Návrh vývojové metodiky pro webové aplikace

A Proposed Web Applications‘ Development Methodology

Bc. Jan Ovesný

Diplomová práce
2014

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Ovesný**
Osobní číslo:	**A11442**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Počítačové a komunikační systémy**
Forma studia:	**prezenční**
Téma práce:	**Návrh vývojové metodiky pro webové aplikace**
Téma anglicky:	**A Proposed Web Applications' Development Methodology**

Zásady pro vypracování:

1. **Proveďte rešerši vývojových metodik pro webové aplikace.**
2. **Charakterizujte webový vývojový cyklus.**
3. **Představte agilní metodiky SCRUM a Kanban, zaměřte se na vhodné vlastnosti pro webový vývoj.**
4. **Navrhněte vhodné rozšíření agilní metodiky pro vývoj webových aplikací.**
5. **Dokumentujte upravenou metodiku pomocí popisu pracovního prostředí.**
6. **Vyhodnoťte vytvořené řešení.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. 1. vyd. Brno: Computer Press, 2004, 278 s. ISBN 80-251-0342-0.
2. KNIBERG, Written by Henrik. Scrum and xp from the trenches: how we do scrum. [S.l: C4Media Inc.], 2007. ISBN 978-143-0322-641.
3. SOMMERVILLE, Ian. Softwarové inženýrství. 1. vyd. Brno: Computer Press, 2013, 680 s. ISBN 978-80-251-3826-7.
4. KNIBERG, Henrik. Kanban vs Scrum. Crisp AB. Viitattu, 2009, 1: 2011.
5. COCCO, Luisanna, et al. Simulating Kanban and Scrum vs. Waterfall with System Dynamics. In: Agile Processes in Software Engineering and Extreme Programming. Springer Berlin Heidelberg, 2011. p. 117-131.
6. RIVERO, José Matías, et al. Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering. Information and Software Technology, 2014.

Vedoucí diplomové práce: **Ing. Radek Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **7. února 2014**

Termín odevzdání diplomové práce: **27. května 2014**

Ve Zlíně dne 7. února 2014

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Karel Vlček, CSc.
ředitel ústavu

ABSTRAKT

Cílem práce je navrhnout metodiku, která bude vhodnou pro užití při vývoji webových aplikací. Za tímto účelem představím v teoretické části některé vývojové metodiky pro vývoj aplikací a v části praktické pak na základě kladů a záporů vyberu jednu, a tuto metodiku přizpůsobím tak, aby splňovala požadavky pro vývoj webových aplikací. Mezi představenými metodikami budou zástupci jak klasických, tak agilních postupů.

Za účelem mé práce se však zaměřím převážně na metodiku SCRUM, která je pro implementaci webových aplikací schůdným řešením. Taktéž bude zdokumentován popis pracovního prostředí jako takového a představeny jednotlivé nástrojů užité při řízení vývoji.

Klíčová slova: Agilní, metodiky, webové, aplikace, SCRUM

ABSTRACT

The aim of this thesis is to propose a methodology that will be suitable for use in web development. For this purpose in the theoretical part I'll introduce some development methodology for web application and in the practical part based on the pros and cons selected one, and this methodology modify to meet the requirements for web application development. Among the featured methodologies I will include representatives of both traditional and agile processes.

For the purpose of my work, I will focus mainly on the SCRUM methodology, which is suitable for implementation of web based applications. It will also be documented the working environment and introduced individual instruments used in management of development.

Keywords: Agile, Methodology, web, applications, SCRUM

„Individuals and interactions over processes and tools“

„Working software over comprehensive documentation“

„Customer collaboration over contract negotiation“

„Responding to change over following a plan“

Manifesto for Agile Software Development

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdané verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 6.5.2014

…..…….
podpis diplomanta

OBSAH

ÚVOD

V dnešní době, kdy je zapotřebí rychle a hlavně efektivně vyvíjet webové aplikace, je stále méně prostoru pro dříve hojně užívané klasické metodologie vývoje softwaru. V dnešní době, kdy zpoždění dodávky aplikace může vést v extrémním případě až ke krachu firmy je dodržování termínů kritické. V dnešní době, kdy se funkcionalita finálního produktu mění již v průběhu vývoje, je potřeba umět na tyto změny reagovat a hlavně včasně změny provést, již prostě není moc prostoru pro klasické metodiky.

Přesně proto se již dříve a také nyní hojně přechází k právě agilním metodikám, kterým se úspěšně výše zmíněné problémy více či méně daří eliminovat.

Spousty agilních metodik už „urazili“ nemalý kus cesty a podařilo se jim za pomocí odborníků a samotného užívání vykrystalizovat do dnešní podoby, kdy jsou nedílnou součástí každé větší firmy, zabývajícím se ať už výrobou, nebo vývojem aplikací.

Bez agilní metodiky by to již prostě nešlo.

I. TEORETICKÁ ČÁST

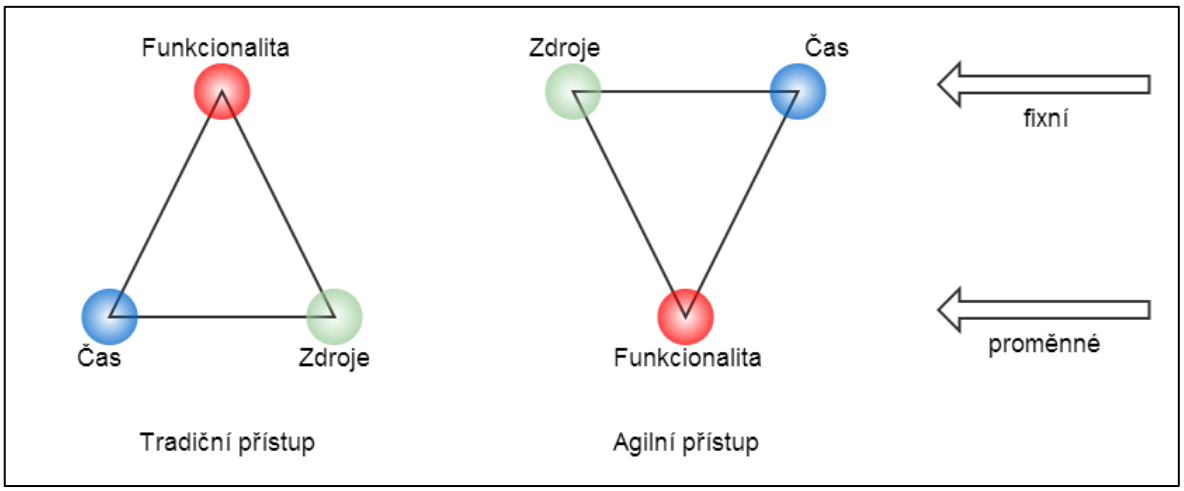
1 VÝVOJOVÉ METODIKY

Softwarové metodiky vývoje softwaru se zabývají tvořením softwaru, nikoliv však co se týče technické stránky jako spíše organizačního aspektu implementace.[1] Jako dva hlavní pohledy na věc můžeme dané metodiky rozdělit podle toho, jak je k celému vývoji přistupováno a to buď to klasicky nebo jednou z agilních metodik.

Pro představení základního rozdílu mezi těmito metodikami si zaveďme 3 proměnné[2]:

1. Funkcionalita:
    - veškeré funkce, které by v dané aplikaci měli být dostupné
2. Čas:
    - čas, potřebný k dosažení dané funkcionality
3. Zdroje:
    - hardware
    - software
    - vývojáři, testeři, manažeři

Z pohledu tradičního přístupu je Funkcionalita fixní prvek, Čas a Zdroje prvek proměnný. U agilní metodiky je tomu právě přesně naopak, a tedy Funkcionalita je proměnná a Čas se Zdroji fixní. Lépe je daná situace názorná na obrázku č.1



Obrázek 1 - Tradiční vs agilní přístup [2]

1.1 Modely procesů

1.1.1 Vodopádový model (VM)

Jak již název napovídá Vodopádový model, jako takový, není metodika, ale modelem životního cyklu. Nicméně, pohlížíme-li na metodiku jako posloupnost kroků, která vede k vytvoření kvalitního softwaru je právě Vodopádový model vhodným kandidátem k uvedení stejně tak Spirálový model (SM), který si představíme později.[2]

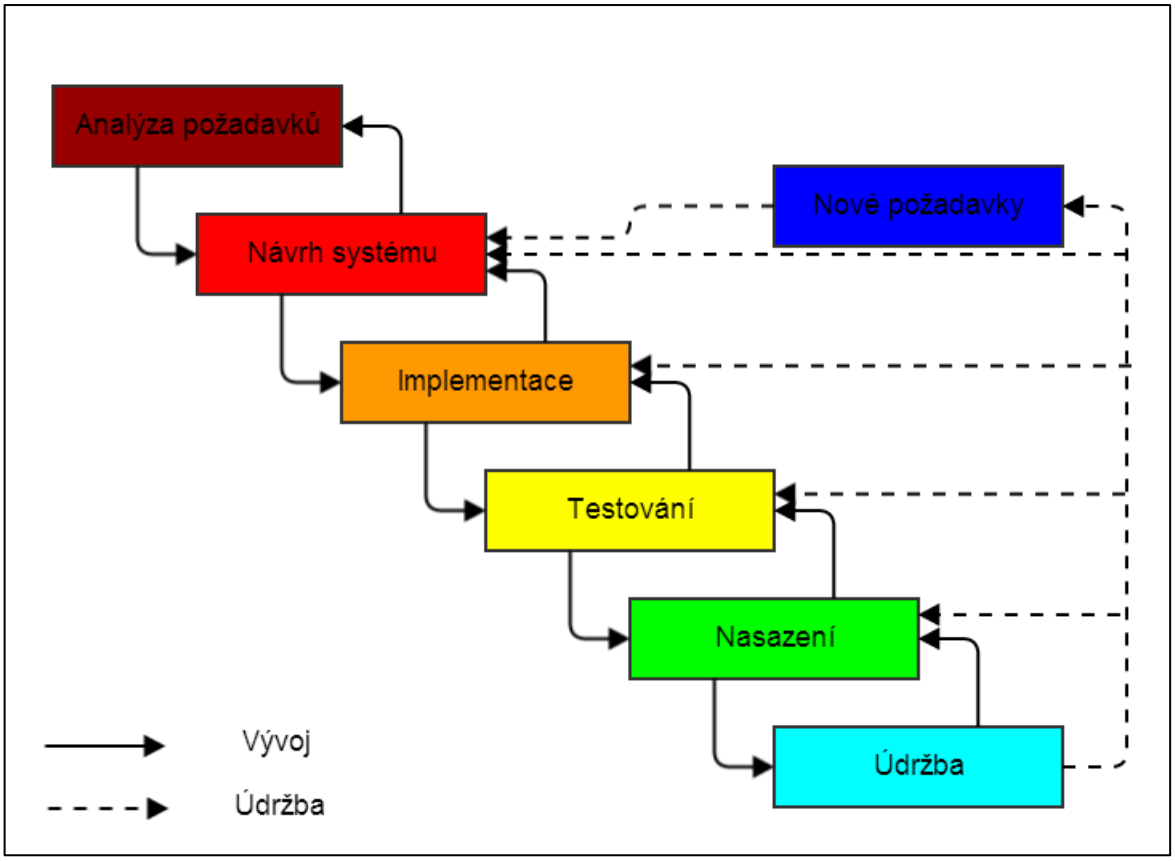
VM byl první model (1970), který byl představen a tvořil významný mezník v softwarovém inženýrství. Slovo, které nejlépe popisuje tento model je slovo linearita. Veškeré fáze vývoje jsou „naskládány" jedna za druhou a jediný možný přechod z jedné fáze do druhé je po skončení předchozího dílčího kroku.[3]

1.1.1.1 Fáze projektu

Jak již bylo dříve zmíněno, posloupnost kroků je pevně daná a má sekvenční charakter, který se v průběhu vývoje striktně dodržuje: [3]

a) Analýza požadavků – Nejprve je potřeba seznámit se se zákazníkem, analyzovat jeho potřeby a definovat problémy, které v průběhu implementace mohou nastat a tím celý vývoj výrazně zpomalit
b) Návrh systému – Po důkladné analýze je potřeba navrhnout řešení daného problémů s ohledem na požadavky zákazníka, cenu a čas potřebný k realizaci daného projektu.
    a. Návrh řešení – slovní popis celé funkcionality
    b. Návrh architektonický – jednotlivé objekty, metody, návaznost…
c) Implementace – samotné kódování a realizace projektu v konkrétním programovacím jazyce a dané platformě
d) Testování – před tím, než je aplikace vypuštěna do světa, je potřeba ji důkladně otestovat. Testování je vhodné provádět jak s pozitivními scénáři, tak negativními
e) Nasazení – Instalace aplikace u zákazníka s případným doladěním nastavení pro konkrétní využití.
f) Údržba – udržování aplikace v chodu, oprava případných chyb, vydávání updatů

1.1.1.2 Schéma



Obrázek 2 – Vodopád schéma [2]

1.1.1.3 Výhody

- Jednoduché na použití a pochopení
- Účinný na menší projekty s jasnými a pochopenými požadavky
- Jasně definované kroky
- Proces a výsledky jsou dobře dokumentovány [3]

1.1.1.4 Nevýhody

- Software je funkční až na úplném konci procesu
- Může být nevhodný pro více komplexní projekty
- Nevhodný pro stálé pokračující projekty
- Integrace je prováděna až na konci vývoje, což znemožňuje následně reagovat na případné problémy
- Pružnost – v daném případě spíše nepružnost celého modelu [3]

1.1.1.5 Shrnutí

Model si v dnešní době nenajde mnoho prostoru pro využití, neboť dnešní aplikace jsou většinou velkého rázu a tak je spíše volen jiný model, případně metodika. Uplatnění si však může najít v oblasti menších projektů, jako jsou projekty školní nebo jednoúčelové aplikace s jednoduchou funkcionalitou.

1.1.2 Spirálový model (SM)

U Spirálového modelu stejně tak jako u VM se nejedná opět o metodiku jako spíš model, nicméně je dobré jej zmínit, ať vidíme vývoj a směr jakým se softwarové inženýrství vyvíjelo. Především to byly nedostatky VM, které vedly k vzniku zcela nového modelu, u jehož zrodu stál Barry Boehm[2].

Jako základní rozdíl mezi VM byl zaveden:

a) iterativní přístup
b) opakovaná, důsledná analýza

Iterativní přístup je rozuměno, že vývoj se odehrává ne v jedné předem dané kaskádě, ale v menších, plně funkčních iteracích vývoje. Na konci každé iterace je tedy představen funkční (i když nekompletní) produkt s přidanou novou funkcionalitou.

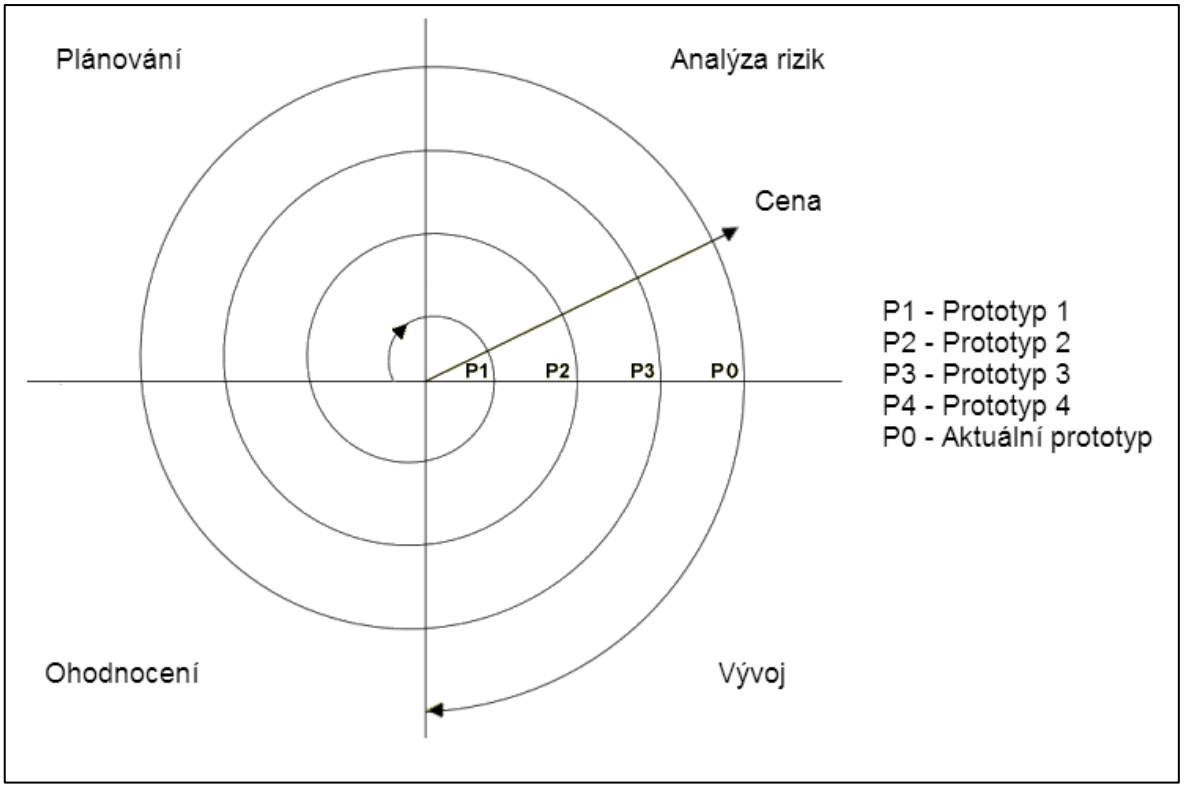
Důsledná analýza je zde prováděna jak na začátku prvotního zahájení vývoje, tak v každé další iteraci.[2]

1.1.2.1 Fáze projektu

Oproti VM, jsou u Spirálového modelu pouze 4 hlavní části, které zaštiťují celý vývoj a zjednodušují chápání daného modelu[4]:

a) Plánování – plánovaní konkrétní funkcionality, která bude v dané iteraci implementována.
b) Analýza rizik – vždy před samotným vývojem je vždy provedena analýza rizik pro danou iteraci, aby se určily pokud možno všechna reálná rizika jako jsou čas, náklady, správná funkcionalita, schopnost konkurence…
c) Vývoj – psaní kódu, testování aplikace.
d) Ohodnoceni – zákazník ohodnotí danou iteraci, poskytne zpětnou vazbu.

