Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Agilní metodiky vývoje SW

Martin Dulovec

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Dulovec

Informatika

Název práce

Agilní metodiky vývoje software

Název anglicky

Agile Methods o Software Development

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku metod vývoje softwaru.
Hlavním cílem práce je zhodnocení agilních metod vývoje softwaru.
Dílčí cíle práce jsou:
- charakterizovat přístupy k vývoji SW,
- zhodnotit nástroje pro kooperativní vývoj SW,
- formulovat návrh využití agilních metod pro malé vývojářské týmy.

Metodika

Analytická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů.

V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny nástroje a metody vhodné pro agilní vývoj a formulován návrhy využití agilních metod pro malé vývojářské týmy.

Doporučený rozsah práce

35

Klíčová slova

SCRUM, FDD, TDD, vývoj software, agilní metodiky

Doporučené zdroje informací

Astels, David: Test Driven Development: A Practical Guide, Prentice Hall 2003. ISBN-13: 978-0131016491

Highsmith, Jim: Agile Software Development Ecosystems, Addison-Wesley 2002. ISBN-13: 978-0201760439

KADLEC, V. *Agilní programování : metodiky efektivního vývoje softwaru.* Brno: Computer Press, 2004. ISBN 80-251-0342-0.

PALMER, S R. – FELSING, J M. *A practical guide to feature-driven development.* Upper Saddle River, NJ: Prentice Hall PTR, 2002. ISBN 0130676152.

SCHWABER, K. – BEEDLE, M. *Agile Software Development with Scrum.* Upper Saddle River: Prentice Hall, 2002. ISBN 0-13-067634-9.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 19. 9. 2018

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Agilní metodiky vývoje SW" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 13.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Michalu Stočesovi, Ph.D. za rady, připomínky a metodické vedení této práce.

Agilní metodiky vývoje SW

Abstrakt

Agilní metodiky vývoje software jsou rychle se rozvíjejícím odvětvím softwarového inženýrství, kterému odborníci předpovídají velkou budoucnost. Agilní pojetí vývoje se snaží o rychlé dodání kvalitního produktu. Práce uvádí do problematiky agilního vývoje software a zaměřuje se na metodiky SCRUM, Lean Development a Extremní Programovaní.

V Teoretické části je uveden popis tradičních metodik pro vývoj SW, kterými jsou Vodopádový model, Spirálový model a model Rational Unified Process (RUP). Další část tvoří rozpracováný popis tradičních Agilních metodik (SCRUM, Lean Development, Crystal metodika, Extremní Programovaní, FDD) a jejich principy.

V Praktické části je provedeno zhodnocení nejčastěji používaných metodik Agilního vývoje. Práce také obsahuje zhodnocení nástrojů k Agilnímu vývoji SW, které se využívají na menších ale i na velkých projektech.

Klíčová slova: SCRUM, FDD, TDD, Lean Development, Crystal metodika, Extremní Programovaní, vývoj software, agilní metodiky

Agile SW development methodology

Abstract

Agile software development methodologies are a rapidly evolving software engineering that experts predict a great future. The agile concept of development seeks to deliver a high-quality product quickly. The thesis deals with agile software development and focuses on the SCRUM, Lean Development and Extreme Programming methods.

The theoretical part describes the traditional methodologies for SW development, such as the Waterfall Model, the Spiral Model and Extreme Programming (RUP). The next part is an elaborate description of traditional Agile Methods (SCRUM, Lean Development, Crystal Methodology, Extreme Programming, FDD) and their principles.

In the Practical part the evaluation of the most frequently used methods of Agile Development is performed. The exam also includes an evaluation of tools for agile development of SW, which are used for smaller but also for large projects.

Keywords: SCRUM, FDD, TDD, Lean Development, Crystal Methodology, Extreme Programming, Software Development, Agile Methodology

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Oblast vývoje software jde stále kupředu mílovými kroky. Už když spatřily světlo světa první počítače, bylo nutné vyvíjet software, který by z jednoduchého stroje udělal univerzálního pomocníka. Software je to, co dnes můžeme najít v počítači. Přitom cena je mnohdy tím, co určuje kvalitu softwaru celého počítačového systému. Když chceme kvalitní software, je potřeba kvalitního vývojového procesu. Z toho vyplývá, že kvalitě vývoje software je věnována vysoká pozornost a jsou do ní investovány vysoké peněžní prostředky.

V posledních letech si odborníci v této oblasti přestali stěžovat na nevýhody aktuálních metod vývoje software a začali podnikat kroky na zlepšení, zrychlení a zkvalitnění vývojového procesu. Začátkem to bylo na úrovni firem, ve kterých daní lidé pracovali. Po nějakém čase se jejich nové metody osvědčily v praxi a byly představeny na veřejnost. Postupem času se začaly objevovat nové přístupy a metodologie, které měly překvapivě hodně společného. Rychlý vývoj v různě dlouhých cyklech, dodávky nových betaverzí, úzký kontakt s klienty a uživateli, zpětné vazby, zvýšení efektivity a výsledné produktivity. Všechno to jsou principy, které se velkou intenzitou objevují ve všech těchto nových přístupech. Přístupy nazvány lehké (lightweight), později pak agilní (agile).

Práce je zaměřena na principy a funkce daných metodik, pochopit jejich účely a strategie a poskytnout alespoň stručný náhled na jejich použitelnost. Jelikož agilních metodik je mnoho a rozsah práce mi nedovolí věnovat se do podrobna každé z nich, vybral jsem tři konkrétní oblasti, které podrobněji popíšu, ostatní jen stručně představím. Danými částmi jsou, SCRUM Development Process, Extrémní programování a Lean Development. Na závěr se motodiky pokusím porovnat.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je tematicky zaměřena na problematiku metod vývoje softwaru. Hlavním cílem práce je zhodnocení agilních metod vývoje softwaru. Dílčí cíle práce jsou:

- Charakterizovat přístupy k vývoji SW.
- Zhodnotit nástroje pro kooperativní vývoj SW.
- Formulovat návrh využití agilních metod pro malé vývojářské týmy.

2.2 Metodika

Analytická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů. V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny nástroje a metody vhodné pro agilní vývoj a formulovány návrhy využití agilních metod pro malé vývojářské týmy.

3 Teoretická východiska

3.1 Tradiční metodiky

V první kapitole bude představena problematika, které vedla ke vzniku agilních metodik, si nejdříve řekneme něco o tradičních metodikách životního cyklu software. Poskytne nám to lepší pohled do problematiky a pochopení historických souvislostí. Problémem klasických metodik byla jejich příliš velká složitost a malá flexibilita. Tradiční metodiky pomalu přestaly být vhodné malým projektům. Často požadovaly přesnou dokumentaci a nejrůznější revize a schvalování. Ani postupné zlepšování těchto metodik nebylo možno vyřešit zmíněné problémy. Bylo nutné udělat ve vývoji software radikální změnu. Ze standardních metodik jsem vybral tři zástupce, které byli milníkami v historii softwarového inženýrství. Představíme vodopádový model, spirálový model a Rational Unified Process. [11]

3.1.1 Vodopádový model

Vodopádový model je považován za ucelenou metodiku vývoje software. Charakterizuje se znaky, které jsou sekvenčně seřazené fáze. Schvalovací proces, který je mezi jednotlivými fázemi, musí projít všechny dokumenty, aby mohl vývoj pokročit do další fáze. Základy vodopádového modelu jsou tyto fáze[7][11]:

1. Definování problému, poznávání zákazníka a cílové oblasti
2. Specifikace a Analýza požadavků
3. Navržení
4. Implementace
5. Testy a integrace
6. Provoz s údržbou

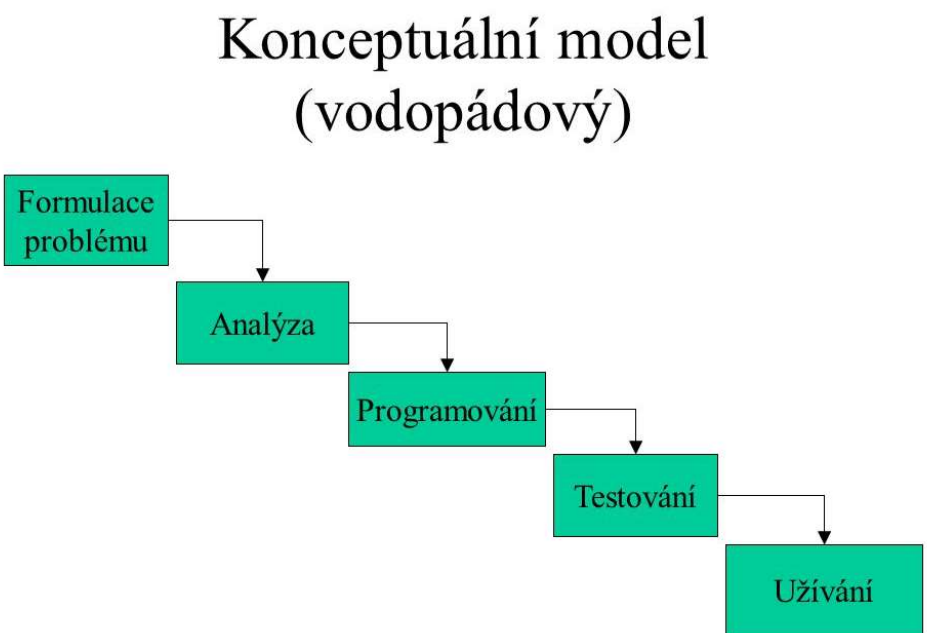
Byl predstaven vodopádový model, který je striktně sekvenční. V modelu se můžeme pohybovat bud o fázi dopředu nebo zpět. Pokud zjistíme ve fázi implementace, že jsme v návrhu udělali chybu, můžeme se vrátit o krok nebo-li fázi zpět, provést redesign a vrátit se do fáze implementace. Při přechodu k implementaci je důležité, aby všechny dokumenty byly schváleny. Teprve v kroku údržby se může vývoj vrátit do jakéhokoliv z předchozích kroků a umožnit tak jakýchkoliv úprav[7].

