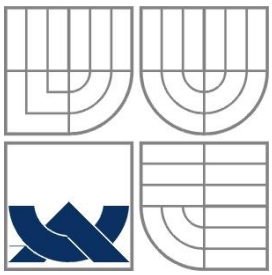
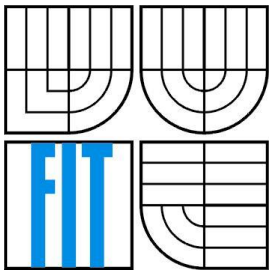


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AGILNÍ METODY VÝVOJE SOFTWARE

SOFTWARE DEVELOPMENT AGILE METHODS

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ DREISEITEL

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. JITKA KRESLÍKOVÁ, CSc.

BRNO 2015

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2014/2015

Zadání diplomové práce

Řešitel: **Dreiseitel Jiří, Bc.**
Obor: Management a informační technologie
Téma: **Agilní metody vývoje software**
Software Development Agile Methods
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s agilními metodami pro vývoj software, jejich parametry a způsoby nasazení pro různé druhy projektů.
2. Zaměřte se na metodu Scrum-ban, její specifikaci a smysl, fáze, požadavky, výstupy, metriky a návaznost na ostatní projektové procesy.
3. Použijte metriky, vybrané po dohodě s konzultantem firmy Siemens CZ, k evaluaci existujícího projektu a navrhněte oblasti zlepšení.
4. Navrhněte automatizované nasazení metody v konkrétním prostředí systému pro projektové a organizační řízení Jira.
5. Navrhněte a implementujte zásuvné moduly (plug in) pro systém Jira tak, aby podporovaly metodiku Scrum-ban v rámci existujících organizačních procesů vybraných po dohodě s vedoucí.
6. Použijte metriky k evaluaci nových postupů a výsledky porovnejte s původním měřením. Zhodnoťte úspěšnost použitých metod a předložte návrhy ke zlepšení.

Literatura:

- Cohn, M.: Agile Estimating and Planning, Prentice Hall, 2007, ISBN 0-1314-7941-5.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů zadání 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kreslíková Jitka, doc. RNDr., CSc.**, UIFS FIT VUT
Konzultant: Řezáč Jakub, Ing., Siemens CZ
Datum zadání: 1. listopadu 2014
Datum odevzdání: 27. května 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce popisuje základní principy agilního vývoje software a uvádí jejich výhody oproti tradičnímu přístupu. Kombinaci agilních metodik Scrum a Kanban používá společnost Siemens v Brně v rámci organizačních procesů, proto se práce zaměřuje především na metodu Scrum, Kanban, Scrum-ban a porovnání těchto metod mezi sebou. Práce se dále zabývá analýzou existujícího projektu, výsledek této analýzy byl předložen konzultantovi ve společnosti Siemens společně s návrhy na zlepšení. Po vzájemné domluvě byl navržen a implementován zásuvný modul pro systém JIRA. Tento modul rozšiřuje současné možnosti analýzy existujících projektů z pohledu agilních metodik, což vede k podpoře při rozhodování v rámci agilních procesů. Závěrem práce je diskutován přínos implementovaných řešení a možné směry dalšího rozvoje.

Abstract

The thesis deals with basic principles of agile software development and describes their advantages compared to the traditional approach. Siemens, s.r.o in Brno uses combination of Scrum and Kanaban as part of organizational processes, therefore the thesis focuses mainly on the method Scrum, Kanban, Scrum-ban and compares these methods with each other. The thesis also analyzes the existing project and results were consulted with the representative of the Siemens company together with the proposals for improvement. JIRA plugin was designed and implemented base on our mutual agreement. This module extends the current tool for analyzing agile projects, this leads to better support of decision making in the context of agile processes. Benefits and further upgrades are consulted at the end of the thesis.

Klíčová slova

Vodopádový model, agilní metodiky, extrémní programování, Scrum, Kanban, Scrum-ban, JIRA, JIRA Agile, měření a procesů

Keywords

Waterfall model, Agile, Extreme Programming, Scrum, Kanban, Scrum-ban, JIRA, JIRA Agile, measuring process

Citace

Dreiseitel Jiří: Agilní metody vývoje software, diplomová práce, Brno, FIT VUT v Brně, 2015

Agilní metody vývoje software

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní docentky RNDr. Jitky Kreslíkové, CSc.
Další informace mi poskytl Ing. Jakub Řezáč ze společnosti Siemens, s.r.o. Všechny literární prameny a publikace, z kterých jsem čerpal, jsou uvedeny v závěru práce.

……
Bc. Jiří Dreiseitel
27.5.2015

Poděkování

Děkuji za poskytnuté materiály a podklady docentce RNDr. Jitce Kreslíkové, CSc. a Ing. Jakubu Řezáčovi. Za jejich vstřícný přístup a čas, který mi byl věnován. Dále bych rád poděkoval svým rodičům a blízkým za neustálou podporu v průběhu celého studia.

Obsah

Seznam obrázků

1 Úvod

Vývoj software je stejně stará disciplína, jako první počítače. Během svého vývoje se stejně jako počítače, musela tato disciplína vypořádat s řadou problémů a prodělat velké množství změn. Vývoj začal revolučním vodopádovým modelem, od kterého se odvíjely další metody a postupy, až dospěl k dnešním sofistikovaným přístupům vývoje software. Jedním z takových zástupců je metodika od společnosti IBM s názvem Rational Unifed Process.

V posledním desetiletí se však začaly objevovat úplně nové metodiky, které se snaží zaměřit na rychlejší a levnější dodávku software. Jsou hromadně označované jako agilní. Těmto metodikám se věnuje stále více odborné veřejnosti a je jim předpovídána slibná budoucnost.

1.1 Cíl a struktura práce

Dlouhodobým záměrem společnosti Siemens je zefektivňování organizačních procesů především prostřednictvím automatizace. Jádro pro řízení těchto procesů představuje systém JIRA, který centrálně slouží k organizaci veškerých úkolů v rámci softwarového návrhu, vývoje i testování. Z tohoto důvodu vzniká potřeba analýzy dat řízených tímto systémem tak, aby bylo možné kvantifikovat účinnost nastavených postupů. Získané hodnoty pak slouží nejen k trvalé kontrole vývoje organizace, ale také jako argumentace při rozhodování o přistoupení k lokálním a systémovým změnám. Protože se tato rozhodnutí realizují na úrovní segmentového managementu, je potřeba poskytovat naměřená data v jasné formě, která řídícím pracovníkům umožňuje učinit často i velmi drahá rozhodnutí bez komplexní znalosti použitých metodik. Cílem diplomová práce je analyzovat používané nástroje, definované business procesy i konkrétní technická nastavení a navrhnout rozšíření, na jejichž základě je možné přehlednou formou podpořit či uskutečnit výše popsaná rozhodnutí v rámci agilních procesů vývoje, vzhledem k tomu, že společnost Siemens praktikuje agilní vývoj pomocí metodiky Scrum a Kanban.

V úvodní kapitole 2 se obecně zabývám metodikami pro vývoj software. Nejdříve se stručně věnuji tradičnímu přístupu vývoje software, který vychází z vodopádového modelu. Na tuto podkapitolu pak navazuji seznámením s agilními metodikami, začátek je věnován vzniku těchto metod a jejím hlavním principům. Ve zbylé části kapitoly se věnuji popisu vybraných agilních metodik.

Následující kapitola 3 je věnována podrobně metodě Scrum. Jsou zde vysvětleny role, které tato metodika zavádí, dále nejčastěji používané artefakty, vysvětlení hlavních činností v průběhu Scrum a na závěr je vysvětlena funkce a popis Scrum tabule.

Další kapitola 4 je věnována metodice Kanban. Kapitola obsahuje seznámení se stručnou historií metodiky a popis její základní charakteristiky. Dále je v kapitole popsána Kanban deska, která je nejdůležitějším artefaktem této metodiky a v poslední části je porovnání metod Scrum a Kanban.

V kapitole 5 popisuji kombinaci dvou výše uvedených metodik a to metodiku Scrum-ban. Lze zde nalézt její základní principy a popis metrik, sloužících k měření výkonosti efektivnosti zavedených procesů.

Kapitola 6 je věnována analýze a evaluaci již existujícího projektu na základě. Dále je kapitola věnována návrhu zlepšení formou zásuvného modulu pro systém JIRA.

Následující kapitola 7 obsahuje návrh a implementaci pro systém JIRA. V úvody kapitoly se věnujeme popisu systému JIRA, tak aby čtenář mohl pochopit principy a funkci tohoto nástroje. Další část je věnována použitým prostředků při implementaci nástroje. Následuje podkapitola 7.3, která je věnována specifikaci zásuvného modulu a popisu struktury jeho implementace. V závěrečné podkapitole 7.4 se věnuji způsobů testování pro odhalení chyb a nasazení v produkčním prostředí.

Poslední kapitola 8 je věnována diskuzi o celkovém přínosu práce a tomu, jaké by mohly být další směry vývoje a zlepšení implementovaného řešení. Zhodnocení práce vychází z kvalifikovaného hodnocení zástupce společnosti Siemens a z reálného nasazení implementovaného modulu.

Diplomová práce přímo navazuje na semestrální projekt, který byl zpracován během zimního semestru. Tento projekt se zabýval agilními metodikami a jejich porovnáním. Uvedené kapitoly byly převzaty do textu práce.

2 Metodiky vývoje software

Úvodní kapitola práce je věnována základnímu seznámení se metodikami pro vývoj software. První část kapitoly je věnována tradičním metodikám a v druhé části se věnuji seznámení s agilním přístupem při vývoji software.

2.1 Tradiční metodiky

Tato práce je zaměřena na agilní metody, při vývoje software. Pro pochopení agilního přístupu je potřeba seznámení i s původními tradičními postupy, které byly předchůdcem agilního vývoje. Tyto metodiky jsou stále používané, vyvíjely se dlouho dobu a postupem času zmohutněly. Problém nastává především u vývoje menších softwarových systémů, kde tyto metodiky nejen že kvalitě výsledného produktu nepřispěly, ale naopak ji ještě snížily prodražením vývoje softwaru z důvodu dodržování řady byrokratických pravidel, aniž by se dosáhlo přiměřeného efektu. Přesto agilní přístup nelze uplatnit na všechny typy projektů, popřípadě mu může bránit struktura a zavedené procesy v dané společnosti. Vhodnost výběru metodiky spočívá nejen na velikosti projektu, ale především na způsobu dodávání produktu zákazníkovi.

Metodiky lze tedy rozdělit na tradiční a agilní. Pro tradiční metodiky platí, že je kladen silný důraz na jednotlivé fáze – získání požadavků od zákazníka, monitorování systému, návrh, implementace apod. Takový vývoj obvykle trvá delší dobu a může být značně neefektivní. V této části práce se nachází jednoduchý popis vybraných tradičních metodik, které jsou využívány při vývoji softwarových produktů. Jedná se o tradiční vodopádový model životní cyklus a spirálový model životního cyklu.

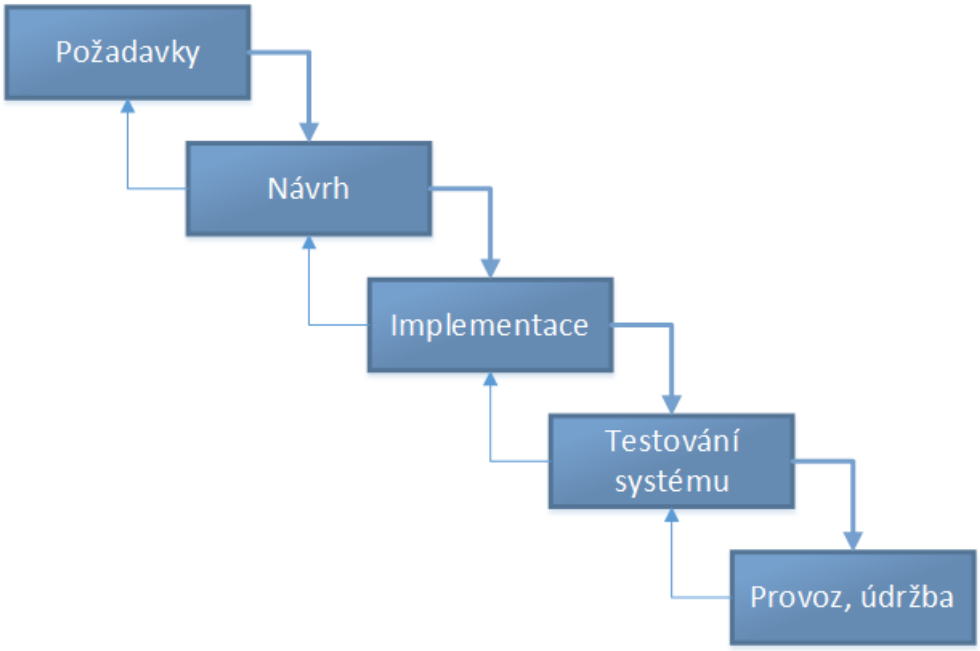
2.1.1 Vodopádový model životního cyklu

Základním modelem životního cyklu software je tzv. vodopádový model. Tento model je postaven tak, že jsou jednotlivé etapy seřazeny za sebou, tedy následující etapa začíná až po ukončení etapy předcházející. Vodopádový model je velmi přirozený a zřejmě i nejstarším model životního cyklu vývoje software. V dnešní době je model využíván především k ukázce, jak vývoj probíhat nemá. Je vyhovující pouze pro jednodušší projekty, kde je výhodou jeho jednoduchost a snadné použití. Grafické znázornění je obrázku 2.1.

Nevýhodu při použití vodopádového modelu je především uživatel, který není schopen stanovit všechny požadavky. Problém pak nastává v poslední fázi projektu, kde při předání spustitelné verze softwaru vznese uživatel celou řadu nových požadavků, nebo opraví původní. Problém vyplívá z toho, že i při zjištění nedostatků se lze vrátit pouze do předchozí úrovně. Zákazník tedy může do vývoje zasahovat pouze na začátku a na konci životního cyklu vývoje software. Místo toho aby byl software

nasazen a používán, dostaneme se v životním cyklu prakticky na úplný začátek, je tedy nutné provést novou analýzu požadavků související s úpravou návrhu, kde výsledek toho všeho je nová implementace a testování. To všechno je velmi problematické, protože se s tím v životním cyklu vůbec nepočítalo a analytici či návrháři už mohou pracovat na novém projektu. Vodopádový model tak neodpovídá vývoji reálných projektů, kdy se jednotlivé etapy překrývají nebo jdou dokonce v jiném pořadí.

Pro nasazení v praxi má vodopádový model vážné nedostatky, jeho důležitost plyne právě z toho, že z něho vychází ostatní modely životního cyklu software. Jeho použití v praxi je každopádně lepší než neřízený chaos a v případě jasných a předem dobře definovaných požadavků, může vést k úspěchu projektu.



Obrázek 2.1: Vodopádový model životního cyklu software, zdroj vlastní

2.1.2 Spirálový model životního cyklu

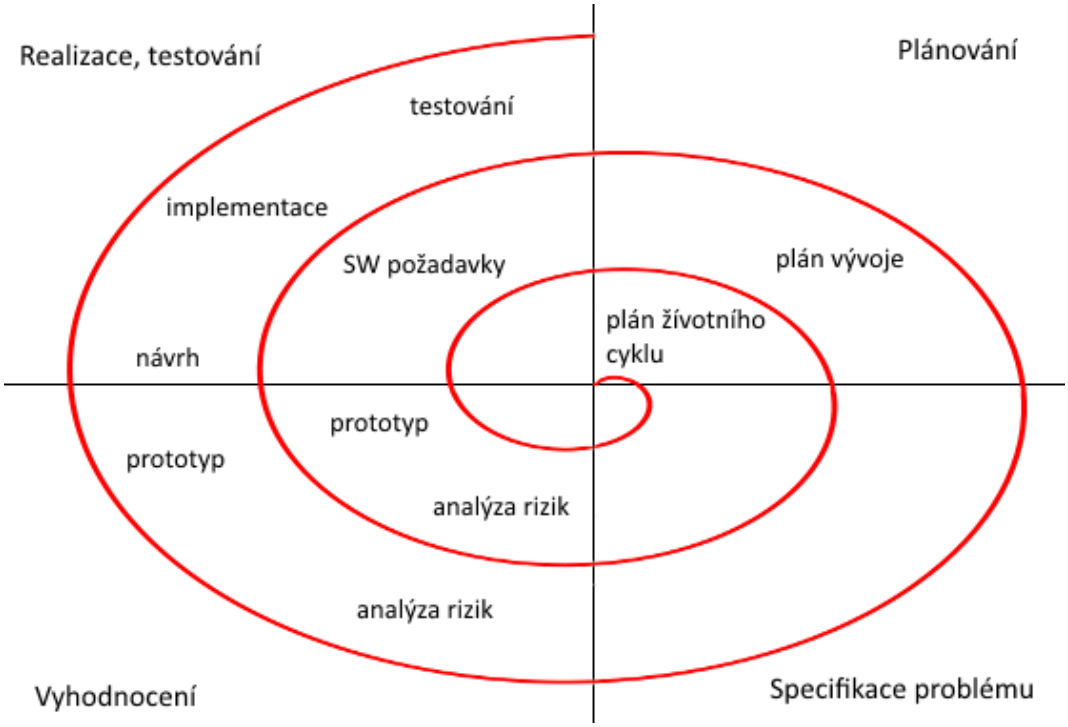
Spirálový model se oproti vodopádovému liší v zavedení iterací pomocí prototypů[1] a klade velký důraz na analýzu rizik. Etapy vývoje software se opakují vždy na vyšším stupni zvládnuté problematiky. Po každém ukončení etapy nedostává uživatel verzi softwaru s omezenou funkcionalitou, ale pouze prototyp. Ukázka jedné z verzí spirálového modelu životního cyklu je na obrázku 2.2.

Vývoj probíhá od středu spirály ven. Pravý dolní kvadrant znázorňuje specifikaci problémů, stanovení cílů, alternativ, omezujících podmínek apod. Z pravého dolního kvadrantu se přesouváme do levého dolního kvadrantu, kde je potřeba provést analýzu rizik. Následuje horní levý kvadrant, ve

[1] **Prototyp** se po použití může zahodit a software se vytváří znovu, to přispívá k jednoduššímu a lepšímu systému, za cenu tvorby prototypů. V některých případech se zahazovat nemusí a může se stát finálním produktem.

kterém probíhá samotná realizace, která zahrnuje návrh, implementaci, testování apod. Poslední kvadrant zajišťuje plánování dalšího postupu.

Výhodou spirálového modelu je, že již od začátku existují spustitelné verze softwaru, takže chyby jsou odhaleny co nejdříve, což snižuje cenu při jejich odstraňování. Dalším plusem je nezávislost na konkrétní metodice, možnost neúspěchu je zde ošetřena analýzou rizik, která vede k lepšímu plánování a zdůraznění na co se zaměřit. Nevýhodou spirálového modelu je však jeho náročnost na řízení a lze jen těžko získat celkový přehled o projektu. Zároveň lidé, kteří stanovují rizika, musejí být experty ve svém oboru, aby celý projekt neskončil krachem.

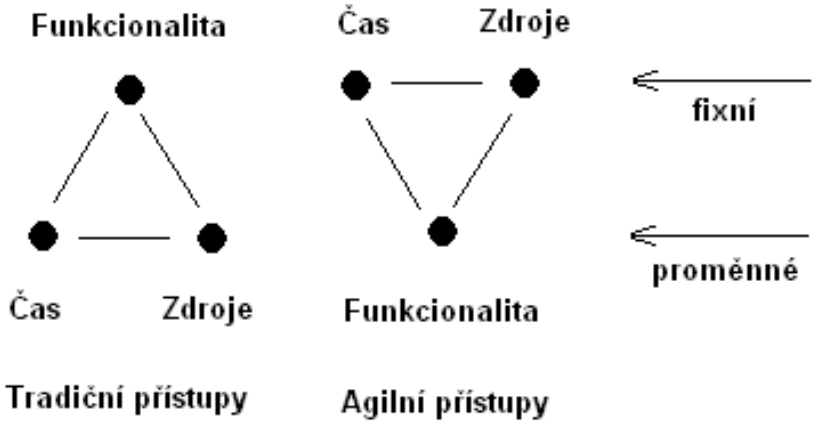


Obrázek 2.2: Ukázka spirálového modelu životního cyklu, zdroj vlastní

2.2 Agilní metodiky

Agilní znamená dynamický, rychlý, interaktivní, přizpůsobivý, iterativní, zábavný, hravý, rychle reagující na změnu a mnohé jiné. Jedná se o jiný způsob života, upřednostňování jiných hodnot a to reálného výsledku před striktními procesy, upřednostnění změny před naplánovaným.

V následující kapitole se budeme zabývat vznikem a principem agilních metodik. Tato skupina metodik vznikla za účelem iterativního a inkrementálního vývoji software. Cílem těchto metodik je co nejlevněji a co nejdříve dodat produkt zákazníkovi, který splňuje všechny jeho požadavky, je kvalitní a zároveň umožňuje provádět následnou údržbu a podporu. Agilní přístup rozhodně není omezen pouze na programování, jeho uplatnění můžeme najít také v Business Intelligence[1], popřípadě v marketingovém plánování. Jedná se o protiklad již výše uvedeného vodopádového modelu. Základní rozdíl mezi agilním a tradičním přístupem si lze ukázat na obrázku 2.3. U agilních přístupů je funkcionalita proměnná, ale čas a zdroje jsou neměnné. Na rozdíl od tradičního přístupu, kde je funkcionalita neměnná, což vede k tomu, že nejsme schopni zachovat původně naplánované zdroje a čas.



Obrázek 2.3: Rozdíl mezi tradičním a agilním programováním, zdroj [3]

2.2.1 Vznik agilních metodik

Každá z agilních metodik je svým způsobem specifická, ale všechny jsou postaveny na stejných principech a hodnotách. Za oficiální vznik agilních metodik můžeme považovat podepsání *Manifestu agilního vývoje software* v roce 2001. Tím byla vytvořena *Aliance pro agilní vývoj software*. Manifest obsahuje čtyři stanovení takové, že tvrzení na levé straně mají větší váhu, než tvrzení na pravé straně. Manifest dle [1] zní:

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

[1] **Business Intelligence** jsou dovednosti, znalosti, technologie, aplikace, kvalita, rizika, bezpečnostní otázky a postupy používané v podnikání pro získání lepšího pochopení chování na trhu a obchodních souvislostech.

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

2.2.2 Hlavní principy agilních metodik

Manifest tedy deklaruje zásadní pravidla, které je třeba při vývoji software uplatňovat a na jejichž základě bylo definováno 10 hlavních principů agilních metodik, které jsou v následujících odstavcích blíže charakterizovány [2]:

1. **Nejvyšší prioritou je včas a kontinuálně dodávat software, který zákazníkům přináší hodnotu.**
   Zákazníka zajímá především fungující software v každé iteraci, a zda poskytnutá funkcionalita uspokojí jeho potřeby. Proto zákazníka tolik nezajímá dokumentace, diagramy nebo integrace stávajících systémů. Důležité je neustálé ověřování hodnoty pro zákazníka, protože ta je předmětem změny.

2. **Změnu požadavků je možné provést i v pozdějších fázích vývoje, protože tím může zákazník získat konkurenční výhodu.**
   Agilní přístup se snaží realizovat změny efektivně a zároveň řídit rizika negativních dopadů. Proto je potřeba realizovat krátké iterace, na jejichž konci je dodávka fungujícího software. Agilní přístup, oproti tradičním metodám, klade důraz na zkrácení cyklu dodání přírůstku.