1.1.2.2 Schéma



Obrázek 3 - Spirala schéma [15]

1.1.2.3 Výhody

- Existence prototypů
- Požadavky mohou být měněny v průběhu vývoje
- Zákazník vidí základ aplikace hned po první iteraci[4]

1.1.2.4 Nevýhody

- Řízení projektu je více komplexní
- Konec projektu není znám na začátku vývoje
- Velké množství dokumentace pro jednotlivé iterace[4]

1.1.2.5 Shrnutí

I když byl v minulosti hojně používán tento model, v současné době je nahrazován právě agilními metodikami, které lépe reflektují skutečné potřeby vývoje softwaru. Své uplatnění však stále může najít v méně komplexních systémech a aplikací.

1.2 Klasické metodiky

1.2.1 Rational Unified Process (RUP)

Rational Unified Process je nyní první zmíněná opravdová metodika vývoje softwaru, jež by se dala charakterizovat, jako propracovaná, objektově orientovaná iterativní metodika. Metodika byla původně vymyšlena firmou Rational, která byla následně odkoupena (2 bil USD) softwarovým gigantem IBM. Jako zásadní otázky, co se vývoje softwaru týče, na které metodika odpovídá jsou: kdo, co, kdy a jak[2]. Na to abychom mohli správně odpovědět na následující otázky, je potřeba zavést následující 4 elementy[2]:

1. Pracovníci (workers):
    - odpovídají na otázku **kdo**
    - chování pracovníků je popsáno pomocí činností
    - odpovědnost je definována ve vztahu k meziproduktu
2. Činnost (activities):
    - Odpovídá na otázku **jak**
    - jedná se o jednotku práce jednotlivce či skupiny
    - výsledkem je vytvoření nebo modifikace meziproduktu
3. Meziprodukty (artifacts):
    - Odpovídá na otázku **co**
    - jde o hmatatelné výsledky projektu, za něž jsou odpovědní pracovníci
    - jsou využity pracovníky jako vstupy jejich činností a zároveň výstupy
4. Pracovní procesy (workflows):
    - Odpovídají na otázku **kdy**
    - jedná se o posloupnost činností vedoucích k vytvoření kýženého výsledku
    - proces může být modelován pomocí UML

1.2.1.1 Fáze projektu

Fáze projektu pro metodiku RUP jsou opět definovány ve 4 na sebe navazujících krocích, které jsou vykonávány v iteracích. Je zde však zásadní posun v komplexnosti a názornosti vytížení jednotlivých rolí projektu v dané fázi. Z diagramu jde velice dobře vyčíst, jak jsou jednotlivé role v průběhu celého vývoje zatěžovány a tím je možno předejít náhlému nedostatku zdrojů v dané fázi[2].

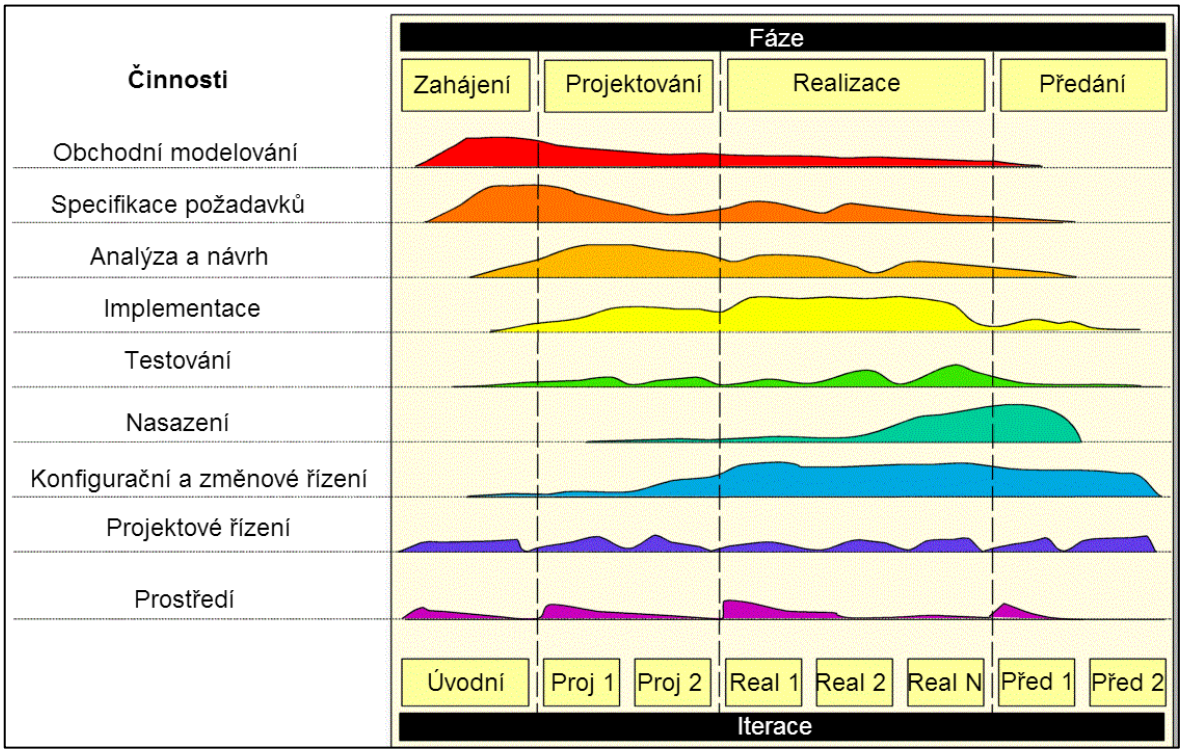
a) Zahájení (inception) – v této fázi je zahájeno prvotní plánování, kooperace se zákazníkem, manažerem, uživateli, vývojáři… Stanovují se případy užití, hranice projektu, cena, je představena alespoň jedna rozumná architektura

b) Projektování (Elaboration) – vytvoření architektury, jsou plánovány jednotlivé části systému, do jisté míry jsou diskutována rizika projektu, jsou specifikovány vlastnosti

c) Realizace (Construction) – tvoření kódu je v této fázi dominantním prvkem, nechybí však ani rozvíjení architektury, průběžné testování, vyhodnocování kvality nebo dodání prvotní betaverze zákazníkovi.

d) Předání (Transition) – jedná se o dodání produktu zákazníkovi. V této fázi je však nezbytné také zaškolení koncových uživatelů a doladění samotného produktu podle nově vzniklých požadavků.

1.2.1.2 Schéma



Obrázek 4 - RUP schéma [16]

1.2.1.3 Výhody

- Metodika je dostatečně obecná, aby se mohla modifikovat přesně za účelem konkrétního projektu.
- Na druhou stranu, je metodika velice detailně popsaná a zdokumentovaná.
- Využívá iterativní přístup.
- Spousta online materiálu dodávaná společně s metodikou.
- Objektově orientovaný přístup k vývoji.
- Provázanost s notací UML[2].

1.2.1.4 Nevýhody

- Stejně tak, jako propracovanost a rozsáhlost metodiky je považováno za výhodu, v menších týmech a jednodušších aplikacích může být tato skutečnost spíše na škodu.
- Neboť se jedná o silně komerční produkt, pro menší nebo studentské týmy bude prakticky nedostupný[2].

1.2.1.5 Shrnutí

Jak již bylo dříve zmíněno, metodika je velice propracovaná a popisuje fáze projektu velice detailně. Pro velké projekty je pak užití této metodiky přínosem, kdežto u menších týmů může být naopak přítěží.

Metodiku lze aplikovat takřka do jakéhokoliv týmu, neboť metodiku lze upravit přímo na míru jednotlivým požadavkům firmy. Je však za potřebí dostatečně zkušených a zběhlých projekt manažerů s dostatkem času a zdrojů pro implementaci.

Jelikož je metodika hojně používaná existuje k ní spousty dokumentací, ať už přímo od firmy Rational, tak prostřednictvím třetích stran. Navíc se metodika ještě stále vyvíjí a tak se uživatel nemusí obávat, že by používal nástroj, který je již zastaralý a nereflektuje aktuální situaci na poli vývoje softwaru.

1.2.2 Unified Software Development Process(USDP -> UP)

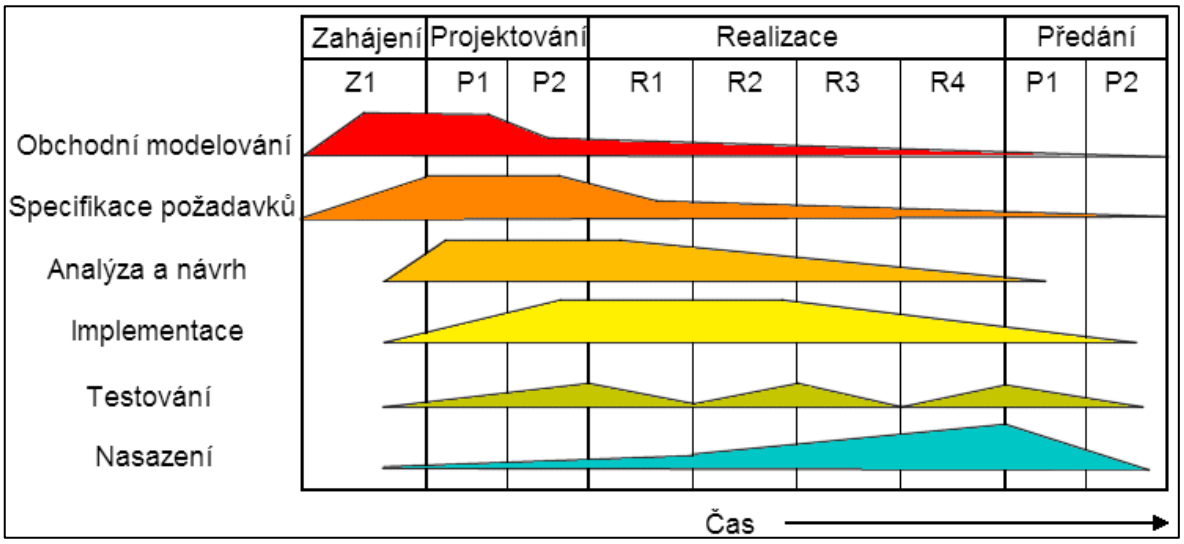
Tato metodika je až podezřele podobná metodice RUP avšak s jedním, pro většinu projektů, zásadním rozdílem a tím je cena. Ačkoliv jsou si metodiky velice podobné, UP vás nebude stát ani korunu, neboť je k dispozici zcela zdarma.

UP je jakýmsi otevřeným standardem od autorů jazyka UML, který je rovněž obsáhnut v komerční verzi RUP a tak jsou fáze projektu poměrně stejné[2].

1.2.2.1 Fáze projektu

e) Zahájení (inception) – realizace podmínek proveditelnosti, zachycení požadavků. Cílem dosáhnout shody napříč všemi účastníky vývoje
f) Projektování (Elaboration) – návrh architektonického základu aplikace, vylepšení odhadu rizik, tvorba plánu implementační fáze
g) Realizace (Construction) – zpřesnění plánů a samotná realizace aplikace
h) Předání (Transition) – předání finálního produktu zákazníkovi, školení uživatelů, dodání dokumentace

1.2.2.2 Schéma



Obrázek 5 - UP schéma [17]

1.2.2.3 Výhody

- Není tak komplexní jako RUP a tedy je vhodná pro menší projekty
- Cena je nulová[2]

1.2.2.4 Nevýhody

- Není tak komplexní jako komerční podoba RUP
- Nepopisuje do hloubky všechny procesy[2]

1.2.2.5 Shrnutí

Jelikož tato metodika vychází ze stejných základů jako RUP, dá se u ní předpokládat určitý stupeň komplexnosti a propracovanosti, nikoliv však na tak vysoké úrovni. S ohledem ale na to, že se jedná o bezplatný produkt, jeho nasazení ve firmě nebude zdaleka tak nákladné jako RUP a tak si jej firma může vyzkoušet, osahat a případně vybrat více vyhovující metodiku.

1.3 Agilní metodiky

1.3.1 Extrémní programování (EP)

Další s předních agilních metodik zabývajícím se vývojem softwaru je Extrémní programování. Možná se zaleknete slovíčka extrémní v názvu, nicméně po nastudování dané metodiky zjistíte, že to není až tak kritické.

První projekt založený na metodice extrémního programování byl odstartován 6. Března 1996. Ačkoliv od tohoto data uběhlo teprve pár let, metodika se úspěšně rozšířila a prokázala svůj potenciál po celém světě ve velikostně nejrůznějších společnostech[5].

Extrémní programování je právě tak úspěšné díky kladení důrazu na uspokojení zákazníka. Metodika umožňuje vývojářům reagovat s jistotou na změny požadavků zákazníka a to dokonce i v pozdější fázi projektu [5]. Jednou ze zajímavostí EP je, že se nelpí na samotném návrh a dokumentace celého systému, nýbrž jen a pouze na zdrojový kód, jako na základního nositele informace[2].

Z předchozího výkladu víme rozdíl mezi klasickým a agilním postupem. EP však tuto skutečnost lehce modifikuje a přidává 4. proměnnou a tou je šíře zadání. Nyní tedy máme:

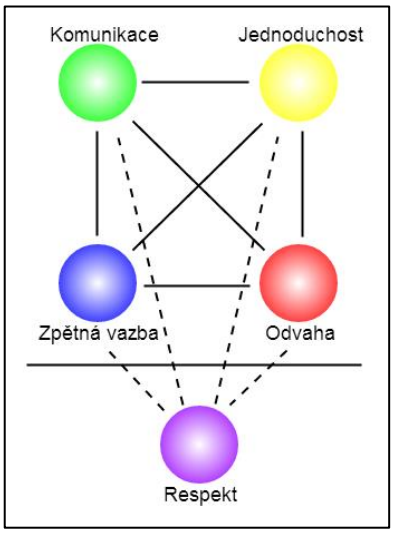
- Kvalitu – odpovídá funkcionalitě
- Čas – ten jen stejný jako u klasických metod
- Náklady – odpovídají zdrojům
- Šíře zadaní – naše nová proměnná

Princip těchto 4 proměnných spočívá v tom, že vývojová skupina vybere a dosadí hodnotu za libovolné 3 proměnné z čehož se 4. proměnná dopočítá. Pokud tým není s výsledkem 4. proměnné spokojen, je potřeba modifikovat některé vstupní parametry. Manažeři a zákazníci mají však tendenci stanovovat všechny 4 proměnné, což pak vede k zásadnímu snížení kvality výsledného produktu[2].

1.3.1.1 Čtyři hodnoty EP

EP vyznává čtyři základní hodnoty, kterými se řídí a jedna tzv. podprahová, která leží mimo teoretický čtverec, nicméně s nimi úzce souvisí. Jsou to tyto hodnoty: komunikace, jednoduchost, zpětná vazba, odvaha a podprahový respekt [2].

a) Komunikace – jako v každém jiném projektu je komunikace nedílnou součástí úspěchu. Nastane-li jakýkoliv problém, z největší pravděpodobnosti je způsoben nedostatečnou, ba dokonce nulovou komunikací
b) Jednoduchost – EP klade také důraz na efektivní implementaci, pokud možno co nejprostší funkcionality. Jako základní otázku tohoto bodu se EP ptá: „Jak vypadá nejjednodušší věc, která může ještě fungovat?“[2].
c) Zpětná vazba – mezi nejúčinnější zpětnou vazbu je považováno testování. Právě testování jednotlivých částí nám dá přesný nezkreslený pohled o daném postupu a průběhu implementace.
d) Respekt – respektem se rozumí vůči dalším kolegům, ke kterým nejsme lhostejní a snažíme se jim pomoci, jak jen to jde. Zajímáme se o jejich práci, její konkrétní průběh a výsledky.