To je ale z aktuálního pohledu nevhodné, jelikož z praxe víme, že změny v daných požadavcích nastávají dříve než v kroku údržby, navíc někdy není list požadavků dostupný

ve formě dokončené ani v době, kdy probíhá krok specifikace daných požadavků. Při vývoji modelu je komunikace se zákazníkem potřeba pouze v úvodu projektu při specifikování požadavků, pak až v závěru při kroku údržby a předání[7][11].

Vodopádový model byl prvním, který definoval posloupnost vývojových kroků, je jeho výhodou jednoduchost. Model je velice jednoduchý na pochopení, řízení a práci podle něj. Každý krok je zakončen zpracováním daných dokumentů, přechody mezi danými kroky zajišťuje schvalovací řízení. Kontrolovat lze v jaké části se projekt aktuálně nachází a jaká část požadované specifikace je již splněna. Pro malé projekty je výhodou, že je model jednoduchý, ale pro některé větší už to tak dobré není, protože brání pokrýt všechny aspekty rozsáhlých projektů. Vývoj dle vodopádového modelu není až tak moc flexibilní, veškerý vývoj probíhá přesně podle naplánované posloupnosti kroků. Jestliže v průběhu implementace narazíme na nový požadavek, je potřeba znovu projít krokami analýzy, specifikace požadavků a návrhu, včetně schválení všech dokumentů. To daný vývoj zdržuje, pokud se dodatečné objevují požadavky v pozdních krocích projektu[11].

Nevýhodou je, že v průběhu vývoje zákazník celou dobu neví, co vlastně vývojářský tým dělá a kdy bude projekt dokončen. Často se pro tento daný způsob dodávání používá pojem „velký třesk". Přitom případy, kdy je zákazník spokojen s výsledkem po dodání v dohodnutém termínu je zcela velice málo. Nejhorší situace nastane,



Obrázek 1: Vodopádový životní cyklus[22]

když analytik vytvoří specifikaci ze spatně nasbíraných podkladů, nepochopí zákazníka a ten pak po několika měsíčním vývoji a po velkém množství finančních prostředků prijde

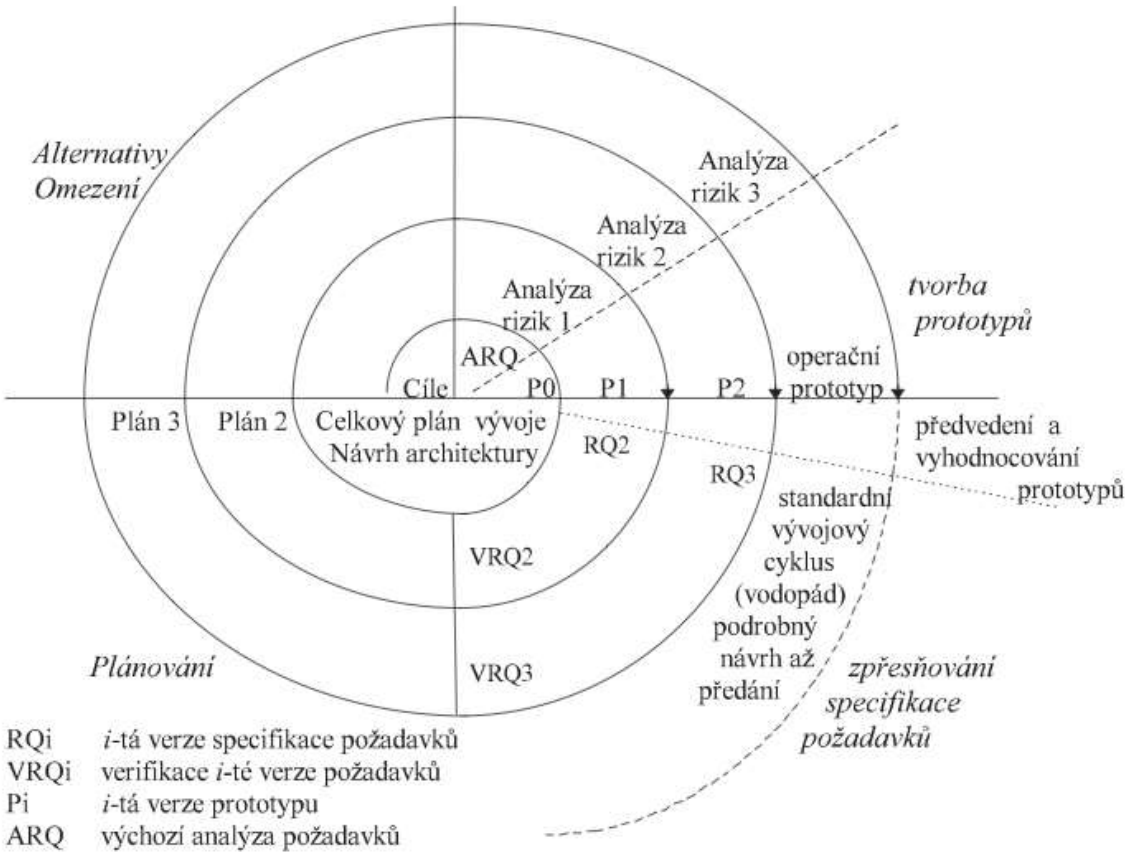
na to, že nedostal to, co vlastně očekával. Ve své dané době byl vodopádový model revoluční a vznikla podle něj spousta nových softwarových produktů. Dnes se častěji používá již jako referenční model pro srovnávání s ostatními modely[7][11].

3.1.2 Spirálový model

Vodopádový model zmíněný v předchozí kapitole krátce po svém vzniku přestal být dostačující. Proto se softwaroví inženýři pokoušeli vymyslet nějaký nový přístup, kterým by se odstranily nedostatky vodopádového modelu. V roce 1985 k tomu došlo, kdy Barry Boehm přišel se svým spirálovým modelem. Jak v případě vodopádu, tak i v případě spirály mluvíme stále o modelu, jelikož to nejsou metodiky vývoje software. Jsou to modely životního cyklu softwarového produktu. Rozdílem je to, že model životního cyklu popisuje jasné kroky, kterými softwarový produkt prochází. Metodika určuje, v jakém kroku se přesně má dělat, k jakému výsledku se má nakonec dojít a jaké postupy se mají použít. Spirálový model vychází právě z modelu vodopádového, který přináší dvě nové a zásadní vlastnosti - podrobnou analýzu rizik a iterativní přístup. Proto se také někdy o něm mluví, že patří do skupiny přístupů, které jsou řízeny riziky (risk-driven approach). Důsledkem toho byl iterativní přístup u velkého množství projektů u kterých se nepodařilo stanovit úplnou a přesnou specifikaci požadavků, to způsobilo, doslova nepoužitelnost vodopádového modelu. Ukázalo se, že lepším způsobem jak stanovit na počátku daný rámec architektury a funkčnosti celého daného systému a postupně ho rozpracovávat do menších detailů. Opakováním kroků vývoje, nebo-li iterace, znamenalo přelom v chápání životního cyklu. Druhým důležitým pojmem v souvislosti s iterativním postupem je analyza rizik. Ta se provádí v každém cyklu a určuje směr vývoje projektu. Riziko, které je ústředním pojmem, je chápáno jako situace, která může ohrozit projekt. Při každé analýze je dobré zvážit její nebezpečnost a pravděpodobnost výskytu. Podrobná a častá analýza rizik má většinou za úkol dostatečně dopředu odhalit dané nevhodné řešení nebo skryté problémy, které mohou ohrozit průběh projektu[12].