3. **Dodávaní fungujícího software v intervalech týdnů až měsíců, s preferencí kratší periody.**

4. **Uživatelé a vývojáři spolupracují denně na projektu.**
   Mění se koncept specifikace požadavků. Základem je přesvědčení, že není možné dohodnout a podepsat všechny požadavky již na začátku projektu. Proto dochází na začátku pouze k definování hrubých požadavků, které se na základě každodenního jednání s uživateli upřesňují a mění. Tím je zodpovědnost na projektu přenášena i na uživatele.

5. **Motivovaní jedinci, kteří mají vytvořené podmínky pro práci a mají podporu vedení, jsou klíčovým faktorem úspěchu projektu.**
   Je potřeba motivovat všechny lidi účastnící se projektu a zároveň musí být projekt podporován všemi zainteresovanými stranami. Jsou to lidé, kteří rozhodují o úspěchu či neúspěchu projektu.

6. **Nejefektivnějším způsobem přenosu informací v rámci vývojového týmu je osobní komunikace.**
   Dokumentace, která bývá u agilních metodik považovaná za nedostatečnou, není cílem projektu. Smyslem dokumentace je pochopení projektu, kterého se s menšími náklady a větší efektivitou dosáhne pomocí přímé komunikace v týmu.

7. **Primární mírou úspěchu je fungující software.**
   Pro úspěch projektu je třeba mít fungující systém. Plnění plánu a existující dokumentace návrhu je pro úspěch nedostačující.

8. **Agilní procesy předpokládají „zdravý" vývoj.**
   Je kladen důraz na vymezení pracovního prostoru, ve kterém zůstane tým v dobré kondici. Nedochází pak k přesčasům a snižování produktivity práce.

9. **Perfektní technické řešení i návrh.**
   Pro agilní přístup je velmi důležitá kvalita návrhu řešení, která je nezbytná pro realizaci změn. Návrh je iterativní proces, který je prováděn v průběhu projektu, není tedy plně dokončen před začátkem implementace.

10. **Zásadním požadavkem je jednodušší řešení, tj. umění maximalizovat množství neudělané práce.**
    Při vývoji software je možné aplikovat celé množství metod a technik. V agilním projektu je kladen důraz na to, aby dané postupy byly co nejjednodušší.

11. **Nejlepší architektury, požadavky a návrhy vznikají ze samo organizujících se týmů.**
    Tento princip zdůrazňuje kreativitu lidí, častou komunikaci a přizpůsobování metodiky.

12. **Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování i zvyklosti.**

2.3 Popis vybraných agilních metodik

V této kapitole se budeme věnovat stručnému popisu vybraných agilních metodik. Hlavní téma práce bude zaměření na metodiku Scrum, proto si ostatní metodiky jen přiblížíme a ukážeme si jejich základní vlastnosti a charakteristiky, tak aby čtenář získal základní orientaci v dané problematice.

2.3.1 Extrémní programování

Metodika extrémního programování (XP) je určena především pro malé až středně velké týmy, při vyvíjení software, u kterého se mění zadání nebo není jasně definované. Autory této metodiky z počátku 90. let jsou Kent Beck, Ward Cunningham a Ron Jeffries. Jedná se o provádění tradičních činností, které jsou dovedeny do extrému. Díky tomu se extrémní programování dokáže lépe přizpůsobit měnícím se požadavkům zákazníka a dodávat software vyšší kvality. Popis metodiky viz [2], [3].

Charakteristika metodiky

Jak již bylo uvedeno výše, extrémní programování vychází z principů a postupů běžných při vývoji softwaru, které však dovádí do extrémů:

- Pokud se osvědčí revize zdrojového kódu, budeme kód neustále revidovat (**párové programování**).
- Pokud se osvědčí testování, budou všichni vývojáři neustále testovat (**testování jednotek**) a budou také testovat zákazníci (**testování funkcionality**).
- Osvědčí-li se návrh, zařadíme jej do každodenní činnosti (**refaktorizace**).
- Pokud se nám osvědčuje jednoduchost, pak vždy volíme nejjednodušší řešení, které vyhovuje požadovaným funkcím (**to nejjednodušší, co ještě může fungovat**).
- Jsme-li přesvědčeni o důležitosti architektury, budou ji všichni neustále definovat a propracovávat (**metafora**).
- Jestliže se přesvědčíme o důležitosti testování integrace, budeme integrovat a testovat několikrát denně (**nepřetržitá integrace**).
- Pokud se osvědčují krátké iterace, uděláme je opravdu krátké: vteřiny, minuty a hodiny nikoli týdny či měsíce a roky (**plánovací hra**).

Při tom všem slibuje extrémní programování snížení projektových rizik, zlepšení schopností reagovat na změny a vylepšení produktivity během celého životního cyklu systému.

Klíčové pojmy extrémního programování

- **Komunikace** – XP se zaměřuje na udržení řádných komunikačních toků, tak že využívá postupů, které nelze bez komunikace provádět. Dále je definován speciální člen týmu tzv. kouč, který řeší problémy v komunikaci a znovu navazuje vztahy mezi lidmi.
- **Jednoduchost** – XP sází na to, že je lepší udělat něco jednoduchého dnes a zítra případně zaplatit za nějakou změnu, než platit dnes za složitou věc. Pro realizaci jednoduchosti podle XP je nutné nedívat se moc dopředu na záležitosti, které se budou implementovat zítra, příští týden či měsíc.
- **Zpětná vazba** – Zpětná vazba v XP probíhá ve dvou paralelních časových rovinách. První z nich je v řádu minut a dnů a týká se testů jednotek, tedy testů od vývojářů. Další je v řádu týdnů a měsíců kde se realizují testy funkcionality od zákazníka.

- **Odvaha** – Pokud tým zjistí, že dopředu cesta nevede, je nutné chybu opravit i v případě, že to znamená zahození poloviny již udělané práce. V kombinaci s předchozími třemi body, se odvaha stává velmi ceněnou a užitečnou věcí.

Extrémní programování je velmi „lehká“ metodika, která zavádí specifické praktiky jako párové programování, refaktorizace, testy před implementací a další. Jedná se o velmi disciplinovaný proces, který však není podrobně popsán, ale je realizován velmi kvalifikovanými a disciplinovanými vývojáři za podpory vývojových nástrojů.

2.3.2 Metodika Lean Development[1]

Metodika Lean Development je především souhrnem pravidel, jejichž používání by mělo zefektivnit a zrychlit vývojový proces. Autorem této metodiky je Robert Charette a pochází z principů *Lean Manufacturing* a *Total Quality Management* na oblast vývoje software. Popis metodiky viz [2].

Charakteristika metodiky

Inspirací pro metodiku Lean Development se staly postupy uplatňované ve výrobě zejména v 80. letech (lean production – štíhlá výroba). Základem metodiky je koncept *dynamické stability*, tedy schopnost přizpůsobit se rychle a efektivně daným požadavkům. Je spjata se schopností vytvářet stabilní, neustále se zlepšující vnitřní procesy. Ty mají potom obecnou platnost a přizpůsobují se širokému okruhu produktů. Cílem je vytváření softwaru tolerantního ke změnám s třetinovou lidskou prací, třetinovým časem, třetinovou investicí do nástrojů či metod a s třetinovou námahou přizpůsobit se novému tržnímu prostředí.

10 pravidel aplikace štíhlé výroby ve vývoji softwaru

1. **Odstranit zbytečné** – Odstranění všeho co nepřináší hodnotu konečnému produktu, jako dokumenty, diagramy a modely, které při vývoji pohlcují software, ale nejsou nutnou součástí finálního produktu.
2. **Minimalizovat zásoby (minimalizovat artefakty)** – Minimalizace dokumentace pomocí dosažení určité úrovně abstrakce v dokumentaci. Lepší 10 stran pravidel a dokumentace odchylek, než 100 stran detailní specifikace produktu.
3. **Maximalizovat tok (zkrátit čas vývoje)** – Je potřeba redukovat práci na procesu, iterativní vývoj je aplikací tohoto principu.
4. **Vývoj řízený poptávkou (rozhodovat se co nejpozději)** – Praktiky vývoje softwaru, které dokáží přizpůsobit dodávku softwaru požadavkům uživatelů, představují v měnícím se prostředí konkurenční výhodu. Pokud je návrh zařazen na začátek životního cyklu, tak se s největší pravděpodobností dostaneme do rozporu s požadavky.

[1] Pro tuto metodiku Lean Development neexistuje ekvivalentní český název.

5. **Pracovníci s rozhodovací pravomocí (rozhodovat co nejníže)** – Vývojář musí pochopit, jak jeho práce přispívá celkovému cíli, musí vědět co a do kdy vykonat a musí mít možnost rozhodovat.
6. **Uspokojovat požadavky zákazníků (nyní i v budoucnu)** – Častým důvodem neúspěchu projektů jsou způsobeny chybějícími, nekompletními nebo nesprávnými požadavky zákazníka. Odpovědí na tento problém jsou praktiky detailní specifikace uživatelských požadavků, které uživatel odsouhlasí. Problém je v tom, že uživatel není schopný dohlédnout všechny potřeby a úplná specifikace trvá dlouho. Tím se zvyšuje riziko, že požadavky nebudou odpovídat skutečným potřebám.
7. **Zavést zpětnou vazbu** – Pokud není možné definovat detailně všechny požadavky předem, tak je potřeba zavést zpětnou vazbu a doplňovat je postupně. Provádíme pak změny v průběhu vývoje produktu.
8. **Odstranit lokální optimalizaci** – V době neustálých změn nemá smysl provádět optimalizaci stávajícího řešení. Popis metody viz [2].
9. **Partnerství s dodavateli** – Hodnota pro zákazníka je důležitá, proto pro urychlení a zlepšení kvality je možné i nakupovat komponenty.
10. **Zavést kulturu neustálého zlepšování** – Vytvořit vhodné podmínky pro tým a motivací zlepšovat procesy při vývoji software.

Lean Development, na rozdíl od většiny metodik, které se zabývají taktickou úrovní managementu, pracuje na strategické úrovni s vazbou na podnikovou strategii.

2.3.3 Metodika Feature-Driven Development[1]

Agilní metodika od autorů Jeff de Luca a Peter Coad, zachovává procesní řízení a zdůrazňuje úlohu modelování při vývoji. V metodice se nachází více formálních požadavků a kroků oproti extrémnímu programování. Popis metodiky viz [2], [4].

Charakteristika metody

Metodiku Feature-Driven Development (FDD) lze rozdělit na dvě fáze. Získávání seznamu vlastností tzv. features pro implementaci a implementaci těchto vlastností jednu po druhé. Vše je založeno na iterativním vývoji, vývoj začíná vytvořením celkového modelu a pokračuje posloupností dvoutýdenních iterací, ve kterých se provádí návrh i realizace pro jednotlivé užitné vlastnosti.

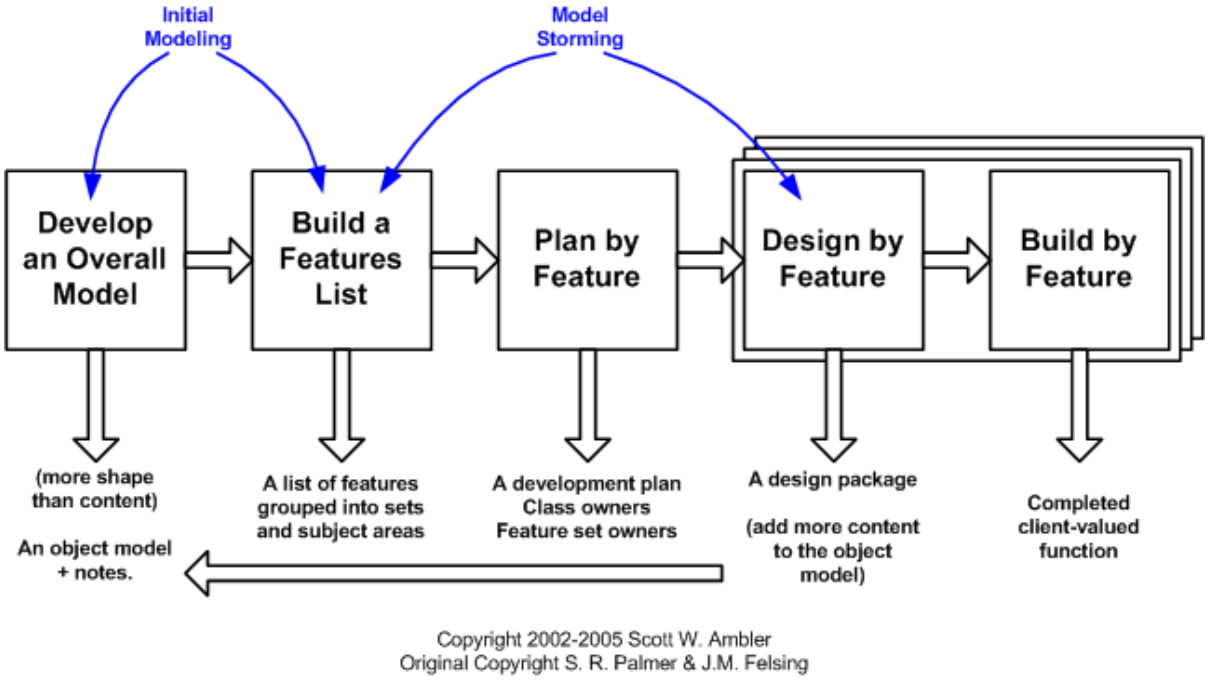
Užitná vlastnost (feature) – malý výsledek užitečný z pohledu zákazníka, který je srozumitelný, měřitelný a realizovatelný spolu s dalšími v rámci dvoutýdenní iterace. Jak lze vidět na obrázku 2.4, FDD se skládá z 5 procesů:

- Vytvoření celkového objektového modelu (Develop an Overall Model).

[1] Pro metodiku Feature-Driven Development neexistuje ekvivalentní český název.

- Sestavení seznamu užitných vlastností (Build a Feature List).
- Plánování pro užitnou vlastnost (Plan by Feature).
- Návrh pro užitnou vlastnost (Design by Feature).
- Realizace pro užitnou vlastnost (Build by Feature).

Cílem FDD je fungující produkt, nikoliv splnění předepsaného procesu. Přesto se definují „lehké“ procesy, které jsou popsány cca na 2 stránkách textu a které umožňují rozšiřovat metodiku na větší projekty, rychleji zapojit nové zaměstnance, stanovit priority, zaměřit se na přínosy.



Obrázek 2.4: Posloupnost vývojových fází metodiky, zdroj [4]

2.3.4 Crystal metodiky

Pro Crystal metodiky se používá označení „rodiny“ metodik, které jsou určeny pro různé typy projektů. Jejím autorem je Alistair Cockburn. Popis metodiky viz [2], [5], [6].

Charakteristika metodiky

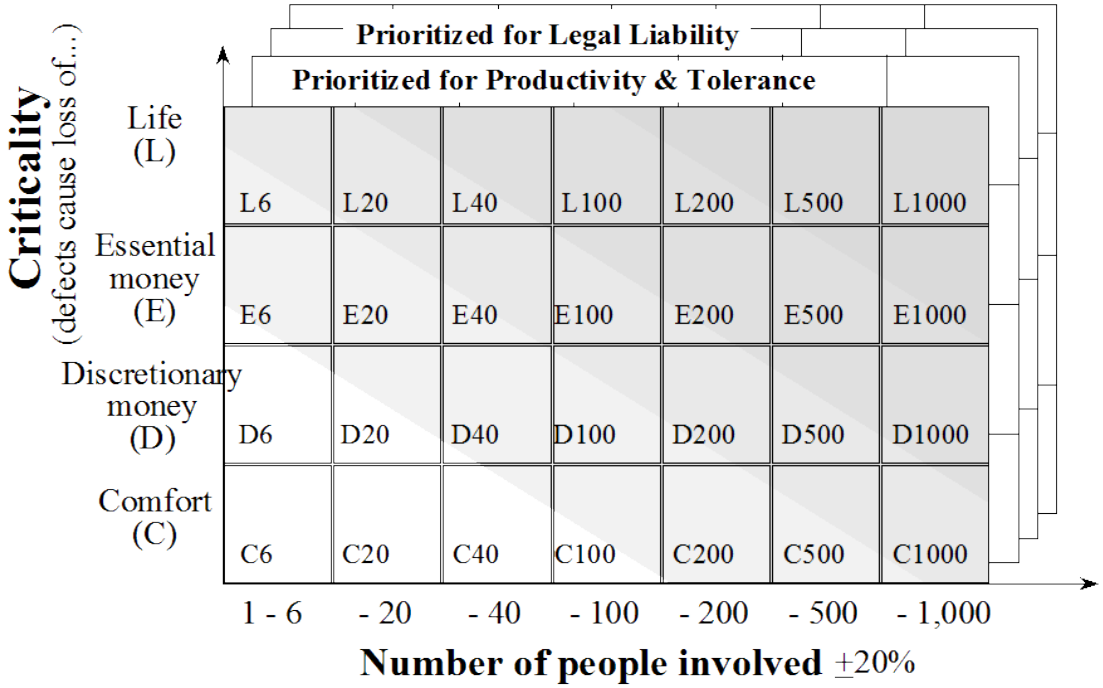
Jádrem všech metodik Crystal je síla komunikace a lehkost produktu. Jednotlivé metodiky jsou pojmenované podle barev a navzájem se odlišují svou vhodností pro různě velké týmy a různě složité projekty. To lze vidět na obrázku 2.5, vodorovná osa *x* určuje počet členů týmu a svislá osa *y* důležitost systému. Třetím rozměrem je hledisko, pro které je metodika optimalizovaná (produktivita, trasovatelnost apod.). Pro barvy jednotlivých metodik platí následující pravidlo, že čím tmavší barva je, tím je metodika robustnější a vhodnější pro rozsáhlejší, případně kritičtější projekty.

Vhodnost metodiky pro projekt se volí na základě tří následujících vlastností projektu:

- **Velikost vývojového týmu**, tedy kolik lidí se účastní vývoje.

- **Kritičnost projektu**, která ohodnocuje dopad selhání systému a může nabývat čtyř hodnot:
  - *Ztráta komfortu (C)* – nejnižší dopad.
  - *Malá ztráta peněz (D)* – má větší dopad.
  - *Velká ztráta peněz (E)* – tentokrát firma ztrácí nezanedbatelnou část financí.
  - *Ztráta lidských životů (L)* – nejhorší možný dopad.
- **Priorita projektu**, tato vlastnost vyzdvihuje stránku projektu, která má pro nás největší význam.

Crystal představuje skupinu metodik pro různé druhy projektů, které se liší důležitostí, velikostí týmu a kritériem optimalizace. Přínosem rodiny metodik je princip škálování podle typu projektu a důraz na lidský faktor.



Obrázek 2.5: Rodina metodik Crystal, zdroj [6]

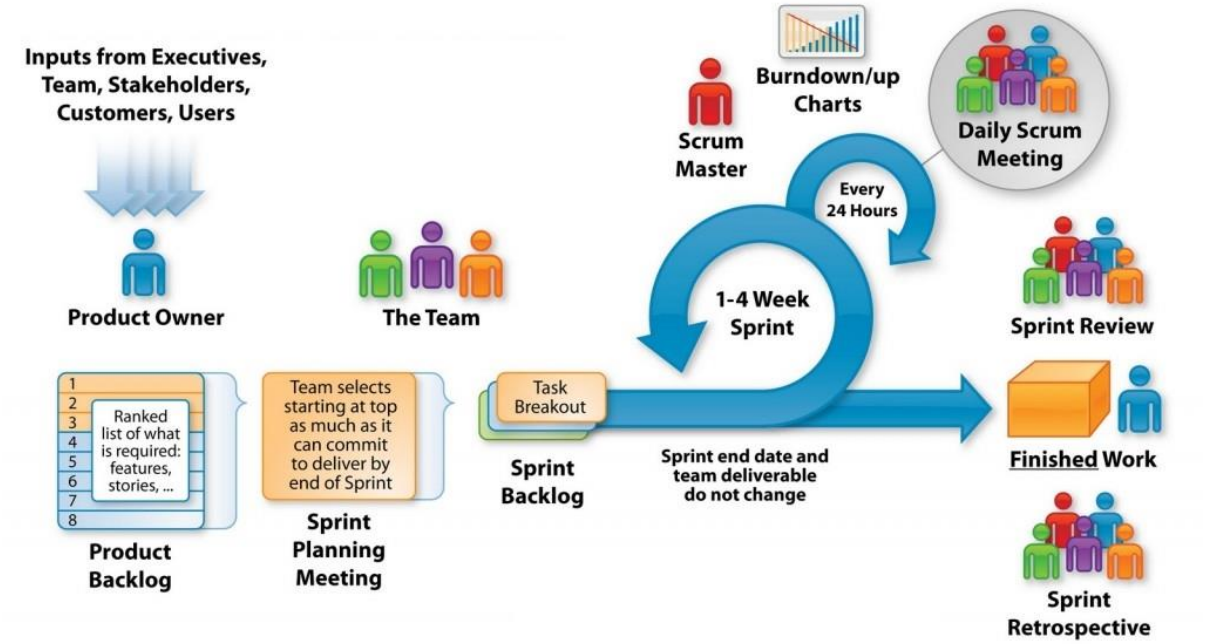
2.3.5 Ostatní metodiky

Agilních metodik je celá řada, liší se vhodností pro jednotlivé typy projektů nebo taky velikostí projektového týmu, který se na řešení projektu podílí. Dalšími metodikami jsou:

- **Dynamic Systems Development Method** – má ze všech metodik nejlépe propracovaný systém školení a kvalitní dokumentaci. Je populární jak v Evropě, tak v USA.
- **Adaptive Software Development** – „lehká“ metodika, kde je kladen důraz na fázi učení.
- **Agilní modelování** – aplikuje principy extrémního programování na oblast modelování.
- **Scrum, Kanban Scrum-ban** – těmto metodikám se budeme věnovat v samostatných kapitolách.

3 Scrum

Tato kapitola je věnována jedné z nejpopulárnějších agilních metodik a metodice Scrum. Hlavní specifičností této metodiky je zapojení do procesu všechny účastníky, přičemž každý účastník má svou roli. Základem je, že nad úkolem pracují všichni, kdo mají zájem o dosažení cíle. Popularita této metodiky vychází především z její jednoduchosti. Základní princip Scrum metodiky je na obrázku 3.1.



Obrázek 3.1: Princip Scrum metodiky, zdroj [9]

3.1 Role

V metodice Scrum se zavádí tři následující role, které se podílejí na celém procesu a jsou základním kamenem:

- Scrum mistr (Scrum Master)
- Vlastník produktu (Product Owner)
- Scrum tým

3.1.1 Scrum mistr

Nejedná se o vedoucího týmu v klasickém slova smyslu. Scrum mistr pracuje jako mezičlánek mezi týmem a jakýmkoliv rušivým elementem zvenku. Jedná se o nejdůležitější roli celé metody, je zodpovědný za úspěch na projektu. Scrum mistr nerozdává úkoly týmu, ale jeho cílem je vytvořit a udržovat samostatný, efektivní a samoorganizující tým. Cíle a povinnosti lze shrnout do následujících bodů:

- motivuje tým k lepším výsledkům
- odstraňuje problémy a překážky
- pomáhá týmu dosáhnout jeho cílů
- odpovídá za dodržování postupů a procesů v týmu
- chrání tým před vnějšími vlivy, které by ho mohly odvádět od soustředěné práce na definovaném cíli
- vytváří přátelskou a důvěryhodnou atmosféru

Scrum mistr provádí každodenní schůze a sleduje pokrok týmu pomoci zvolených artefaktů. Může se podílet na vytvoření produktového katalogu společně s vlastníkem produktu.