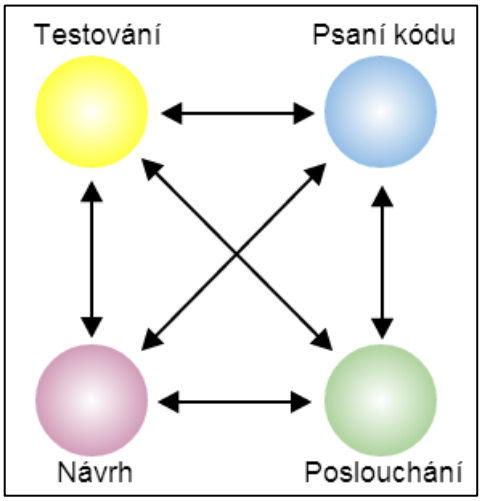


Obrázek 6 – EP hodnoty [2]

1.3.1.2 Čtyři činnosti EP

Jako základní činnosti EP si můžeme uvést následující[2]:

1. Testování – Dříve než je napsán samotný modul aplikace, je nejprve vyvinut automatizovaný test, který danou funkcionalitu otestuje. Automatizovaný je z toho důvodu, aby byly výsledky neovlivněny lidským faktorem a mohli se spouštět i několikrát denně dle potřeby.
2. Psaní zdrojového kódu – fáze samotného programovaní, kdy na konci každé dílčí části je spuštěn automatizovaný test.
3. Poslouchání – vývojář musí naslouchat jak zákazníkovi, tak kolegům, aby bylo vůbec možné implementovat správné a funkční části kódu
4. Navrhování – navrhovaní stejně tak jako psaní kódu, testování a naslouchání patří mezi každodenní rutiny vývoje EP. Chceme-li se vyhnout slepým uličkám, do kterých může každý projekt sklouznout je potřeba správně a efektivně navrhovat.



Obrázek 7 – Činnosti EP [2]

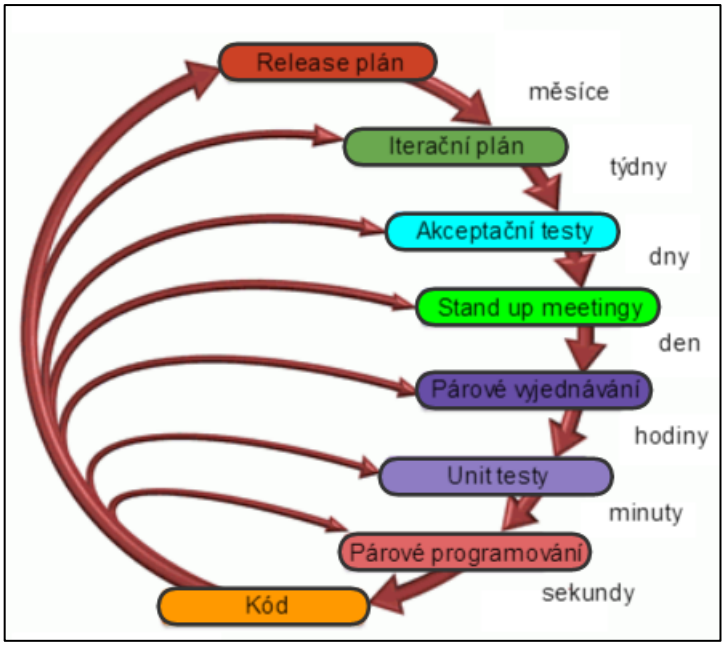
1.3.1.3 Dvanáct základních postupů

Sice jsme si již dříve představili některé uznávané činnosti a hodnoty metodiky EP, nicméně ty jsou spíše obecného rázu a tak je potřeba zavést další, konkrétní postupy, které definoval Kent Beck[6].

1. Plánovací proces – zaznamenat požadovanou funkcionalita zákazníka konfrontovanou s cenou implementace. Taktéž nazývaná „Planning game“
2. Malé vydání – dodat malé průběžné verze, obvykle max. co dva týdny.

3. Metafora – všichni členové týmu používají běžná jména a příběhem popisují fungování celého systému.
4. Jednoduchý návrh – program by měl obsahovat potřebné minimum pro jeho správnou funkčnost. Nikoliv se snažit předvídat budoucnost a implementovat funkce, které by mohly, ale nemusely nikdy být využity. Je ekonomičtější vyvinout jednoduchou věc nyní a za rok zaplatit trochu více za případnou změnu funkcionality, než implementovat dnes robustní řešení, které se v budoucnu nemusí vůbec využívat
5. Testování – každá implementovaná část musí projít automatizovaným testem, který existoval již před samotným kódováním funkcionality.
6. Refaktorizace – dojdeme-li do slepé uličky, nebo najdeme vhodnější řešení již implementované části, je potřeba kód "rozebrat" a znovu vhodněji složit.
7. Párové programování – dva programátoři sedí u jednoho PC, kdy vždy jeden programuje a druhý kontroluje napsaný kód, zda-li splňuje očekávanou kvalitu a funkcionalitu. Po čase se role obrací.
8. Kolektivní vlastnictví – všichni programátoři jsou zodpovědní za celý kód, nikoliv jen za část, kterou sami implementovali.
9. Kontinuální integrace – vývojový tým integruje právě vyvíjenou funkcionalitu i několikrát denně.
10. 40ti hodinový pracovní týden – XP tým dodržuje 40ti hodinový pracovní týden, bez velkého množství přesčasů, aby byla zajištěna pozornost a soustředění všech účastníku vývoje.
11. Zákazník na blízku – zákazník je v průběhu vývoje součástí týmu a tak je od něj očekáváno, že bude k dispozici na zodpovězení všech otázek týkajících se požadavků a správné funkcionality systému. Na blízku je zde míněno opravdu na pracovišti společně s vývojáři.
12. Standart zdrojového kódu – Všichni programátoři dodržují stanovené standardy zdrojového kódu, což umožňuje lepší orientaci a čitelnost v programu stejně tak jako společné vlastnictví celého kódu.

1.3.1.4 Schéma



Obrázek 8 – EP schéma [18]

1.3.1.5 Výhody

- Metodika zaměřená na kódování bez zbytečného papírování
- Produkt je velice rychle dodáván, i s požadovanými změnami v různých fázích vývoje
- Výsledná cena produktu je nižší s ohledem na rychlost vývoje

1.3.1.6 Nevýhody

- Nedostatek plánování a dokumentace může zhoršit orientaci v obsáhlejších kódech
- Přílišná refaktorizace může být zbytečná a zpomalovat celý projekt
- Ne každému může vyhovovat programování v páru

1.3.1.7 Shrnutí

Ačkoliv se EP zdá být velice progresivní metodikou s rychlým postupem a hmatatelnými výsledky již během prvních dní vývoje, pro rozsáhlé projekty a velké týmy je tato metodika takřka nepoužitelná a mohla by vést k zániku projektu, nebo minimálně k nedodržení smluveného termínu.

2 SCRUM A KANBAN

2.1 SCRUM

2.1.1 Historie

Názor, kdo vlastně vynalezl SCRUM se napříč publikem liší a tak není zcela možné určit na které straně je pravda. Někteří tvrdí že SCRUM byl vynalezen Jeffem Shuterlandem, John Scumniotales a Jeffem McKanem v roce 1993. Jiní zase tvrdí, že byl vynalezen již v roce 1986 pány Hirotaka Takeuchi a Ikurijo Nonaka[7]. Pro nás, jako pozorovatele již vytvořené metodiky, to nemusí být až tak zásadní problém, neboť si jej teď zkusíme představit bez ohledu na původního autora.

2.1.2 Úvod

Agilní metodika SCRUM, stejně tak jako ostatní agilní metodiky využívá iterativní a inkrementální postup, jejímž cílem je především dosáhnout efektivního vývoje softwaru.

2.1.3 Charakteristika

V této metodice, oproti Extrémnímu Programovaní, odpovídá každý člen týmu za svou množinu objektů, kterou sám implementuje a spravuje. Mezi základní rysy popisující SCRUM jsou[8]:

- Jednoduchost
- Srozumitelnost
- Extrémně obtížný pro dokonalé zvládnutí

Jelikož je SCRUM spíše procesní rámec, je možné ho svým způsobem vylepšovat a zdokonalovat.

V teorii SCRUMu jsou určeny tři základní pilíře, na kterých celá metodika stoji:

- Transparentnost
- Kontrola
- Adaptace

2.1.3.1 Transparentnost

Veškeré činnosti, se kterými se v průběhu vývoje setkáváme, by měli být zaznamenány v systému, do kterého mají všichni přístup, všichni mu rozumí a používají stejné termíny a názvosloví.[8]

2.1.3.2 Kontrola

Aby se proces neodchýlil od vytyčených standardů, je potřeba jej neustále kontrolovat. Není však žádané, aby přílišná kontrola procesu omezila, ba dokonce zamezila samotnému vývoji. V ideálním případě kontrola prováděna kvalifikovanými lidmi přímo na pracovišti[8].

2.1.3.3 Adaptace

Dojde-li však už k jakékoliv odchylce v procesu, je potřeba na ni správně a hlavně v čas reagovat, aby nedošlo k nepřijatelnému produktu, který by nebyl akceptován. Tato změna musí být provedena v co nejkratším čase, abychom minimalizovali budoucí odchylku[8].

Mezi nástroje vhodné adaptace slouží během sprintu (sprint si vysvětlíme později) následující 4 činnosti:

- Plánování sprintu
- Denní meetingy
- Vyhodnocení sprintu
- Retrospektiva Sprintu

2.1.4 SCRUM termíny

Abychom se správně orientovali v celé terminologii je potřeba si vysvětlit pár základních termínů, které se ve SCRUMu používají.

- Sprint – časově ohraničené období vývoje, které by mělo trvat jeden týden až měsíc
- Backlog – jedná se o souhrn artefaktů (funkcionalit), které je potřeba implementovat
- Releas – již naprogramovaná funkcionalita, připravená k dodání zíkazníkovi
- Blocker – určitá skutečnost/fakt bránící dalšímu postupu ve vývoji
- Bug – chyba v aplikaci
- Ticket – obsahuje popis Bugu / funkcionalita

2.1.5 SCRUM tým

SCRUM metodologie má jasně definované jen a pouze tři základní role s kterými se celý proces vystačí a to jsou:

- Vlastník produktu (Product owner)
- Scrum master
- Vývojový tým

2.1.5.1 Vlastník produktu

Jediná osoba zodpovědná za úspěch celého projektu, která má následující povinnosti[8]:

- Zodpovídá za jasnou formulaci položek v Backlogu a jeho přehlednost a dostupnost
- Nastavuje priority položek v Backlogu.
- Dohled nad kvalitou odvedené práce.

Vlastník produktu může delegovat jednotlivé úkoly na jiné členy týmu, zodpovědnost za odvedenou prací však bude náležet právě jemu.

2.1.5.2 Vývojový tým

Je tým, který dodává patřičnou funkcionalitu vždy na konci sptrintu. Mezi jeho základní rysy patří[8]:

- Jsou sebeorganizující se a tedy jim nikdo nedává pokyny, jak mají co dělat.
- Jsou multifunkční a tak jsou schopni dodat patřičnou funkcionalitu.
- Tým není rozdělen na vývojáře a testery. Vystupuje jako celek.
- Za konečný výsledek zodpovídá celý tým, nikoliv jedinec.

2.1.5.3 Scrum master

Zodpovídá za obecné dodržování pravidel a praktik SCRUMu napříč celým týmem a dále pomáhá vůči jednotlivým rolím[8]:

1. Vlastník produktu
    - Pomáhá vylepšit správu Backlogu.
    - Dohlíží na správnou správu Backlogu.
    - Pomáhá s vysvětlením a aplikací agilní metodiky.
    - Moderuje SCRUM meetingy.

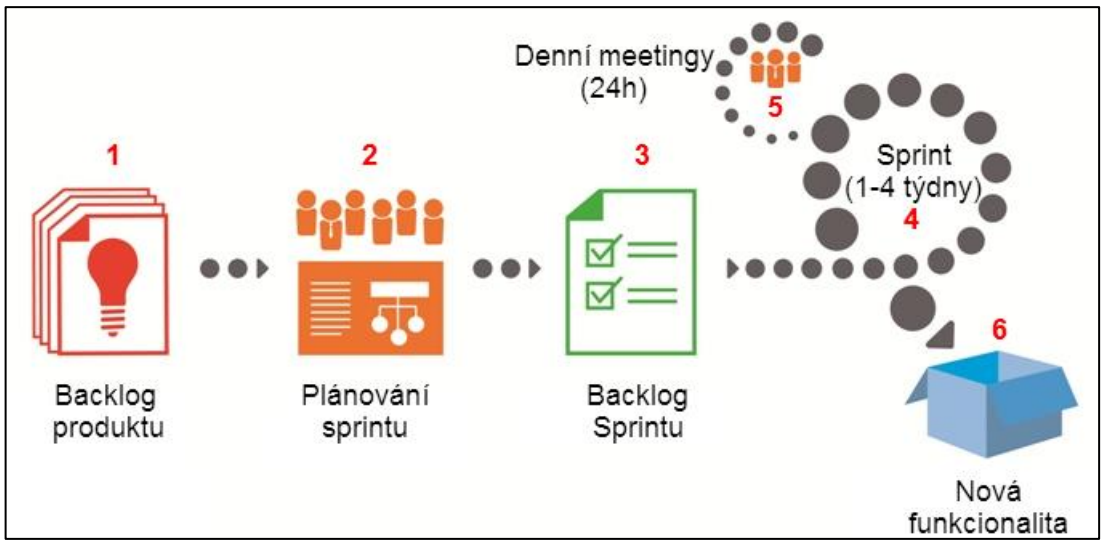
2. Vývojovému týmu
    - Snaží se odstranit „blockery“.
    - Dává týmu jasnější pohled na metologii SCRUMu.
    - Vede tým k tomu, aby byly multifunkční a seberganizující se.
3. Společnosti
    - Provádí školení v otázce SCRUMu
    - Plánuje implementaci SCRUMu
    - Iniciuje vhodné změny v organizaci
    - Spolupráce s dalšími Scrum mastery v organizaci

2.1.6 Fáze projektu

Jednotlivé fáze projektu jsou poměrně jednoduché a chceme-li, aby byl SCRUM správně dodržován je nezbytně nutné, aby je všichni členové týmu dokonale pochopily.

1. Produktový Backlog je naplněn artefakty, které je potřeba implementovat.
2. Artefakty jsou prozkoumány, vysvětleny => připraveny pro zahájení vývoje
3. Z produktového Backlogu jsou určité artefakty (na základě priority) přeneseny do Backlogu sprintu
4. Samotný vývoj je zahájen
5. Každý den se všichni účastní SCRUM meetingu
6. Sprint je ukončen, nová funkcionalita je dodána

2.1.7 Schéma



Obrázek 9 - SCRUM schéma [19]

2.1.8 Výhody

- Transparentnost všech akcí (artefaktů), na čemž si SCRUM zakládá
- SCRUM mastera usnadní nasazeni samotného SCRUMU
- SCRUM master odstraňuje „blockery“
- SCRUM master moderuje meetingy

2.1.9 Nevýhody

- Platí se navíc funkce SCRUM mastera
- Příliš časté meetingy mohou odvádět od „opravdové“ práce

2.1.10 Shrnutí

Metodiku SCRUM považuji za vhodnou všude tam, kde se pracuje na rozsáhlých projektech, ve více časových pásmech a tím pádem ve zhoršených „komunikačních podmínkách“. Díky své transparentnosti metodiky a SCRUM masteru můžou být však tyto problémy eliminovány na minimum.

2.2 Kanban

2.2.1 Historie

Původní Kanban systém byl založen v roce 1947 mužem jménem Taiichi Ohno, který byl tehdy zaměstnancem japonské firmy Toyota Motor Corporation. Jedním z důvodů proč tento nástroj vůbec vznikl, byla nedostatečná produktivita podniku ve vztahu k jeho konkurentům v USA. Ohno popsal koncepci takto: "Mělo by být možné organizovat tok materiálu ve výrobě podle stejného principu, jak fungují supermarkety a to tak, že když spotřebitel vykoupí určitou částku výrobku z regálu, je tato skutečnost zaznamenána zaměstnancem a vybrané zboží se okamžitě doplní."[9]

2.2.2 Úvod

Kanban jako takový je hlavně způsob řízení procesů pouze na základě skutečného vytížení/spotřeby zdrojů. Cílem tohoto systému je tedy zvýšit dostupnost těchto zdrojů a eliminovat výpadky vývoje, kvůli právě nedostatku požadovaných zdrojů.

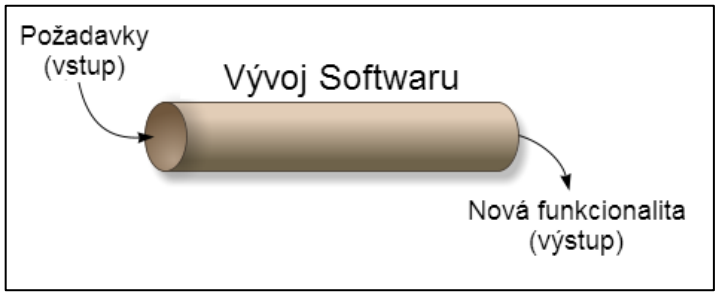
Metoda Kanban je často zaměňován s pojmem just-in-time. Ve skutečnosti je Kanban jedním z hlavních nástrojů JIT systému.[9]

2.2.3 Charakteristika

Proces vývoje software může být chápán, jako trubka s požadavky na nové funkce na jejím vstupu a hotovou funkcionalitou na jejím výstupu.