Hlavní výhodou spirály je nezávislost na konkrétní dané metodice. Která je při samotném vývoji použita. Jak jsem již uvedl, model je komplexní a sympatizuje s většími projekty, ale u menších projektů je to bohužel opačně. Nehodí se pro malé projekty, protože pro ně je příliš komplikovaný. Nevýhodou je, že model zdědil od modelu vodopádového to, že předání probíhá až po dokončení posledního cyklu koncového produktu. V každém cyklu se sice vytvoří prototyp, který se může obecně týkat různých

malých částí daného systému a ani nemusí být použitelný v provozu. Tím pádem je tady zavedena zpětná vazba, který problém „velkého třesku" nevyřeší. Musíme nutně zmínit, že model spoléhá na metodiku, která se používá při vývoji a nezabývá se rozpracováním jednotlivé činnosti, které se provádí v průběhu vývoje. V softwarovém inženýrství je spirálový model důležitým milníkem. V minulosti byl používán k řadě projektů. V dnešní době se už ale využívá méně. Důvod je ten, že jsou na světě propracovanější metodiky a pro současné typy aplikací je model Spirála strnulý a neobratný. [12]



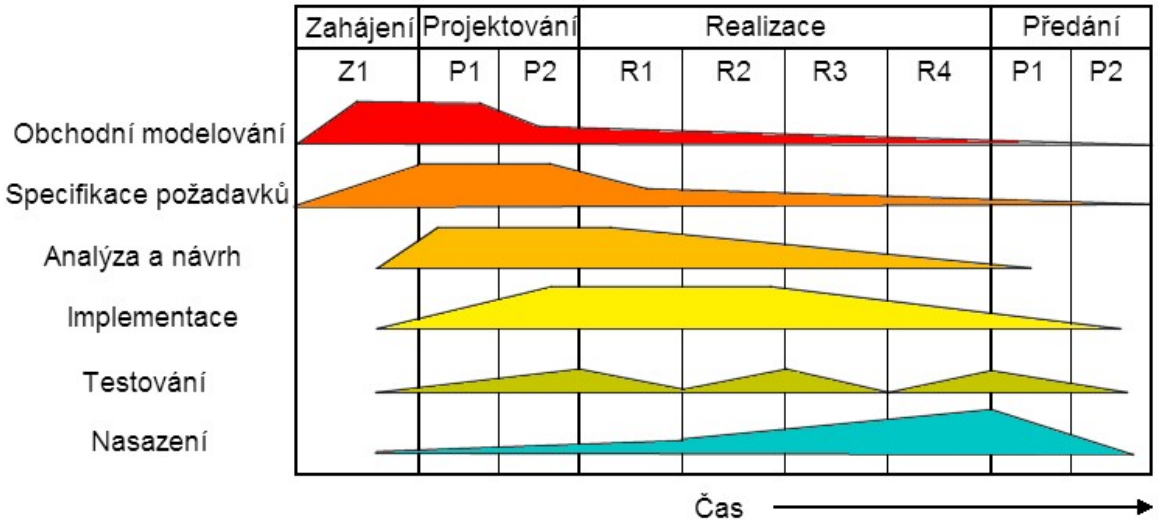
Obrázek 2: Spirálový životní cyklus[23]

3.1.3 Rational Unified Process

Rational Unified Process (RUP) je objektově orientovaný iterativní přístup k životnímu cyklu software. Metodiku byla původně vytvořena společností Rational Software Corporation RUP patří do skupiny tzv. Přístupů, které jsou řízeny případy použití, to znamená, že jako základní element je chápán případ použití definovaný jako posloupnost akcí prováděných daným systémem nebo uvnitř systému, která poskytuje danou hodnotu uživateli systému. Pro modelování větší části procesů se využívá prostředků jazyka jménem UML
stále pracuje, tak též můžeme najít spousty dostupných dokumentů, které podporují nástroje třetích stran, které se zabývají RUP [2].

Rational Unified Process obsahuje čtyři základní fáze. Každá z daných fází obsahuje několik dalších iterací. Před začátkem nové iterace musí být splněna dříve nadefinovaná kritéria předchozí iterace. Fáze inception (zahájení) definuje účel, rozsah daného projektu a jeho obchodní kontext. Ve fázi elaboration (projektování) je potřeba zanalyzovat požadavky zákazníka, celého daného projektu a definovat základy architektury. Fáze construction (realizace) je nejdelší a probíhá zde tvorba zdrojových kódů. V poslední fázi transition (předání) může být projekt předán danému zákazníkovi nebo do dalšího cyklu [16][2].

1. Správa požadavků
2. Iterativní vývoj software
3. Ověřování kvality software
4. Architektura založená na komponentách
5. Vizuální modelování
6. Řízení změn software



Obrázek 3: Vývojový cyklus v metodice RUP[24]

3.2 Agilní metodiky

V této části práce se zaměříme na problematiku agilních metodik. Nejdříve se podíváme, co je agilní programování a vysvětlím Manifest agilního programování, stručně popíšu nejznámější agilní metodiky a na závěr tři z metodiky podrobněji popíši.

3.2.1 Manifest agilního vývoje softwaru

Cílem manifestu je zlepšit metodu vývoje SW. Základními principy agilního přístupu jsou[18][4][10]:

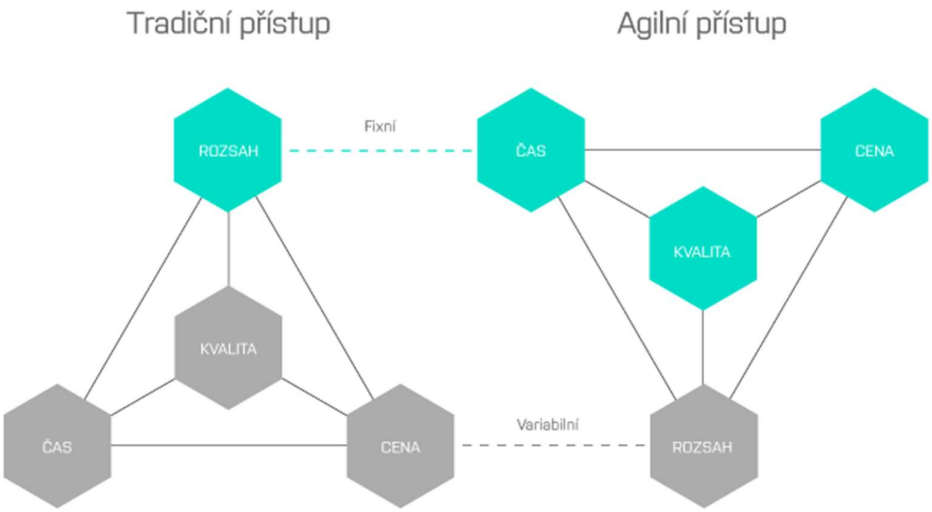
1. Lidé a jejich spolupráce mají vždy přednost před procesy a nástroji
2. Funkční software je vždy důležitější než obsáhlá vyčerpávající dokumentace
3. Spolupráce s klientem má přednost před domlouváním smluv
4. Reagovat na změny je vždy důležitější než dodržování zadaného plánu

3.2.2 Základní principy manifestu

1. Nejvyšší hodnotu je vždy uspokojit zákazníka rychlým a trvalým dodáním kvalitního software
2. Změnové požadavky (Change Requesty) jsou vítány a to nejen ke konci vývoje. Agilní procesy jsou zpracovány tak, aby zákazníkovi přinášely výhody
3. Dodávat spávně funkční software často, v daných intervalech týdnů až měsíců se snahou o kratší intervaly mezi danými dodávkami
4. Businessu a developeři musí na projektu denně komunikovat a spolupracovat
5. Základem projektu je motivace lidí. Musí mít korektní pracovní podmínky, důvěru a podporu, že svou práci odvedou
6. Nejlepší cesta k efektivnímu sdílení informací během vývoje je osobní komunikace
7. Hlavním kritériem postupu je funkční software
8. Agilní procesy propagují udržitelný vývoj. Sponzoři, vývojáři i uživatel musí být schopni dodržovat stálý výkon neomezeně dokud je třeba bez stanoveného konce
9. Trvalá kontrola co nejvyšší úrovně technického řešení a správný návrh aplikace posiluje agilní přístup
10. Jednoduchost je umění co nejvíce práce vůbec nedělat - je nezbytná
11. Návrh, nejlepší architektura
    a požadavky pocházejí ze samoorganizovaného týmu
12. Tým pravidelně vyhodnocuje práci a upravuje postupy tak, aby byl co nejefektivnější. [18][10]

3.2.3 Principy agilních metodik

Na obrázku si řekneme něco o základních rozdílech mezi agilními a tradičními programovacími přístupy.



Obrázek 4: Rozdíly tradičních a agilních přístupů[25]

Na začátku každého vývojového procesu stanovíme požadavky, které se nebudou měnit. Měnit můžeme jen čas a zdroj. Co program bude umět víme, ale nevíme jak dlouho a za kolik se bude program vyvíjet[14].