3.1.2 Vlastník produktu

Další rolí, kterou Scrum zavádí, je role vlastníka produktu. Jeho starostí je definování vize projektu a zajištění transparentní komunikace mezi týmem, zákazníkem a firmou. Je reprezentant strany zákazníka a člověk odpovědný za vývoj produktu. Primárním cílem vlastníka produktu je aby komunikoval tak, aby všichni věděli, kam jdeme, proč a jak, tedy seznámení s produktem. Shrnutím funkcí vlastníka produktu jsou:

- řídí návratnost investice (ROI)[1] celého projektu
- je zodpovědný za formování a vize produktu
- řídí očekávání zákazníků a všech zainteresovaných stran
- je prostředníkem mezi týmem a zákazníkem, spolupracuje s oběma
- je zodpovědný za vytvoření a koordinaci produktového katalogu
- je zodpovědný za přijetí kódu na konci každé iterace
- definuje priority projektu

Vlastník produktu je týmu k dispozici dle potřeby, ale na rozdíl od scrum mistra už s týmem ne vždy sedí ve stejné místnosti. Taky tato role je pro úspěch projekt klíčová, vyžaduje komunikativního člověka se silnou znalostí produktu, většinou se jedná o projektového manažera.

Zástupce vlastníka produktu (Product Owner Proxy)

Ve velkých korporacích může být vlastník produktu příliš vzdálený od týmu, třeba v případě, kdy jsou distribuované týmy po celém světě a společnost vyrábí globální produkty. Jedná se o prostředníka mezi vlastníkem produktu a týmem, tedy člověk na straně byznysu, ale zároveň je v denním kontaktu s týmem a rozumí komplexně celému systému. Musí zastoupit vlastníka produktu, odpovídat na otázky týmu, činit rozhodnutí a nést za tato rozhodnutí zodpovědnost.

[1]**Návratnost investice (ROI)** znamená poměr výnosu či změny hodnoty investice k investovanému kapitálu.

3.1.3 Scrum tým

Scrum tým je samoorganizující, multifunkční a vzájemně zastupitelný. Je zodpovědný před vlastníkem produktu za splnění všech úkolů v průběhu sprintu. Tým se sám dohodne na tom, kdo na čem bude pracovat, kdo komu pomůže a kterou znalost by si měl začít pozvolna budovat, aby byl jako celek co nejefektivnější. Funkcí scrum týmu jsou:

- zodpovědnost na ocenění produktového katalogu
- rozhodnutí o designu a implementaci produktu
- je zodpovědný za produkt před vlastníkem produktu
- společně s scrum mistrem pozoruje svůj pokrok

Velikost týmu je typicky do deseti lidí a skládá se z lidí s různými zručnostmi. Proto jsou všichni členové týmu v některé míře vývojáři, analytici nebo testeři. Tým je pak schopen sám sebe organizovat pro práci nad konkrétním úkolem, což dovoluje rychlé a pružné reagování na rozdílné typy úkolů.

3.2 Artefakty

Nejpoužívanější artefakty používané v metodě Scrum jsou:

- Sprint
- Produktový katalog (Product Backlog)
- Sprint katalog (Sprint Backlog)
- Burdown graf

3.2.1 Sprint

Iterace je základním principem, které agilní metody využívají. Nejznámější, která v metodě Scrum dostala pojmenování, je *sprint*. Vychází z toho, že pro lidi jsou pravidelně opakující se procesy, z psychologického hlediska příjemné a snadno si na něj zvyknou. Proto je vývojový proces ve Scrum rozdělen do pravidelných cyklů, *sprintů* ve kterých tým dodává hotovou funkcionalitu. Výsledky jsou pak prezentované pokaždé včas a vzhledem k procesu plánování dostaneme pokaždé to, co jsme čekali.

Na konci každého sprintu je *recenze (Review)* nebo také demo pro zákazníky, kde je prezentovaná hotová funkcionalita. Proto délka *sprintu* musí být taková, aby tým stihl něco dokončit a od prezentovat.

Sprint je tedy fixní časový úsek, ve kterém tým zpracovává požadovanou funkcionalitu z *katalogu sprintu* a na jehož konci tuto funkcionalitu prezentuje na zákaznickém demu v rámci *recenze*. Délka *sprintů* bývá obecně 1 – 4 týdny.

3.2.2 Produktový katalog

V každém projektu je potřeba zaznamenávat, co všechno se plánuje udělat. Tento seznam může být vytvářen softwarovým týmem, produktovým manažerem nebo přímo zákazníkem. Ve Scrum se takový seznam nazývá *produktový katalog* a má ho na starosti již výše zmíněný vlastník produktu. Jednotlivé položky v tomto seznamu by měly odpovídat jednotlivým funkcionalitám produktu, tedy něčemu co zákazníkovi přináší hodnotu.

Vlastník produktu je zodpovědný za obsah, dostupnost a prioritizaci katalogu, Scrum mistr, tým a všechny zúčastněné strany na vytvoření úplného *produktového katalogu* spolupracují.

Osvědčeným způsobem jak zaznamenávat funkcionalitu v katalogu je pomocí položek tzv. *User Story*. Funkcionalita je tedy vždy popsána z pohledu zákazníka, nikoli z pohledu technika nebo projektového manažera. Vlastník produktu pak používá *produktový katalog* během plánování sprintu, tak aby mohl vysvětlit klíčové body. Tým pak vybírá položky, které může udělat za jeden sprint.

Každý *produktový katalog* má určité vlastnosti, tak aby se lišil od obyčejného tzv. „To-Do" seznamu:

- Jedná se o živý dokument, který se může měnit během projektu.
- Položky v katalogu řadíme podle priority do pyramidy, na vrcholu jsou nejdůležitější položky.
- Každá položka má své ohodnocení.
- Každá položka má cenu pro zákazníka a vede k dokončení plánované funkcionality.

Položky katalogu - User Story

Položky popisují funkcionalitu, ne tu existující, ale tu kterou chceme přidat nebo změnit. Není to tedy seznam technických aspektů. Je zaměřena vždy na obchodní hodnotu a popisuje přínos, který od ní očekáváme.

User Story položka musí být jednoznačně popsatelná, vytvářet obrázek, nezávislá, přinášet hodnotu a také malá. Pokud je *User Story* příliš velká, tak může být těžké říct co je jejím obsahem a co už ne, pro tým tak může být neuchopitelná. Základní formát *User Story* je:

*„Jako **uživatel** chci **funkcionalitu**, abych dostal **byznys hodnotu**"*

Takto zapsaná *User Story* nám říká, co chceme dělat, ale i pro koho to chceme dělat a proč.

Plánovací poker (Plannig Poker)

Jedná se metodu k ohodnocení funkcionality, kterou agilní týmy využívají. Cílem hry a týmu je se domluvit. Vlastník produktu nejprve představí funkcionalitu tak, aby všichni členové týmu rozuměli funkcionalitě z obchodního pohledu. Poté každý člen týmu tajně ohodnotí funkcionalitu vlastním bodovým hodnocením (může vybírat z přesně dané sady bodového ohodnocení). Ohodnocení jsou zveřejněna až poté, co činnost dokončí všichni členové týmu. Poté je prováděna diskuze nad nejnižším

a nejvyšším bodovým ohodnocením. V praxi je tento postup obtížně aplikovatelný kvůli časové náročnosti, proto největší přínos je seznámení všech členů týmů s jednotlivou funkcionalitou.

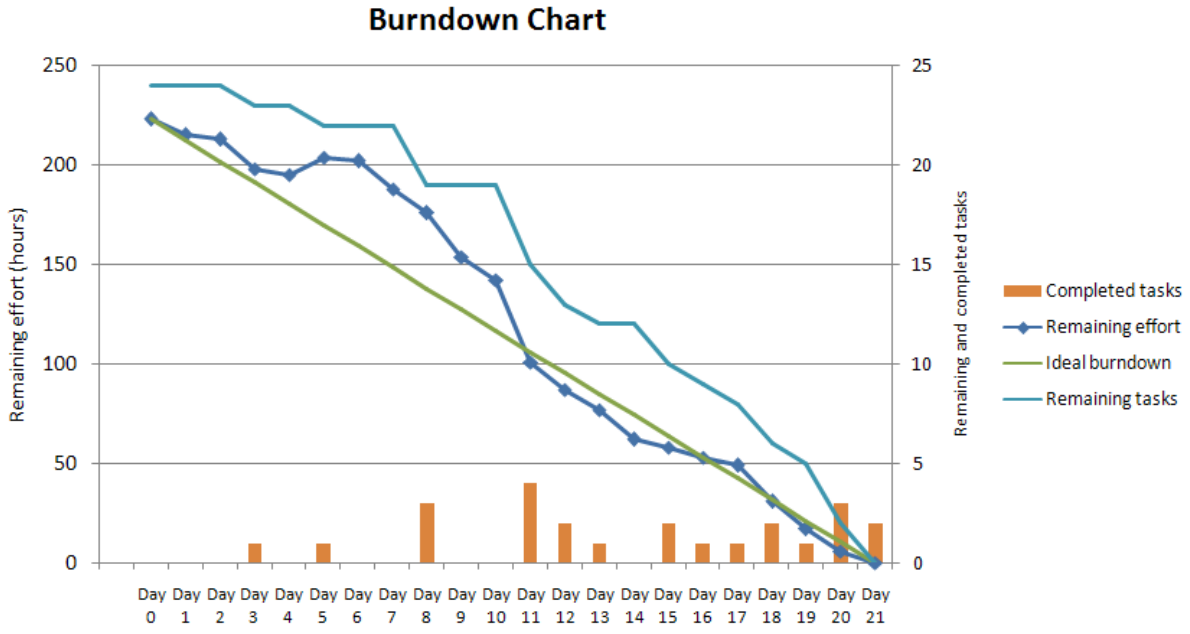
3.2.3 Katalog sprintu

Katalog sprintu je menší součástí produktového katalogu a obsahuje prioritní funkcionality, které se tým zavázal dodat v rámci jednoho sprintu. Je výstupem *sprint plánovací* schůze. Po ukončení sprintu je tento seznam dodán jako jediný přírůstek funkčnosti. Termín sprintu je pevně daný, proto jsou úkoly v tomto katalogu časově omezené a tým je zodpovědný za dodržování těchto termínů. *Katalog sprintu* zahrnuje všechnu práci, která vede k nezbytnému splnění cíle daného sprintu.

3.2.4 Burndown graf

Pro ukázání stavu celého projektu a predikci jeho dokončení se používá *Burndown graf*. Je to sloupcový graf, ze kterého každý sprint spálí určité množství bodů alias funkcionality. Jedná se o jeden z ukazatelů, zda vývoj produktu probíhá tak, jak má. Ukázka grafu je na obrázku 3.2.

Na vodorovné ose *x* je zobrazen čas a na ose svislé *y* součet bodů jednotlivých úkolů buď ve sprintu, nebo celého úkolu. Mezi počtem bodů a počtem dní obsažených ve sprintu je rovná čára, která



Obrázek 3.2: Ukázka Burndown grafu, zdroj Wikipedia

představuje ideální průběh. Z grafu lze vyčíst, zda tým pracuje příliš rychle nebo pomalu, podle čeho reaguje scrum mistr. Je-li postup na projektu rychlejší než plán, tak může scrum mistr přidat další úkoly. Naopak pokud je tým příliš pomalý, tak musí identifikovat důvody, proč se nedaří pracovat se stanovenou rychlostí. Jedno z řešení je odebrání některých úkolů zpět do produktového katalogu. Tyto přesuny může provádět pouze scrum mistr.

3.3 Činnosti v průběhu Scrum

Základní činnosti lze vidět na obrázku 3.1. Jedná se tedy o:

- Revize produktového katalogu (Product backlog refinement).
- Plánování sprintu (Sprint plannig).
- Denní Scrum schůze (Daily Scrum meeting).
- Recenze sprintu (Sprint review).
- Retrospektiva sprintu (Sprint retrospective).

Revize produktového katalogu

Jedná se o pokračující činnost v rámci celého Scrum projektu, činnost zahrnuje následující:

- revize dodržení pořadí úkolů v produktovém katalogu
- zrušení položek nebo snížení jejich úrovně, pokud již není důležitá
- vytvoření nové položky nebo zvýšení její úrovně, pokud se stává důležitější
- dekompozice položek do menších částí
- ocenění jednotlivých položek

Přínos této činnosti je především v přípravě na další sprinty, věnujeme zvláštní pozornost přípravě položek, které teprve budou přicházet k realizaci.

Plánování sprintu

Jedná se o schůzku týmu, kterou začíná každý sprint. Tým během tohoto setkání spolupracuje nad volbou prací pro budoucí sprint. Tým diskutuje nad každou položkou produktového katalogu, aby bylo dosaženo nejlepšího společného chápání dané problematiky, což je důležité k úspěšnému dokončení prací nad ní. Schůzka je časově omezená, proto je důležitá kvalita produktového katalogu, to je také důvod důležitosti předchozí revize katalogu.

Schůzka je rozdělená do dvou etap:

1. *Jakou práci budeme dělat?* – Rozhodnutí kolik položek bude zařazeno do sprintu, tým se rozhoduje na základě svých schopností, výkonosti a současné kapacity.
2. *Jak bude práce provedena?* – Provádí se design a plánování, prioritní práce jsou rozděleny do délky jednoho dne nebo méně.

Denní Scrum schůze

Jedná se o komunikační setkání v rámci týmu, tak aby bylo zajištěno, že jsou všichni na správné cestě. Během toho setkání mohou hovořit pouze členové týmu, scrum mistr a vlastník produktu. Další zainteresované strany se mohou setkání zúčastnit, ale mohou pouze poslouchat a sledovat jeho průběh.

Denní schůze jsou rozhodně klíčovým prvkem metodiky Scrum. Řídí se transparentností, důvěrou a lepším výkonem. Schůzky jsou časově omezené a neměly by trvat déle než 15 min. Doporučený formát je takový, že každý člen týmu odpoví na následující otázky:

- co jsem dokončil včera
- co dokončím dnes
- jaké vidím problémy

Recenze sprintu

Provádí se na konci každého sprintu, tým a zúčastněné strany přezkoumávají jeho výstup. Délka této schůze by měla být přibližně jedna hodina. Snažíme se získat zpětnou vazbu od zákazníka na dokončenou práci.

Ústředním bodem diskuze je tedy inkrement dokončený v průběhu daného sprintu. Tohoto neformálním setkání se může zúčastnit kdokoliv, ovšem rozhodnutí ohledně budoucnosti a aktualizaci produktového katalogu provádí vlastník produktu.

Provedení *recenze sprintu* je zcela v kompetenci týmu a sám tým musí najít svůj vlastní způsob, jak to provést. Dává každému přehled o současném produktovém inkrementu. Dalším důležitým prvkem je, že dochází k diskuzi na téma, jak probíhal právě dokončený sprint, jaké kdo měl nápady a připomínky, zkoumá se produktový katalog a mluví se o možném termínu dokončení dalšího sprintu.

Retrospektiva sprintu

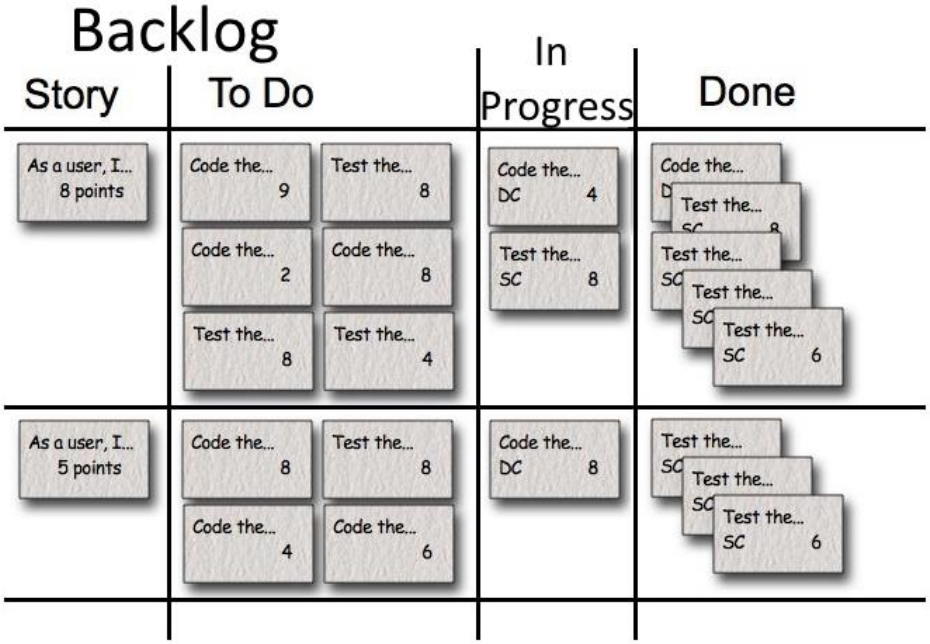
Retrospektiva je velmi užitečná agilní praktika, využívána i v neagilních týmech. Jedná se o efektivní nástroj získání zpětné vazby. *Retrospektiva sprintu* je na konci každého sprintu a její cílem je zhodnocení jak to dopadlo s ohledem na proces, vztahy mezi lidmi a nástroji. Doba trvání takové schůzky je přibližně jedna hodina a skládá se obecně z těchto fází:

- **Úvod** – připomenutí pravidel, dřívějších schůzek a potvrzení aktuálního plánu.
- **Sběr dat** – získávání informací: co funguje a co by šlo zlepšit.
- **Hlubší porozumění informacím** – identifikace příčin problémů, pochopení podstaty a hledání prostoru pro zlepšení.
- **Brainstorming možností** – základní kámen retrospektivy, abychom se mohli posunout dále.
- **Shrnutí konkrétních akcí** – závěr, shrnutí konkrétních kroků.

Retrospektiva pomáhá ke zlepšení vlastního týmového procesu, který vždy zůstane v rámci týmu.

3.4 Scrum tabule

Scrum tabule je jednoduchý vizualizační nástroj. Její provedení může být jako obyčejná tabule nebo zeď, kde se mohou lepit lístečky. Je velmi flexibilní, kdykoliv ji může kdokoliv změnit a začít s ní pracovat jinak.



Obrázek 3.3: Příklad Scrum tabule, zdroj vlastní

Jak taková tabule vypadá lze vidět na obrázku 3.3. Obecně stačí, když má tři sloupce:

- Katalog sprintu (Sprint Backlog)
- Ve vývoji (In Progress)
- Dokončeno (Done)

Na začátku sprintu se všechny naplánované funkcionality dají do sloupce Sprint Backlog a ideálně se rozpadnou ještě na jednotlivé úkoly. V průběhu sprintu se potom každý den jednotliví členové týmu zavazují k dokončení konkrétních úloh s tím, že se snaží minimalizovat rozpracované úlohy a dokončit již naplánované funkcionality.

V jednu chvíli může mít každý člen rozpracovanou pouze jednu úlohu, ale více členů může pracovat na jedné úloze. Abychom mohli pokračovat v práci na další úloze, musíme svou současnou úlohu dokončit, nebo někomu předat. Proto by měl mít každý člen svou unikátní značku, tak aby bylo z tabule hezky vidět, v jakém stavu která funkcionalita je a kdo na čem pracuje.

4 Kanban

Tato kapitola je věnovaná metodice Kanban. Kanban je původně technika pocházející z Japonska a řídil se jím počet lidí v chrámu. Princip byl takový, že kdo vstupoval do chrámu tak získal lístek, při východu ho zase odevzdal. Takto bylo dosáhnuto toho, že v chrámu nebylo nikdy více lidí, než byla povolená kapacit, protože bez lístku se do chrámu nemohl nikdo dostat. Posléze byl tento princip implementován v japonských továrnách na řízení výroby. Jednotlivé díly se dodávaly „just in time“, tedy až když byly potřeba, každý díl tak měl pouze omezenou frontu, kde mohly být připravené zásoby [1].

4.1 Základní charakteristika

Kanban je velmi flexibilní proces, který se používá tam kde se Scrum nebo XP úplně nehodí. V porovnání s ostatními agilními metodikami narážíme na problém, že Kanban nám v podstatě nic nenařizuje. Všechno si volíme sami a sami také rozhodujeme. Stačí dodržet tři základní principy:

- omezit rozpracovanou práci – work in progress
- minimalizovat čas průchodu – lead time
- provádět vizualizaci vývoje

Celý systém řízení vývoje je tedy postaven na vizualizaci průběhu práce, omezení paralelních činností a sledování nutné doby ke splnění jednotlivých požadavků ve snaze optimalizovat proces. Implementace Kanban se liší podle aktuální potřeby.

Principy Kanban

- **Začněte s tím, co děláte teď** - nepředepisuje konkrétní sadu rolí nebo procesních kroků. Vycházíme z rolí a procesů, které máme a používáme. Kanban je technika řízení změn.
- **Souhlas s evolučními změnami** – tým musí souhlasit se zavedením změn. Rozsáhlé změny se mohou zdát efektivněji, ale souvisejí s vyšší poruchovostí kvůli odporu a strachu v organizaci.
- **Respekt k aktuálnímu procesu, rolím a zodpovědnosti** – je potřeba předcházet nežádoucímu strachu v týmu. Prvky, které současně fungují a stojí za zachování, můžeme zachovat. To nám umožní širší podporu iniciativy v zavedení Kanban metodiky.
- **Podpora na všech úrovních** – musí být podporováno na všech úrovních organizace od vrcholného managementu až po jednotlivé členy týmu.

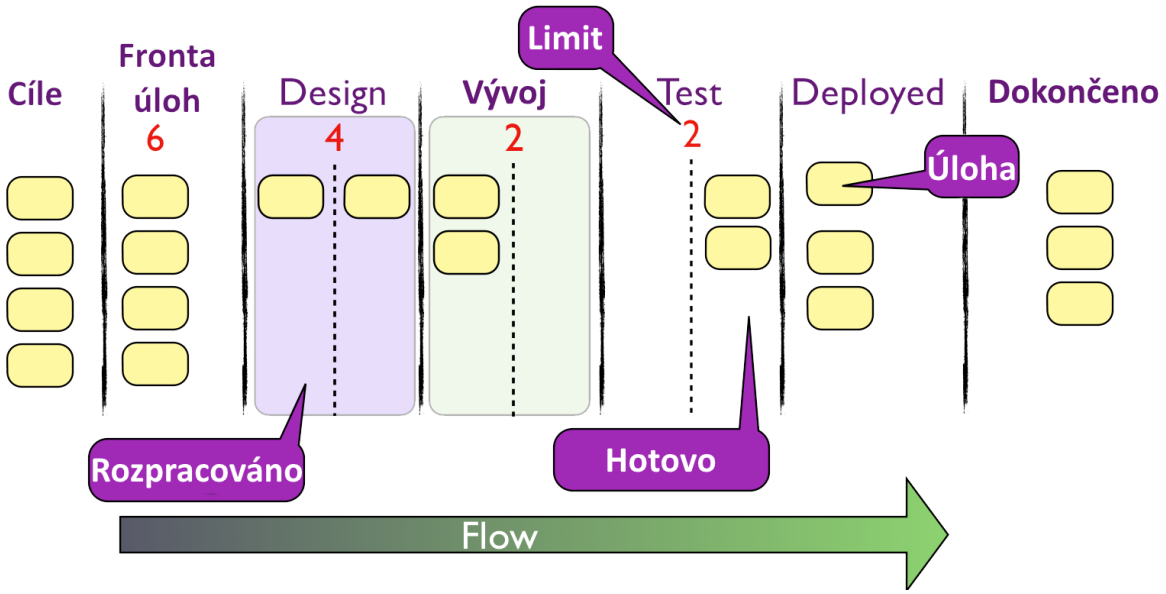
Praktiky Kanban

- **Omezení rozpracovaných prací** – přepínání kontextu nemá žádný pozitivní efekt, proto se k němu Kanban staví negativně. Říká, že je potřeba nastavit explicitní limit, kolik úkolů může být v daném stavu pracovního procesu (Workflow).

- **Vizualizace postupu** – Tok práce je neviditelný. Známým způsobem vizualizace toku práce je použití karet a desky se sloupci. Každý sloupec pak představuje jednotlivý stav nebo krok v toku práce. Vizualizace toku práce vede k pochopení, jak práce prochází.

