	30	-8.125	30
Tomáš Opletal	0.00%	1		0		0

Obr. 15 Stav týmu v iteraci (zdroj: vlastní zpracování)

Nasazení do iterace

Každý z členů týmu nemusí být mít během iterace dostupný (dovolená, pracovní volno, atd.) Nasazení do iterace identifikuje, na kolik procent se bude vývojář věnovat práci, která je naplánovaná.

Individuální koeficient

Zkušenosti jednotlivých členu reprezentuje *Individuální koeficient*. Při nulovém koeficientu je výsledek podobný jako při nulovém *nasazení do iterace*. Zároveň umožnuje porovnat dva odlišně zkušené/rychlé programátory od sebe odlišit, přestože se budou oba věnovat v iteraci na sto procent.

Předpokládané úsilí

Každý z nich se mohl věnovat v každém sprintu jinak intenzivně. Stejně tak je každý jinak zkušený a rychlý. Tyto proměnné veličiny jsem zohlednil při výpočtu položky *Předpokládané úsilí*. Tato položka představuje počet jednotek, který by měl během

iterace zvládnout dokončit. Při nulovém nasazení do iterace je *Předpokládané úsilí* nulové.

Naplánováno

Tato položka reprezentuje součet Story Pointů, které jsou přiřazeny vývojáři v dané iteraci. Toto číslo by nemělo přesáhnout předpokládané úsilí. Jestliže se tak stane, znamená to, že by mohl být vývojář příliš zatížen a nemusí být dokončeny všechny naplánované User Stories v iteraci. V takovém případě by měla být práce nebo do iterace naplánována.

Index vytíženosti

Ideální index vytíženosti je nulový. Představuje o kolik je vývojář zatížen nebo naopak nevytížen. Vysoké pracovní zatížení může vést k častějším chybám.

Realizováno

Zobrazuje, kolik jednotek práce vývojář v iteraci vykonal.