Uvnitř dané trubky bude proces, který může mít charakter od velice neformálního procesu až k velice konkrétnímu a detailnímu procesu. Pro názorné představení si nyní uvedeme tu první variantu složenou ze tří základních částí:

1. analýza požadavků
2. vývoj
3. testování a nasazení

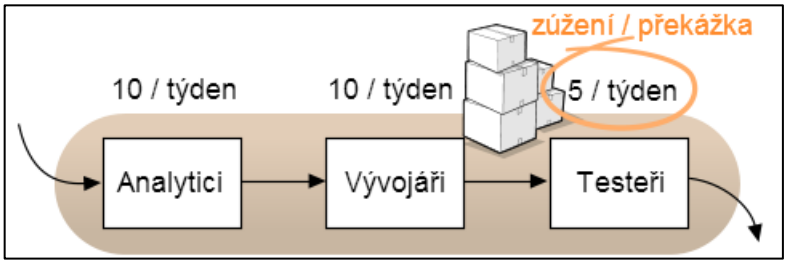


Obrázek 10 – Zjednodušení Kanbanu [10]

2.2.3.1 Princip zúženého „hrdla“

Chceme-li, aby byl vývojový proces stabilní a kontinuální, musíme se vyhnout tomuto zúžení /překážka (uvědomme si však, že toto zúžení je jen a pouze přeneseného významu; v našem případě budeme toto zúžení chápat jako nedostatek zdrojů). zúžení /překážka v potrubí omezuje průtok a stejně tak v našem případě bude tato překážka omezovat náš vývoj. Propustnost potrubí, jako celku, je omezena na propustnost zúžení.[10]

Příklad: v případě, že testeři jsou schopni testovat 5 funkcí týdně, zatímco vývojáři a analytici mají schopnost produkovat 10 funkcí týdně, bude propustnost potrubí, jako celku, jen 5 funkcí týdně, protože testeři zde budou vystupovat právě jako naše zúžení/překážka.



Obrázek 11 – Kanban zúžení [10]

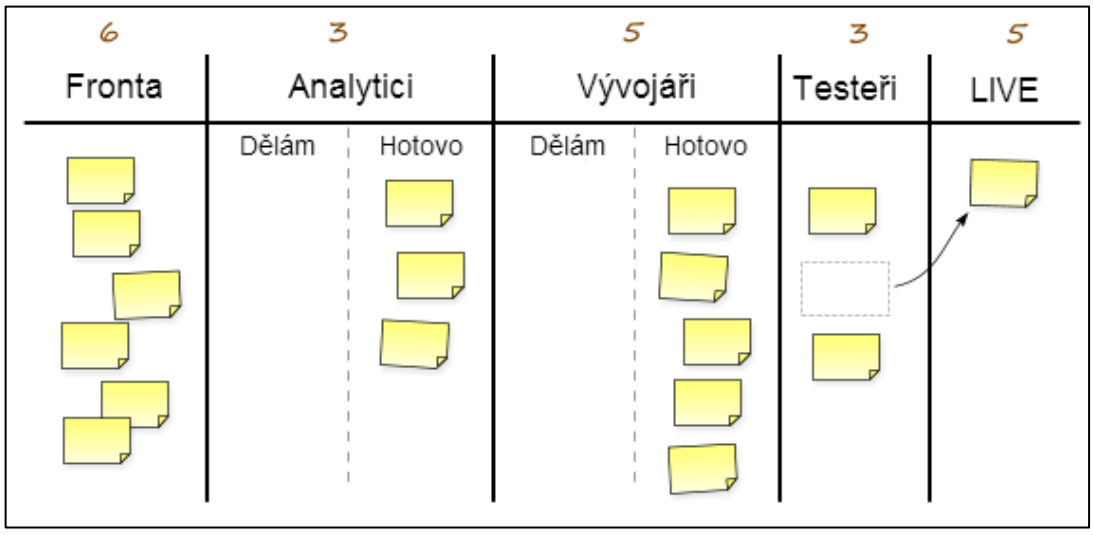
2.2.4 Princip

Kanban je neuvěřitelně jednoduchý, ale zároveň neuvěřitelně silný. Ve své nejjednodušší podobě se systém skládá pouze z velké desky na stěnu a s kartami, nebo poznámkami umístěných ve sloupcích s čísly na vrcholu.

Karty představují pracovní položky („featury“), které protékají procesem vývoje, který je reprezentován jednotlivými sloupci. Čísla v horní části každého sloupce jsou limity na počet karet povolených v každém sloupci.

Limity jsou zásadní rozdíl mezi Kanban nástěnkou a jakým jiným vizuálním „storyboardem“. Omezení množství rozdělané práce v každém kroku procesu zabraňuje nadprodukci jednotlivých týmů a odhaluje překážky dynamicky, takže je můžete řešit dříve, než se projekt dostane do kritického bodu, kdy jeden tým nebude stihat kontinuálně řešit zadané úkoly[10].

2.2.5 Schéma



Obrázek 12 – Kanban storyboard [10]

2.2.6 Shrnutí

Kanban jako takový je velice jednoduchý na vysvětlení a pochopení a však jako vizuální nástroj nesmírně účinný. Přestože tato metodika prvotně vznikla ve výrobním prostředí, není vůbec obtížné ji aplikovat na vývoj softwaru. Užití této metodiky je pak vhodné využít u týmů, kteří sedí v přinejmenším v jedné budově a mají tuto tabuli na dosah.

2.2.7 Výhody

- Jednoduché na vysvětlení, pochopení
- Přehlednost
- Účinnost

2.2.8 Nevýhody

- Pro globální týmy potřeba implementace virtuální tabule
- Při více úloh může být tabule značně nepřehledná
- Jednotlivé úkoly se mohou z nástěnky ztratit

II. PRAKTICKÁ ČÁST

3 ROZŠÍŘENÍ AGILNÍ METODIKY PRO VÝVOJ WEBOVÝCH APLIKACÍ

3.1 Analýza

Abychom se správně rozhodli, jakou metodiku vybrat je nezbytné provést alespoň základní analýzu potřeb vývojového týmu webových aplikací za účelem zjištění, jaké požadavky jsou kladeny.

3.1.1 Komunikace

Komunikace je jednoznačně jeden z nejdůležitějších prvků každého vývojového týmu. Pokud by tým správně nebo dostatečně nekomunikoval, nebylo by možné včasně reagovat na změny jednotlivých programátorů v kódu a tím by se celý princip agilních metodik vytratil. Taktéž je potřeba komunikovat a radit se mezi sebou, pokud nastane jakýkoliv problém, bránící vývojovému týmu zdárného nasazení patřičné funkcionality to provozu.

3.1.2 Přehlednost

Každý artefakt ve sprintu by měl být jasně zadán s jasnými požadavky a měla by se vést jeho historie aktivit od založení až po jeho nasazení/konec. Každý člen vývojového týmu by měl nést plnou zodpovědnost za daný artefakt nebo alespoň za část, s kterou aktivně pracoval nebo vytvářel a měnil.

3.1.3 Jednoduchost

Žádný vývojový tým nebude odvádět dobrou práci, pokud bude nucen pracovat v chaotickém a nepřehledném pracovním prostředí, v kterém by se ztrácely artefakty, nebo bude zahlcen v nadměrném množství „papírovací“ práce, která je nejen otravná, ale v mnoha případech i zbytečná.

3.1.4 Udržovatelnost a správa

Celý proces by měl být jednoduchý na údržbu a správu. S jednotlivými artefakty by se mělo při vývoj smět pohodlně manipulovat a přenášet v případu potřeby z jedné iterace vývoje do druhé.

3.2 SCRUM pro webové aplikace

Na základě představených životních cyklů a jednotlivých metodik, ať už klasických či agilních a jednoduché analýzy potřeb se jako nejvhodnější řešení jeví agilní metodika SCRUM, na jejímž základě postavím realizaci procesu pro vývoj webových aplikací. Celý proces s jednotlivými fázemi bude realizován v této kapitole.

Jako hlavní přednosti SCRUMu, kterými převyšuje ostatní metodiky, vidím v:

a) Agilním přístupu
b) Přítomnosti SCRUM mastra
c) Volitelné délce sprintu
d) Denních meetinzích
e) Dostatečné (nikoliv přehnané) dokumentaci
f) Transparentnost sprintu a artefaktů v něm
g) Vlastník produktu zodpovědný za dodání, nikoliv manager
h) Schopnost dodat produkt ve stanoveném čase
i) Pravidelná zpětná vazba od zákazníka
j) Měřitelná produktivita (pomocí „burndown“)

3.3 Sprint

Jak jsme si již dříve řekli, SCRUM je metodika agilní, která se skládá z jednotlivých fází vývoje. Od plánování přes vývoj až po nasazení. My si nyní pokusíme metodiku přizpůsobit každodenním potřebám vývoje webovým aplikacím v týmu, který je geologicky (tedy i časově) distribuován po celém světě.

Jelikož se jedná a vývoj webových aplikací je vhodné volit délku sprintu 2 týdny což je dostatečně krátká doba na rychlé reakce změn požadavků, na druhou stranu dostatečně dlouhá doba na vývoj velkého množství funkcionalit a opravu chyb (bugů), která není zahlcena přílišným a hlavně častým plánováním, jak by tomu bylo, kdybychom zvolili sprint o minimální přípustné délce v celkovém trvání jednoho týdne.

3.4 Plánování

Plánování každého sprintu probíhá vždy na jeho začátku a je rozděleno na dvě části: neveřejné a veřejné plánování; probíhají chronologicky za sebou a je na nich rozebírána

podoba konkrétního sprintu. Obě tyto plánování by neměli zabrat více jak jeden den sprintu dohromady.

Obrázek 13 – Plánování [20]

3.4.1 Neveřejné

Neveřejné plánování probíhá za přítomnosti Vlastníka produktu, který na základě byznys přínosu stanový, které tikety se budou v nadcházejícím sprintu implementovat a SCRUM mastera, který pomůže sestavit celou podobu sprintu a dohlédnout na to, že jsou veškeré položky řádně popsány a je jejích funkcionalita přesně vytýčena.

V této částí plánování (jedná-li se o vůbec prvotní plánování) jsou veškeré tickety (bugy / funkcionality) zadány do systémového backlogu a následně po analýze přesunuty do sprint backlogu, z kterého se budou přidělovat jednotlivým developerům.

Veškeré požadavky by měli obsahovat následující položky:

- Typ požadavku – oprava bugů která vznikla v průběhu předchozí implementace nové funkcionality / nová funkcionalita
- Priorita – stanovená priorita tiketu na základě byznys přínosu, který oprava / implementace přinese do celého projektu
- Popis – popis jak a kdy se BUG projevuje (očekávané chování / aktuální chování) / popis nové funkcionality

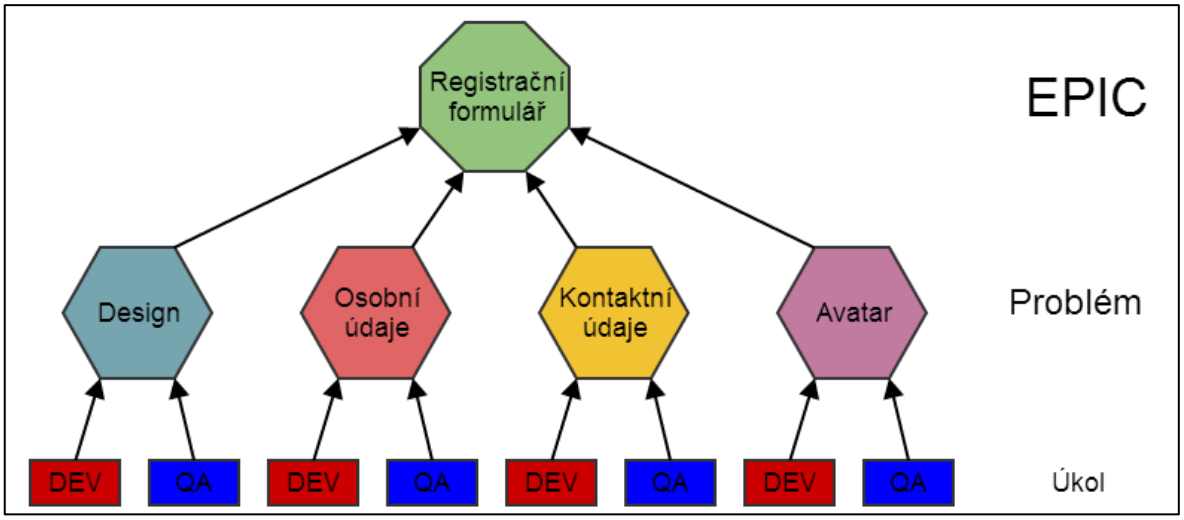
- Akceptační kritéria – jak se bude aplikace chovat s opraveným BUGem / jak se bude aplikace chovat s novou funkcionalitou

3.4.1.1 Organizace tiketů

Jelikož mohou být nové funkcionality složitějšího rázu, je potřeba je rozdělit na několik menších tiketů, které jsou pak možné zadat více developerům najednou. Aby toto bylo možné musíme se blíže seznámit s hierarchickou strukturou tiketů. Jako dostatečnou hierarchii se dá považovat 3 stupňová obsahující následující prvky:

- Epic
  - Sdružuje veškeré problémy k celé dané funkcionalitě
- Problém (Příběh / Chyba)
  - Jeden problém je složen z minimálně jednoho DEV úkolu a jednoho QA úkolu
- Úkol (pro developera (DEV), pro testera (QA))
  - Elementární jednotka popisující prací DEV/QA

Příklad: Představme si, že je potřeba naprogramovat jednoduchý registrační formulář. Jako Epic zde bude právě sloužit „Registrační formulář“. Tento Epic v sobě bude obsahovat jednotlivé problémy (příběhy), kterými může být např. „design stránky“, „osobní údaje“, „kontaktní údaje“ a „avatar“. Každý takto vytvořený problém v sobě bude obsahovat min. jeden úkol pro developera(DEV) a minimálně jeden úkol pro testera(QA). Tímto jednoduchým návrhem máme pokrytou základní funkcionalitu. Schéma této struktury by mohlo vypadat následovně:



Obrázek 14 - Hierarchie tiketů

3.4.2 Veřejné

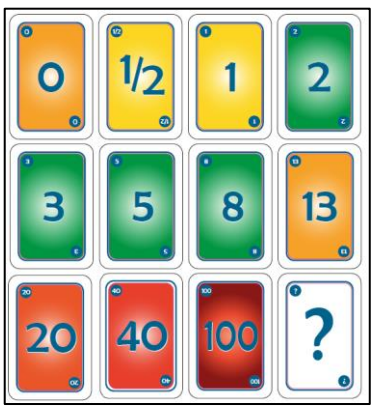
Tohoto plánování už se účastní všichni členové týmu, tedy i vývojový tým složený z vývojářů a testerů. V této fázi plánování se stanovuje odhadovaná časová náročnost jak z pohledu vývoje tak testování. Pro časový odhad náročnosti se může využít takzvaný „Planning poker“ nebo také „Story point poker“ (ten si vysvětlíme později). Jsou-li veškeré položky patřičně vysvětleny a časově ohodnoceny, Scrum master je přesune do sprint backlogu, kde čekají na zahájení sprintu a přidělení jednotlivým developerům.

Položky by měli být přesunuty na základě byznys přínosu ($) a časové náročnosti. Neměli by se však přesouvat selským rozumem, pouze na základě $.

Příklad: Funkcionalita A má byznys přínos 50 a časovou náročnost 25. V systémovém backlogu však máme dvě podobné funkcionality B, každá s byznys přínosem 40 a časovou náročností však pouze 10. Jednoduchou matematikou lze spočítat, že funkcionalita A má koeficient přínos 50 / 25 = 2 kdežto koeficient přínosu pro obě funkcionality B dohromady 80 / 20 = 4. Zde je jasně vidět, že implementací dvou méně časově náročných funkcionalit lze dosáhnout dvakrát tak velkého byznys přínosu, než kdybychom implementovali právě „nejhodnotnější“ funkcionalitu A.

3.4.2.1 Story point poker

Pomocí story point pokeru lze odhadovat časová náročnost jednotlivých tiketů. V rámci veřejného plánování jsou jednotlivým tiketům přiřazovány číselné hodnoty 1,2,3,5,8,13,20,40,100 na základě stanovení referenční hodnoty 8. Pro tuto hodnotu(8) se zvolí již implementovaná funkcionalita, které jsou si všichni členové vědomi i její časové náročnosti a od této hodnoty se odvíjí každé další odhadování časové náročnosti pro další tikety.



Obrázek 15 – Story point poker [21]

Příklad: V minulosti se implementovala funkcionalita, jejihž časová náročnost odpovídá časové náročnosti 8:

- developeři jsou si vědomi, kolik času strávili daný tiket implementovat
- testeři jsou si vědomi kolik času strávili daný tiket otestovat.