4.2 Kanban tabule

Základním vizualizačním nástroje je *Kanban tabule*, která musí splňovat již výše uvedené základní principy. Tato tabule může vypadat podobně jako na obrázku 4.1.



Obrázek 4.1: Kanban tabule, zdroj vlastní

Sloupce jdou opět z leva doprava, stejně jak Scrum tabule.

- **Cíle projektu** – Nepovinný sloupec, kde se mohou uvést cíle projektu, tým pak vidí, co všechno se po nich chce.
- **Fronta úloh** – Uložení úkolů do fronty, které jsou připraveny k provedení. Vykonávána je úloha s nejvyšší prioritou.
- **Návrh (Design)** – Od tohoto sloupce až po „dokončeno“, se sloupce mohou střídat. Tým si rozhodne, kterými kroky musí úkol projít, než se dostane do stavu „hotovo“. Zde se může provádět návrh kódu, když je diskuze ukončena, úkol se přesune do dalšího sloupce.
- **Vývoj** – Zde se úkol pozdrží tak dlouho, doku nejsou dokončeny všechny jeho funkce. Po dokončení se přesune do dalšího sloupce, nebo se může vrátit do předchozího sloupce, pokud je nutné upravit architekturu návrhu.
- **Testování** – Pokud jsou nalezeny během testů chyby, úkol se vrací zpět. Pokud ne, tak postupujeme dále.
- **Nasazení (Deployed)** – Nasazení úloh.
- **Dokončení** – Po dokončení všech prací nad úkolem se nálepka může přesunout sem.

Pro naléhavé úkoly může být zavedeno zvláštní místo na desce. Jedná se o takzvanou expedici, která není na obrázku ukázána. Ta může obsahovat pouze jeden naléhavý úkol a práce nad ním musí nastat okamžitě, aby byly dokončeny co nejdříve.

Nejdůležitějším prvkem *Kanban tabule* jsou číslice uvedené v každém sloupci. Jedná se o limit úkolů, které mohou být současně v daném sloupci řešeny. Čísla se volí experimentálně a závisí na počtu členů v týmu.

4.3 Kanban vs. Scrum

Porovnání obou metodik podle [7].

Podobnosti

- Jedná se o Lean a Agile metodiky.
- Limitují rozdělané práce.
- Využívání transparentnosti pro řízení a zlepšování procesu.
- Používají stejný způsob plánování.
- Software je dodáván v pravidelných a krátkých intervalech.
- Samoorganizované týmy.
- Rozdělování prací na menší části.
- Plán verzí je optimalizovaný na základě empirických údajů.

Rozdíly

Scrum	Kanban
Časově omezené iterace jsou předem určené.	Časově omezené iterace jsou nepovinné.
Tím se zavazujeme dodat specifikované množství práce v dané iteraci.	Závazek je nepovinný.
Rychlost se používá jako standardní metrika pro plánování a zpracování zlepšení.	Jako standardní metrika pro plánování a zlepšení procesu se používá **čas průchodu (Lead Time).**
Jsou vyžadované multifunkční týmy.	Jsou povolené týmy specialistů.
Položky se musí rozdělit na menší, takže je možné jejich dokončení v rámci jednoho sprintu.	Není předepsaná velikost položek.
Je vyžadován Burndown graf.	Není předepsán žádný typ diagramu.
Rozpracovaná práce je limitována nepřímo sprintem.	Rozpracovaná práce je limitovaná přímo stavem.
Je vyžadovaný odhad.	Odhad je nepovinný.
Do rozpracované iterace není možné přidávat další položky.	Nové položky mohou být přidané kdykoliv to kapacita povoluje.

Katalog sprintu je vlastněn jedním týmem.	Kanban tabule může být sdílena více týmy nebo jednotlivci.
Jsou předepsané role.	Nepředpisuje žádné role.
Po každém sprintu je Srum tabule smazána.	Kanban tabule je trvalá.
Vyžaduje se produktový katalog s uvedením priorit.	Priority úkolů jsou volitelné.

Kanban se liší od Scrum především ve své orientaci na úkoly. Neexistují sprinty, tým tedy pracuje nad úkolem od začátku až do konce. Prezentace úkolu se provádí až po dokončení úkolu. Není zde potřeba odhadnutí času pro dokončení úkolu, protože by se stejně jednalo o nepřesný odhad. Manažer má za úkol vytvořit frontu prioritních úkolů, a tým musí co nejvíce z těchto úkolů vykonat. Kromě přidávání úkolů do fronty a určení jejich priority, není po manažerovi vyžadována žádná další práce.

Proč to funguje?

- **Proč funguje Scrum?**
  - multifunkční tým
  - časové omezení sprintu
- **Proč funguje Kanban?**
  - omezení paralelních prací
  - viditelnost procesu pro všechny
  - řízený pracovní postup (Workflow)

5 Scrum-ban

Srum-ban je metodika, založena na bázi tahu, která kombinuje prvky Scrum a Kanban. Jedná se o plnohodnotný Scrum s použití technik z Kanban.

- Použití Scrum, abychom byli agilní.
- Použití Kanban, abychom neustále zlepšovali svůj proces.
- Scrum-ban = scrum + kanban.

Tým méně pracuje nad plánováním práce, která již byla dříve akceptována během plánovací schůze a o to více se věnuje práci nad produktovým katalogem. Časově ohraničené sprinty v metodě Scrum-ban nejsou používané, ale například meetingy a jiné praktiky jako plánování, recenze, retrospektiva musí probíhat nadále, ale nepříliš často.

5.1 Principy Scrum-ban

Omezení rozpracovaných úloh (Work In Progress)

Ve Scrumu omezuje množství právě probíhajících prací délka sprintu. To Scrum-ban omezuje práci pomocí omezení rozpracovaných úloh (WIP) ve sloupcích stejně jako Kanban. Cílem Scrum-ban je posouvání karty na pracovní desce zleva doprava. Pokud počet karet v jednom sloupci přesáhne maximum, které je stanoveno, celý tým by měl začít tento úkol řešit, aby ho pomohl danou kartu posunout dále. Bez ohledu na to, jakou funkční roli mají členové týmu.

Plánovací schůze

Plánovací schůze by měly probíhat tak často, jak je potřeba. Pokud se nedaří týmu pravidelně splnit úkoly z vrcholu katalogu při standardním tempu, tak je schůze povinná.

Recenze produktu (Review)

Jedná se jediný způsob, jak získat od klienta zpětnou vazbu, která vede ke správnému pochopení toho, nad čím tým pracuje. Tyto schůze se zákazníkem, kde se prezentuje hotová práce, by měly probíhat v pravidelných intervalech.

Retrospektivní schůze

Obecně platí pravidlo, že by retrospektiva měla probíhat po každé recenzi. Jedná se o jednu z nejužitečnějších částí agilního procesu, a proto jí musíme věnovat pozornost.

Denní schůze (Stand-up meeting)

Nejužitečnější formátem schůze je zaměření na tok karet po desce. Otázky, které si klademe ve Scrum, můžeme aplikovat i na karty. Skupina karet se pohybuje mezi sloupci a každá karta se popisuje a zjišťuje se, co je nezbytné pro její pohyb směrem doprava. Tento přístup poskytuje pochopení týmu a informuje každého o významných architektonických a designových rozhodnutích.

5.2 Metriky pro Scrum-ban

Metriky pro Scrum-ban se zaměřují na měření nejzákladnějších ukazatelů, které mohou identifikovat silné a slabé stránky procesů. Většina měřených ukazatelů tedy souvisí s Kanban deskou, respektive s pohybem lístků po tabuli.

5.2.1 Měřené údaje

Počet úkolů v jednotlivých sloupcích

- Ukazatel důležitý pro plánování, pokud počet rozpracovaných úkolů klesne pod určitou mez, můžeme začít s plánováním další práce. Tak zamezíme situacím, kdy některý člen z týmu nebude moci na ničem pracovat. Počet úkolů má i svou horní mez a to z důvodu motivace členů týmu a omezení pocitu zahlcení, ale taky zejména proto, že existuje přímá úměra mezi poklesem kvality a množstvím rozpracované práce.

Doba zpracování požadavku od zahájení práce – Cycle time

- Doba potřebná k dokončení prací nad úkolem v jedné kartě, která je měřena od okamžiku, kdy se poprvé začala zpracovávat.
- V průběhu času se provede statistická analýza všech karet v rámci projektu. Tím získáme průměrnou délku cycle time a směrodatnou odchylku.
- Ukazatel důležitý z hlediska plánování na makroúrovni, tedy sečteme počet karet a vynásobíme průměrnou délkou cyklu.

Doba zpracování požadavků od vstupu do fronty – Lead time

- Lead time slouží pro sledování času vyřešení daného problému. Jedná se o čas z pohledu zákazníka, tedy doby kdy je požadavek opravdu zařazen do fronty, bez ohledu kdy se na něm začne pracovat.

Doba, kterou úkol strávil v jednotlivých sloupcích

- Ukazatel s cílem vytvoření dat o tom, jak dlouho běžně trvá přesun kartičky (dané kategorie a priority) z jedné fáze do druhé.

- Získaná data můžeme použít pro stanovení hranic, jejichž překročení nás upozorní na to, že s průběhem řešení úkolu pravděpodobně není něco v pořádku a nastalá situace by se měla řešit.

Kolik času na požadavku stráví jednotlivé role

- Jak dlouho lidé plnící určitou roli v týmu vlastnili kartičku v průběhu řešení.
- Jaký to byl druh úkolu.

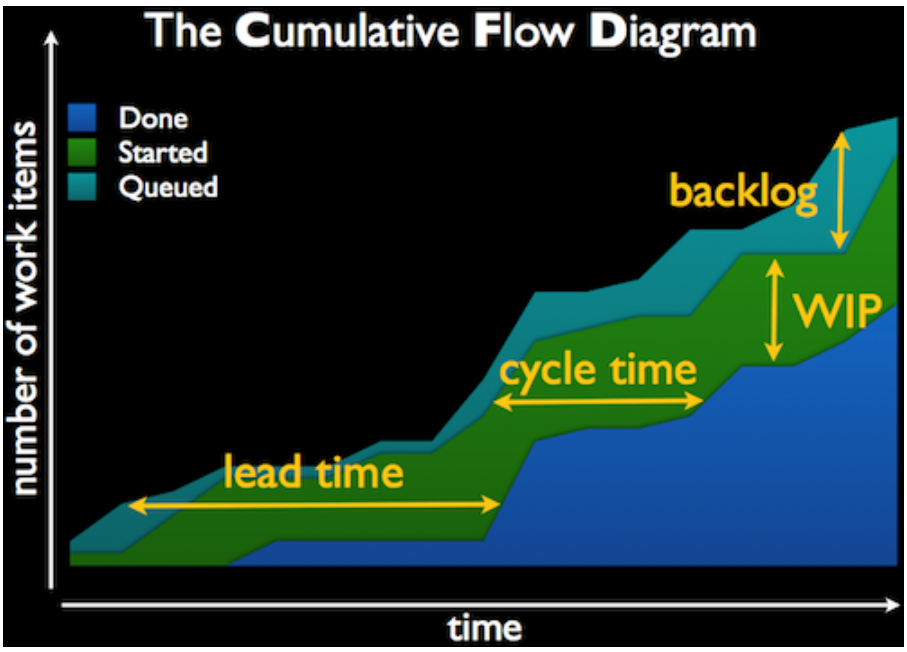
Z výše uvedených změřených údajů lze získat další data, která pomohou dospět k cíli celého měření, tedy identifikaci úzkých hrdel procesů.

Rozpracované práce

- Work in progress (WIP)
- WIP = celkový počet úkolů – (počet úkolů v produktovém katalogu + počet dokončených požadavků)

Distribuční rozdělení práce

- Pokud známe počet úkolů v jednotlivých sloupcích, lze pak zjisti, zda nedochází ke kumulaci úkolů v rámci jedné fáze. Pokud se tak děje, tak to znamená, že v dané fázi se nachází nějaký problém s dokončením práce, nebo že další fáze není připravena a zpracované požadavky nemůže přijmout.
- Pro vizualizaci se používá Kumulativní Flow Diagram. Lze z něho vyčíst průměrnou dobu cyklu a WIP.



Obrázek 5.1: Ukázka Kumulativního Flow diagramu [10]

Doba reakce

- Výpočet doby reakce je velmi jednoduchý.
- Reaction time = Lead time – Cycle time.
- Tento ukazatel nám říká, jak dlouhá doba uplyne, než se na nějakém požadavku skutečně začne pracovat. Na základě tohoto parametru lze hlídat, aby se důležité požadavky opravdu řešily dříve než ty méně podstatné.

5.2.2 Zhodnocení výsledků měření

Naměřené ukazatele mohou sloužit k predikci následujícího:

- **Délka cyklu požadavku** – pro jednotlivé kategorie úkolů jsou známé průměrné délky cyklu a směrodatná odchylka doby trvání. Při plánování lze pak predikovat, kolik času zabere jeho dokončení od doby, kde se na něm začne pracovat.
- **Doba řešení problémů –** tím, že sledujeme dobu řešení jednotlivých požadavků od vstupu do fronty, pak stejným postupem jako v předchozím bodu lze určit, za jak dlouho bude problém vyřešen od jeho zařazení do fronty.
- **Zapojení rolí při řešení požadavků –** pro konkrétní úkoly lze říci, kolik času na nich stráví konkrétní role v týmu. Můžeme tak určit zda je dostatek zdrojů pro řešení aktuální iterace a zda nedojde k přetížení některých rolí.

5.3 Porovnání Scrum, Kanban a Scrum-ban

Porovnání zvolených metodik, je uvedeno v tabulce 5.1.

	Srum	Kanban	Scrum-ban
Iterace	1 – 4 týdenní sprinty	Průběžná práce vedoucí k výsledkům v rámci jednoho týdne, nebo splnění cíle v delších iteracích	Průběžné práce v krátkých cyklech pro plánování a delších pro dosažení výsledků
Omezení práce	Omezení délkou sprintu	WIP limity	WIP limity
Odhad délky práce	Odhad na začátku sprintu	Volitelné	Volitelné
Metriky	Burndown graf	Kumulativní Flow Diagram, Lead Time, Cycle Time	Průměrná délka cyklu (Avarage Cycle Time)

Schůze	plánovací, denní, rcenze, retrospektiva	Není povinné	Pro zlepšení procesu a šíření znalostí
Role	Tradiční Scrum role	Nejsou předepsány	Tým + potřebné role
Členové týmu	Multifunkční tým	Multifunkční tým, jsou povoleny i specializace	Specializovaný tým
Rozsah úkolů	Úkoly odpovídající délce sprintu	Libovolně rozsáhlé	Libovolně rozsáhlé
Nové úkoly v rámci iterace	Zakázané	Povoleno vložení do fronty	Povoleno vložení do fronty
Tabule	Každý sprint se maže	Po celou dobu neměnná	Po celou dobu neměnná
Pravidla	Definované procesy	Flexibilní procesy, jen pár omezení	Mírně omezené procesy
Prioritizace	Prostřednictvím produktového katalogu	Volitelné	Doporučeno při každém plánování
Vhodné pro	Rozsáhlé týmové projekty, kde délka projektu je větší než rok	Projekty údržby, řízení výroby	StartUp projekty, vývoj nových produktů, projekty údržby

Tabulka 5.1: Porovnání Scrum, Kanban a Scrum-ban

6 Analýza existujícího projektu návrh zlepšení

Součástí práce je analýza existující projektu ve společnosti Siemens. Pro tuto analýzu byl vybrán energetický projekt EON EniM[1], zabývající se vývojem plně automatizovaných měřících zařízení elektřiny u koncového zákazníka pro energetickou skupinu E-On. Pro analýzu bylo využito nástrojů poskytující systém JIRA viz. kapitola 7.1. Na základě evaluace tohoto projektu a po konzultaci se zástupcem společnosti Siemens, jsem navrhl rozšíření, které přispěje k rozhodování během agilních procesů vývoje.

6.1 Měření a analýza získaných dat

Pro měření a analýzu stavu projektu jsem využil tří nástrojů, které poskytuje systém JIRA, jedná se o:

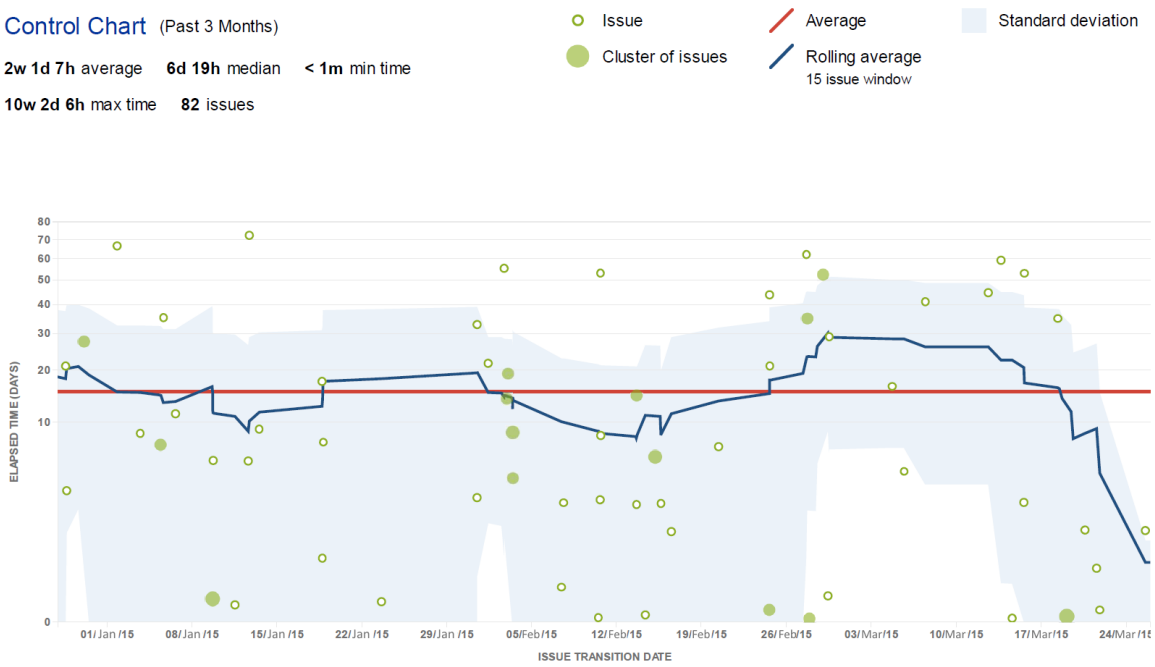
- Control Chart
- Kumulativní Flow diagram
- Burdown graf

Pro *Control Chart* jsem analyzoval projekt v období od začátku roku 2015 do konce března roku 2015. *Kumulativní Flow diagram* byl vytvořen pro dobu konání celého projektu a poslední *Burndown graf* byl zobrazen pro poslední sprint v rámci projektu.

Analýza Cycle Time

Pro tuto analýzu jsem použil nástroj *Control chart* (na obrázku 6.1). Jedná se o nástroj v sytému JIRA, který vizualizuje dobu zpracování od začátku práce (cycle time). Výhodou tohoto grafu je, že se automaticky vypočítá průměrná hodnota, medián a maximální hodnota. Na vodorovné ose *x* je doba trvání projektu a na svislé ose *y* je vyznačena doba zpracování jednotlivchých úkol§ v rámci projektu. Z grafu lze vyčíst, že průměrná doba zpracování úkolu je 2 týdny 1d a 7h. Oproti tomu medián této je 6d 19h. Což značí o tom, že se v projektu vyskytují úlohy, které mají výrazně vyšší dobu zpracování. Nejextrémnější hodnota je 10 týdnů 2dny 6h. To signalizuje, že se v projektu vyskytují špatně dekomponované úlohy. Ale jinak, většina úloh byla zpracována pod průměrnou hodnotou, tedy byly dekomponovány správně.

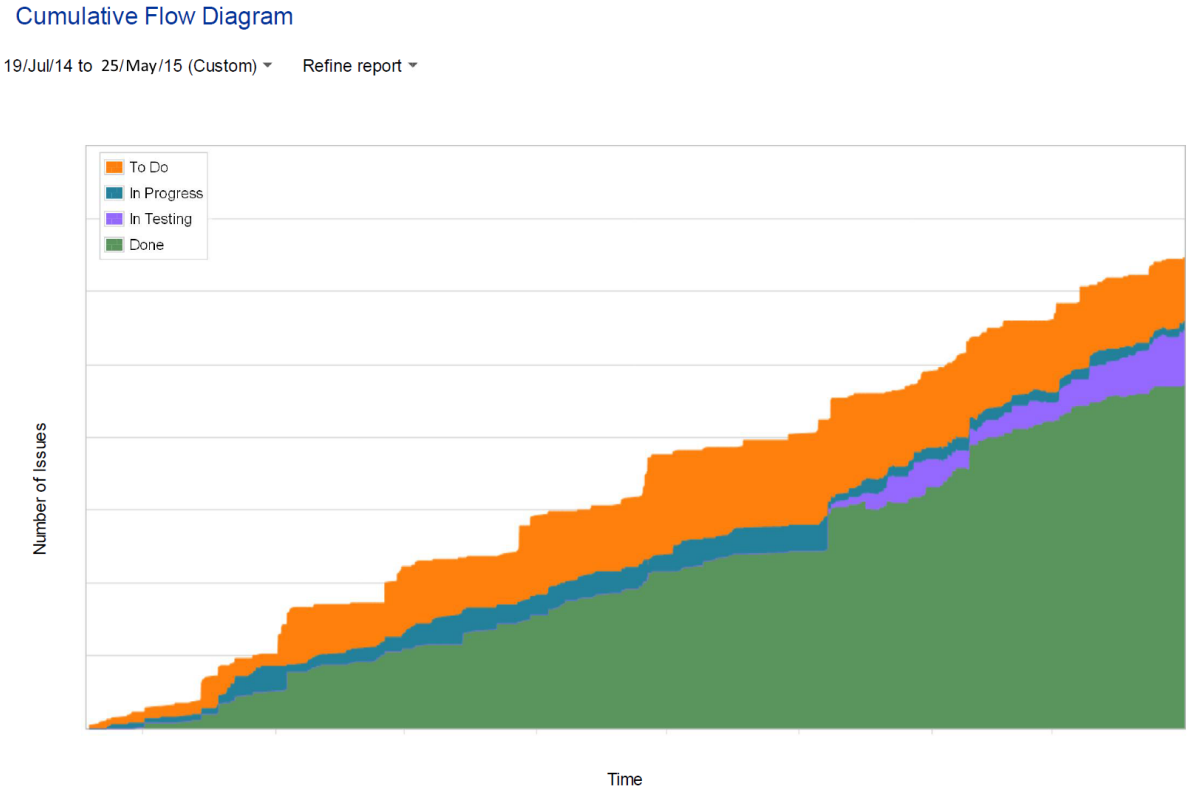
[1] Veřejné informace k projektu lze najít na webových stránkách:
http://www.siemens.com/press/pi/ICSG201407050e



Obrázek 6.1: Control Chart projektu EON EniM

Analýza kumulativního distribučního rozdělení práce

Pro tuto analýzu jsem využil kumulativního Flow diagramu. Tento diagram je na obrázku 6.2. Lze vidět, že množství úloh v čase přibližně lineárně rostlo. Další zajímavou informací je, že v poslední třetině projektu, je většina úloh spíše ve stavu testování, než ve stavu vývoje. To znamená, že počet

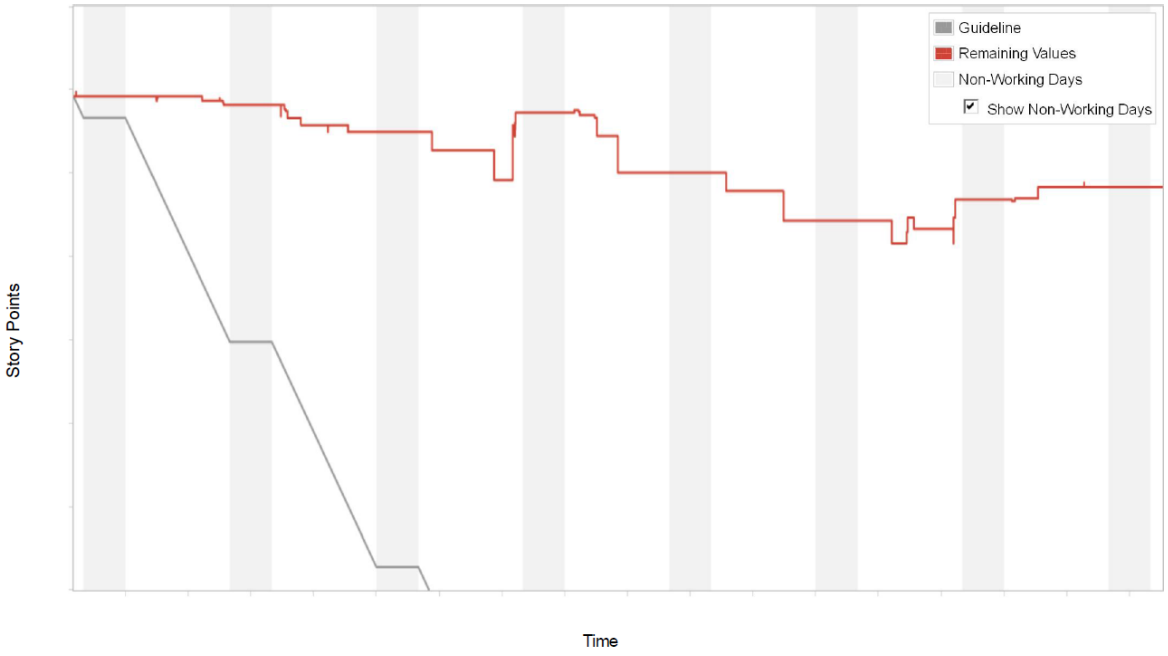


Obrázek 6.2: Kumulativní Flow diagram projektu EON EniM

zpracovávaných úloh z produktového katalogu se tak snižuje a naopak narůstá počet rozpracovaných úloh (WIP) a počet dokončených úloh se oproti minulosti snižuje, to značí zpomalování prací na dokončení projektu.