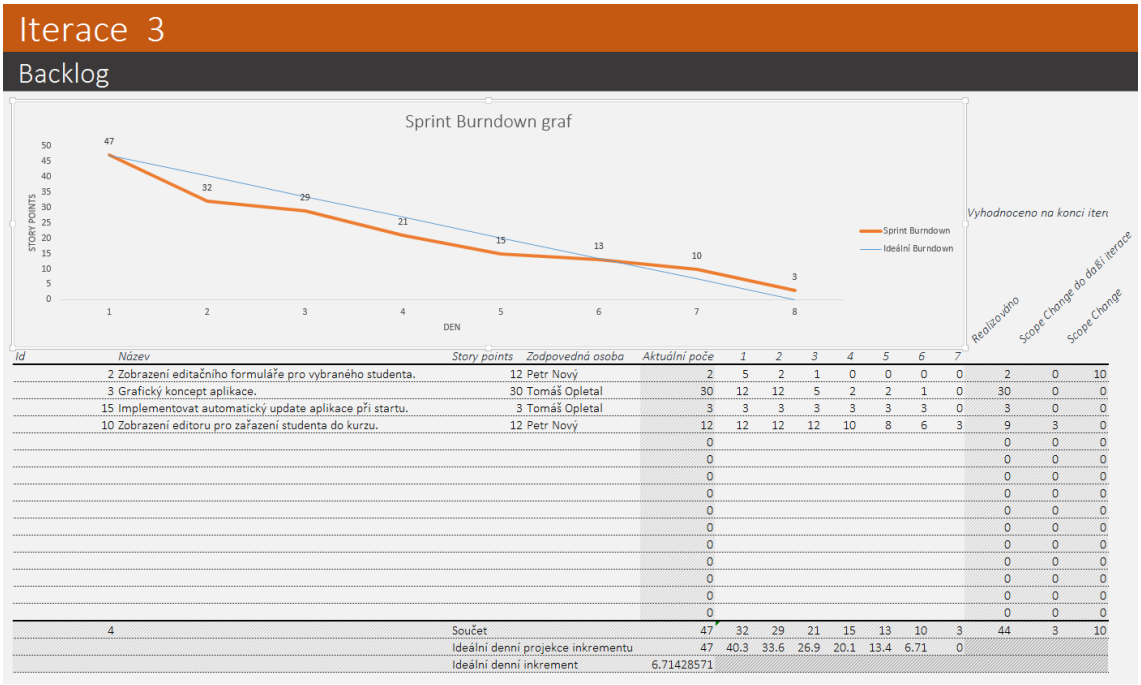
Pracovní dny v iteraci

Během iterace nastanou dny nepracovní (víkendy, svátky, apod.). Stejně tak může mít každý z vývojářů naplánovanou dovolenou. Aby bylo možné zjistit jako množství práce je schopný ve sprintu vykonat, vytvořil jsem kalendář pro jednotlivé dny v iteraci, ve kterém je možné individuálně označit, zda se pro vývojáře o pracovní den, či nikoliv. Nepracovní dny jsou ve výpočtu ignorovány.

	1	2	3	4	5	6	7
Člen týmu	*01/03 (Fri)*	*02/03 (Sat)*	*03/03 (Sun)*	*04/03 (Mon)*	*05/03 (Tue)*	*06/03 (Wed)*	*07/03 (Thu)*
Petr Nový	✓ 1	✗ 0	✗ 0	✓ 1	✓ 1	✓ 1	✓ 1
Tomáš Opletal	✓ 1	✗ 0	✗ 0	✓ 1	✓ 1	✓ 1	✓ 1

Obr. 16 Pracovní dny v iteraci (zdroj: vlastní zpracování)

Iterace - Sprint Backlog



Id	Název	Story points	Zodpovedná osoba	Aktuální poče	1	2	3	4	5	6	7			
2	Zobrazení editačního formuláře pro vybraného studenta.	12	Petr Nový	2	5	2	1	0	0	0	0	2	0	10
3	Grafický koncept aplikace.	30	Tomáš Opletal	30	12	12	5	2	2	1	0	30	0	0
15	Implementovat automatický update aplikace při startu.	3	Tomáš Opletal	3	3	3	3	3	3	3	0	3	0	0
10	Zobrazení editoru pro zařazení studenta do kurzu.	12	Petr Nový	12	12	12	12	10	8	6	3	9	3	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
				0								0	0	0
4		Součet		47	32	29	21	15	13	10	3	44	3	10
		Ideální denní projekce inkrementu		47	40.3	33.6	26.9	20.1	13.4	6.71	0			
		Ideální denní inkrement		6.71428571										

Obr. 17 Iterace – tým (zdroj: vlastní zpracování)

Každá iterace má vlastní záložku se informacemi o User Stories, které jsou relevantní pouze k dané iteraci. Každá položka může byt zjemněna na drobnější pod úkoly. Každý den se zapíše do patřičného pole, kolik práce zbývá k dokončení User Story.

Za každou User Story by měl být někdo odpovědný, což reprezentuje políčko O*dpovědná osoba*.

V případě, že byla User Story přesunuta z předcházející iterace z důvodu, že nebyla dokončena, sloupec *Scope Change* reprezentuje práci již vykonanou. *Aktuální počet* je rozdíl mezi *Story Points* a *Scope Change*.

V horní části je Sprint Burndown graf, kde je možné během sprintu pozorovat průběh práce. Oranžová linka reprezentuje aktuální zbývající práci, zatímco modrá reprezentuje její ideální trajektorii.