Tato hodnota bude tedy hodnotou referenční pro každé další odhadování nových Bugů / funkcionalit.

3.5 Vývoj

3.5.1 Prostředí

Celé prostředí, ve kterém probíhá vývoj, by mělo být odděleno od testovacího prostředí, aby bylo možné v průběhu vývoje již testovat dokončené funkcionality jiných developerů.

Z tohoto prostého důvodu je potřeba mít alespoň 3 (ideálně však 4 prostředí): DEV, QA, STG, PROD. V případě pouze tří stupňového vývojového prostředí lze DEV a QA sloučit do jednoho. My si však ukážeme právě čtyř stupňové prostředí.

3.5.1.1 DEV prostředí

V DEV prostředí je zahájen prvotní vývoj všech nových funkcionalit a oprava bugů. Prostředí nemusí být napojeno na zbytek systémů. V tomto prostředí může být velké množství rozpracovaných kódů v jakémkoliv stádiu vývoje. Jen hotové a plně funkční kódy se z tohoto prostředí mohou přesunout na QA prostředí.

3.5.1.2 QA prostředí

V tomto prostředí probíhá první část testování samotné funkcionality kódu testery. Prostředí je napojeno na zbytek systému, nebo aspoň na část, s kterou kód komunikuje napřímo.

3.5.1.3 STG prostředí

Aby se mohla ověřit skutečná funkčnost aplikace dříve, než je vpuštěna do produkčního prostředí je potřeba mít přesnou kopii tohoto prostředí dostupnou k plnému otestování všech nových funkcionalit.

3.5.1.4 PROD prostředí

PROD, nebo-li produkční prostředí je prostředí, na kterém již pracují reální uživatelé a tak je hlubší testování takřka nemožné. Na tomto prostředí se provádí pouze rychlý test základní funkcionality.

3.5.2 Developeři

3.5.2.1 Analýza

Samotný vývoj začíná, jakmile jsou tikety přiřazeny developerům. Není však možné začít okamžitě programovat, aniž by se nejprve provedla alespoň základní analýza požadovaného tiketu. V průběhu analýzy developer zjišťuje případné nesrozumitelnosti v tiketu, snaží se objasnit celou funkcionalitu od Vlastníka projektu, pokud není jasně stanovena již z dřívějšího neveřejného plánování. Také je potřeba si správně nastavit prostředí, ověřit si že má developer přístup všude tam, kam daná funkcionalita bude zasahovat a komunikovat případně s jinačími systémy.

Jakmile je analýza dokončena a prostředí správně nakonfigurováno, začíná samotné psaní kódu.

3.5.2.2 Vývoj

Developer, na základě popisu v tiketu, programuje konkrétní funkcionalitu, při čemž by měl dbát zásad správného formulování kódu:[11]

- Názvy by měli být popisné
- Užívání konstant
- Krátké funkce
- Jednoúčelové funkce
- Funkce s malým počtem argumentů

Nedílnou součástí všech aplikací by měla být možnost logovat akce, které se v daný čas v dané aplikaci spustily, čí provedli. Implementace logování většinou nebývá součásti zadání, avšak je běžnou praxí jej do programu zahrnout. Velice se tím usnadní případné řešení špatně (v krajních případech vůbec) fungujících aplikacích.

3.5.2.3 Dokumentace

Každá nová funkcionalita by měla být dobře zdokumentována. Tato dokumentace by měla obsahovat seznam metod, které se v aplikaci užívají, jednotlivé vstupy do nich a jednotlivé výstupy z nich. Taktéž může obsahovat jednoduchý popis samotného kódu pomocí pseudokódu.

Příklad:

Metoda: Vytvoř kontakt

Vstup:

Proměnná	Povinná položka	Typ	Poznámka
ID	Ano	Integer	
FirstName	Ano	String	Jméno
LastName	Ano	String	Příjmení
Email	Ano	String	
Tel	Ne	Number	
Address	Ano	text	adresa

Tabulka 1 – Vytvoř kontakt (vstup)

Výstup:

Proměnná	typ	poznámka
ID	Integer	
Status	String	OK
ErrorMsg	text	Chybové hlášení

Tabulka 2 – Vytvoř kontakt (výstup)

Pseudokód:

1. Načti všechny proměnné
2. Vygeneruj jedinečné ID
3. Ulož záznam
4. Vypiš ID a k němu Status / chybové hlášení
5. Ukonči program

3.5.2.4 Revize kódu

Pro každý napsaný kód by měla proběhnout revize od jiných zkušených developerů, kteří mohou poukázat na případnou nečitelnost kódu, slabá místa, špatně používání zdrojů, či jiných lehkých nebo dokonce hrubých porušení zásad. Touto revizí, nejen že se sníží riziko vypuštění nevhodného kódu do aplikace, ale také se samotný developer může poučit a příště vyvarovat stejným chybám.

3.5.2.5 Jednotkový test

Každý developer by měl po skončení všech předchozích kroků (analýza, vývoj, dokumentace, revize kódu) provést alespoň základní ověření funkcionality. Toto ověření je opravdu pouze na bázi základní funkcionality a v žádném případě není dostatečným otestováním. Opravdové a hloubkové end-to-end (E2E – od konce-do-konce) otestování provede QA tým.

3.5.3 Testeři (QA)

3.5.3.1 Analýza

Dříve, než je vůbec možné, jakoukoliv aplikaci, či pouhou funkcionalitu řádně otestovat, je nutné se s ní nejprve seznámit. K tomuto účelu nejlépe poslouží právě developery zhotovená dokumentace, popisující celou funkcionalitu. Najde-li tester slabá místa dokumentace je potřeba informovat developera, aby tyto místa doplnil dalšími patřičnými informacemi.

3.5.3.2 Test case

Aby bylo možné funkcionalitu otestovat i za pár týdnů/měsíců je potřeba napsat správný Test case, což je vlastně popsaný krok za krok, kam kliknout, jaká vstupní data vyplnit, na co se při kontrole zaměřit a co nepřehlédnout. Z takto vytvořených test casů se v pokročilé fázi projektu sestaví regresní sada, kterou se ověřuje průběžně předchozí funkcionalita. Každý test case by měl obsahovat:

- jednoduchý popis jakou funkcionalitu pokrývá
- požadavky
- kroky
- odhadovaný čas pro jeho projití

Příklad:

TC017 – Osobní údaje

Popis: Ověření správné funkce vkládání osobních údajů

Požadavky: Přístup do aplikace

Kroky:

Krok:	Očekávaný výstup
1. **Spusťte aplikaci**	Aplikace je spuštěna
2. **Vložte jméno**	Jméno je vloženo
3. **Vložte příjmení**	Příjmení je vloženo
4. **Zvolte pohlaví**	Pohlaví je zvoleno
5. **Stiskneme tlačítko „OK“**	Záznam je uložen
6. **Zopakujeme kroky 2-5 pro jiné vstupní hodnoty**	Aplikace se chová korektně

Tabulka 3 - Příklad Test casu

Klíčová slova: Manuální, Regresní kandidát

Odhadovaný čas: 10 minut

3.5.3.3 Výkonnostní test

Nejen že aplikace musí být napsaná správně po funkční stránce, ale také po stránce optimalizační. Výkonnostní test by měl tedy ověřit, jak se aplikace chová na prahu stability např. zahlcením velkým množstvím vstupních dat v krátkém časovém úseku. Tyto testy, už jen z jejich povahy na generování vysokého počtu vstupních dat za krátký časový úsek, takřka není možné provádět manuálně, ale je potřeba napsat automatizovaný test, nebo alespoň generátor těchto dat. Při výkonnostním testu bychom měli simulovat minimálně stejné množství vstupních dat, které bude muset aplikace zvládat obsluhovat po nasazení do opravdového provozu.

3.5.3.4 Test bezpečnosti

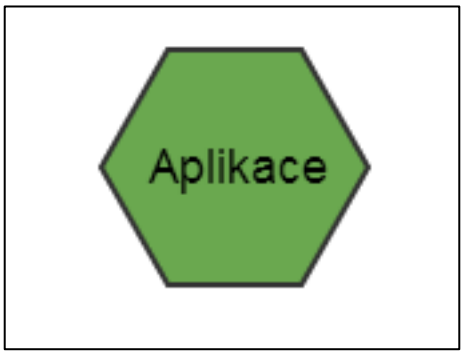
Každá aplikace by také měla splňovat alespoň základní požadavek na bezpečnost. Při bezpečnostních testech se pak zaměřuje pozornost na slabá místa v aplikaci, které by

případný útočník mohl napadnout a zneužít. Je potřeba otestovat, že pouze uživatelé s patřičným oprávněním mohou přistupovat k jednotlivým funkcím webové aplikace a mohou modifikovat jen to, na co má právě daná role práva.

Obrázek 16 - Bezpečnotní test [22]

3.5.3.5 Jednotkový test

Jakmile je developer s prací hotovo přichází na řadu tester. Ten stejně tak jako developer otestuje samotnou funkcionalitu, zda-li se opravdu chová tak, jak by měla a plní předem stanovený účel. Jak již bylo dříve zmíněno, testuje se jak pozitivní tak negativní metodou a tedy vkládáním očekávaných a nevhodných dat, za účelem otestování nepřípustných stavů aplikace a ošetření výjimek.

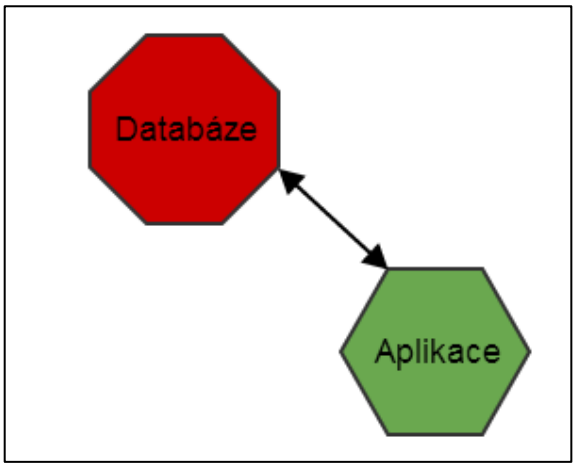


Obrázek 17 - Jednotkový test

3.5.3.6 Integrační test

Je-li funkcionalita připojena k dalším komponentám systému je nutné otestovat vzájemná komunikace všech „sousedních“ systému, s kterými aplikace přímo komunikuje. Máme-li

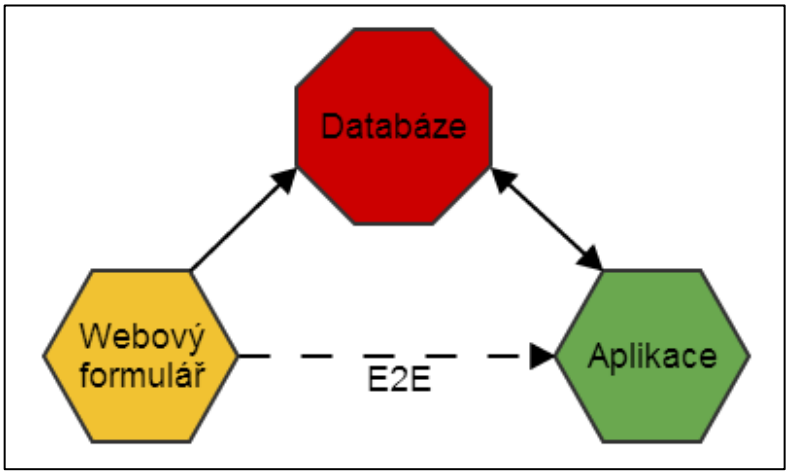
aplikaci, která zpracovává data z databáze, musíme ověřit, že tato komunikace probíhá v pořádku a že má databáze své zdroje dostupné a aplikace má k těmto zdrojům přístup.



Obrázek 18 - Integrační test

3.5.3.7 End-to-end test

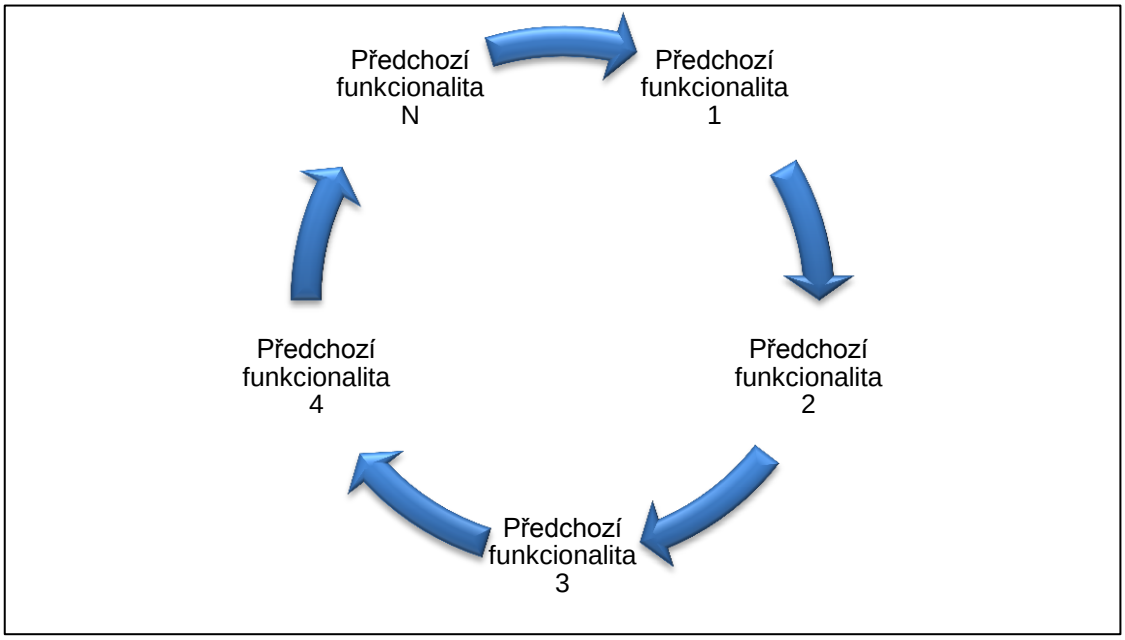
Máme-li systém složený z více samostatných komponentů a implementujeme novou funckionalitu jednoho systému, která ovlivňuje funkci druhého systému je potřeba tyto návaznosti otestovat. Dejme tomu, že na webové stránce se vyplní registrační formulář, ten putuje do databáze a z databáze si jej vytáhne a zpracuje třetí aplikace (námi právě dodaná). Je potřeba otestovat, zda-li celá tato posloupnost akci se správně provede a do naší aplikace vstupují správná data z registračního formuláře přes databázi.



Obrázek 19 – E2E test

3.5.3.8 Regresní test

Regresní testování se provádí vždy na konci každého sprintu, aby se ověřila neporušenost původní funkcionality implementováním nových částí kódu. Toto testování se již provádí na STG prostředí, které je identické s produkčním prostředím s tím rozdílem, že na STG prostředí je přidaná nová funkcionalita ze sprintu. Do regresní sady je tedy potřeba zahrnout veškeré test casy, které pokrývají funkcionalitu, s kterou nové přírůstky kódu přímo pracují. V našem případě to budou test casy pokrývající funkcionalitu na straně databáze.



Obrázek 20 - Regresní test

3.5.4 Vlastník produktu

3.5.4.1 Akceptační test (UAT)

Jakmile je vše naprogramováno a otestováno, dříve než dojde na nasazení nových funkcionalit, je potřeba je představit klientovi (v případě SCRUMu Vlastníkovi produktu), který ověří, že se aplikace chová tak, jak bylo požadováno a zmíněno v daných tiketech. Tímto testováním nám Vlastník produktu v ideálním případě dá zelenou a tým je připraven na nasazení nových funkcionalit do produkčního prostředí. V opačném případě sdělí Vlastník produktu, co je špatně implementováno a tým je nucen funkcionalitu opravit/změnit a dodat aplikaci znova pro nové UAT. Druhá varianta je méně častá, neboť tiket by měl obsahovat dostatek informací o dané funkcionalitě a tedy by nemělo dojít

k nepochopení funkcionality nové funkce a taktéž je Vlastník produktu stále k dispozici k případnému objasnění nesrovnalostí.

Obrázek 21 - Akceptační test [23]

3.6 Nasazení

3.6.1 Načasování

Jakmile se sprint blíží ke konci je potřeba začít uvažovat, jak proběhne samotné nasazení nových přírůstků do produkčního prostředí. Jedná-li se o změny přímo ovlivňující používání aplikace koncovými uživateli, není možné, aby se přírůstky nasazovali „za provozu“. Je však možné informovat uživatele, aby v daném časovém rozmezí, kdy se bude provádět nasazení, se systémem nepracovali, nebo provádět změny v dobu, kdy je provoz aplikace nulový, což může být v pozdních večerních hodinách, nebo o víkendu.