Analýza Burndown grafu

Posledním graf, na kterém jsem provedl analýzu je již známý *Burndown graf* sloužící k predikci dokončení prací ve sprintu. Graf je na obrázku 6.3. Z grafu jsem vyčetl, že průběh sprintu rozhodně neodpovídá předpovídané hodnotě a nedaří se tedy „spalovat“ úkoly podle plánu. To může souviset s předchozím měřením a to, že se zvyšuje počet rozpracovaných úloh, protože úlohy stagnují ve stavu testování.



Obrázek 6.3: Burndown graf projektu EON EniM (poslední sprint)

6.2 Návrh zlepšení

Předmětem zlepšení bude zásuvný modul pro systém JIRA. Tento modul byl zvolen na základě evaluace existujícího projektu a pak především toho, že požadavkem společnosti Siemens je analýza používaných procesů, tak abychom jednoduše mohli vyhodnotit problematické části projektů vzhledem k běžícím sprintům což současné nástroje systému JIRA neumožňují. Počet položek k implementaci byl zvolen s ohledem na časové omezení k vypracování diplomové práce tak, aby bylo modul možné nasadit aspoň v rámci testovacího provozu a vyhodnotit základní aspekty projektů, popřípadě odhalit chyby v implementaci. Na základě požadavků zástupce společnosti Siemens jsme se dohodli na implementaci modulu, který umožní vytvářet reporty (viz. 7.1.4) nad daným projektem.

Výstupem prvního reportu budou dva grafy zobrazující analýzu všech úloh (rozdělených podle typů) v rámci celého projektu. A to vzhledem k tomu kolik daná úloha absolvovala v rámci projektu

sprintů a kolik procent úloh bylo v rámci jednotlivých sprintů uvedeno do stavu „Resolved“ nebo „Close“. V rámci reportu budou zobrazeny i podpůrná data v tabulce pro identifikaci jednotlivých úloh.

Výstupem druhého reportu bude graf zobrazující analýzu všech úloh (rozdělených podle typy) v rámci jednotlivých sprintů. Pro každý sprint bude zobrazený průměrný čas, kolik úloha strávily času, než se dostaly do zvoleného stavu. V rámci reportu budou zobrazena i detailní data v tabulce, tak aby u každé úlohy byl zobrazen přesný čas.

Výstupem třetího reportu bude graf analyzující průběh konkrétního sprintu. Tedy vizualizace změny statusu každé úlohy v rámci běhu sprintu. Informace z toho grafu by měly doplňovat předchozí report. Výstupem reportu budou i detailní data pro každou úlohu, ukazující jednotlivé časy, kolik úloha strávila v daném stavu v průběhu sprintu.

7 Návrh a implementace modulu pro systém JIRA

Tato kapitola se zabývá popisem systému JIRA, seznámí čtenáře s potřebnými prostředky pro vývoj zásuvného modulu a na konec popisuje výslednou implementaci nástroje včetně postupů při testování funkčnosti.

7.1 JIRA

JIRA patří mezi nejoblíbenější nástroje pro podporu agilního vývoje. Výhodou je, že se jedná o velmi robustní nástroj, který nabízí širokou podporu pro různé metodiky vývoje. První verzi uvedla na trh australská společnost Atlassian již v roce 2002, což vypovídá o pevném zázemí produktu. Společnost Atlassian má dále ve svém portfoliu další produkty podporující vývoj, nasazení a správu software [8].

JIRA je multiplatformní webový nástroj přístupný přes prohlížeč. Provoz je závislý na licenci, může běžet na vlastních serverech uživatele anebo na serverech poskytovatele. Vytváří podporu pro projektový management správu defektů a požadavků. Koncept systému JIRA je v následujících podkapitolách.

7.1.1 Projekt

Projekt v systému JIRA je kolekce Issue[1] a odpovídá požadavkům dané organizace. Příklady projektů, které lze řešit s pomocí systému JIRA jsou:

- Vývoj softwarového produktu.
- Marketingová kampaň.
- HelpDesk systém.
- Pořádání rozsáhlé události (koncert, sportovní událost, apod.)

Každá Issue patří jednomu projektu a každý projekt má jméno a klíč. Klíč Issue se pak skládá z klíče projektu a pořadí. Demonstrace je na obrázku 7.1.

Komponenta projektu

Issue v projektu jsou rozděleny do logických skupin, kde každý projekt může obsahovat různý počet komponentů v závislosti na tom, jak potřebujeme. Ukázka je na obrázku 7.2.

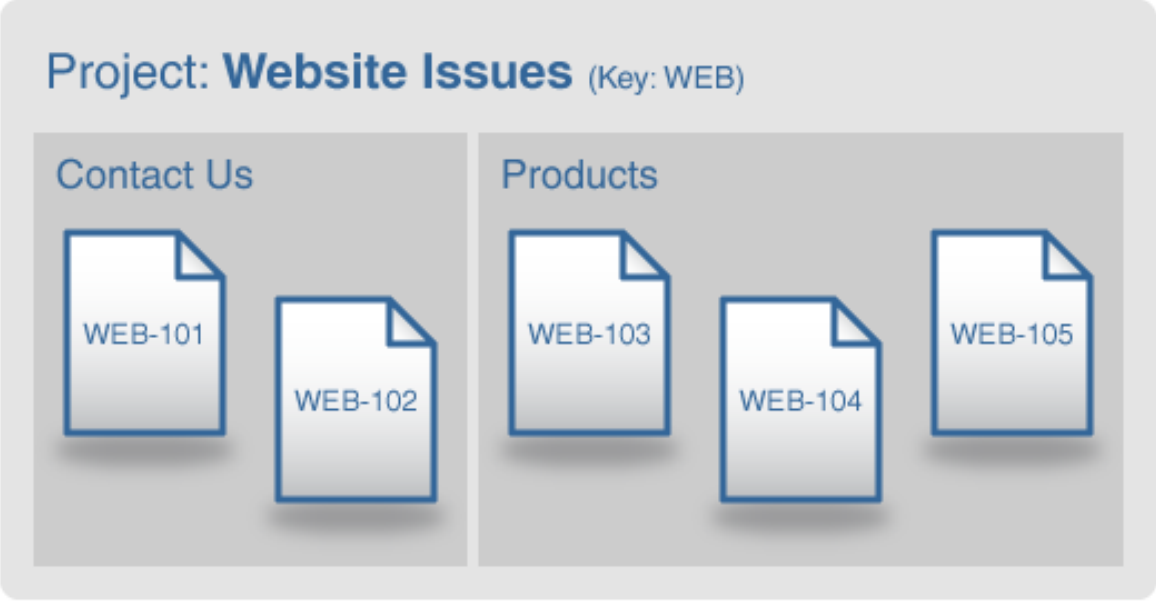
[1] **Issue,** přeloženo do češtiny problém nebo úkol. Je popsána v kapitole 7.1.3.

Verze projektu

Pro některé typy projektu je užitečné verzování projektů. Verze mohou být ve třech stavech: *Vydáno (Released), Nevydáno (Unreleased)* nebo *Archivováno (Archived).* To se využívá především u projektů, kde se vyvíjí softwarový produkt. Issue pak obsahuje dva záznamy:

- **Které verze Issue ovlivňuje (Affects Version)** – například softwarové chyby se nacházejí ve verzích produktů 1.1 a 1.2.
- **Verze, ve kterých je Issue vyřešena (Fix Version)** – například Issue se nacházela ve verzích 1.1 a 1.2, ale ve verzi 2.0 již byla vyřešena.

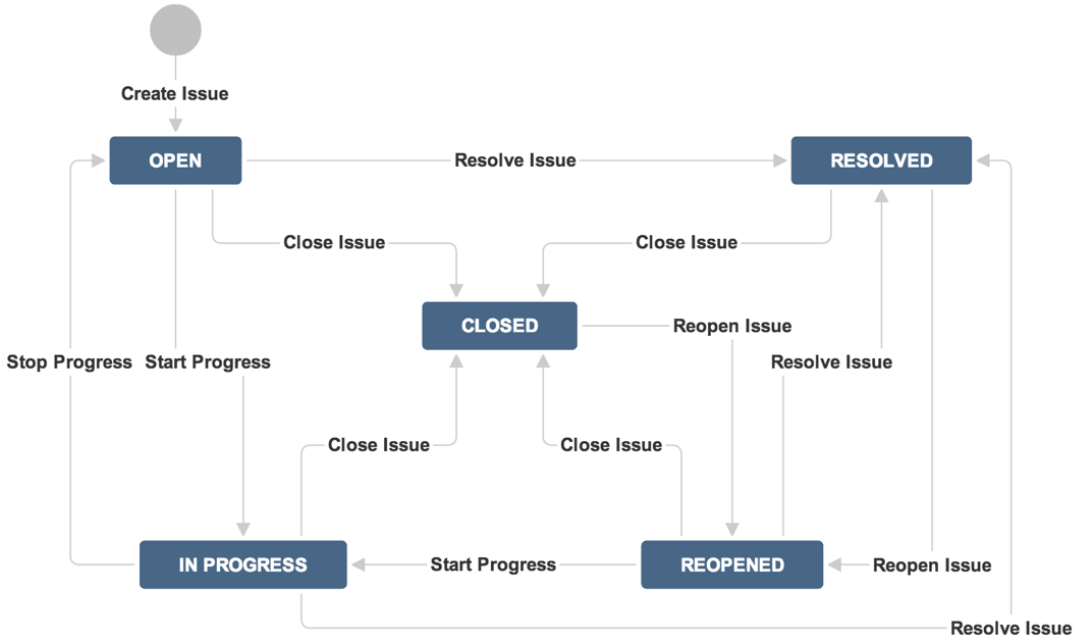
Obrázek 7.1: Ukázka konceptu projektu, zdroj [13]



Obrázek 7.2: Ukázka komponent projektu, zdroj [13]

7.1.2 Workflow

JIRA Workflow je sada stavů a propojení mezi nimi, kterými prochází Issue v rámci svého životního cyklu. Tento pracovní postup lze administrátorem samozřejmě libovolně upravovat dle potřeby aktuálního projektu. Na následujícím obrázku 7.3, je ukázáno implicitní Workflow pro JIRA projekty.



Obrázek 7.3: Implicitní Workflow projektu, zdroj [13]

7.1.3 Issue

Issue reprezentující jednotlivou úlohu či problém v rámci projektu, může být konfigurována přesně na základě požadavků dané organizace. Základní vlastnostmi Issue jsou rozvedeny v následujícím textu.

Issue Type

JIRA může využívat velké množství typů Issue. Implicitní typy jsou v následujícím seznamu, avšak tento seznam může být libovolně upravován administrátorem.

- **Bug (chyba)** – problém, který brání ve správné funkci výsledného produktu.
- **Improvement (vylepšení)** – vylepšení existující funkčnosti.
- **New Feature (nová funkčnost)** – nová funkčnost produktu.
- **Task (úkol)** – úkol, který musí být splněn-
- **Custom Issue** – libovolná Issue definovaná podle potřeby organizace.

Priority

Indikuje důležitost Issue. Implicitní seznam priorit je v následujícím seznamu, ale i tento seznam může být libovolně upravován administrátorem.

- **Highest (nejvyšší)** – tento problém zablokuje progres.
- **High (vysoká) –** indikuje, že Issue je příčinou problémů a vyžaduje pozornost.
- **Medium (normální)** – indikuje, že Issue má výrazný vliv na výsledný produkt.
- **Low (nízká)** – indikuje, že Issue má nízký vliv na výsledná produkt.
- **Lowest (nejnižšší)** – vliv Issue na produkt je minimální.

Status

Každá Issue se právě nachází v daném statusu, který odpovídá aktivnímu Workflow. Většinou Issue začíná ve stavu „Open" a končí ve stavech „Resolved" a pak „Closed". Seznam stavů může být libovolně upravován administrátorem.

- **Open (otevřeno)** – Issue je vytvořena a připravena k práci na ní.
- **In Progress (v procesu)** – Issue začala být aktivně zpracovávána.
- **Resolved (vyřešeno)** – Issue bylo dokončena a čeká na verifikaci od vedoucího.
- **Reopened (znovu otevřeno)** – Issue již byla vyřešena nebo uzavřena, ale byla opět vrácena pro nesplnění funkčnosti.
- **Closed (dokončeno)** – Issue je dokončena a potvrzena vedoucím.

Resolution

Issue může být dokončena mnoha způsoby, jenom jeden z nich je stav „Fixed" indikující, že byla Issue úspěšně implementovaná. Resolution je nastaveno ve chvíli, kdy je změněn status Issue. Seznam Resolution může být libovolně upravován administrátorem.

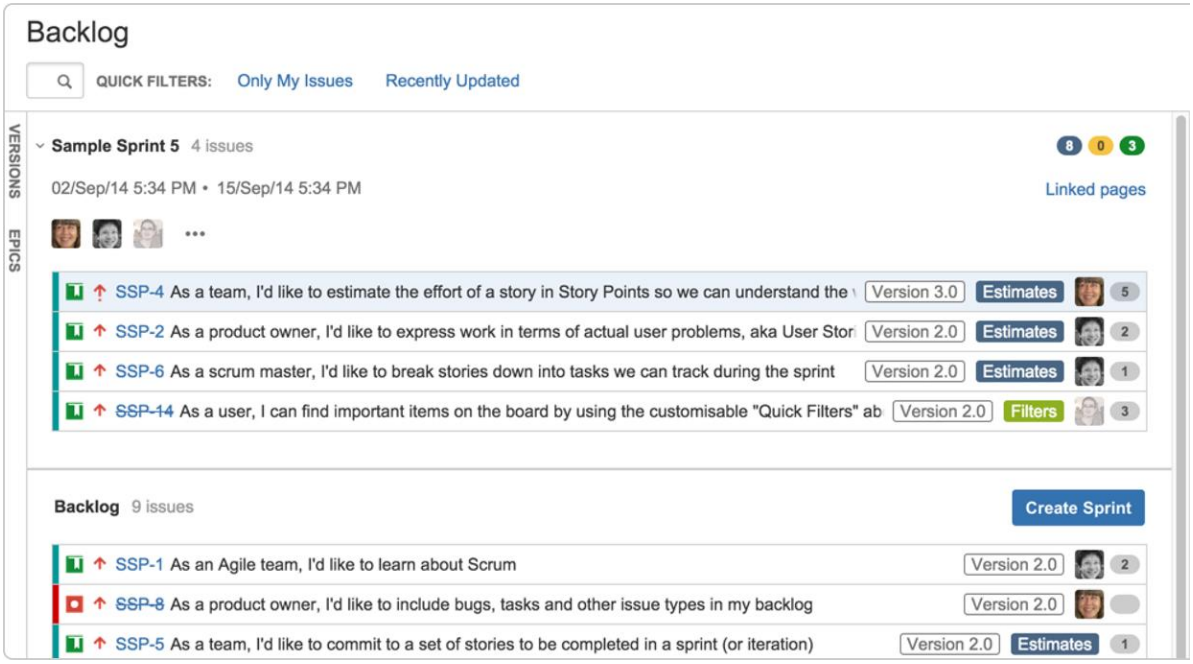
- **Fixed** – Issue byla úspěšně dokončena a byly splněny požadavky.
- **Won't Fix** – Issue nelze zpracovat.
- **Duplicate** – Issue je kopií existující Issue.
- **Incomplete** - není dostatek informací pro dokončení práce na Issue.
- **Cannot Reproduce** – Issue nemůže být dokončena nebo právě není dostatek informací k dokončení prací nad úkolem. Pokud získáme více informací, bude Issue znovu otevřena.
- **Won't Do** – stejné jak Won't Fix, ale je k dispozici pouze u softwarových projektů.

7.1.4 JIRA Agile

JIRA Agile je placený rozšiřující modul, který přidává do systému rozsáhlou kolekci funkcí podporující agilní vývoj projektů pomocí metodik Scrum a Kanban. Pro administraci a průběh projektu jsou pak nejdůležitější tři níže popsané obrazovky.

Backlog (produktový katalog)

Obrazovka, kde se vytváří, plánují a organizují sprinty konkrétního projektu. Uživatel zde může vytvořit libovolný počet sprintů, do kterých se pak přiřazují jednotlivé Issue. U každého sprintu pak lze naplánovat jeho počáteční a koncové datum, avšak v jednom projektu může být aktivní pouze jeden sprint. Na obrázku 7.4 lze vidět typickou ukázku této obrazovky. Máme zde spuštěný „Sample Sprint 5“, kterému jsou přiřazeny 4 Issue. Lze vidět taky datum spuštění sprintu a předpokládaný konec sprintu. Dole pod naplánovaným sprintem je zbytek produktového katalogu, ve kterém se nachází 9 Issue.



Obrázek 7.4: Ukázka plánování sprintů na obrazovce Backlog, zdroj [14]

Scrum Board (Scrum tabule)

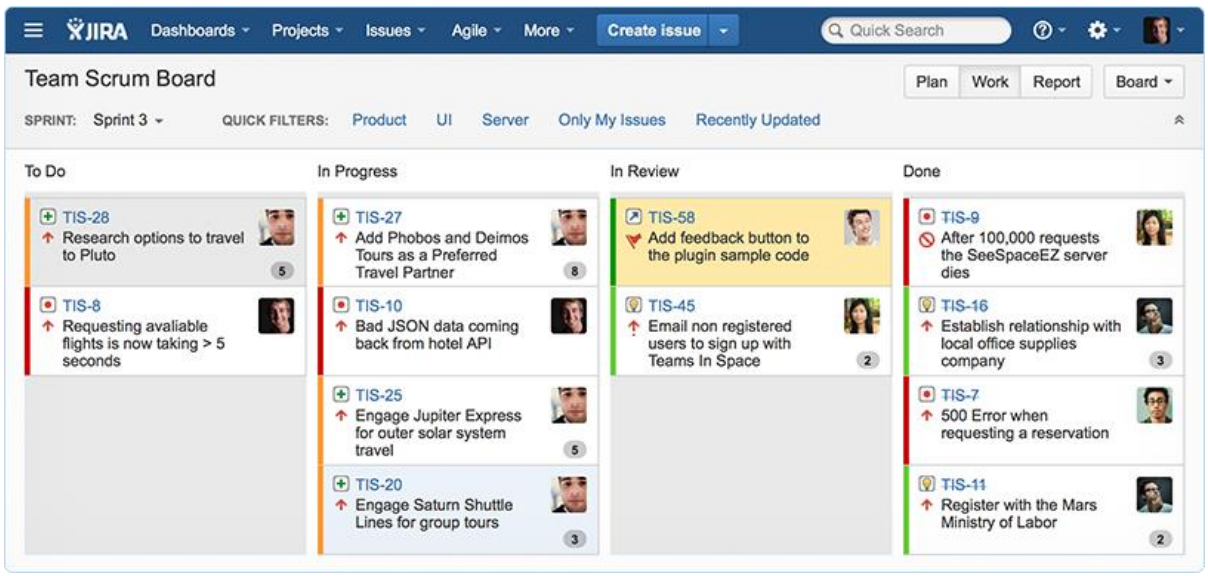
Jedná se o nejdůležitější obrazovku při vývoji pomocí Scrum. Slouží pro zobrazení všech Issue aktuálního sprintu a jejich stavů. Členové týmu tak získávají rychlý přehled o celkovém stavu sprintu, což vede k efektivnějšímu rozhodování při dělení práce. Na obrázku 7.5 lze vidět typickou ukázku obrazovky. Je zde aktivní „Sprint 5“, kde se rozpracované úkoly mohou nacházet ve čtyřech různých stavech. Na Issue můžeme aplikovat různé filtry, které nám jsou k dispozici.

Sprint Reports

Posledním nástrojem je vytváření reportů aktivního sprintu. To se provádí na obrazovce Sprint Reports. K dispozici máme sadu základních reportů, které slouží k základní analýze aktuálního ale i minulých sprintů. Základní sadou reportů jsou tyto:

- **Burdown Chart** – vytvoření Burdown grafu daného sprintu.
- **Control Chart –** slouží k analýze Cycle Time nebo Lead Time.

- **Cumulative Flow Diagram –** vytvoření Kumulativního Flow Diagramu.
- **Epic Report** – ukazuje list dokončených a nedokončených Issue v rámci jedné Epic.
- **Epic Burndown** – vytvoření Burdown grafu zvolené Epic.
- **Release Burdown** – vytváření Burdown grafu týkajícího release výstupu.
- **Sprint Report** – zobrazuje statistiku sprintu s pohledu dokončených a nedokončených Issue, je vhodný pro retrospektivu sprintu.
- **Velocity chart –** zobrazuje graf množství provedené práce v každém sprintu a může sloužit k predikci množství provedené práce v budoucnu.
- **Version report** – zobrazuje postup týmu v porovnání vydaných verzí projektu.



Obrázek 7.5: Ukázka Scrum tabule na obrazovce Scrum Board, zdroj [8]

7.2 Prostředky pro vývoj zásuvného modulu

Společnost Atlassian je velmi otevřená tomu, aby vývojáři mohli vytvářen podpůrné rozšíření pro jejich produkty. Tyto rozšíření pak lze buď přímo nasadit v rámci fungující organizace nebo dokonce prodávat na „Atlassian Marketplace“, což je webový obchod s rozšířeními pro produkty společnosti Atlassian.

Vývojář si může vybrat jaká je jeho cílová platforma a na základě toho je mu poskytnutý příslušný soubor vývojářských nástroj (SDK). Náš modul je plánován pro nasazení na vlastním produkčním serveru, tomuhle řešení odpovídá níže popsaný Atlassian Plugin SDK. V následujících podkapitolách bude přiblíženo, jak probíhá vývoj rozšiřujícího modulu pro systém JIRA.