3.3 Průběh

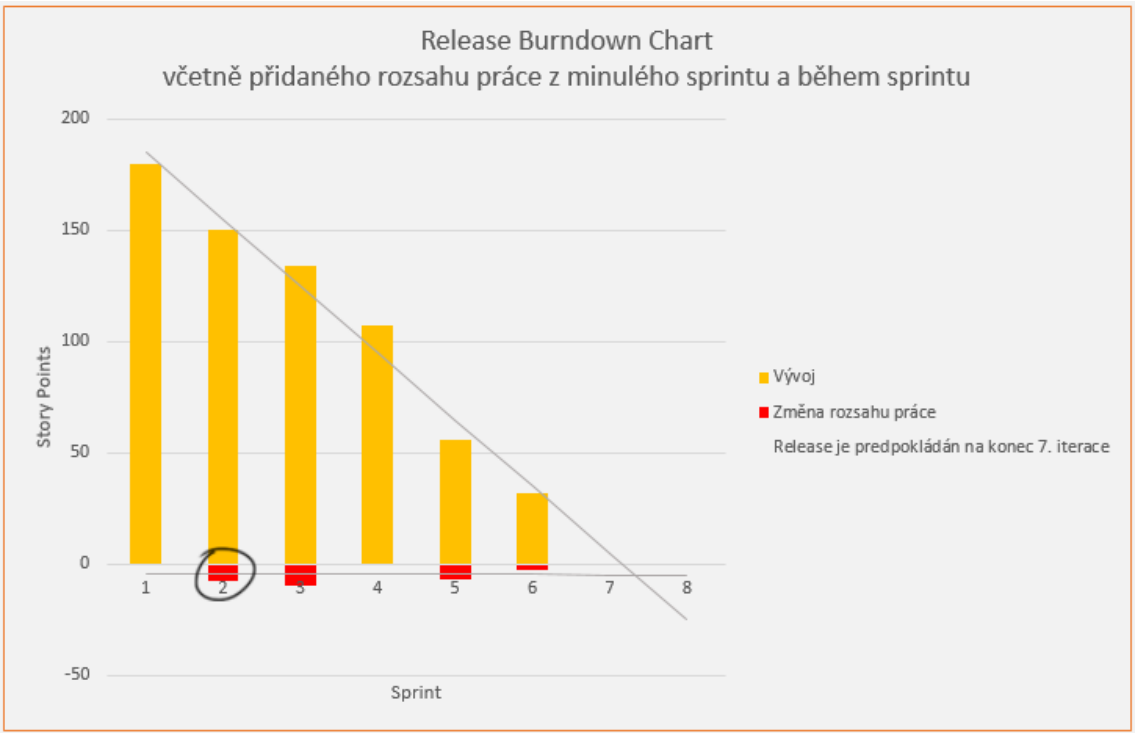
Délka iterace byla stanovena na sedm dní. Osmý den (první v další iteraci) se tým setkal, aby se naplánoval sprint následující.

První iterace proběhla bez hodnocení nebo plánování. Prvním úkolem byl tzv. sketchup aplikace (tzn. rozmístění jednotlivých ovládacích prvků v aplikaci). Pro požadovanou aplikaci bylo potřeba pouze dvou návrhů (příloha 1 a příloha 2), nad kterými se pouze diskutovalo o požadovaných funkcích. Další obrazovky aplikace jsou natolik podobné, že ani pro ně nebylo potřeba skeče kreslit. Zároveň se mohlo začít pracovat na jádru aplikace a doménovém modelu.

Do druhé iterace se vybrala podle metodiky Scrum první nejvíce přínosná funkce pro zákazníka, kterou byla přehled studentů s jejich kontaktem. Tato iterace skončila s nedokončenými User Stories. Mohlo to být způsobeno nedostatečným nebo špatným naplánováním user story, technickými nebo jinými překážkami. Znamená to, že se změní naplánovaný rozsah práce v iteraci následující. V Release Burndown grafu se tato změna projeví nárůstem rozsahu práce do negativních čísel (viz. Obr. **18**).

Třetí iterace začínala se zbývající prací z iterace předchozí. Rozsah práce u těchto User Stories by snížen o práci již vykonanou.

Tímto způsobem se postupovalo v každé iteraci. Na konci každé z nich bylo demo (prezentace dodělaných User Stories) a diskuse se zákazníkem pro získání feedbacku.



Obr. 18 Změna rozsahu práce při nedodělání práce v patřičném sprintu. (zdroj: vlastní zpracování)

Po ukončení šestého sprintu se zákazník rozhodl aplikaci začít využívat a vývoj byl pozastaven. Další požadované funkce v aplikaci pro něj nemají aktuálně přidanou hodnotu za dobu nutnou k jejich implementaci a rozhodl se tedy vývoj pozastavit. Díky metodice Scrum to bylo možné bez větších komplikací.