3.6.2 Řešení problémů

Taktéž je potřeba mít připravený záložní postup, vrácení celého systému do původního stavu, kdyby něco nešlo podle plánu a systém se začal chovat po nasazení nové funkcionality nestandardně/neočekávaně. Jsou-li změny prováděny na serveru, který běží na virtuálním stroji, není nic jednoduššího, než před samotným nasazením pořídit kopii aktuální podoby systému (což většina virtualizačních řešení nabízí) a v případě problému pouze přehrát touto zálohou neúspěšné nasazení.

3.6.3 Verifikace správného nasazení

Pokud proběhlo nasazení v pořádku a systém nejeví žádné známky chyb je vhodné tento systém ještě nějaký čas sledovat pro případné problémy, které se můžou vyskytnout ne nutně hned po nasazení. Taktéž je vhodné provést alespoň základní otestování funkcionality systému takzvaný smoke test.

Smoke test nezachází do úplných detailů systému, ale opravdu jen ověření, že základní funkcionalita je zachována. V našem příkladu webového formuláře bychom tedy zkusili zadat jednu registraci a zkontrolovali správné zapsání do databáze a správnou modifikaci provedenou naší aplikací. Je tu jistá podobnost s dříve zmíněným end-to-end testem, ten je však více důsledný oproti pouhému smoke testu.

4 NÁSTROJE UŽÍVANÉ PŘI DOKUMENTOVÁNÍ PROJEKTU

Nástroje užívané v jednotlivých fázích projektu si nyní představíme v následující kapitole. Zaměříme se na úvodní fázi projektování, fázi řízení (průběh sprintu) a konečnou fázi testování na prostředí STG těsně před nasazením.

4.1 Úvodní fáze

4.1.1 UML

Modelující jazyk UML je vhodným nástrojem pro návrh takřka celého projektu od počáteční Úvodní studie přes Případy užití až po Diagram aktivit. Nyní si jednotlivé části návrhu představíme.

4.1.1.1 Úvodní studie

Úvodní studií se rozumí obecný slovní popis dané funkcionality, její rozsah, za jakým účelem se aplikace vytváří, jaké modelové scénáře bude aplikace podporovat, jací uživatelé vstupují do aplikace a jaké mají možnosti interakce s aplikací. Taktéž může být doplněn omezujícími podmínkami a odpovědnou osobou, která za specifikace, nebo celý projekt zodpovídá.

Příklad:

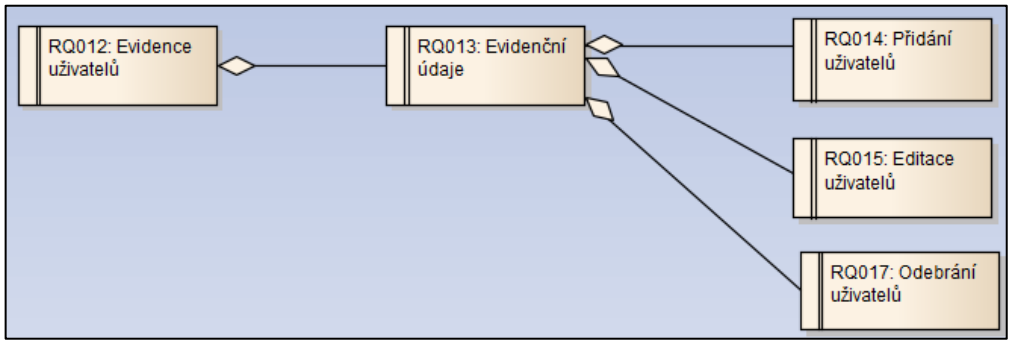
- Popis: Registrační formulář (Základní popis) - Daný dokument popisuje on-line registrační formulář, pro který by měla být nalezena vhodná softwarová realizace. Tento systém se bude používat za účelem registrace nových uživatelů do již funkčních a nasazených stránek. Potřebná data budou uchovávána ve zvolené databázové struktuře. Registrovaným uživatelům bude zpřístupněno zasílání soukromých zpráv dalším registrovaným uživatelům.
- Účel: Systém se vytváří za účelem zefektivnit správu uživatelů na webových stránkách, mít přehled o přistupujících uživatelích a umožnit jim zasílat soukromé zprávy.
- Rozsah: Systém bude mít vytvořeno více uživatelských rolí a podle nich přizpůsobeny možnosti a chování systému.

- Modelový scénář
    - Registrace do systému
    - Přihlášení do systému
    - Editace uživatelů
    - Hledání uživatelů
    - Zasílání zpráv mezi uživateli
    - Smazání uživatele
    - Tvoření reportů
    - Zpětná vazba od zákazníka
- Uživatelé
    - Administrátor
        - Správa moderátorů
        - Správa uživatelů
        - Zasílání zpráv
        - Tvoření reportů
    - Moderátor
        - Správa uživatelů
        - Zasílání správ
        - Tvoření reportů
    - Uživatel
        - Správa svého účtu
        - Zasílání zpráv
        - Zpětná vazba
- Omezující podmínky
    - Uživatelsky přívětivé rozhraní
    - Vhodný design
    - Zabezpečení (účty)
- Odpovědná osoba/osoby
    - Jan Ovesný
    - Karel Novotný

4.1.1.2 Požadavky

Stanovením funkčních a nefunkčních požadavků se definuje celá funkcionalita projektu. Funkční požadavky jsou ty, které zajišťují funkčnost aplikace. Nefunkční požadavky kladou důraz na design a omezení[12].

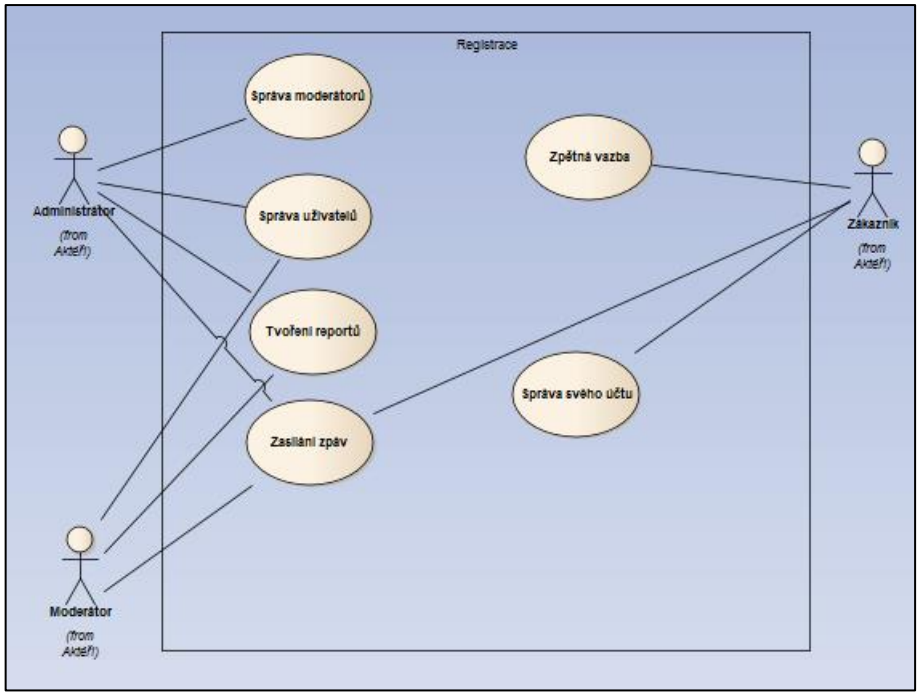
- Funkční požadavek: Evidence uživatelů
- Nefunkční požadavek: Webová aplikace pro zákazníka



Obrázek 22 – UML Funční požadavek (Evidence uživatelů)

4.1.1.3 Případy užití

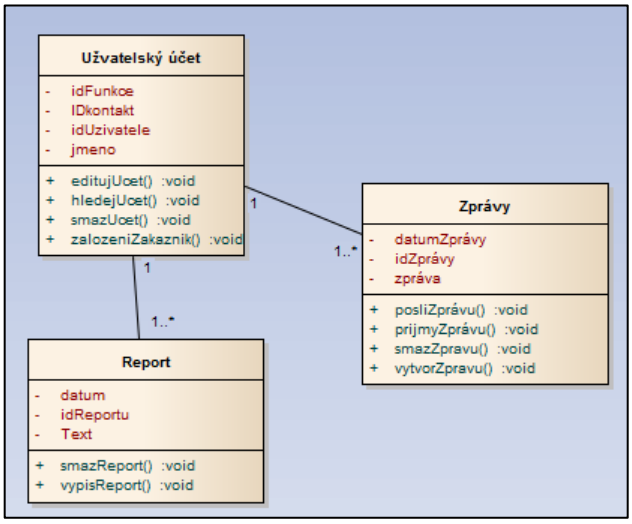
Případy užití jsou zjednodušeně jednotlivé akce, které jednotliví aktéři mohou provádět v dané aplikaci a vycházejí z úvodní studie pro všechny uživatele.



Obrázek 23 – UML Případy užití

4.1.1.4 Diagram tříd

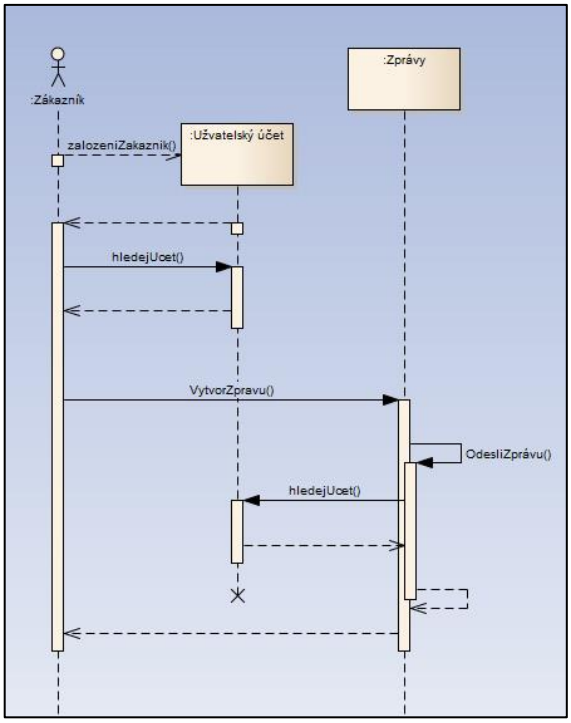
Jedná se o návrh tříd, které se v aplikaci budou vyskytovat. Každá třída obsahuje vnitřní proměnné a metody.



Obrázek 24 - UML Diagram tříd

4.1.1.5 Sekvenční diagram

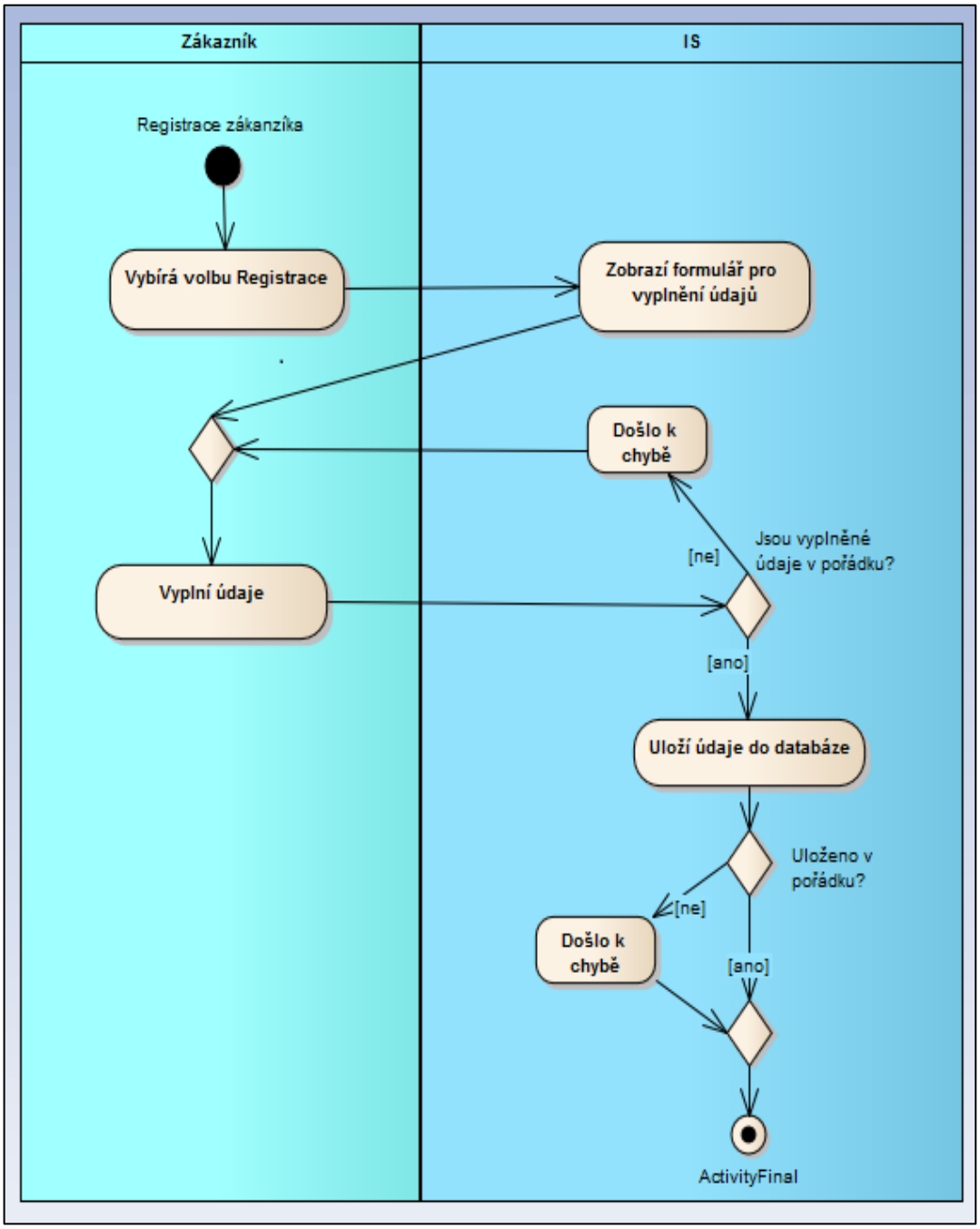
V sekvenčních diagramech je zaznamenán tok dat (zasílání zpráv) a spolupráce objektů pro konkrétní případ užití[13].



Obrázek 25 - UML Sekvenční diagram

4.1.1.6 Diagram aktivit

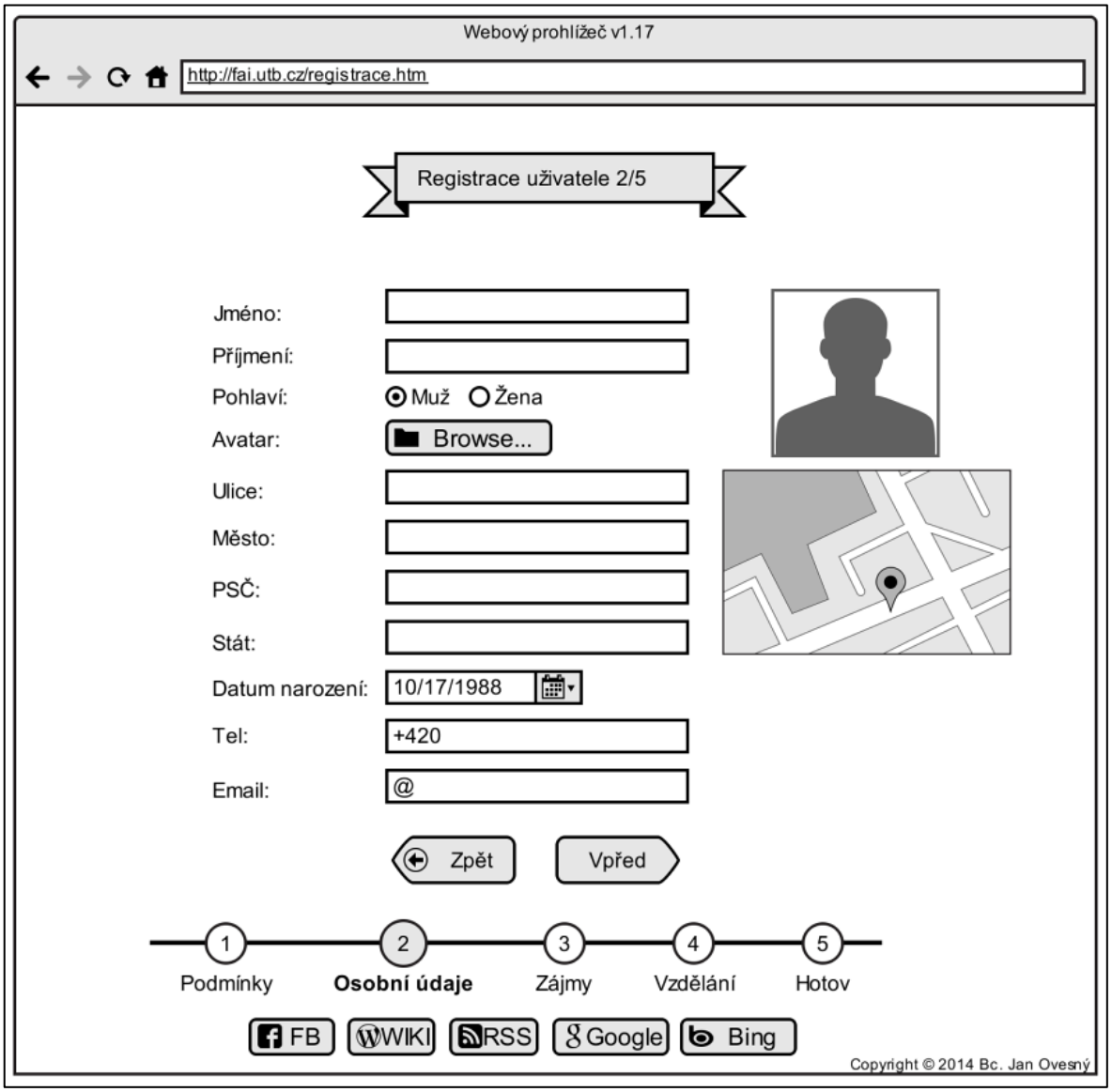
Diagram aktivit popisuje interakci mezi jednotlivými systémy a zaznamenává procedurální logiku, byznys proces nebo pracovní postup. Pomocí diagramu aktivit je také možné modelovat případy užití reprezentovanou jako posloupnost akcí[14].