7.2.1 Atlassian Plugin SDK

Atlassian Plugin SDK nabízí sadu skriptů pro vytváření, instalování a kompilaci rozšiřovacích modulů (plugin) týkajících se produktů společnosti Atlassian. Získat ho lze jednoduše na webových stránkách

společnosti Atlassian, kde je volně ke dostupný ke stažení. Pro zprovoznění a instalaci pod systémem Windows je potřeba mít nainstalovaný Java Development Kit, který obsahuje nástroje pro vývoj aplikací pro platformu Java. Vzhledem k tomu, že produkty společnosti Atlassian jsou webové aplikace, tak je také potřeba zkontrolovat, zda jsou volné příslušné síťové porty, na kterých pak spuštěné aplikace implicitně pracují.

Jak už bylo zmíněno, tak Atlassian Plugin SDK je řízen sadou skriptů pro příkazový řádek, z nichž nejdůležitější jsou tyto:

- `atlas-run` – spustí aplikaci a nainstaluje do ní vyvíjený modul.
- `atlas-debug` – spustí aplikaci v ladícím módu a nainstaluje do ní vyvíjený zásuvný modul.
- `atlas-compile` – zkompiluje zdrojové kódy projektu.
- `atlas-release` – vytvoří výsledný produkt vyvíjeného modulu.
- `atlas-package` – zkompiluje a zabalí vyvíjený modul do JAR balíku.
- `atlas-mvn` – spouští Maven[1] příkazy.
- `atlas-create-jira-plugin` – vytváří kostru pro vytvoření rozšiřujícího modulu systému JIRA.
- `atlas-create-jira-plugin-module` – přidává do existujícího projektu další moduly, rozšiřující funkčnost vyvíjeného zásuvného modulu pro systém JIRA.

Další obrovskou výhodou tohoto vývojového nástroje je velmi podrobná dokumentace a řada návodů pro tvorbu aplikací.

Struktura projektu zásuvného modulu pro systém JIRA

Adresářová struktura projektu vypadá následovně a odpovídá implicitní adresářové struktuře pro nástroj Maven:

```
./plugin
./plugin/pom.xml
./plugin/src
./plugin/src/test/java
./plugin/src/test/resources
./plugin/src/main/java
./plugin/src/main/resources
./plugin/src/main/resources/css
./plugin/src/main/resources/images
./plugin/src/main/resources/js
./plugin/src/main/resources/templates
```

[1] **Maven** je nástroj pro správu, řízení a automatizací vytváření aplikací.

`./plugin/src/main/resources/atlassian-plugin.xml`

`./plugin/src/main/resources/plugin.properties`

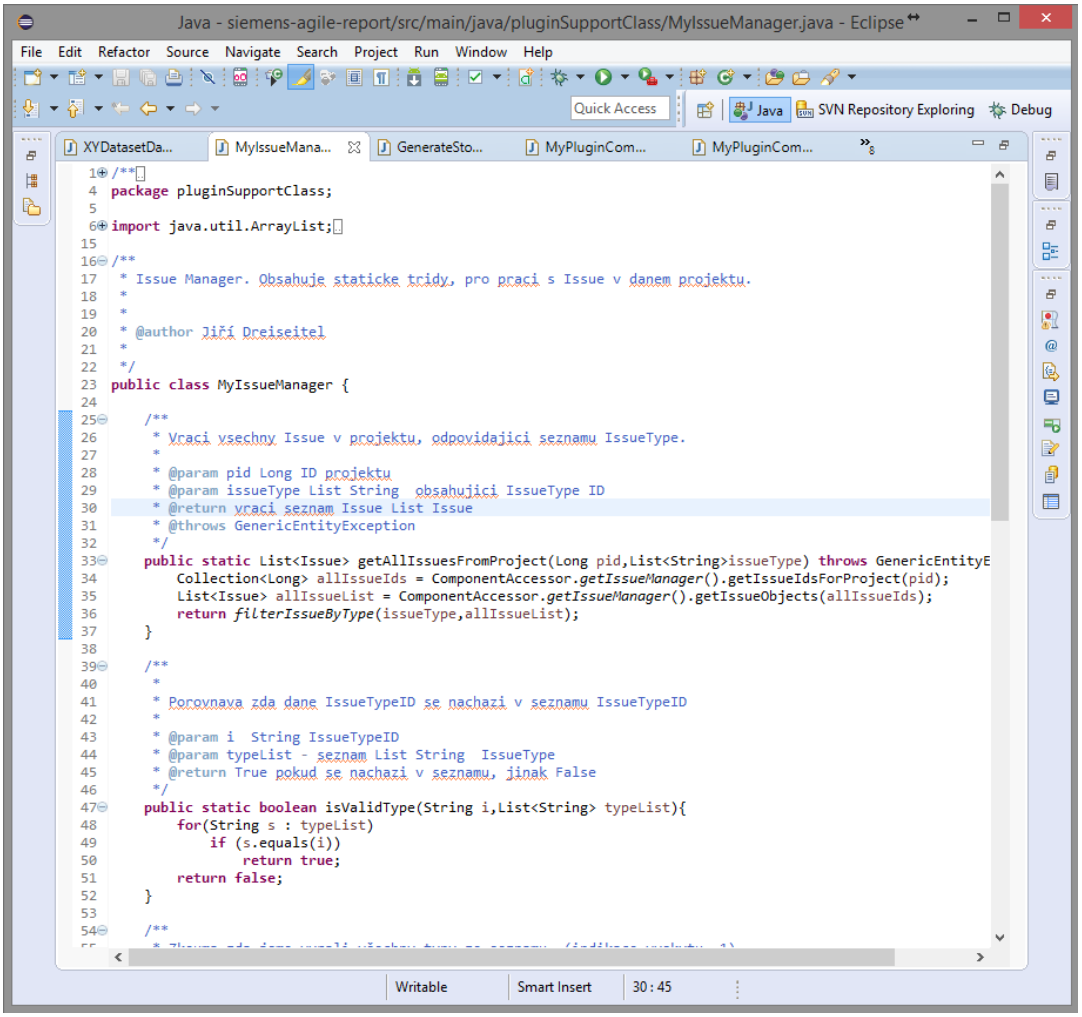
V kořenovém adresáři se nachází XML soubor `pom.xml`, reprezentující **Project Object Model**. Tento model popisuje softwarový projekt jak z pohledu zdrojového kódu, tak i z pohledu závislostí na externích knihovnách, popisu procesu kompilace projektu a různých funkcí s tím spojených. Adresář `src/test/java` obsahuje třídy testů. Adresář `src/test/resources` obsahuje konfigurační soubory pro testy. Důležitějším adresářem je `src/main/java`, který obsahuje kompilované java soubory a adresář `src/main/resources` obsahuje další soubory včetně konfiguračního souboru `atlassian-plugin.xml`. Tento soubor popisuje strukturu aplikace pro zásuvný modul. V konfiguračním souboru jsou popsány důležité elementy jako komponenty, které modul obsahuje a dále popisuje zdroje, které modul využívá. Konfigurační soubor je rozdělen do těchto sekcí:

- `atlassian-plugin` – kořenový element konfiguračního souboru, další elementy jsou jeho potomky.
- `plugin-info` element – obsahuje informace zobrazované administrátorovi, parametry modulu a OSGi bundle instrukce, je potomkem `<atlassian-plugin>` elementu.
- `description` element – popis zásuvného modulu, potomek elementu `<plugin-info>`.
- `version` element – obsahuje informace o aktuální verzi zásuvného modulu, je potomkem `<plugin-info>` elementu.
- `vendor` element – obsahuje odkaz na administrátorskou obrazovku zásuvného modulu, je potomkem `<plugin-info>` elementu.
- `param` element – obsahuje další parametry zásuvného modulu.
- Zbylé elementy v konfiguračním souboru obsahují jakékoliv další moduly, které jsou součástí celého zásuvného modulu.

Při vývoji se používá kombinace kódu v jazyce Java, XML, HTML (JavaScript) a dále speciálního skriptovacího jazyka VTL (Velocity Template Language), který slouží pro odkazování ve webových aplikacích na objekty definovány v kódu Java. K implementaci zásuvného modulu se dále využívá **JIRA Java API (v6.3.15)** a **JIRA Agile Java API (v.6.6.70)**, které poskytují funkce pro přístup k objektům systému JIRA. Pro vytváření grafů byl použit open-source Framework **JFreeChart (v1.0.17)**, pro programovací jazyk Java. Tvorba základní kostry modulu byla vytvořena na základě návodu z knihy [9].

7.2.2 Vývoj modulu v prostředí Eclipse

Velkou výhodou při vyvíjení zásuvného modulu pro systém JIRA je, že lze projekt snadno importovat do open-source vývojové platformy Eclipse. To se provádí jednoduchým příkazem `atlas-mvn eclipse:eclipse`. Po provedení tohoto příkazu je možné projekt importovat. K vývoji byla použita aktuální verze Luna (v4.4.2.) Výhoda využití vývojového prostředí spočívá ve využití a integraci verzovacího systému SVN a dále možností použití režimu Ladění (Debugging), nehledě na to, že Eclipse poskytuje plno další nástrojů pro vývoj aplikací, jako refaktorizaci, měření metrik kódu, revizi kódu a mnohé další. Ukázka práce ve vývojovém prostředí Eclipse je na obrázku 7.6.



Obrázek 7.6: Ukázka práce ve vývojovém prostředí Eclipse, zdroj vlastní

7.3 Zásuvný modul Siemens Agile Report

Tato podkapitola se zabývá konečným návrhem modulu a jeho implementací. Popis je dále doplněn ukázkami uživatelského rozhraní a analýzou zobrazovaného výstupu.

7.3.1 Specifikace požadavků

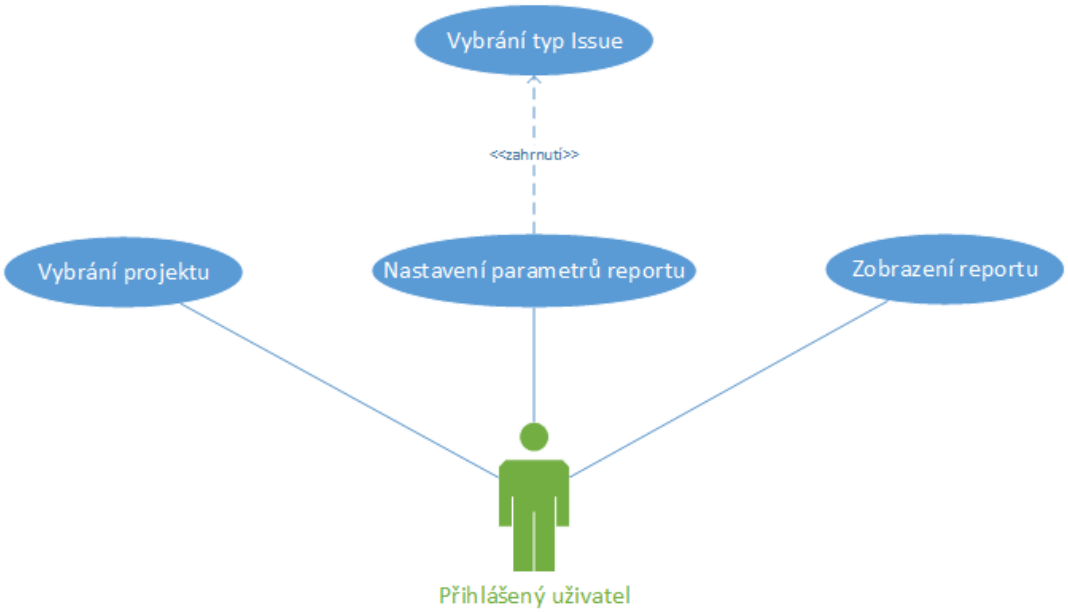
Zásuvný modul pojmenovaný anglicky jako **Siemens Agile Report** je stěžejní částí diplomové práce. Modul se skládá z tří částí, kde každý podle nastavení specifikovaného uživatelem, spouští příslušné reporty. Pro rychlou specifikaci jednotlivých reportů budou využity use case diagramy dle definice UML.

Report Issue In Sprint Charts

U reportu zobrazující analýzu Issue vzhledem k tomu, kolik absolvovala projektů a kolik Issue bylo v každém sprintu dokončeno, může uživatel reportu nastavit parametr, pro které typy Issue nás analýza zajímá viz. 7.1.3.

Výstup analýzy kolik Issue absolvovala sprintů, má být přehledný sloupcový graf tak aby bylo možné jednoduše vyčíst potřebné informace. Položky grafu musí být rozděleny podle parametrů reportu, tedy podle typy Issue. Dále je potřeba zobrazit tabulku s doplňujícími informacemi k příslušnému grafu. Tabulka by měla být rozdělena na části dle parametru reportu, tedy podle typu Issue a pak jednotlivé skupiny Issue rozdělit podle počtu absolvovaných sprintů. Ke každé Issue je potřeba zobrazit doplňující informace jako klíč, popis, ohodnocení Issue, priorita a aktuální status.

Výstup analýzy kolik procent úkolů bylo dokončeno v rámci každého projektu, má být přehledný sloupcový graf, tak aby bylo možné vyčíst potřebné informace. Položky grafu budou rozděleny opět



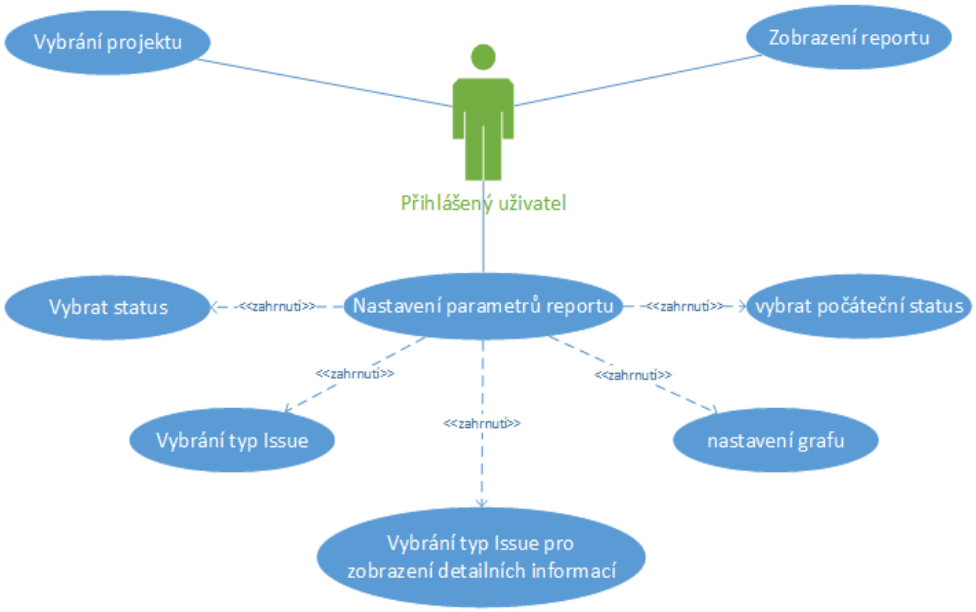
Obrázek 7.7: Use case reportu Issue In Sprint Charts

podle parametrů reportu stejně jak u výše uvedeného grafu. Tabulka zobrazující doplňující informace bude členit data podle sprintů, kde u každé Issue bude uveden klíč, typ Issue, popis, ohodnocení Issue, priorita a aktuální status.

Report Issue Spend Time Before Get do Selected Status

Pro report, který provádí analýzu toho, jak dlouho trvá než se Issue dostane do vybraného stavu, je požadováno, aby uživatel mohl nastavit následující parametry. Vybrat počáteční stav z aktivního Workflow, které je přiřazeno vybranému projektu. Dalším parametrem je stav, do kterého se Issue v průběhu sprintu měla dostat. Issue lze opět filtrovat podle typu tak jak u předchozího reportu a pak uživatel vybírá, pro které typy Issu chce zobrazit detailní informace. Posledním parametrem je nastavení grafu, tedy v jakých jednotkách bude zobrazován čas.

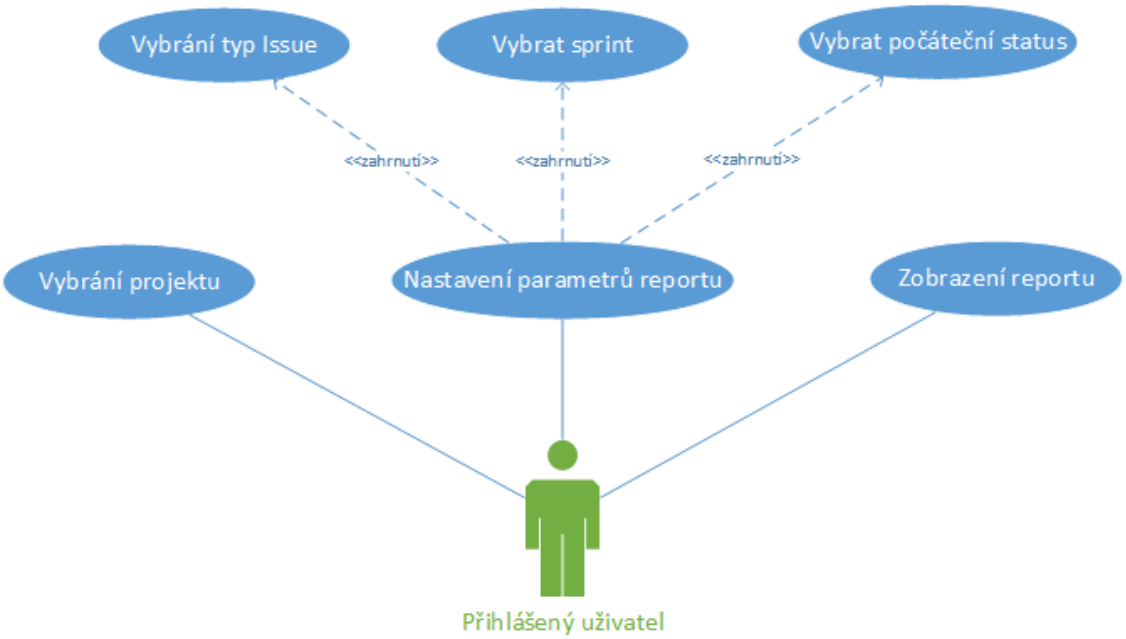
Výstupem reportu má být přehledný sloupcový graf zobrazující data tak, aby bylo možné vyčíst potřebné informace. Položky grafu budou rozděleny podle parametrů reportu, tedy podle typu Issue, které uživatel vybral. Tabulka zobrazující doplňující informace bude členěna na jednotlivé sprinty, další členění v tabulce bude podle parametru reportu, tedy typu Issue, u kterých chce uživatel zobrazit detailní informace. U každé Issue pak bude zobrazen klíč, čas než se dostala do vybraného stavu, její popis, bodové ohodnocení a priorita.



Obrázek 7.8: Use case reportu Issue Spend Time Before Get to Selected Status

Report Issue Spend Time In Status

Pro report, který provádí analýzu konkrétního sprintu, je požadováno, aby uživatel vybral sprint, pro který bude analýza prováděna a potom typy Issue, pro které bude průběh sprintu zobrazen. Dalším parametrem, který uživatel musí nastavit je počáteční stav aktivního Workflow vybraného projektu.



Obrázek 7.9: Use case reportu Issue Spend Time In Status

Výstupem reportu je zobrazení *XY* časového grafu zvoleného sprintu, kde budou zaznamenány změny stavu každé Issue (v trvání sprintu), která byla vybrána k analýze. Pro tento graf bude vytvořena také doplňující tabulka, kde u každé Issue bude zobrazen klíč, typ Issue, bodové ohodnocení a kolik času strávila v daném stavu v průběhu sprintu.

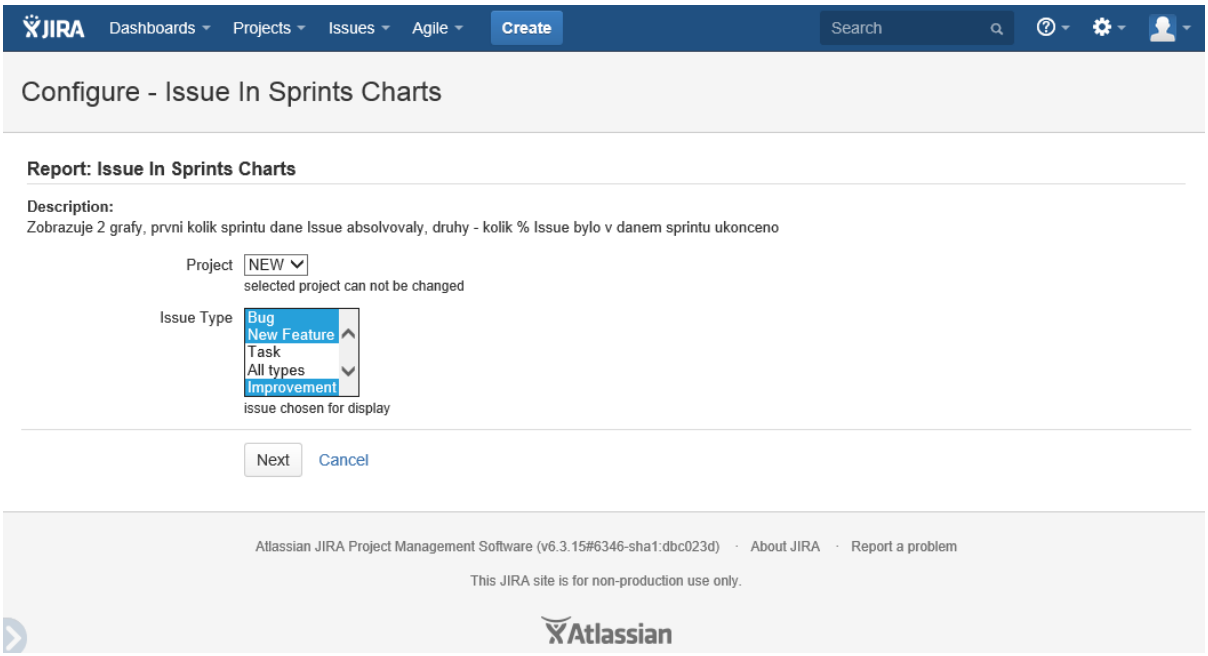
7.3.2 Report Issue In Sprint Charts

Po spuštění[1] reportu se nám zobrazí obrazovka odpovídající navrhovanému případu užití. Na této obrazovce může uživatel nastavit parametry reportu. U reportu *Issue In Sprint Charts*, může uživatel nastavit pouze typy Issue, kterých se analýza má týkat. Jedná se o typy přiřazené k vybranému projektu a uživatel může vybrat jednu až více možností. Pokud uživatel nevybere žádnou, automaticky se analýza provede pro všechny typy úkolů. Ukázka úvodní obrazovky reportu je na obrázku 7.10.