Zdroje a čas jsou vždy zadány zadavatelem u agilního přístupu jako neměnné. Během vývoje se jedná a přehodnocují se priority se zákazníkem, to znamená, že funkcionalita je proměnná. Proto se čas vývoje stanový na nejdelší možný a stanový se i vyšší náklady[14].

Principy agilních metodik [14]:

1. Vývoj v krátkých iteracích – zákazník je průběžně obeznámen s aktuálním stavem vývoje
2. Důraz je kladen na komunikaci v týmu - Zařídit fungující komunikaci v týmu je důležitá část integrace.
3. Komunikace se zadavatelem - Zadavatel je většinou členem vývojového týmu, neustále komunikuje s týmem a podílí se na testech a návrhu.
4. Automatizované testování - Při častých změnách systému by testy měli být automatizované v důsledku správnosti.

Díky agilnímu manifestu často docházelo ke zvyšování v počtu agilních metodik. Nejvýznamnější metodiky: SCRUM Development Process, Extrémní programování, Lean Development, Feature Driven Development, Metodika Crystal a Testy řízený vývoj[14].

3.2.4 Extrémní programování

Extrémní programování (XP) přineslo do vývoje software mnoho nového. Jsou dvě věci zvláště důležité pro kontrolu kódu[6].

První věc je programování ve dvojicích. To přináší možnosti pro trvalou křížovou kontrolu kódu a zároveň Test Driven Development (TDD) zabraňuje index, off-by-one, compile-time pro jejichž odhalení byla primárně určena revize kódu[6][2].

Druhá věc XP je zavádění konceptu nákladů podle toho, kdy je chyba nalezena. Jinak řečeno, čím později ve vývojovém cyklu je nalezená chyba, tím vyšší je cena na její nápravu (exponenciálně roste). Je jednoduché si to představit – chyba odhalená na produkčním prostředí stojí 100 ×–1 000 × víc než chyba odhalená a opravená ve fázi specifikací [6][2].

V dnešní době se vývojářský tým často řídící ideou agilního vývoje posune od daných požadavků k psaní kódu a poté k testování v rozmezí dnů nebo hodin. Když si představíme dva projekty, které mají stejnou specifikaci, jeden realizovaný agilně ve čtrnáctidenních iteracích a druhý klasický šestiměsíční („vodopád“), může ten agilní najít chybu v jednom či dvou týdnech, vůbec než dojde k nějaké revizi v případě druhého projektu. A to samozřejmě znamená nižší náklady[15][2][6].

Velké rozšíření internetu přineslo snížení nákladů na opravu chyb. Dnes je běžné, že se denně instalují automatické softwarové balíčky.

Dohromady to znamená, že revize kódu se snížily a zároveň klesla i rizika, která měla revize kódu pokrývat. To znamená, že mohou být zajištěny nástroji, jako jsou rapidní testování, TDD nebo párové programování [15][2].

1. **Plánovaní hry:** Dochází k vytváření určitých představ, co je zákazníkovi možné splnit. Dále se vytvori tým vývojářů, který nastíní hrubý plán, který pak dále konzultuje, zadání od zákazníka.

2. **Malé verze:** Nově vzniklé verze se často uvolňují v co nejmenší konfiguraci; nově představují celkový funkční mezikrok. Četnost uvolňování závisí na velikosti aplikace: U většiny velkých systémů jsou postupné verze dodávány pomaleji, než tomu je u menších aplikací.
3. **Metafora:** Všechen vývoj je veden pod taktovkou jakéhosi příběhu. Tento příběh nám „vypráví“ o tom, jak by měl systém nakonec fungovat. Metafora však neilustruje jenom funkce, cíle a požadavky, ale postupně začíná nahrazovat co bývá nazýváno „architektura“.
4. **Jednoduchý návrh:** Systém bývá navrhováný(a vyvíjený) tak „štíhlý“ a velmi jednoduchý, jak můžeme. Ptáme se „na co nejmenší konfiguraci, která nám může fungovat?“ Případné komplikace jsou průběžně odstraňovány, jak se objevují. Pokud je to potřeba, můžeme zakomponovat další nové prvky.
5. **Testování:** Testování je jedním ze základních pilířů XP. Neustále jsou psány testy jednotek, test modulu musí existovat dříve, než modul samotný. Vývoj nepokračuje, dokud test neproběhne bez chyby. Zákazníci naopak dodávají jakési testy funkcionality – tedy testy, které ověřují dokončení funkcí (nebo přesněji funkčnosti). Žádná funkce bez zautomatizovaného testu prakticky neexistuje.
6. **Refaktorizace:** Refaktorizace je změna zdrojového kódu, která nemá vliv na změnu funkčnosti aplikace. Změny se provádí jak je potřeba: kvůli duplicitám, pro jednoduchost architektury a systému, pro zvýšení efektivity. Důležitá věc je, že nesmí dojít ke změně chování aplikace a ke změně struktury systému. Opravy nemůžeme nazvat refaktorizací, znamená to pouze nefunkčnost aplikace.
7. **Párové programování:** Zdrojový kód vždy píše dvojice programátorů; sdílí se jeden monitor ve dvojici, počítač a klávesnice. Každa dvojice má vnitřní strukturu: jeden z programátorů, v daném okamžiku pracuje s klávesnicí, vyvíjí nějakou metodu X a přemýšlí o nejlepším způsobu jak metodu X implementovat. Druhý programátor pak uvažuje více nad celkovou a globální strategii: bude daná architektura fungovat spravně? Nebude moc složitá nebo neefektivní? Máme všemy automatizovanými testy podchyceny situace, které mohou nastat? Níže se nad tímto programováním zamyslíme vice.

8. **Společné vlastnictví:** Všichni mohou měnit jakékoliv místo ve zdrojovém kódu, a to kdykoliv. Tento bod znamená zvýšení potřebného koeficientu, tzv. truck faktoru, jehož hodnota 1 znamená, že každé části zdrojového textu rozumí právě jeden expert. V XP všichni přijímají odpovědnost za celý systém. Realizace tohoto bodu ovšem znamená, že tým musí mít propracované techniky konfiguračního řízení, především pak sdílení kódu a verzování, aby nedocházelo k nekonzistenci změn.
9. **Nepřetržitá integrace:** Systém je sestavován několikrát denně, když však je dokončen nový úkol. XP doporučuje mít jeden počítač vyhrazený pouze pro danou integraci.
10. **Čtyřicetihodinový pracovní týden:** Dlouhodobé přesčasy ústí v neefekti vitu a přetěžování lidských zdrojů. Je nepřípustné pracovat dva týdny přesčasů za sebou. Tohoto bodu se drží všichni vývojáři, kteří si stěžují na stres, nedostatek času a přetížení.
11. **Zákazník na pracovišti:** Součástí týmu se stává i skutečný živý uživatel behem doby vývoje, který je k dispozici ideálně na plný úvazek, aby spolupracoval na testech, odpovídal na otázky, určoval priority a řešil spory. Tento bod často zmiňují odpůrci Extrémního programovaní: zákazníci nemají rádí to, že musí vývojářský tým „vodit za ruku“. V tom případě by se zákazníkovi vyplatilo uvolnit nějakého člověka pro průběžn ou konzultaci s vývojářským týmem, neboť by produkt vypadal podle představ příslušného pracovníka. Nestávalo by se, že po dokončení vývoje by zákazník prohlásí „tohle jsem přeci nechtěl“ Ačkoliv nelze reálně očekávat, že by zákazník nějakého „full-time“ pracovníka opravdu uvolnil. To nic nemení na veci, že je nutné se snažit konzultovat stav vývoje se zadavatelem.
12. **Standardy pro psaní zdrojového textu:** Připomeňme, že hlavním nosičem „dokumentace“ v Extrémním programování je zdrojový kód. To znamená, že musí být psán podle pevně stanovených pravidel, kde umožní komunikovat prostřednictvím zdrojového textu. Zdrojový text, který je psaný bez pravidel (nemá žádnou kulturu) nemůžeme brát jako dokumentaci celého projektu. [15][2][6]

Pro menší skupinky lidí a menší vývojové projekty, kde je zapotřebí rychlý vývoj, je tato metodika vhodná. Jelikož je žádoucí vyvíjet SW v co nejjednodušší a nejkratší formě, určitě najdeme výhodou, že lidé jsou rádi, když vidí blízký a konkrétní cíl, když daný SW funguje. Pro spokojenost programátorů, nesmíme opomenout délky pracovní doby (viz. bod 10.), která se nepřekračuje. Avšak nejjednodušší část může být zároveň nejsložitější[6].