Obrázek 26 - UML Diagram aktivit

4.1.2 Mockup

Při prvotním návrhu designu jakékoliv webové aplikace je takzvaný mockup takřka nezbytnou součástí. Správně navržený mockup umožní developerům vhodně rozvrhnout jednotlivé elementy na stránku, tak jak si Vlastník produktu žádá. Mockup je vlastně takový prvotní návrh, jak by celá stránka měla vypadat a kterého by se měl developer držet. Pokud by se v budoucnu implementovaný návrh aplikace Vlastníkovi produktu nelíbil, mockup je kromě samotných požadavků, jedinou „obranou“, kterou developer může argumentovat, proč stránka vypadá tak, jak vypadá.



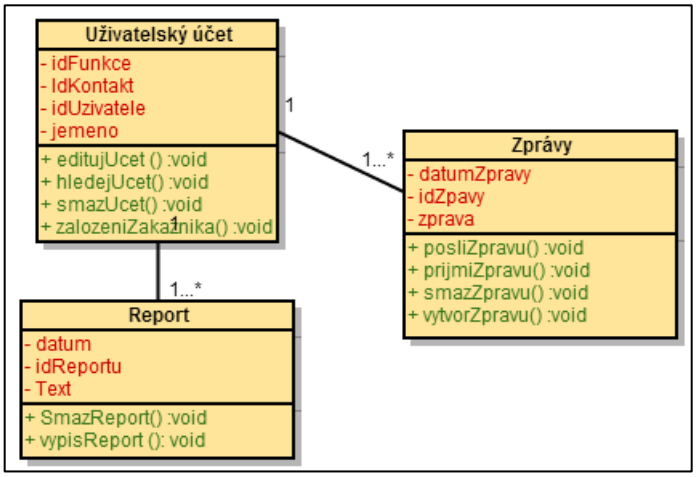
Obrázek 27 – Mockup registračního formuláře

4.1.3 Gliffy

Gliffy je velice užitečný nástroj, který nám umožňuje navrhovat různorodé diagramy a do určité míry muže nahradit modelovací jazyk UML, díky podporované sadě objektů UML ve verzi 1.0. Gliffy je však více obecný nástroj, který je možno užívat i tam, kde UML použít nelze, jako je například zachycení modelu infrastruktury firemní sítě.

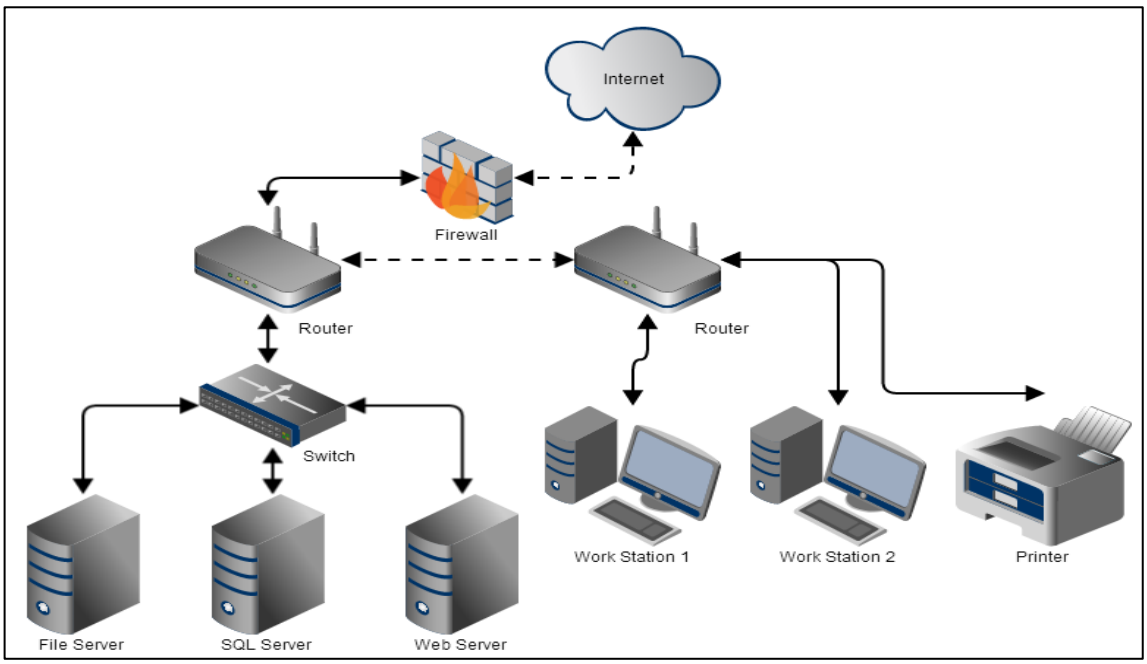
Příklad:

UML



Obrázek 28 - Gliffy UML

Infrastruktura



Obrázek 29 - Gliffy infrastruktura

4.2 Fáze řízení

Pro řízení projektu je dobré mít patřičný nástroj, který nám umožní kontrolovat a dohlížet na položky plánované pro jednotlivé sprinty. Právě takovým nástrojem je JIRA kterou si představíme.

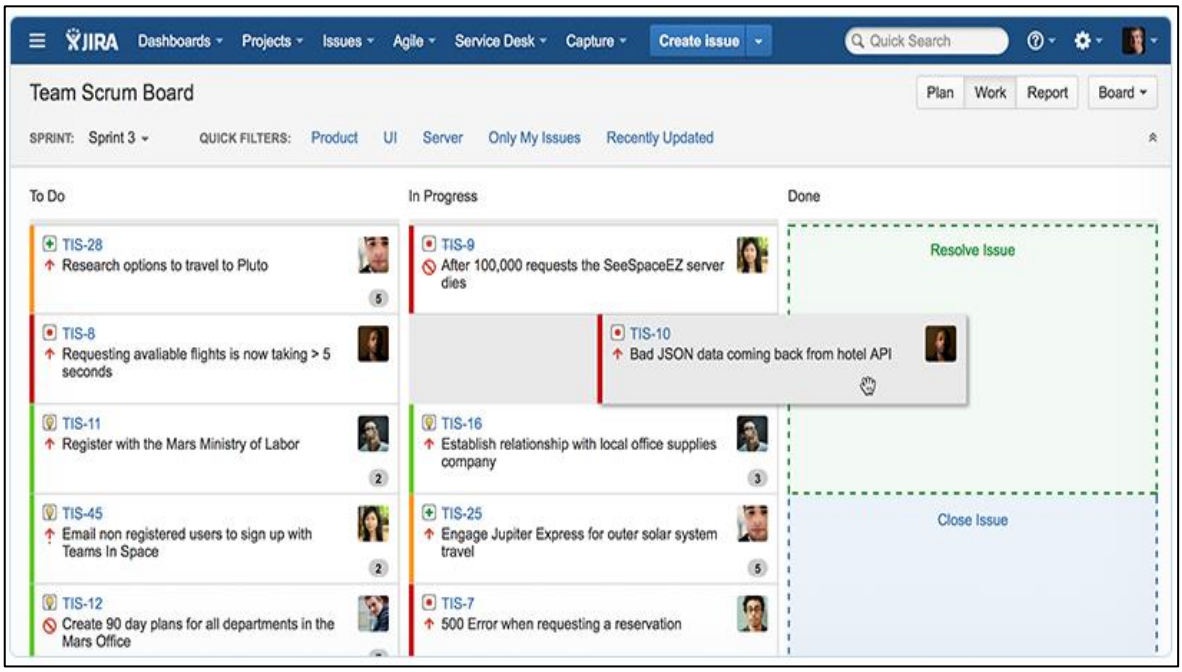
4.2.1 JIRA

JIRA je software zabývající se správou tiketů v průběhu sprintu. Aplikace je velice intuitivní a dodává do projektu patřičnou přehlednost, v jakém stavu se konkrétní tiket nachází. Aby byla dodržena přehlednost a jednoduchost, je dobré si stanovit tři až čtyři základní stavy, do kterých může tiket přecházet.

Příklad:

3 stavový systém (SCRUM)

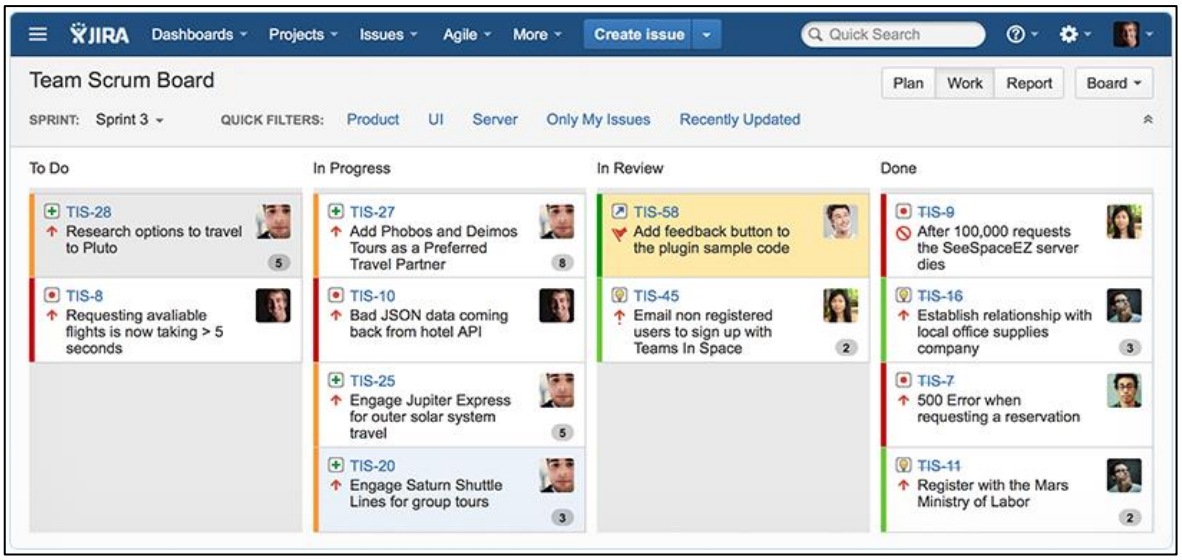
- Připraven pro vývoj – veškeré tikety ze sprint backlogu, které se budou v nadcházejících týdnech implementovat
- Vývoj – tikety, kterými se jednotlivý vývojář/tester aktuálně zabývá
- Hotov – veškeré tikety, u kterých bylo dosaženo stavu splňující podmínku „Hotovo“



Obrázek 30 - Jira 3 stavový systém [24]

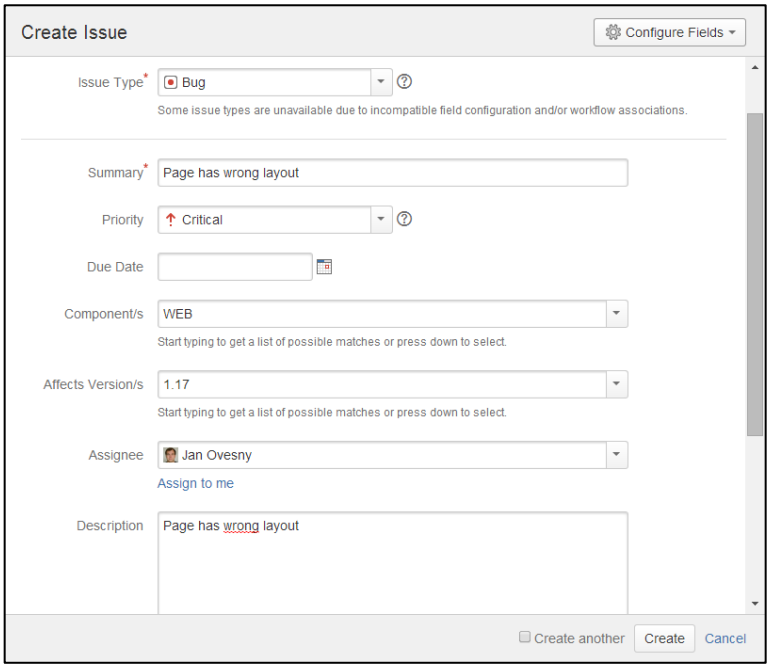
4 stavový systém (Kanban)

- Připraven pro vývoj – viz. 3 stavový systém
- Vývoj - tikety, kterými se jednotlivý vývojář aktuálně zabývá
- Testování - tikety, kterými se jednotlivý tester aktuálně zabývá
- Hotov – viz. 3 stavový systém



Obrázek 31 – JIRA 4 stavový systém [24]

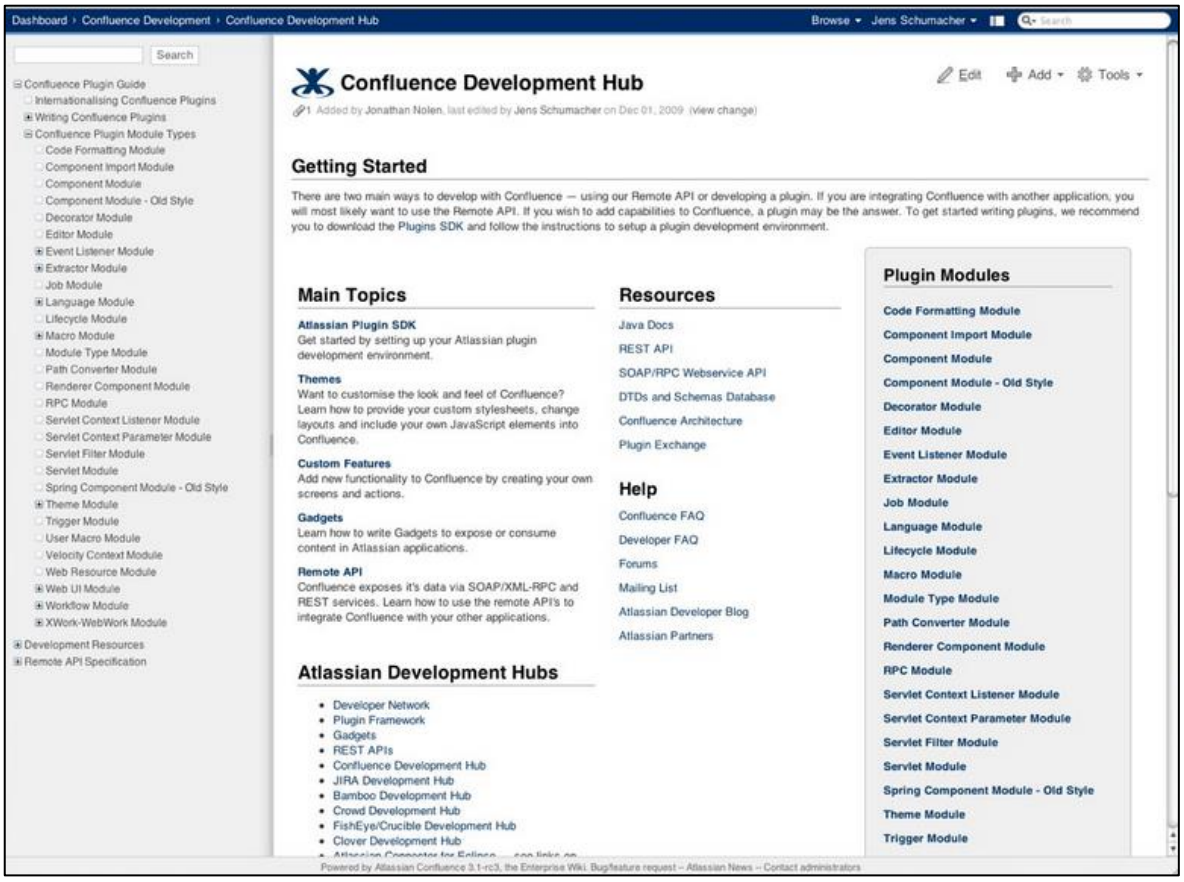
Aby bylo tikety možné spravovat, je potřeba nejprve tikety vytvořit, což je v prostředí JIRy velice jednoduché.