Další vývoj je předpokládán v několika následujících měsících. Pokračovat se bude přidáním dalších user story, úpravou nebo diskuzí s týmem nad položkami, které již v seznamu požadavku existují.

ZÁVĚR

Scrum může pomoci organizacím dodávat kvalitnější software při dodržování hodnot a principům sepsaným v agilním manifestu. Jak píše jeden ze zakladatelů metodiky Scrum: *"…přestože se Scrum stává více mainstreamem než radiálním pojetí projektového managementu, adaptace Scrumu (nebo agilních praktik) selhává…"*

Jeff Sutherland, http://jeffsutherland.com/scrumhandbook.pdf.

Z vlastní zkušenosti mohu konstatovat, že softwarové společnosti v České Republice (v menším měřítku i jinde v Evropě) nechtějí z různých důvodů na agilní metodiky přejít.

Při správné adaptaci agilních praktik v nich vidím velký potenciál pro zkvalitnění služeb softwarových společností, a zároveň i spokojenosti zákazníka. Agilní praktiky se mohou adaptovat nejen ve firmách softwarových, ale například v reklamních, designerských, apod. společnostech. Toyota byla jedna z prvních automobilek, která adaptovala několik oddělení na Scrum – Toyota Prius byl prví automobil, který byl vyvíjen agilní metodou Scrum).

Existují certifikační kurzy a instituce, které se věnují osvětě při praktikování metod scrum. Doufám, že stále více nejen softwarových společností budou adaptovat agilní praktiky.

SEZNAM POUŽITÉ LITERATURY

1. TELECOM EGYPT. Presentation of Project Management Body Of Knowledge PMBOK. *ICU - Committed to connecting the world.* [Online] 2010. [Citace: 1. 1 2013.] http://www.itu.int/ITU-D/arb/COE/2010/ProjectManagement/Documents/Doc03-PM-for-ICT-workshop.pdf.

2. SVOZILOVÁ, A. *Projektový management.* Praha : Grada, 2006. str. 351. ISBN 80-247-1501-5.

3. COHN, M. *Agile Estimating and Planning.* New Jersey : Prentice Hall, 2006. str. 330. ISBN 978-0-13-236435-5.

4. SHARMA, S. C. *Operation Research, Pert, CPM & Cost Analysis.* New Dehli : Discovery Publishing House, 2006. str. 309. ISBN 81-8356-102-0.

5. DUBAKOV, M. Why MS Project Sucks for Iterative Development? Part II. *Edge of Chaos.* [Online] 31. 1 2006. [Citace: 6. 11 2012.] http://www.targetprocess.com/blog/2006/01/why-ms-project-sucks-for-iterative.html.

6. CAGLEY Jr., T. *Mastering Software Project Management.* Fort Lauderdale : J. Ross Publishing. str. 390. ISBN 978-1-60427-036-1.

7. THE STANDISH GROUP. *CHAOS Report.* Boston : The Standish Group, 2009. str. 4.

8. —. CHAOS Report. *Spinroot.* [Online] 1995. [Citace: 15. 8 2012.] http://www.spinroot.com/spin/Doc/course/Standish_Survey.htm.

9. NELSONM, R. R. Track in The Snow. [editor] David Rosenbaum. *CIO.* 1. Září 2006, stránky 28-30.

10. SMITH, G., SIDKY, A. *Becoming Agile.* New York : Manning Publications Co., 2009. ISBN 1933988258.

11. HANGSMITH, J. *Agile Project Management.* Boston : Addison-Wesley Professional, 2009. str. 277. ISBN 0321658396.

12. BECK, KENT, a další. Manifesto for Agile Software Developement. *Agile Manifesto.* [Online] 2001. [Citace: 28. 10 2009.] http://agilemanifesto.org/.

13. BECK, KENT, a další. Principles behind Agile Manifesto. *Agile Manifesto.* [Online] 2001. [Citace: 28. 10 2009.] http://agilemanifesto.org/principles.html.