V dnešní době se hledají jen složitosti řešení a tím pádem může být problém s vývojem softwaru nejlehčím možným způsobem. Při ostychu někdy může být obtížné požádat kolegy o radu a pokračovat dále. U XP je těžké si zvyknout spolupracovat s týmem lidí, nepokračovat jako individualista [15].

3.2.5 Lean Development

Autorem je Robert Charette [9], který vychází z principů, použité v 80. letech pro efektivnost výroby automobilů v Japonsku, kde se výroba nazývala Lean Manufacturing. Základní vlastností této metodiky je odstranění všeho co není důležité.

Definice metodiky dle Kadlece [2]: *„Lean Development vede k systematickému přístupu a k identifikování a eliminaci možných zdrojů, plýtvání v průběhu celého vývojového procesu, který se pokouší dodat zákazníkovi perfektní produkt a splnit tak jeho požadavky.“* Základ metodiky je dovést vývojový proces k efektivní, optimální a hlavně kvalitativní podoby. Lean Development je spíše více zaměřen na strategii. Základní cíl této metodiky je vyvoj software, za co nejméně času, vystačit si s třetinou rozpočtu, snížit tvorbu chyb o celou třetinu běžného množství. Není to však metodika popisující způsob řízení týmu a vývoje , nýbrž nabízí více pravidel a principů, kde při dodržování pravidel, zajistí ekonomický a efektivní výsledek.

Metodika se opírá o deset základních pravidel [2][8][9]:

Odstranění zbytečností - Odstranit všechno, co konečnému produktu nepřinese žádnou hodnotu.

Minimalizace zásob - Není vždy nutné vyvíjet a vytvářet artefakty (prototypy, modely, dokumenty), které nejsou aktuálně potřebné či se v budoucnosti nepřiblíží k zákazníkovi. Důležité však je udržet počet artefaktů na co nejmenším počtu, které jsou nutné k udržení hodnoty systému.

Maximalizace toku (zkrácení času vývoje) - Pokud potřebujeme maximali- zovat tok, rozdělíme proces do menších částí kde se provede iterativní vývojový proces. Tento proces také přichází s výhodou, kde zákazník může průběžně sledovat prototypy a doplňit požadavky i za chodu. [2][8][9]

Vývoj daný poptávkou - Princip je provést všechna dané rozhodnutí nejpozději, a to z důvodu, který zadavatelé nejsou často schopni definovat aktuální potřeby a už vůbec ne budoucí. *„Všechny klíčové rozhodnutí, která se učiní, by měla být provedena v nejzazším možném okamžiku, kdy už je nevyhnutelné rozhodnout se o dalším směřování projektu. Čím dříve se učiní rozhodnutí, tím se často zvyšuje pravděpodobnost, že jej budeme muset, ač nechceme, později měnit“* [2][9].

Rozhodovací pravomoc pracovníků - „Vývojáři musí vždy chápat, jak jejich provedená práce přispívá celkovému cíli, musí vědět, co mají vykonat a do kdy a musí mít možnost rozhodovat“ [8].

Uspokojovaní požadavků zákazníka - Neúspěch projektu často závisí na neúplnosti, či správnosti zadaných požadavků. Lepším způsobem, než je přimění uživatele k podpisu rozsáhlé specifikace projektu je vytvoření partnerství. Zvyšujeme tím pravděpodobnost kontaktu se zákazníkem, který dokáže lépe zahrnout zákazníkovy požadavky do vývoje.

Zavedení zpětné vazby - Pokud nemůžeme v úvodu projektu zadefinovat všechny požadavky, tak můžeme tyto požadavky postupně doplňovat do vývojového procesu zavedením zpětné . Během vývoje bude často docházet ke změnám a možným chybám. Z prevence se nejdříve napíší testy dané komponenty. [2][8][9]

Odstranění lokální optimalizac - Marnění času a energie na jednotlivé komponenty, je zbytečne pokud jejich optimalizace nepřináší celkové zvýšení efektivnosti systému.

Partnerství s dodavateli - Nejdůležitější je pro zákazníka výsledná hodnota. Pro urychlení práce a zlepšení kvality může být výhodné např. nakoupit hotové knihovny.

Zavedení kultury k neustálému zlepšování - Vytvoření lepších pracovních podmínek a lepší motivovaci pro zaměstnance k maximálnímu výkonu.

Z těchto popsaných pravidel vznikla metodika Lean Development, která definuje sedm principů. Najdeme i zastánce pouze 7 principů. Jedním je Marry Poppendick [9]. Lean Development neudává komplexnost vývoje SW. Ukazuje jen na to, čeho se můžeme vyvarovat a jakého postupu se při deseti krocích vývoje držet. Při splnění těchto kroků můžeme zaručit efektivní a rychlý vývojový proces, kdy velmi často dochází k uspokojení zákazníkových zadaných požadavků[2][8][9].

3.2.6 SCRUM

První seznámení s agilní metodikou nazívající se Scrum. Ten se datuje roku 1986. Pro tuto konkretní metodiku je nejdůležitější komunikace se zákazníkem a maximum spolupráce s členy v týmu. Nejvhodnější je pro menší týmy o pár členech, ale je možné i zapojení více týmů. Úkolem členů v týmu je přezkoumání realizovaných požadavků na každodenní bázi a zadání dalších kroků. Scrum dokáže pracovat s měnícími se požadavky[3].

Vlastník (Product Owner) vytvoří produktu ve Scrumu. Sestaví požadovaný seznam (Product backlog), který se seřadí podle důležitosti. Vývoj nazývaný Sprinty běží v iteracích. Vývojový tým si naplánuje množství práce, kterou by měl během každého Sprintu stihnout. Tuto práci obsahuje Sprint backlog, který udává detail úkolů z Product backlogu v daném nejbližším období. *„Součást Sprintu jsou povinné schůzky, Denní schůzka (Daily Scrum), Plánovací schůzka (Sprint Planning), Retrospektiva daného sprintu (Sprint Retropective) a Vyhodnocení sprintu (Sprint Review), na kterých probíhají odhady práce, zpětná kontrola vykonané práce, plánování činností, přezkoumaní a vyhodnocení přírůstku - dokončení dané práce, úprava backlogu, návrh na zlepšení. Pokud se stanovené cíle splní, tak je produkt hotový a dokončen“* [17][3].

Praktiky Scrumu můžeme rozdělit na čtyři důležité body: artefakty, činnosti, role a pravidla viz. Obrázek 5.

1. **Role:** Lidé ve Scrumu mají určitou roli, která obsahuje svou funkci. Metodika má tři základní role: Product Owner(Vlastník produktu), Tým vývojářů a Scrum master. Ti dávají dohromady Scrum tým. Tato skupina si sama vede práci a měla být soběstačná.
2. **Vlastník produktu:** Je člověk, který zodpovídá za zákazníkovi investice a hájí zájmy zákazníka. Jeho úkolem je zadat projekt, rozhodnout o rozpočtu, rozsahu a termínech projektu. Seznamuje s dalším postupem vývojáře, určuje požadavky, stanovuje prioritní vlastnosti projektu a hledá reálná řešení. Také spolupracuje se všemi, kteří se na projektu účastní. Hlavní úkol je ověřovat plnění požadavků.
3. **Scrum master:** Je zodpovědný za dodržení pravidel Scrumu. Jeho hlavní rolí je pomáhat v komunikacích mezi vlastníkem produktu a vývojářskym týmem. Zároveň se snaží Scrumu udržet v mezích. Jeho úkolem je pomáhat

týmu řešit problémy, dodržovat cíle, tým inspirovat k mnohem lepším výsledkům než doposud a vést ho k samostatnosti. Velmi usnadňuje práci týmu a snaží se být prostředníkem, který se snaží zavést principy, zvyklosti Scrumu a hodnoty.