Obrázek 32 – JIRA tiket

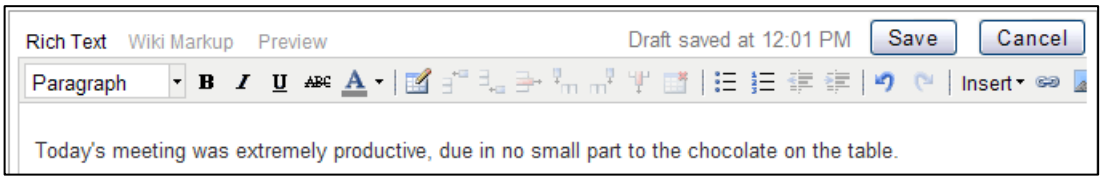
4.2.2 Confluence

Confluence je velice užitečný nástroj, díky kterému je možné spravovat a zveřejňovat různé informace pro celý tým. Díky stromové architektuře je opět dodržen vysoký standart přehlednosti.



Obrázek 33 – Confluence [25]

Vytváření nových dokumentů je díky vestavěnému textovému editoru více než snadné a díky podobnosti s jinými editory taktéž intuitivní. Velkou výhodou je pak tlačítko „zpět“, které jistě všichni známe z produktu Microsoft Word.

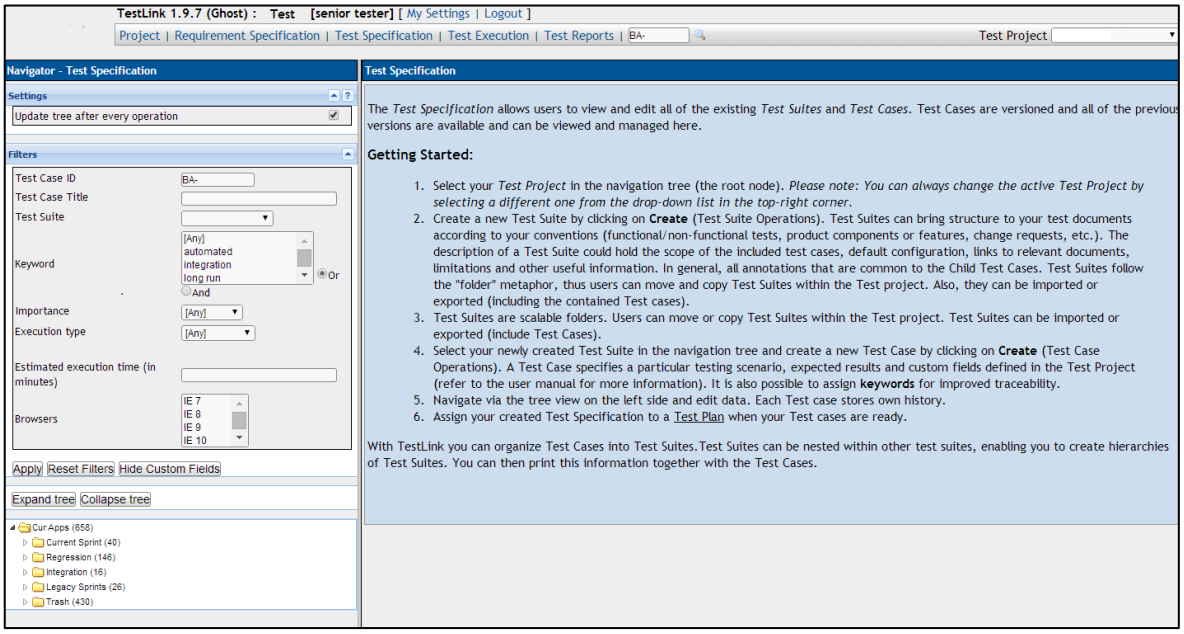


Obrázek 34 - Confuence editor [25]

4.3 Fáze finálního testování

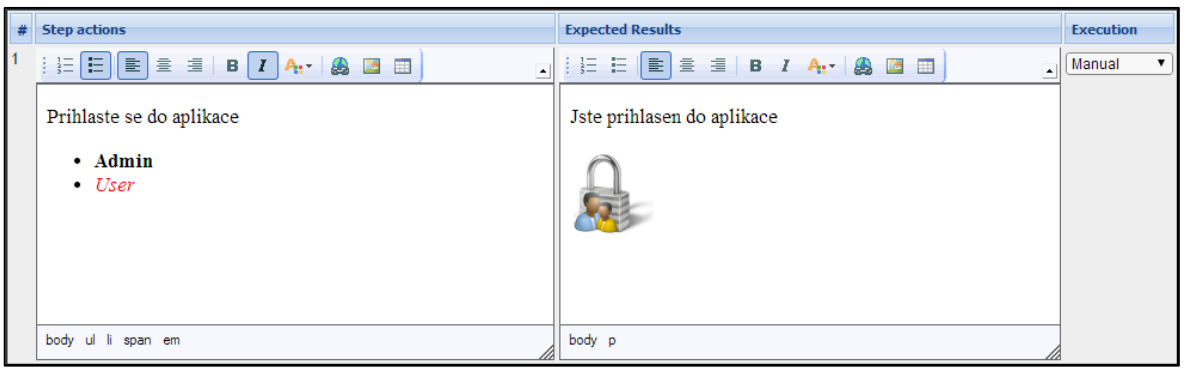
4.3.1 TestLink

Jak jsme si již dříve řekli, před samotným nasazením nové funkcionality do produkce, je potřeba provést regresní testování již nasazených částí kódu. Pro správu celé regresní sady je pak vhodným nástrojem TestLink, který nám umožňuje spravovat Test casy, sledovat reporty jednotlivých testování, nebo dokonce určit odhadovanou časovou náročnost celé regresní sady.



Obrázek 35 – Testlink

Obdobně jako u Confluence tak i v Test linku je obsažen jednoduchý textový editor, díky kterému je možné upravovat jednotlivé Test casy efektivním způsobem.



Obrázek 36 - Test link: Test case editor

5 VYHODNOCENÍ

Nasazením této metodiky vidím přínosy hned v několika zásadních odvětvích, které jsou kritické pro vývoj webových aplikací.

5.1 Rychlost vývoje

Díky tomu, že vývojový tým se stará právě pouze o vývoj a není nijak brzděn případným papírováním a přílišnou dokumentaci je efektivnost celého týmu na vysoké úrovni. Taktéž díky délce sprintu trvající pouhé dva týdny je tým schopen dodávat nové funkcionality ve velice krátkém čase, což je u webových projektů klíčové.

5.2 Zpětná vazba

Jelikož je nasazení nových funkcionalit na konci každého sprintu spjato s UAT testováním Vlastníkem produktu, což je svým způsobem zpětná vazba, není možné, aby se do produkčního prostředí dostávala nežádoucí funkcionalita, kterou by bylo možné implementovat ať už to nepochopením tiketu nesoucí informaci o dané funkcionalitě, nebo jen špatnou implementací developera.

5.3 Nulová chybovost implementace

Každá funkcionalita, jakmile je naprogramována, podléhá celé řadě procesů, kterými se eliminují případné chyby kódu. Díky procesům jako jsou revize kódu, jednotkových testů a následného testování testerů na po sobě jdoucích vývojových prostředích je výskyt chyb při konečném nasazení nulový.

5.4 Komunikace

Díky denním meetingům, je dosaženo potřebného stupně komunikace, bez které by nasazovat jakoukoliv funkcionalitu bylo takřka nemožné. Taktéž přítomnost Vlastníka produktu na meetinzích a objasňování případných dotazů ke konkrétním funkcionalitám je, hlavně ve fázi vývoje, velkým pozitivem.

5.5 Přehlednost

Tím, že na celé řízení projektu byl využit nástroj JIRA v kombinaci s Confluence portálem nám byla zachována vysoká úroveň přehlednosti celého vývoje, z čehož také vyplývá

odpověď na otázku, co vše již bylo implementováno/otestováno a co nám ještě zbývá implementovat/otestovat do konce daného sprintu případně konce vývoje celého projektu.

ZÁVĚR

Cílem diplomové práce bylo seznámit se s klasickými a agilními vývojovými metodikami, zejména se však zaměřit právě na ty agilní a vybrat vhodného kandidáta, kterého jsem pak přizpůsobil požadavkům při vývoji webových aplikacích ve firmě.

Nejprve byla potřeba vybrat jednotlivé zástupce ze dvou klíčových, výše zmíněných skupin vývojových metodik a na základě jejich kladů a záporů společně s jednoduchou analýzou potřeb vývojového týmu, vybrat nejvhodnějšího zástupce.

Jako nejschůdnější kandidát byla vybrána agilní metodika SCRUM, která za pomocí vhodných nástrojů splňuje požadavky vyplívající z analýzy.

Dále byly představeny fáze vývoje webové aplikace od plánování, přes průběh vývoje a testování až po nasazení do ostrého provozu s podrobným popisem každého dílčího kroku a jednotlivých úkolech, co jaký člen vývojového týmu musí splnit, aby byla implementace úspěšně dokončena. Důraz byl kladen na zachycení vývojového (pracovního) prostředí a na jednotlivé úlohy (řazeny chronologicky), které musí každý developer a tester, jakožto vývojový týmu, splnit.

Taktéž byly představeny nástroje, které jsou nedílnou součástí vývoje, ať už webových, či desktopových aplikací. Samotné nástroje byly také logicky uspořádány podle toho, v jaké fázi vývoje jsou převážně používány a tedy opět plánování, vývoj (neboli řízení) a finální testování (těsně před nasazením).

Jak již bylo v teoretické části zmíněno, metodika SCRUM má již za sebou určitou historii, díky které se metodika vyvinula do velice užitečného nástroje, a díky své flexibilitě je ji možné nasadit v nejrůznějších firmách zabývajícími se nejrůznějšími problémy.

ZÁVĚR V ANGLIČTINĚ

The aim of this thesis was to get acquainted with classical and agile development methodologies, with focus on the agile ones and choose a suitable candidate, which I adapted to the requirements of the development of web applications in the companies.

First goal was to select individual representative of this two groups above and based on their pros and cons along with a simple analysis of the needs of the development team choose the most appropriate representative.

As the most suitable candidate was chose Agile SCRUM because by using appropriate tools it meets the requirements taken from the analysis.

Furthermore there were introduced web application development phases starting with planning, through the development and testing to deployment into production with a detailed description of each step and each sub-tasks, which the member of the development team must meet to successfully complete the implementation. Emphasis was placed to capture the development (working) environment and the specific task (in chronological order), which every developer and tester needs to finish.

There were also introduced the tools that are an important part of the development, whether web applications or desktop applications. The actual tools are logically arranged, depending on what stage of development are largely used: planning, development (or management) and final testing (just before deployment).

As already mentioned in the theoretical part, SCRUM methodology has already it’s own history so the methodology developed into a very useful tool, and thanks to its flexibility, it can be deployed in a variety of companies dealing with various problems.

SEZNAM POUŽITÉ LITERATURY

[1] PHILLIPS, Andrew. Software Development Methodologies: Introduction. *Codeproject.com* [online]. 2010 [cit. 2014-04-29]. Dostupné z: http://www.codeproject.com/Articles/124732/Software-Development-Methodologies

[2] KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. 1. vyd. Brno: Computer Press, 2004, 278 s. ISBN 80-251-0342-0.

[3] SDLC Waterfall Model: Waterfall Model design. TUTORIALSPOINT. *Tutorialspoint.com* [online]. 2014 [cit. 2014-04-29]. Dostupné z: http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

[4] SDLC Spiral Model: Spiral Model design. TUTORIALSPOINT. *Tutorialspoint.com* [online]. 2014 [cit. 2014-04-29]. Dostupné z: http://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

[5] WELLS, Don. Extreme Programming: A gentle introduction. *Extremeprogramming.org* [online]. 2009, 8.10.2013 [cit. 2014-04-29]. Dostupné z: http://www.extremeprogramming.org/

[6] Webopedia: Extreme Programming. *Webopedia.com* [online]. 2014 [cit. 2014-04-29]. Dostupné z:http://www.webopedia.com/TERM/E/Extreme_Programming.html

[7] KRISHNAMURTHY, Venkatesh. A Brief History of Scrum: A Confusing Origin Story. *Techwell.com* [online]. 2012 [cit. 2014-04-29]. Dostupné z: http://www.techwell.com/2012/10/brief-history-scrum

[8] SCHWABER, Ken a SUTHERLAND. Průvodcem Scrumem. *Scrum.org* [online]. 2013 [cit. 2014-04-29]. Dostupné z: https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-CS.pdf#zoom=100

[9] KOMENTARZY, Brak. KANBAN. *En.system-kanban.pl* [online]. 2013 [cit. 2014-04-29]. Dostupné z: http://en.system-kanban.pl/kanban/

[10] PETERSON, David. What is Kanban?. *Kanbanblog.com* [online]. 2014 [cit. 2014-04-29]. Dostupné z:http://www.kanbanblog.com/explained/

[11] HOŘEJŠEK, Michal. Jak psát hezký kód I. *Zdrojak.cz* [online]. 2010 [cit. 2014-04-29]. Dostupné z:http://www.zdrojak.cz/clanky/jak-psat-hezky-kod-i/

[12] REJNKOVÁ, Petra. Příklady použití diagramů UML 2.0: Diagram případů užití. *Uml.czweb.org* [online]. 2009 [cit. 2014-04-29]. Dostupné z: http://uml.czweb.org/pripad_uziti.htm

[13] REJNKOVÁ, Petra. Příklady použití diagramů UML 2.0: Sekvenční diagram. *Uml.czweb.org* [online]. 2009 [cit. 2014-04-29]. Dostupné z: http://uml.czweb.org/sekvencni_diagram.htm

[14] REJNKOVÁ, Petra. Příklady použití diagramů UML 2.0: Diagram aktivit. *Uml.czweb.org* [online]. 2009 [cit. 2014-04-29]. Dostupné z:http://uml.czweb.org/diagram_aktivit.htm

[15] Other Life Cycle Models. TECHNOLOGYUK. *Technologyuk.net* [online]. 2014 [cit. 2014-04-29]. Dostupné z:http://www.technologyuk.net/computing/sad/other_models.shtml [16] http://projects.staffs.ac.uk/suniwe/wholesite.html

[17] REFSLUND, Anders. Iterative Methodology. *Computerscience* [online]. 2013 [cit. 2014-04-29]. Dostupné z:http://refslund.me/cs/systemdevelopment/process/iterative

[18] Withfriendship.com. HIOX INDIA. *With friendship* [online]. 2014 [cit. 2014-04-29]. Dostupné z:http://withfriendship.com/user/cyborg/extreme-programming.php

[19] Scrum Alliance: Why Scrum?. SCRUM ALLIANCE. *Scrumalliance.org* [online]. 2014 [cit. 2014-04-29]. Dostupné z:http://www.scrumalliance.org/why-scrum

[20] CAYP's Next Open Planning Meeting. CAPITAL ALLIANCE OF YOUNG PROFESSIONALS INC. *Cayp.org* [online]. 2011 [cit. 2014-04-29]. Dostupné z: http://cayp.org/2011/09/cayps-next-open-planning-meeting/

[21] Posts Tagged Planning poker. RSA. *Rsastories.wordpress.com* [online]. 2012 [cit. 2014-04-29]. Dostupné z:http://rsastories.wordpress.com/tag/planning-poker/

[22] STATE ASSESSMENT INFORMATION: TEST SECURITY TRAINING TOOL. SOUTHWEST PLAINS REGIONAL SERVICE CENTER.*Swprsc.org* [online]. 2013 [cit. 2014-04-29]. Dostupné z:http://www.swprsc.org/vnews/display.v/SEC/Home%7CState%20Assessment%20Information%3E%3ETest%20Security%20Training%20Tool

[23] Software Quality Assurance: Acceptance Testing. CLEAR SYNERGY TECHNOLOGIES INC. *Clear-synergy.com* [online]. 2013 [cit. 2014-04-29]. Dostupné z: http://clear-synergy.com/SDLC/SQA.html

[24] JIRA Agile: Go agile with ease. ATLASSIAN. *Atlasian.com* [online]. 2013 [cit. 2014-04-29]. Dostupné z: https://www.atlassian.com/software/jira/agile

[25] Confluence 3.2 Beta Release Notes. ATLASSIAN. *Confluence.atlassian.com* [online]. 2009 [cit. 2014-04-29]. Dostupné z: https://confluence.atlassian.com/display/DOC/Confluence+3.2+Beta+Release+Notes

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

VM	Vodopádový model
SM	Spirálový model
P1	Prototyp 1
RUP	Rational Unified Process
IMB	International Business Machines
UML	Unified Modeling Language
USDP	Unified Software Development Process
UP	Unified Process
EP	Extrémní programování
JIT	Just in time
DEV	Development
QA	Quality ansurance
STG	Staging
PROD	Production
ID	Identifikátor
Tel	Telefon
Msg	Zpráva
E2E	End-To-End
UAT	User acceptance test
FB	Facebook
WIKI	Wikipedia
RSS	Rich Site Summary
$	Byznys přínos

SEZNAM OBRÁZKŮ

SEZNAM TABULEK