14. TAKEUCHI, H. a NONAKA, I. The New New Product Development game. *Harward Business Review.* [Online] 1986. [Citace: 1. 5 2012.] http://hbr.org/1986/01/the-new-new-product-development-game/.

15. RUBIN, K. S. *Essential Scrum.* Michigan : Library of Congress Cataloging-in-Publication Data, 2012. str. 452. ISBN 978-0-13-704329-3.

16. BEEDLE, M., a další. *Scrum: A Pattern Language for Hyperproductive Software Developement.* Boston : Addison-Wesley, 1999.

17. JEFFRIES, R. Essential XP: Card, Conversation, Confirmation. *An Agile Software Development Resource.* [Online] 30. 8 2001. [Citace: 8. 5 2012.] http://xprogramming.com/articles/expcardconversationconfirmation/.

18. KNIBERG, H. *Scrum and XP from the Trenches (Enterprise Software Development).* USA : Lulu.com, 2007. str. 140. ISBN 978-1-4303-2264-1.

19. COHN, M. *User Stories Applied.* Boston : Addison-Wesley, 2013. str. 268. ISBN 0-321-20568-5.

20. DAVIES, R. a BIRD, C. Scaling Scrum. *ScrumAlliance.* [Online] 2007. [Citace: 12. 5 2012.] http://www.scrumalliance.org/resource_download/287.

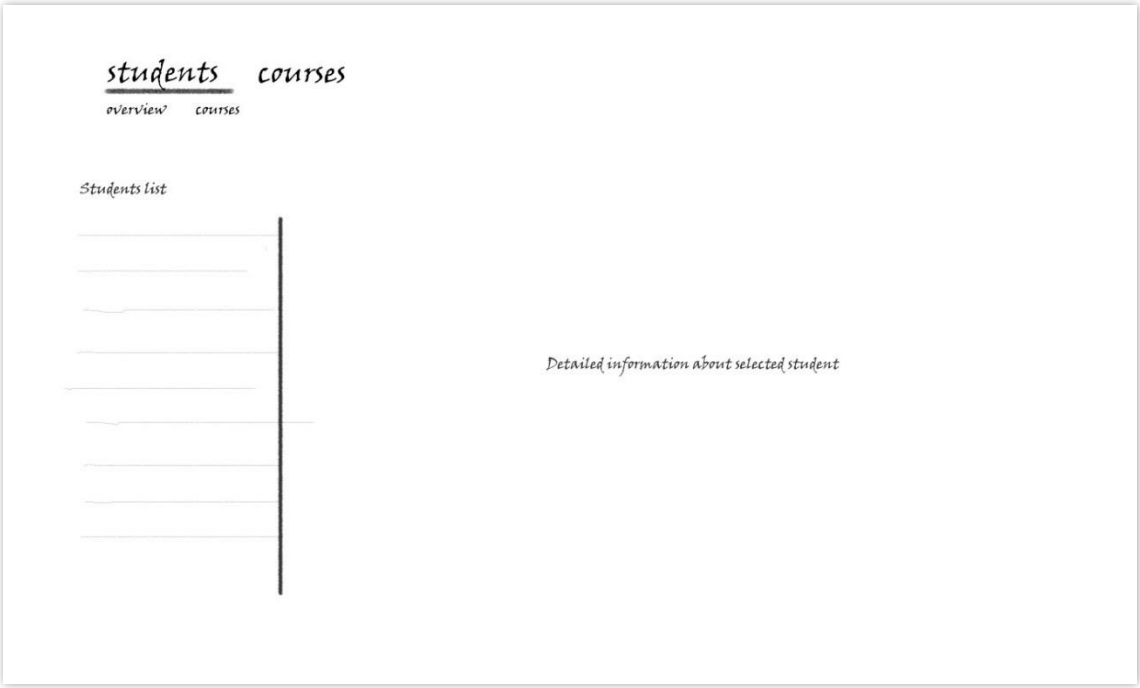
SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

SEZNAM PŘÍLOH

Příloha č. 1 Skeč grafického rozhraní aplikace 1



Příloha č. 2 Skeč grafického rozhraní aplikace 2