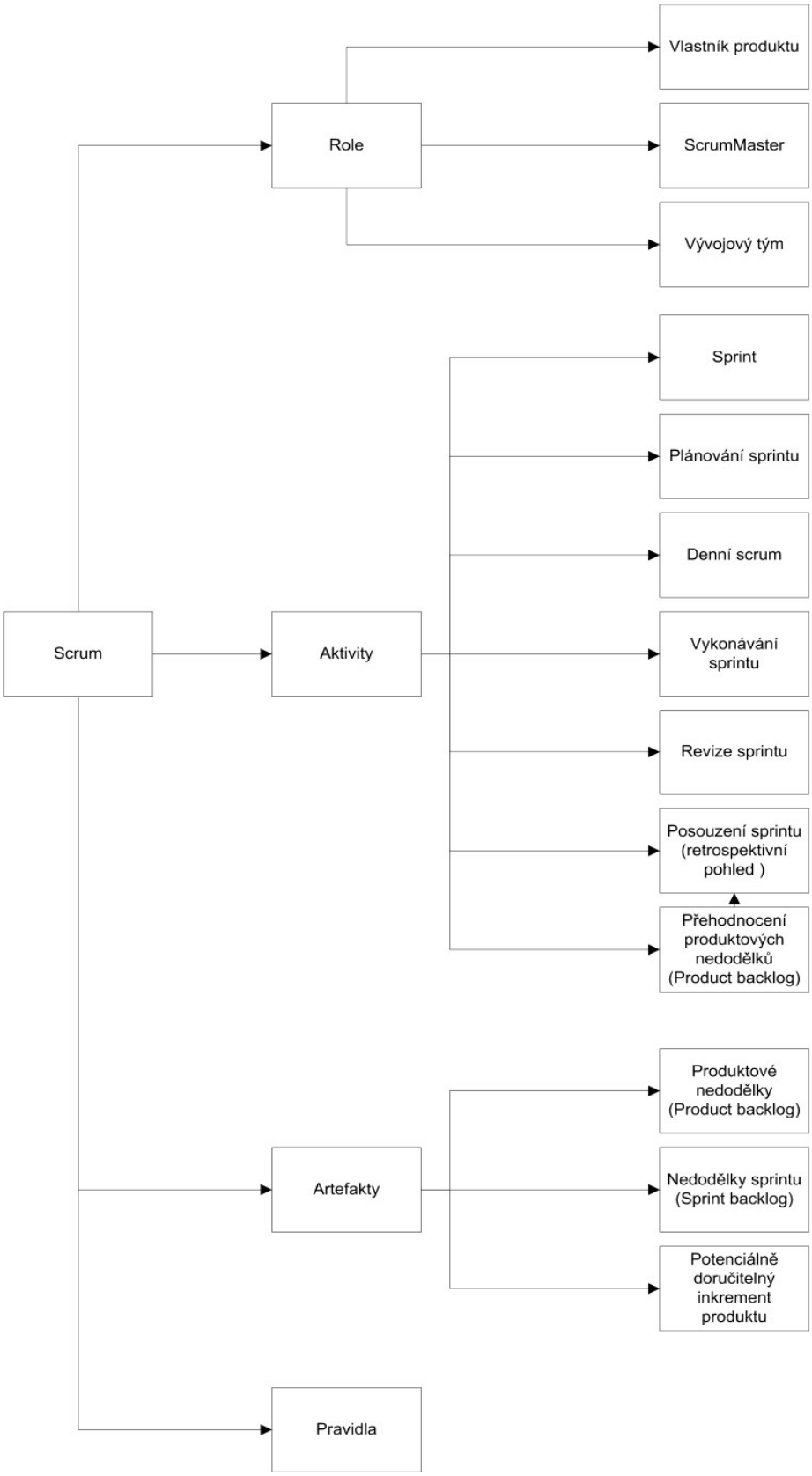
4. **Vývojový tým:** Tým, kde jsou obsažené všechny profese. Členové týmu jsou profesionálové, kteří během sprintu jsou schopni vyrobit přírůstek produktu. Každý vývojářský tým se samostatně spravuje a rozhoduje, nikdo jiný ho neorganizuje. Jeho úkolem je předvádět výsledky svojí odvedené práce vlastníkovi produktu, testovat a vyvíjet produkt a předkládat návrhy ke zlepšení. Během jednoho sprintu tým musí stoprocentne vynaložit úsilí k zadané práci, složení týmu se během daného sprintu nemění.
5. **Artefakty:** *„Artefakty označují entity, které ve Scrumu vznikají, mění se nebo se často se používají a které jsou užitečné pro kontroly a zajištění transparentnosti. Paří k nim Sprint backlog, Product backlog a přírůstek“* [17][3].
6. **Product backlog:** Je to důležitý seznam pro následující vývoj daného produktu, který obsahuje funkce, požadavky, vlastnosti, rozšíření, opravy, technické zlepšení a další práce. Na tomto seznamu jsou položky zapsany ve formě User stories (uživatelských příběhů) a mají určitou důležitost, odhad a popis. Během jednoho sprintu musí byt vše dokončeno a proto jsou rozděleny. Důležitost položek v daném seznamu se odráží od naléhavosti, míry rizika a od přínosu.
7. **Sprint backlog:** Popisuje určitou práci vývojářského týmu, kterou potřebujeme ke splnění Sprintu a k vytvoření přírůstku, který přinese uživateli očekávanou hodnotu. Vývojářský tým může provádět změny ve Sprint backlogu, jelikož denně sleduje položky, které aktualizuje a také upravuje seznam podle aktuálního vývoje. Úkolem Sprint backlog je pomáhaní vývojářskému týmu se zorientovat v kolech a odhadovat čas práce, kterou má za úkol splnit.
8. **Přírůstek:** *„Je to souhrn položek z daného Product backlogu, které byly dokončeny v průběhu minulých a současných Sprintů. Důležité ovšem je, aby byl Scrum týmem odhlasovaný jako „hotovo“ a aby po dokončení byl použitelný pro nasazení. To, zda je práce ve sprintu*

správně dokončena v odpovídající kvalitě posuzuje každý tým sám. Přírůstky se doplňují, testují se a sleduje se jejich kompatibilita" [17][2][3].

9. **Scrum činnosti:** Scrum vždy dodržuje předepsané činnosti, tak aby byla zajištěna jeho transparentnost v procesech, sledování nákladů a jejich kontrola. Díky často předepsaným činnostem Scrumu, které dodržují časové limity, které jsou pravidelné, nedochází ke ztrátám času a prodlevám v průběhu plánování.
10. **Sprint:** Je jaká si iterace ve Scrumu, která je časově omezena a výsledkem je přírůstek produktu. Důležité však je, aby všechny takto naplánované sprinty trvaly stejnou dobu. Sprinty na sebe navazují a mají jasne určený začátek a konec. Další nový sprint vždy startuje po ukončení předešlého sprintu. Po startu nemůžeme provádět změny, z hlediska ovlivnění cíle sprintu. Pokud dochází ke časové ztrátě daného vývoje, měl by se kontaktovat zadavatel a měla by probíhat konzultace, které položky ze současného sprintu v Product backlogu se mohou odstranit. V jíném případě se mohou přidat nové položky.
11. **Plánovací schůzka:** Během Sprintu se plánují činnosti daného Scrum týmu. Plánovací schůzka(Planing) má časový limit, během kterého se domlouvá budoucnost příštího Sprintu. Dochází k vytváření plánu práce, který je nezbytný pro daný sprint.
12. **Denní schůzka:** Vývojářský tým se denně schází, aby mohl projednat plán na další den. Úkolem dané schůzky je kontrola provedené práce od poslední schůzky. Po revizi dne predchozího se udělá odhad a naplánuje se práce na další den.
13. **Vyhodnocení sprintu:** Je to schůzka Scrum týmu, která je neformální s ostatními stakeholdery, která má za cíl přezkoumat přírůstek a jestli to bude nutné, tak Product backlog poupravit.
14. **Retrospektiva sprintu:** Zajímá se o procesy, nástroje a lidské zdroje, které byli pouzity. Plánuje dané kroky, k napomáhání se zlepšovat v činnosti týmu v nadcházející iteraci. Zpětně kontroluje práci Srcum týmu.

Efektivnost využití peněz a času zajišťuje Scrum. Velké projekty jsou často rozděleny na jednodušší a snadno zvládnutelné sprinty. Scrum funguje dobře pro rychleji se rozvíjející projekty. Při průběhu práce je možnost flexibilně reagovat na změny[3].

Díky každodenním schůzkám (Scrum meetings) zjišťujeme stav projektu a následné rozhodování, jak můžeme pokračovat dále, popřípadě změnit směřování projektu pro zefektivnění našeho procesu. Scrum metodika je souhrn vzorů, jak postupovat. Neudává specifika, jak by se měl software vyvíjet[2][3][4][17].



Obrázek 5: Praktiky Scrum metodiky

3.2.7 Vývoj řízený vlastnostmi (FDD – Feature Driven Development)

Feature Driven Development začíná tím, že se vytvoří doménový model popisující celý systém. Převede se do seznamu vlastností (elementárních funkcionalit, které přináší hodnoty uživateli). Vývoj má vždy pět fází (tři sekvenční a dvě iterativní). Doba trvání Iterace jsou dva týdny. Během iterace se vyvíjí konkrétní vlastnosti systému. Zákazník průběžně dostává nové verze produktu a mezivýsledky. Na rozdíl od Extremního porgramování nebo SCRUM je každému programátorovi práce přidělena[5][14].