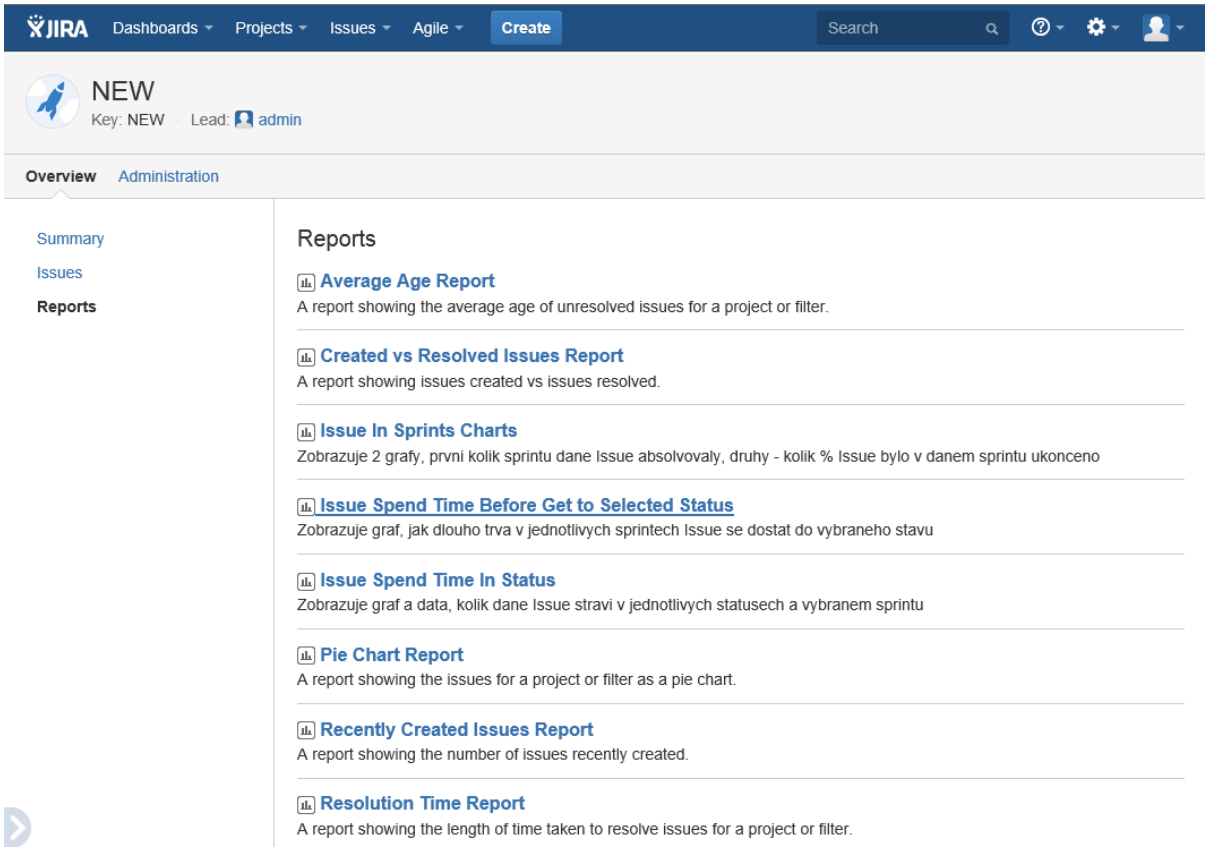
Analýza výstupu

Výstupem reportu jsou dva grafy ukázané na obrázku 7.12. První graf zobrazuje kolik sprintů jednotlivé Issue absolvovaly. Na vodorovné ose *x* je počet sprintů a na svislé ose *y* je odpovídající počet

[1] Reporty se spouští z obrazovky reportů v přehledu projektu, jak je ukázáno na obrázku 7.11.



Obrázek 7.10: Obrazovka reportu Issue In Sprint Charts

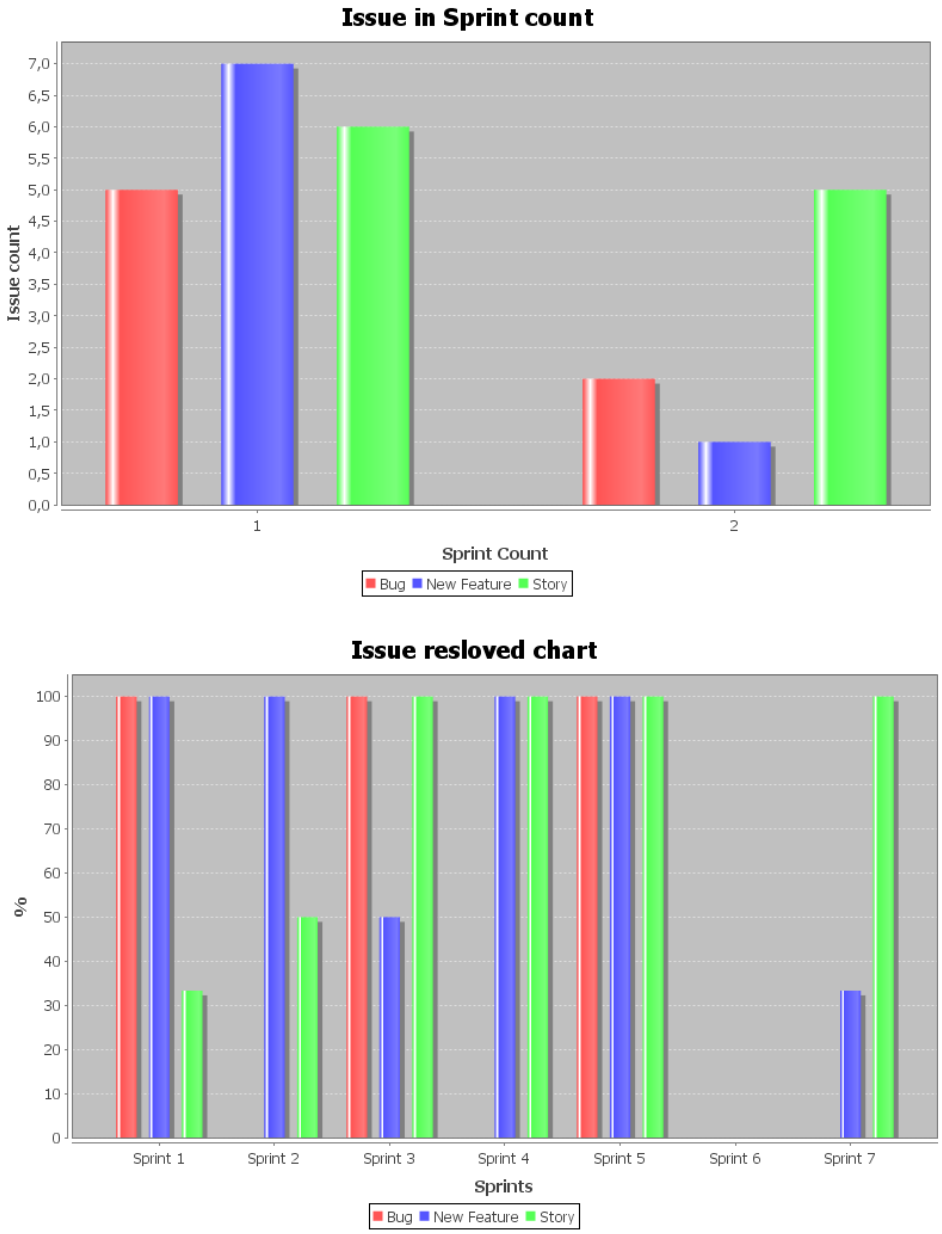


Obrázek 7.11: Obrazovka dostupných reportů v přehledu vybraného projektu

úloh. Sloupce jsou rozděleny podle typu Issue, jak bylo nastaveno v parametru reportu. Z prvního grafu můžeme vyčíst, že všechny vytvořené úkoly v projektu absolvovaly většinou jeden maximálně dva sprinty. Například pro Issue, typu Bug, můžeme z grafu vyčíst, že většina chyb byla opravena během

jednoho sprintu. Ideálně by graf měl vypadat tak, že většina Issue je vyřešena během jednoho sprintu a se zvyšujícím se počtem absolvovaných sprintů by mělo množství úkolů ideálně lineárně klesat. Pokud se stane, že se nějaké úkoly neustále „přelévají“ mezi jednotlivými sprinty, tak to indikuje závažný problém. Poté je potřeba v doplňující tabulce dohledat co jsou to za konkrétní Issue a popřípadě jaké je jejich bodové ohodnocení. Může se stát, že ulohy s vysokým hodnocením mohou být neustále odkládány, to by pak mělo vést tým k tomu, že daný problém byl špatně dokomponován a bylo by ho potřeba rozdělit na větší množství podúloh, tak aby se snížilo její bodové ohodnocení.

Druhý graf nám poskytuje doplňující informace k projektu. Na vodorovné ose *x* máme jednotlivé sprinty a na svislé ose *y* procentuální hodnotu dokončených[1] úkolů v daném sprintu. Sloupce jsou opět

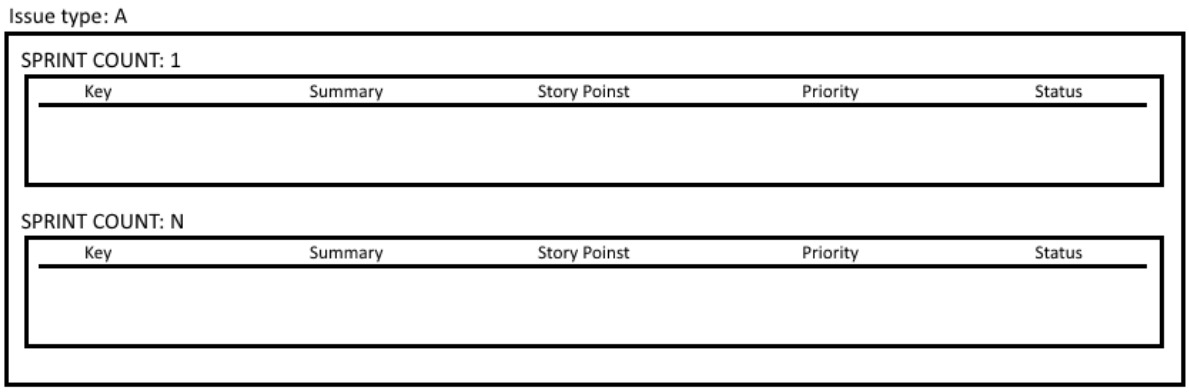


Obrázek 7.12: Grafy na výstupu reportu Issue In Sprints Charts

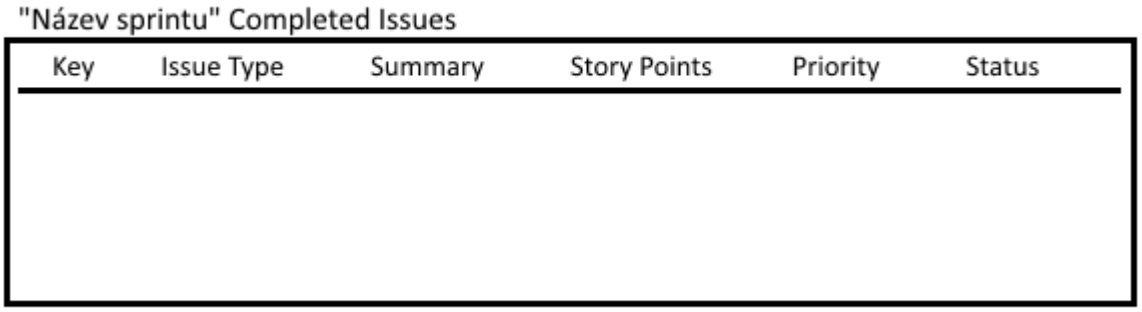
[1] Dokončených znamenám, že byly uvedeny do stavu Resolved nebo Close.

členěny podle typu Issue. Ideální stav je, kdy je v každém sprintu dokončeno 100% všech Issue, naopak hodnota 0% by vypovídala o nějakém závažném problému. Graf vypovídá především o tom, na jaké typy úloh se tým v konkrétním sprintu zaměřil. Pokud by nastala situace, že jsou upřednostňovány jedny typy úloh před druhýma (například řešení nových Feature, před řešením chyb) tak je potřeba tento aspekt zahrnout do plánování. Tak aby Issue byly řešeny rovnoměrně a úkoly byly správně dekomponovány.

Tabulka zobrazující doplňující informace pro první graf je na obrázku 7.13. Takováto tabulka je vytvořena pro každý typ Issue zvolený v parametrech reportu, kde pro každý počet absolvovaných sprintů (osa *x*) jsou vypsané konkrétní Issue s doplňujícími informacemi dle specifikovaných požadavků. Na obrázku 7.14 je tabulka zobrazující detailní informace pro druhý graf. Tahle tabulka je vytvořena pro každý sprint (osa *x*) a v každé této tabulce je kompletní výpis dokončených úloh v průběhu konkrétního sprintu. U každé Issue jsou opět zobrazeny informace dle požadavků.



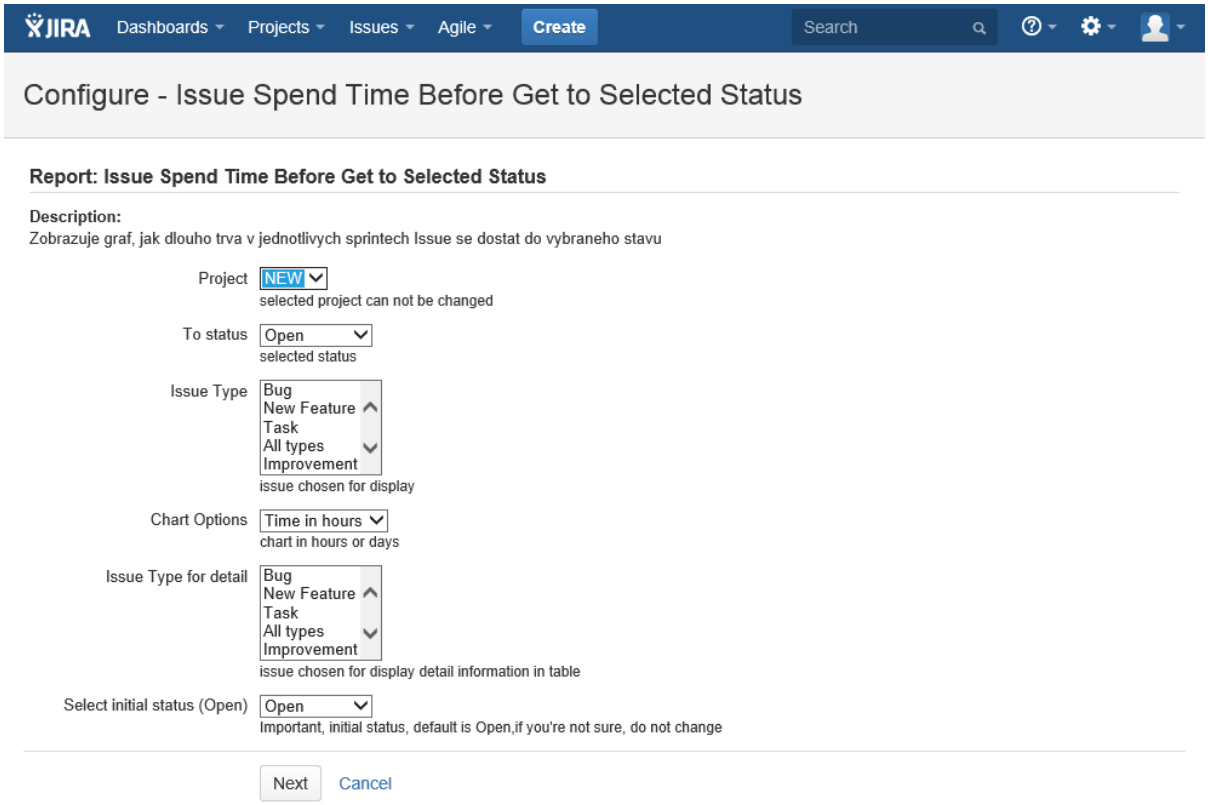
Obrázek 7.13: Formát tabulky pro graf zobrazující kolik Issue absolvovala sprintů



Obrázek 7.14: Formát tabulky pro graf zobrazující procentuální počet dokončených Issue ve sprintu

7.3.3 Report Issue Spend Time Before Get to Selected Status

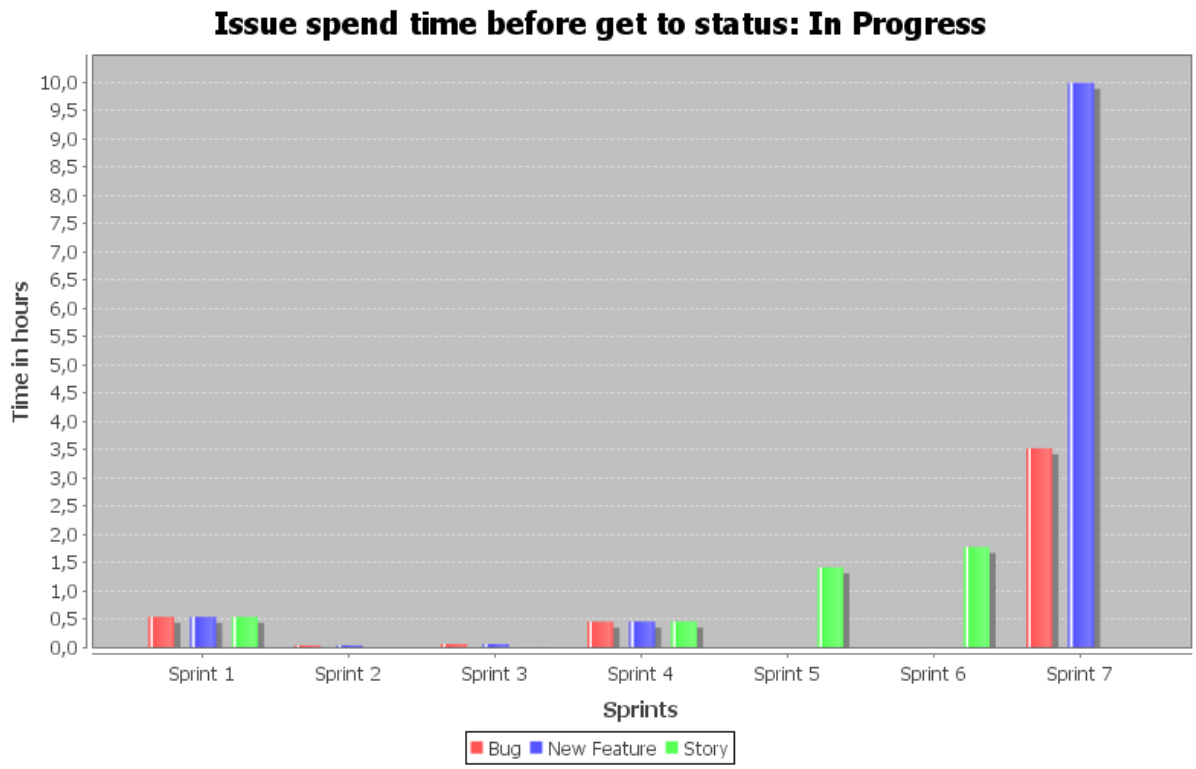
Po spuštění reportu se nám zobrazí obrazovka odpovídající navrhovanému případu užití. Na této obrazovce může uživatel nastavit parametry reportu. U reportu *Issue Spend Time Before Get to Selected Status* nastavuje uživatel, které typy Issue budou podrobeny analýze, ale nejdůležitějším nastavením je aby uživatel vybral stav, do kterého se úkol v průběhu sprintu měla dostat a report tak zanalyzoval, jak dlouho to trvalo. Další doplňující možností je nastavení počátečního stavu (implicitně nastaven na „Open"), report pak bude měřit interval mezi těmito vybranými stavy. Uživatel taky může nastavit, zda bude výsledný čas v grafu zobrazen ve dnech nebo v hodinách. Posledním nastavením je, že uživatel vybere pro které typy Issue chce zobrazovat tabulku s detailními informacemi.



Obrázek 7.15: Obrazovka reportu Issue Spend Time Before Get to Selected Status

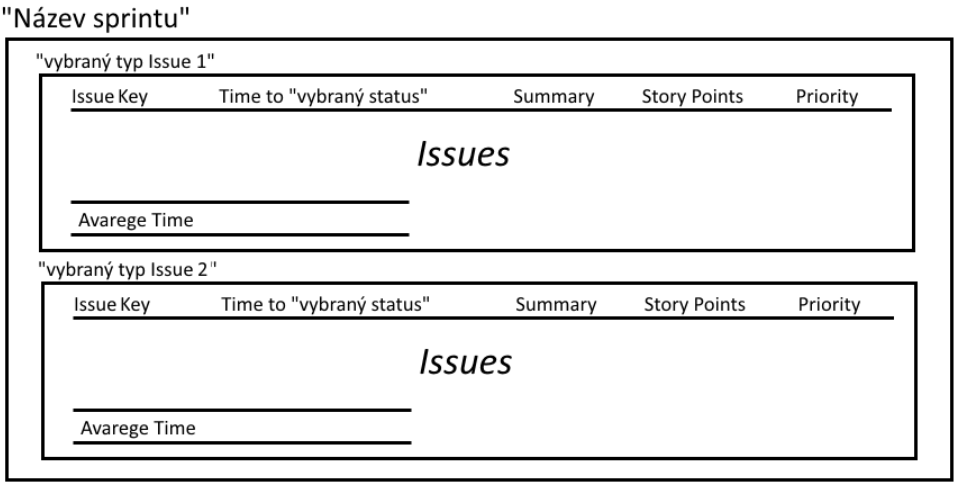
Analýza reportu

Výstupem reportu je sloupcový graf ukázaný na obrázku 7.16. Graf zobrazuje, jaká byla průměrná doba, vybraných typů Issue, než se do staly v konkrétním sprintu do stavu „In Progress". Zobrazená data jsou z testovacího prostředí, proto jsou časy relativně krátké v řádu několika hodin, avšak v praxi mohou být tyto hodnoty v řádu několika dní popřípadě i týdnů. Zásadní informaci, kterou lze z grafu vyčíst je opět porovnání mezi jednotlivými sprinty, tedy jaké byly průměrné časy v konkrétních sprintech. Ideálně by měla průměrná doba být ve všech sprintech přibližně stejná. Pokud nějaká hodnota výrazně vybočuje z obvyklých hodnot, je třeba se zaměřit na to, co se v konkrétním sprintu dělo a co



Obrázek 7.16: Graf na výstupu reportu Issue Spend Time Before Get to Selected Status

tento výkyv způsobilo. Doba než začnou být úlohy v rámci sprintu zpracovávány je závislá na mnoha okolnostech. Především je důležitá správná dekompozice a naplánování sprintu, tak aby mohla být přípravná doba co nejkratší. Dále je potřeba mít dostatek volných členů týmu, kteří mohou na naplánovaných úkolech pracovat. Pokud bychom měřili dobu, než se Issue dostane ze stavu „In Progess“ do stavu „In Testing“ a tato hodnota by byla neúměrně vysoká, tak by to mohlo znamenat, že nemáme dostatek testerů. To by byla indikace pro tým a projektového manažera, že je potřeba se nad tímto probléme zamyslet a najít řešení. Buď změnit zavedené procesy anebo, investovat do toho, aby se tým testerů mohl rozšířit. Informace z tohoto grafu mohou být použity k dalším podobným analýzám.