3.2.8 Metodiky Crystal

V případě metodiky Crystal jde vlastně o metodiku, jak vytvářet a volit metodiky. Tvůrce Alistair Cockburn zformuloval tři základní myšlenky, na které tato metodika reaguje[25]:

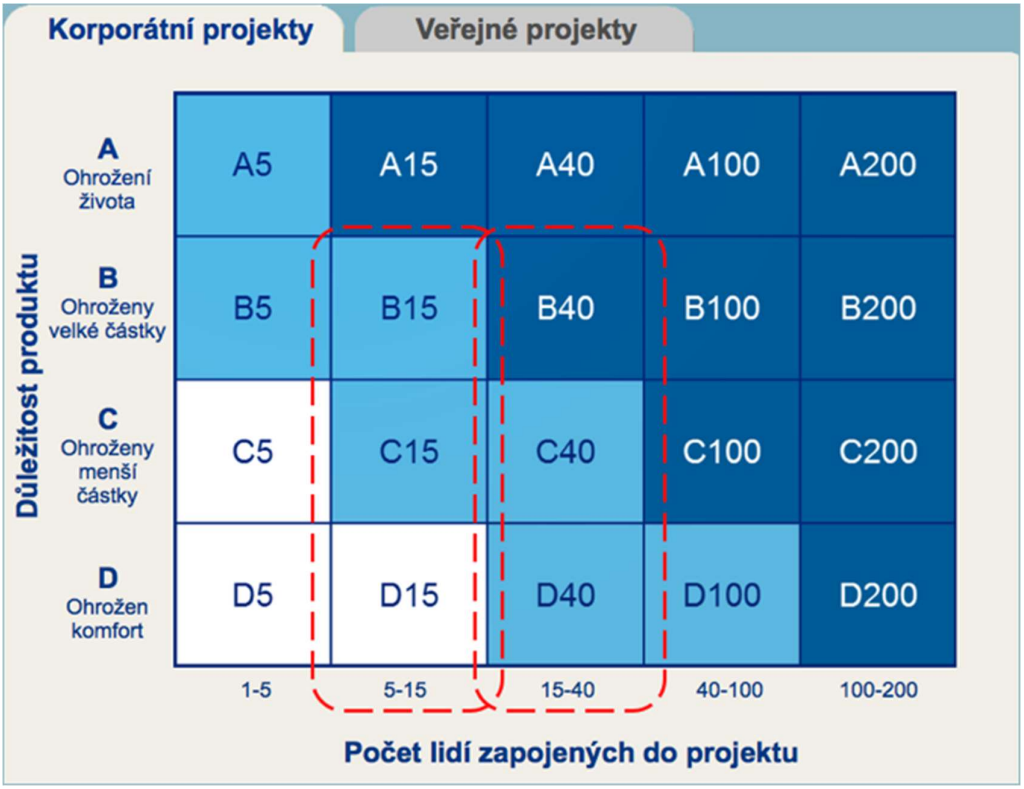
1. Každý softwarový projekt se vyznačuje odlišnou množinou parametrů, což vyžaduje rovněž odlišné zacházení a vedení. V ideálním případě bychom potřebovali pro každý projekt uzpůsobenou metodiku, která by ponechala stranou činnosti a nástroje nevhodné nebo přebytečné a naopak podpořila funkce nezbytné a konkrétní projektové cíle.
2. Úspěch projektu je vždy závislý na dobrých lidských zdrojích. Zlepšení podmínek pro práci projektového týmu se zpravidla projeví na jeho výkonnosti.
3. Základní podmínkou úspěšného projektu je vzájemná, otevřená a průběžná komunikace.

Dlužno říci, že tyto myšlenky mají univerzální platnost a pamatuje na ně dokonce i tradiční metodický standard PMBOK, který doporučuje vedoucímu projektu zvolit právě jen takové nástroje a postupy, jenž posunou projekt vpřed (viz poznámka v předchozí kapitole). Z pohledu metodiky Crystal existuje pět časových okamžiků, v nichž je nutné uvažovat o přizpůsobenim metodiky[25].

1. **Před započetím projektu**, kdy je nutné čerpat ze získaných lessons learnedi v předchozích obdobných projektů a hovořit s jejich účastníky

o slabinách, rizicích i funkčních a osvědčených postupech. Veškeré tyto relevantní poznatky je nutno shromáždit a zaznamenat.

2. **Na začátku projektu**, kdy se rozhoduje o délce iterací, požadovaných vstupech a mezivýstupech, o schvalovacím procesu, umístění pracovníků týmu (řešení lokálních pracovišť či organizace virtuálních týmů), použitých standardech i režimu komunikace či reportování. Všechny takto související parametry metodiky se volí a definují na základě společných úvah a poznatků získaných v předchozím kroku.
3. **Po skončení každé iterace** – v těchto pravidelných periodách bychom měli shrnout a vyhodnocovat, co se tým v uplynulé iteraci naučil, jaké problémy musel řešit a na co je vhodné se zaměřit v dalším průběhu projektu.
4. **V prběhu dalších iterací** – pokud zvolíme delší iterace (více než 3 týdny), je vhodné zařazovat v jejich průběhu pravidelná setkání, během nichž se ptáme na zpomalující a rizikové faktory, a to např. i ve zvolených technologiích a nástrojích.[25][2]



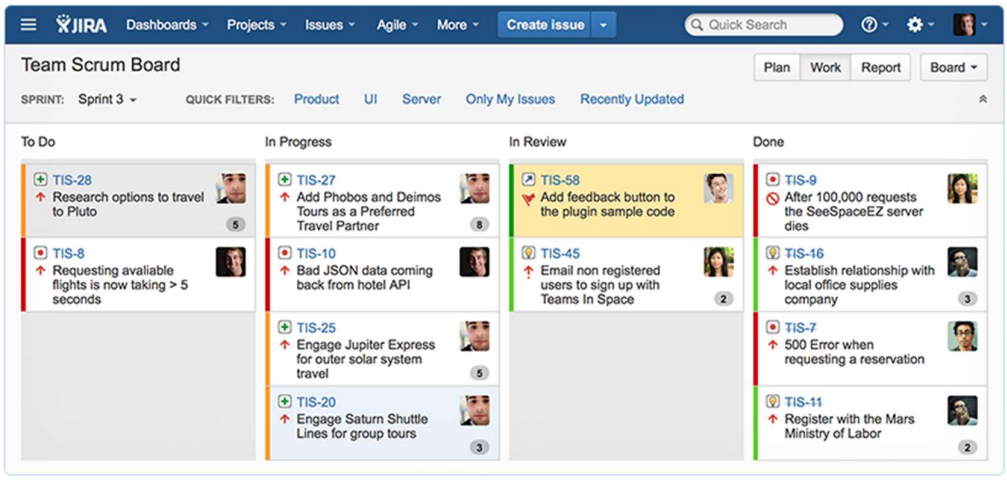
Obrázek 6: Metodika Crystal[25]

Všimněme si, že z pohledu tradiční metodiky PMBOK připomíná tento postup průběžnou analýzu rizik a formování odezvy na ně. Nejde tedy zatím o nic mimořádného ani převratného.Abychom mohli na počátku i v průběhu projektu volit z vhodné množiny postupů a opatření, je třeba vniknout do podstaty produktu, který projektem vytváříme a přesvědčit se, zda mu skutečně rozumíme. Vhodnou veličinou pro takové posouzení může být například míra důležitosti funkcí produktu, které mohou být ohroženy. Jiné nároky budeme zcela jistě klást na bezpečnost bloku jaderné elektrárny, než na kávomat. Míru důležitosti produktu si můžeme rozdělit třeba ve čtyřstupňové škále od A-Ohrožení života až po D-Ohrožení komfortu.[2][25]

3.2.9 Nástroje k vývoji SW

JIRA

Je navržena k podpoře vývojářských týmů v průběhu všech fází vývoje a provozu SW řešení. Celkově zahrnuje efektivní řízení a sledování požadavků a úkolů v projektu (project and task management). Podporuje a usnadňuje proces řízení požadavků a projektů - nabízí uživatelské a flexibilní nástroje pro sledování a řízení pracovníků při plnění úkolů. Orientuje se na podporu dosažení očekávaného výkonu na projektu.[21]



Obrázek 7: Jira nástroj[26]

Hlavní přednosti JIRA aplikace:

- Podpora pro projektové řízení
- Podpora agilních metodik – SCRUM
- Management workflow
- Management for incidents
- Tracker for issues
- Development for Scrum
- Management for Service Level (SLA)
- Road Map, Change Log Release Management
- Neustále dostupné informace přes webové rozhraní pro cely tým
- Vyhodnocování kapacit, sledování, podklady pro fakturaci a detailní evidence
- Historie projektové komunikace
- Podpora pro helpdesk a klientský servis
- Sdílení dokumentů, komunikace a informací v týmu
- Historie evidence, reporty, statistiky
- Přehledné grafy, manažerské reporty
- Řešení požadavků zákazníkem a sledování stavu projektu
- Termínuy dokončení, Uúkoly podle priorit
- Výkonná a stabilní a platforma JAVA Enterprise Edition
- Podpora ORACLE DB, MS SQL, PostgreSQL, mySQL a dalších db serverů

Enterprise Architect

Nástroj Enterprise Architect je vyvíjen australskou společností Sparx Systems.