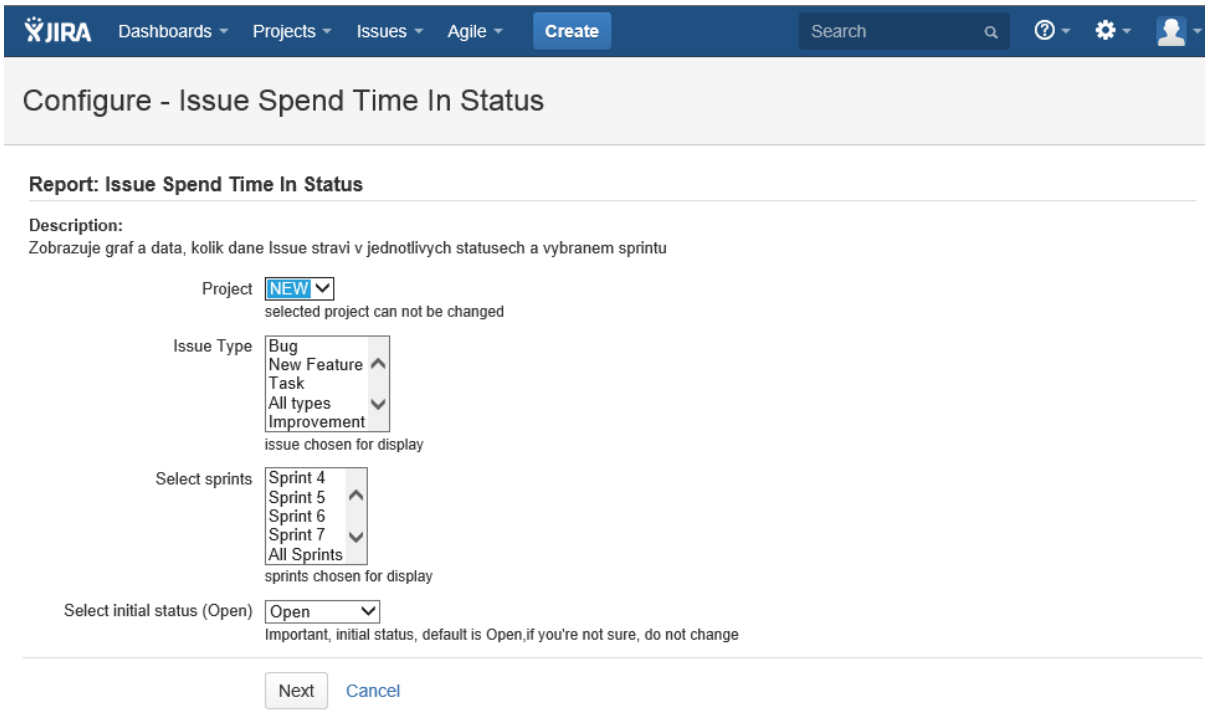


Obrázek 7.17: Formát tabulky zobrazující detailní informace k reportu

Proto, aby analýza mohla být co nejpřesnější a tým mohl identifikovat konkrétní problémy, slouží detailní informace zobrazené v tabulce pod grafem. Struktura této tabulky je na obrázku 7.17. Tabulka je vytvořená pro každý sprint (osa *x*) a dále je členěna na části podle typu Issue, které uživatel vybral k detailnímu zobrazení. Pro každý tento typ je pak zobrazena průměrná hodnota (osa *y*), odpovídající sloupci v grafu. U každé Issue jsou pak zobrazeny informace dle specifikace požadavků na report.

7.3.4 Report Issue Spend Time In Status

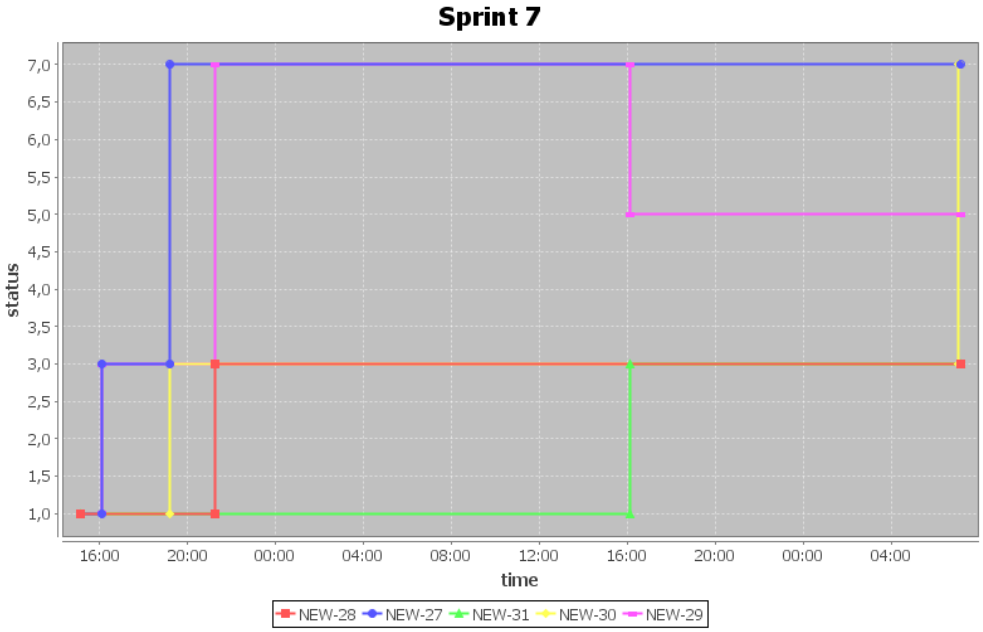
Po spuštění reportu se nám zobrazí obrazovka odpovídající navrhovanému případu užití. Na této obrazovce uživatel nastavuje parametry reportu. U reportu *Issue Spend Time In Status* může uživatel nastavit typy Issue, pro které se analýza bude provádět a pak konkrétní sprinty, nad kterými budou zobrazovaná data. Uživatel může vybrat vždy více možností. Dalším nastavením je nastavení počátečního stavu odpovídajícího aktivního Workflow a to z důvodu správné funkce reportu. Ukázka této obrazovky je na obrázku 7.18.



Obrázek 7.18: Obrazovka reportu Issue Spend Time In Status

Analýza reportu

Výstupem reportu je graf s tabulkou pro každý sprint. Ukázka výstupu je na obrázku 7.19. Název grafu je taky název konkrétního sprintu. Na vodorovné ose *x* je časová osa, rozsah této osy odpovídá délce sprintu od jeho začátku až po jeho dokončení, pokud je sprint stále aktivní, tak je ukončen aktuálním časem. Na svislé ose *y* jsou číselné hodnoty odpovídající jednotlivým stavům. Tyto hodnoty vychází z interního nastavení systému JIRA a nelze je měnit, význam jednotlivých hodnot odpovídá popisům stavů v tabulce pod grafem kde je číslo uvedeno vždy v hranaté závorce (např. statusu „Open“ odpovídá



Key	Issue Type	Story Points	Open [1]	Resolved [5]	Closed [6]	In Progress [3]	Reopened [4]	In Testing [7]
NEW-28	Bug	-	0d 6h 5:51	no data	no data	1d 9h 52:47	no data	no data
NEW-27	Bug	-	0d 0h 56:38	no data	no data	0d 3h 6:24	no data	1d 11h 55:36
NEW-31	New Feature	-	1d 0h 57:49	no data	no data	0d 15h 0:49	no data	no data
NEW-30	New Feature	-	0d 4h 3:7	no data	no data	1d 11h 51:47	no data	0d 0h 3:44
NEW-29	New Feature	-	0d 0h 56:41	0d 15h 1:3	no data	0d 5h 9:4	no data	0d 18h 51:49

Obrázek 7.19: Graf a tabulka na výstupu reportu Issue Spend Time In Status

hodnota 1, statusu „Closed“ odpovídá hodnota 6). Issue jsou rozděleny na jednotlivé linie, kde každá je jinak barevně označena. Změna stavu je znázorněna jako jednorázový skok v čase, kdy změna nastala. Z grafu tak uživatel může vyčíst, jak probíhal sprint z pohledu práce na zvolených úkolech. Důležitou částí je tabulka pod grafem, která doplňuje informace ke grafu. V této tabulce jsou zobrazeny časy, kolik Issue strávila v rámci sprintu v jednotlivých stavech. Hodnoty z této tabulky může tým opět použít k analýze průběhu sprintu. Pokud je v tabulce uvedeno „no data“ tak to znamená, že se konkrétní úloha v průběhu tohoto sprintu do daného stavu vůbec nedostala. Dlouhé intervaly v jednotlivých stavech mohu vypovídat buď o špatné dekompozici problému, jeho vysoké náročnosti anebo neschopnosti týmu problém vyřešit. Smyslem tohoto reportu je doplnit přechozí report, tatk aby uživatel mohl získat ucelené informace pro svoji analýzu. Dále je v tabulce uvedeno o jaký typ Issue se jedná a jaké je její bodové ohodnocení.

7.4 Testování

Důležitou součástí implementace bylo také průběžné testování během vývoje, ale také závěrečné ověření funkčnosti a zkušební nasazení v produkčním prostředí. K testování sloužilo spouštění a instalace do zkušební verze systému JIRA. V tomto systému byly vytvořena testovací data, tak aby bylo možné vyzkoušet co nejvíce pravděpodobných scénářů, které mohou nastat během normálního

provozu v produkčním prostředí. Poslední tetování proběhlo ve zkušebním provozu nad skutečnými daty.

7.4.1 Testovací prostředí

Jako testovací prostředí byl zvolen osobní počítač s 64bitovým operačním systémem Windows 8.1 Professional. Na tomto počítači byla spuštěna trial verze systému JIRA. Licence pro zkušební verzi poskytuje plnohodnotnou verzi systému JIRA (v6.4) na 30 dní pro účely vývoje, testování a představení funkčnosti, nesmí být použita pro komerční účely. Testovalo se především schopností vytvoření reportu, za různých podmínek a nastavení. Další kontrolou bylo, zda výstup reportu odpovídá skutečným datům v systému JIRA.

7.4.2 Způsob testování provozu

Postup při testování ve zkušební verzi JIRA byla následujíc:

1. Nastavení systému na testovací platformě JIRA shodně s nastavením systému používaného společností Siemens.
    a. Nainstalování shodných modulů v odpovídajících verzí.
    b. Vytvořit stejné aktivní Workflow.
    c. Pojmenovat stavy shodnými názvy.
2. Vytvoření kolekce potřebných dat pro testování.
    a. Vytvoření projektu.
    b. Vytvoření kolekce Issue různých typů (produktový katalog).
    c. Naplánování sprintů (katalog sprintu).
    d. Spuštění sprintů.
    e. Simulace práce v rámci sprintu (přesování Issue mezi jednotlivývmi stavy v průběhu času).
3. Vytvoření reportů podle různých scénářů kombinace parametrů.

Scénáře nastavení reportu Issue In Sprint Charts

Vytvoř report s následujícím nastavením parametrů:

1. Nevyber žádné typ Issue pro zobrazení.
2. Vyber všechny typy Issue pro zobrazení.
3. Vyber libovolný počet typů Issue k zobrazení.
4. Vyber jeden typ Issue k zobrazení.

Scénáře nastavení reportu Issue Spend Time Before Get to Selected Status

Vytvoř report s následujícím nastavením parametrů:

1. Nastav cílový status a pak další parametry:

    a. Vyber všechny typy Issue.
    b. Graf nastav na zobrazení v hodinách.
    c. Vyber zobrazení detailů pro všechny typy Issue.
    d. Počáteční stav nastav podle počátečního stavu Workflow.
2. Nastav cílový status a pak další parametry:
    a. Vyber všechny typy Issue.
    b. Graf nastav na zobrazení ve dnech.
    c. Vyber zobrazení detailů pro libovolný výběr Issue.
    d. Počáteční stav nastav podle počátečního stavu Workflow.
3. Nastav cílový status a pak další parametry:
    a. Nevyber žádné typy Issue pro zobrazení.
    b. Graf nastav na libovolné zobrazení.
    c. Nevyber žádné typy Issue pro zobrazení detailních informací.
    d. Počáteční stav nastav podle počátečního stavu Workflow.
4. Výše uvedené scénáře proveď s různými kombinacemi cílového a počátečního stavu Issue.

Scénáře nastavení reportu Issue Spend Time In Status

Vytvoř report s následujícím nastavením parametrů:

1. Použij kombinaci těchto parametrů:
    a. Nevyber žádné typy Issue k zobrazení.
    b. Nevyber žádný sprint k analýze.
    c. Počáteční status nastav dle odpovídajícího Workflow.
2. Použij kombinaci těchto parametrů:
    a. Vyber libovolný počet typů Issue k zobrazení.
    b. Vyber libovolný počet sprintů k zobrazení.
    c. Počáteční status nastav dle odpovídajícího Workflow.
3. Použij kombinaci těchto parametrů:
    a. Vyber všechny typy Issue k zobrazení.
    b. Vyber libovolný jeden sprint k zobrazení.
    c. Počáteční status nastav dle odpovídajícího Workflow.
4. Vyzkoušej další možné kombinace nastavení.

7.4.3 Nasazení v produkčním prostředí

Po důkladném otestování v testovacím prostředí pomocí výše uvedených scénářů byl zásuvný modul nainstalován do systému JIRA společnosti Siemens, s.r.o. Při spuštění tohoto zkušebního provozu modulu byla nalezeno řada chyb, které vedly, nebo povedou ke zlepšení nástroje. Avšak i tak se

podařilo získat potřebné základní informace z projektů a bylo ověřeno, že nástroj je funkční a bude možné ho v budoucnu plnohodnotně používat.

8 Zhodnocení a další vývoj

Během hodnocení aplikace vycházím především z vyjádření zástupce společnosti Siemens Ing. Jakuba Řezáče, který byl hlavním konzultantem v průběhu řešení práce. Bylo splněn cíl diplomové práce a to, že v práci dochází k analýze použitých nástrojů a definovaných byznys procesů. Výsledkem je nabídka sady měření (implementovaný zásuvný modul), na jejichž základě je možné podpořit či uskutečnit rozhodnutí v rámci agilních procesů vývoje.

8.1 Zhodnocení přínosu

Přínos reportu *Issue In Sprints Charts* spočívá především v tom, že může přinést informace o přetíženosti vývojového týmu, což vyplynulo z použití tohoto reportu v rámci zkušebního provozu v produkčním prostředí. Tento naměřený výsledek otvírá otázku ohledně vyšších investic do fáze analýzy požadavků. Ředitele R&D[1] tyto výsledky staví před otázku, jak velké riziko nespokojenosti u zákazníka je ochoten podstoupit při přesouvání určitého objemu požadavků do následujících sprintů. Report také umožňuje identifikovat nejčastěji přesouvané úkoly, což naznačuje potíže v komunikaci pracovních skupin.

Report *Issue Spend Time Before Get To Selected Status* se konkrétněji zaměřuje na přípravu během přechodů úkolů mezi jednotlivými pracovními skupinami vůči typu a doméně zpracovávaného požadavku. Report se nejvíce osvědčil při identifikaci slabě rozpracovaných témat v rámci projektu, ovšem také předkládá reálná trvání přípravy před vývojem, která je manažery často upozaďována na úkor rychlého vypracování nasaditelného produktu.

Poslední report *Issue Spend Time In Status* doplňuje předešlý report o reálný pohled na trvání jednotlivých vývojových fází. Tento report lze ocenit hlavně ve fázi plánování nových projektů, kde používané expertní estimační[2] metody vyžadují postupné upřesňování na základě předchozích zkušeností z projektů podobné domény. Díky úplnému řízení kroků pomocí systému JIRA lze nyní využívat přesná data, místo odhadů založených na předchozích zkušenostech. To vede k vyšší důvěryhodnosti před zákazníky a přináší interní zpětnou vazbu pro prioritní upřednostňování témat ke zlepšení, což také vede k efektivnějšímu směřování investic.

„Diplomová práce prokazuje, že investice do měření kvality a automatizace jsou návratné, významně snižují nejistotu rozhodování a zlepšují vztahy se zákazníkem i v organizaci samotné. Práce též rozpracovává řadu dalších metrik, které by rovněž znamenaly posun při snaze zvýšit hodnocení

[1] **R&D (Research and Development)** – oddělení vývoje a výzkumu.
[2] **Estimace** - kvalifikované odhadování na základě zkoumání, šetření a strukturování dané problematiky [15].

organizace podle CMMI[1]*. "* Citace z vyjádření k diplomové práci od zástupce společnosti Siemens, Ing. Jakuba Řezáče.

8.2 Možný další vývoj

V rámci dalšího vývoje by mohly být do modulu implementovány následující rozšíření. Jedná se především o rozšíření, které byly v průběhu práce konzultovány, ale nakonec nebyly z časových důvodů do výsledné implementace zahrnuty. Velmi prospěšným rozšiřujícím modulem by byl nástroj, který by vizualizoval a porovnával vytížení jednotlivých týmů na projektu. Požadavek pro tento modul vznikl na základě toho, že se stává situace, kdy tým testerů nestíhá testovat všechny implementované úkoly, což vede k výraznému zpomalování prací na dokončení úkolů. Pokud by se analýzou zjistilo, že některé týmy jsou neúměrně vytížené oproti ostatním, tak by to mělo vést k zamyšlení ředitele oddělení na přerozdělení investovaných zdrojů do týmů.

Další možností rozšíření by byl nástroj, který by analyzoval korelaci mezi vzniklými chybami. Jednalo by se o nástroj, který by analyzoval úkoly v rámci projektu, a odhalil by úkoly (Issue) u které by vznikalo velké množství chyb (Bug). Tyto úkoly by byly „vytáhnuty" ze systému a tým by tak mohl analyzovat, co vede ke vzniku těchto chyb.

Poslední možností dalšího vývoje je vylepšení aktuálních nástrojů. Bylo by vhodné, aby implementované nástroje byly převedeny do uživatelského rozhraní tak, že by mohl uživatel interaktivně nastavovat parametry reportu a lépe filtrovat zobrazovaná data. Momentálně je výstup pouze statický, nelze ho nijak měnit. Dále by bylo možné, kdyby samotný report již analyzoval zobrazovaná data a sám by upozorňoval na problematické části a nabízel možné příčiny a řešení. Poslední rozšířením by mohlo být, aby data byla zobrazovaná v přesně daném formátu, tak aby mohl být vytvořený skript, který by nad získanými daty provedl analýzu.

Možnosti rozšíření je skutečně mnoho a rozhodně by bylo možné, po konzultaci se zástupcem společnosti Siemens, nalézt řadu dalších pro společnost přínosných aplikací, které by přesně odpovídali požadavkům projektových manažerů a přispěly by tak k lepšímu a hlavně automatizovanému rozhodování v rámci byznys procesů.

[1] **CMMI (Capability Maturity Model Integration)** - model kvality organizace práce určený pro vývojové týmy.

9 Závěr

Práce se zabývá agilním vývojem software s důrazem na kombinaci metodik Scrum a Kanban. Cílem práce bylo navrhnout řešení, které by přispělo k vylepšení zavedených procesů ve společnosti Siemens, která v rámci organizačních procesů využívá metodiky Scrum-ban. V průběhu řešení práce došlo k analýze a evaluaci existujícího projektu. Při vyhodnocování těchto výsledků a po diskuzi se zástupcem firmy Siemens, bylo navrženo rozšíření v prostředí systému pro projektové a organizační systém JIRA, tak aby doplnilo současné nástroje pro analýzu projektů, a podle potřeb společnosti Siemens pomáhalo ke zlepšení již zavedených organizačních procesů. Předmětem tohoto vylepšení je zásuvný modul, který rozšiřuje možnosti generování projektových reportů.

Implementovaný nástroj obsahuje 3 moduly. První modul analyzuje, kolik sprintů absolvovaly jednotlivé úkoly a jaká byla úspěšnost dokončení úkolu v rámci sprintů. Výsledky tohoto měření slouží k identifikaci nejčastěji přesunovaných úkolů, což může naznačovat potíže v komunikaci pracovních skupin. Druhý modul analyzuje, jak dlouho trvá daným úkolům, než se dostanou do vybraného stavu. Smyslem modulu je především analyzovat přípravnou práci, tedy jak dlouho trvá, než se úkol dostane do stavu „In Progress“. Poslední modul doplňuje předešlý report a složí k analýze průběhu sprintu a zobrazení časů kolik strávily úlohy v jednotlivých stavech.

Současně je implementovaný modul nasazen a testován v systému JIRA společnosti Siemens a byl již použit při analýze existujícího projektu. Výsledky tohoto měření jsou shrnuty v kapitole 8, kde se uvádějí a shrnují celkové přínosy aplikace. Dle názoru zástupce společnosti Siemens bude nástroj velmi užitečný při rozhodování v rámci agilních procesů a to, protože bude možné tato rozhodnutí podpořit přesnými daty, což dříve nebylo možné.

Práce dále rozvíjí možnosti rozšíření a vylepšení, které by vedly k dalšímu zlepšení zavedených procesů. Práce splnila všechny body zadání, pouze v bodu 6 se trochu odklonila od původního plánu a to z důvodu, že navrhnuté řešení neovlivňuje přímo procesy, které by mohly ovlivnit metriky projektu, ale vede k lepší a automatizované identifikaci problémů. Při dlouhodobém používání nástroje se vyhodnocování těchto dat může zapracovat do zavedených procesů a vést tak k lepší efektivitě práce. Z tohoto důvodu nedochází k porovnání s dříve naměřenými výsledky, ale je uvedený přínos během nasazení a použití nástroje v praxi.

Literatura

[1] ŠOCHOVÁ, Z. a E. KUNCE. *Agilní metody řízení projektů*.. Brno: Computer Press, 2014. ISBN 978-80-251-4194-6.

[2] BUCHALCEVOVÁ, A. *Metodiky vývoje a údržba informačních systémů*.. Praha: Grada Publishing, 2005. ISBN 80-247-1075-7.

[3] KADLEC, V. Extremismus nemusí být na škodu: extrémní programování. In: *Živě.cz* [online]. 13. 5. 2003 [cit. 2015-01-07]. Dostupné z: http://www.zive.cz/clanky/extremismus-nemusi-byt-na-skodu-extremni-programovani/sc-3-a-111714/default.aspx

[4] AMBLER, S.. *Agile Modeling* [online]. 2001-2014 [cit. 2015-01-07]. Dostupné z: http://agilemodeling.com/essays/fdd.htm

[5] KOTRLA, T. *Agilní metodiky vývoje software*. Brno: 2006. Diplomová práce. Masarykova univerzita v Brně, Fakulta informatiky [cit. 2015-01-07]. Dostupné z: http://is.muni.cz/th/50755/fi_m/

[6] COCKBURN, A. Crystal light methods. In: *Alistair Cockburn* [online]. 19. 6. 2008, verze 1.5.2015 [cit. 2015-01-07]. Dostupné z: http://a.cockburn.us/1734

[7] KNIBERG, H. a M. SKARIN. *Kanban and Scrum: making the most of both*. C4Media, 2009. ISBN 978-0-557-13832-6. Dostupné také z: http://www.infoq.com/minibooks/kanban-scrum-minibook

[8] Atlassian. *Jira* [online]. 2015 [cit. 2015-05-01]. Dostupné z: https://www.atlassian.com/software/jira

[9] KURUVILLA, J. *JIRA Development Cookbook*. Birmingham: Packt Publishing Ltd. 2011. ISBN 978-1-84968-180-3.

[10] DITTMAR, K. Introduction into SCRUM. In: *Redletterday* [online]. 23. 5. 2014 [cit. 2015-01-07]. Dostupné z: http://redletterday.ch/?p=550

[11] FABÓK, Z. XP with Kanban instead of Scrum. In: *Zsoltfabok* [online]. 11. 2. 2011 [cit. 2015-01-07]. Dostupné z: http://zsoltfabok.com/blog/2011/02/xp-with-kanban-instead-of-scrum/

[12] COHN, M. *Agile Estimating and Planning*. Upper Saddler River, NJ: Prentice Hall, 2005. ISBN 0-13-147941-5.

[13] FLORAC, W. a A. CARLETON. *Measuring the Software Process*. 3. vyd. Boston: Addison-Wesley, 1999. ISBN 0-201-60444-2.

[14] Atlassian Documentation. *JIRA Documentation* [online]. 2015 [cit. 2015-05-01]. Dostupné z: https://confluence.atlassian.com/display/JIRA/JIRA+Documentation

[15] Atlassian Documentation. *JIRA Agile* [online]. 2015 [cit. 2015-05-01]. Dostupné z: https://confluence.atlassian.com/display/AGILE/JIRA+Agile+Documentation

[16] ABZ Slovník cizích slov [online]. 2004 [cit. 2015-05-01]. Dostupné z: http://slovnik-cizich-slov.abz.cz/

Seznam příloh

Příloha 1. CD se zdrojovými kódy a obrázky.

Příloha 1. Obsah CD

Na přiloženém CD se nachází následující adresářová struktura:

`./app/` - zkompilovaný zásuvný modul pro systém JIRA v balíčku JAR
`./thesis/` - text práce ve formátu PDF a .docx včetně použitých obrázků
`./src/` - zdrojové kódy aplikace v jazyce Java včetně dokumentace