Enterpise Architect podporruje UML verzi 2.3, ale s možností dodefinování dalších objektů a jejich vlastností se otevírá prakticky neomezená možnost tvorby vlastních modelů. Je to nástroj, který je schopen podpořit a velice usnadnit celou fázi vývoje software, od definování požadavků na systém, designu až po přípravu testování a dokumentaci systému. [19]

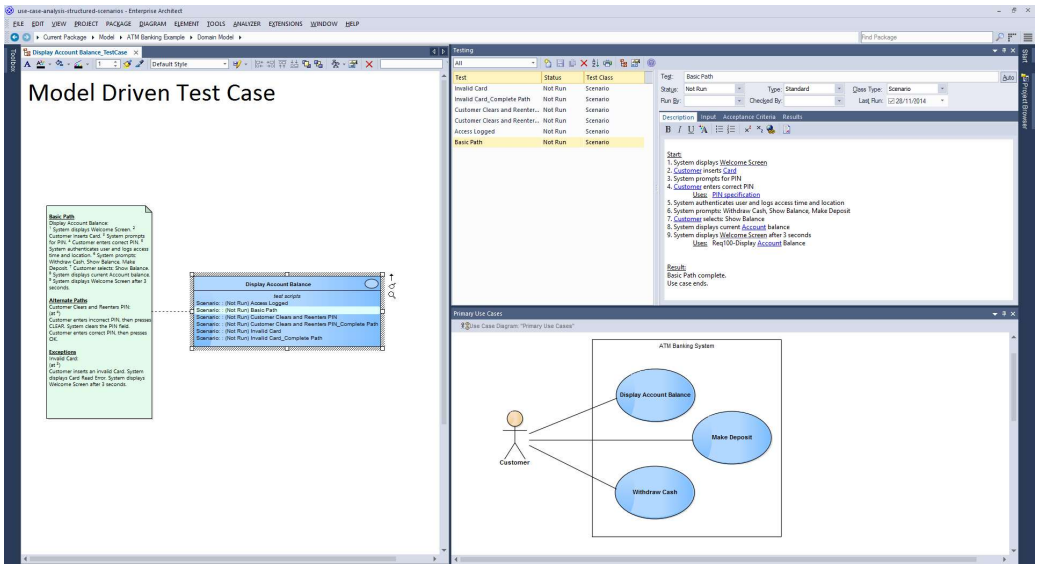
Podpora diagramů UML

Strukturní diagramy

- Diagram tříd (Class diagram)
- Objektový diagram (Object diagram)
- Diagram vnitřní struktury (Composite Structure Diagram)
- Diagram komponent (Component Diagram)
- Diagram nasazení (Deployment Diagram)
- Diagram balíčků (Package Diagram)

Diagramy chování

- Diagram případu užití (Use Case Diagram)
- Diagram činností (Activity Diagram)
- Stavový diagram (State Diagram)
- Diagramy interakcí



Obrázek 8: Enterprise Architect[19]

- Sekvenční diagram (Sequence Diagram)
- Diagram komunikace (Communication Diagram)
- Přehled interakcí (Interaction Overview Diagram)
- Modelování obchodních procesů

Datové modelování [20]

Enterprise Architect umožňuje i tvorbu logického a fyzického datového modelu. Zajímavostí je, že tvorba těchto modelů probíhá pomocí notace UML, tabulky jsou třídy se stereotypem „Table“. Jako vztahy mezi tabulkami je možné použít standardní vztahy známé z objektového a datového modelování – tedy především asociaci a generalizaci.[19]

4 Praktická část

V praktické části dochází k zhodnocení jednotlivých metodik a nástrojů. Především na přizpůsobitelnost nových řešení kladou v potaz agilní metodiky. Pevně definovaný čas je u agilní metodiky nejdůležitější, jeho délka se určuje hned na začátku. Oproti rigorózní model je prioritní funkcionalita. Proto agilní metodiky z časového hlediska redukují danou funkcionalitu jen na nejnutnější funkce a implementují je do zdrojového kódu. Dále, tyto metodiky předpokládají, že často dochází při vývoji ke změnám funkcionality, proto se klade důraz na přizpůsobivost. Pro kontrolu upravených a změněných částí zdrojového kódu dochází k častému testování, aby byla zajistěna a zachována kompatibilita softwaru. Dalším důležitým bodem těchto metod je častá komunikace se zákazníkem. Principy agilních metodik se odvíjí z Manifestu agilního vývoje SW.

4.1 Zhodnocení SCRUMU a Lean Developmentu

Scrum je vývojový softwarový rámec, který je zaměřením na schopné jedince, zatímco Lean Development optimalizuje vývojový proces, který neomezuje chod softwaru. Porovnávané metodiky nám mohou slibit snížení spotřeby peněz a zefektivnění času. Scrum může vyjádřit časově dobu vývoje. Lean Development udává, jak dospět k dané peněžní a časové úspoře podle deseti základních pravidel dané metodiky. Metodiky Scrum a Lean se soustředí na různé vývojevé aspekty, obě metody se zaměřují hlavně na určité body, které vytvářejí základní hodnoty principů, následně sdílející. Lean a Scrum se navzájem vcelku dobře doplňují. V případech, že Scrum nepředepisuje konkrétní dané procesy a může být rozšířen o principy z Lean Development nebo proces.

4.2 Zhodnocení SCRUMU a Extrémního programování

Scrum metodika, zatímco extrémní programování se zabývá spíše produktivitou, je zaměřen na samotný proces vývoje. Přínosem Extrémního programování, je v mnoha společnostech využíváno inženýrských praktik, jako testově řízený vývoj a párové programování. Dané činnosti zrychlují proces vývoje, který omezují potřebu znovu procházet kód. Týmy Extrémního programování pracují pro zákazníka na požadavcích v jasně definovaných prioritách a v daném pořadí. Zatímco Scrum tým nemusí dodržovat

plnění požadavků v daném sledu, ale v rámci jednoho sprintu je může plnit libovolně. Začínat Scrum metodikou je lepší, z mého pohledu, jelikož je více formálně ukotvená oproti Extrémnímu programování, které vyžaduje úzký vztah týmu, schopněplný uvést věci do daného extrému, které známe jako základní kameneny extrémního programování. S ohledem na lidský organismus, aby neklesala rychlost průběhu Vývoje SW, je snahou těchto metodik, dodržování stanovené pevné pracovní doby, 40 hodin týdně. U Extrémního programovaní, během vývoje iterace, je mnohem více benevolentní ke konkrétním daným vlastnostem.

4.3 Zhodnocení nástrojů vývoje SW

Každý nástroj byl číselně ohodnocen podle odborníků z praxe. Zvolená metoda byl brainstorming, kde probíhal řízený rozhovor mezi odborníky z praxe. Jednotlivá kritéria jsou ohodnocena podle hodnotící stupnice od 1 do 10 (bodovací metoda). Minimální hodnota 1 značí, že metodika je nevyhovující požadavkům na projekt. Maximální hodnota 10 značí, že metodika vyhovuje nejlépe požadavkům na projekt.

Zhodnocení nástrojů pro kooperativní vývoj SW						
	Software pro kreslení diagramů		**Software pro sledování požadavků a projektů**		**Software pro sledování testů**	
Nástroje	Enterprise Architect	ProgeCA	JIRA	Microsoft TFS	HP Quality Center	PractiTest REVIEW
Reporting	X	X	9	8	9	7
Tvorba a manipulace	8	7	9	7	8	8
Týmová spolupráce	8	6	8	6	9	6
Jednoduchost a intuitivní UI	7	8	8	9	9	7
Celkové Hodnoce	8	7	9	8	9	7

Tabulka 1: Zhodnocení nástrojů dle kritérií SW

4.4 Výběr metodiky

Z porovnání a výčtu metodik byla vybrána Scrum metodika. Hlavně pro její přínosy, které umožňí sledovat aktuální pracovní postup v projektu a dodání mezi-iteračních výsledků zákazníkovi. Poté také tím, že je možné během sprintu vytvářet software v jakémkoli sledu zadefinovaných požadavků právě běžícího sprintu. Scrum master po konzultaci s vlastníkem daného produktu umožňí přidání dalších User stories z vytvořeného backlogu do sprintu v případě, pokud se vykonály všechny úkony ze sprint backlogu. V jiném případě může Scrum Master na základě konzultace se zákazníkem pár požadavků slevit, aby byl korektně dodán funkční SW z již konkrétní iterace sprintu včas. Hlavním důvoduem pro výběr této z metodik byl zpětnovazebně řízený vývoj Scrumu, který si zakládá na zpětné vazbě Vlastníka produktu po každé ze zadané iteraci vývojového sprintu. Každá metodika se do jisté míry vymezuje oproti ostatním. Přichází s nějakou novinkou, věnuje pozornost určitému aspektu vývojového cyklu. Toto kritérium popisuje na jakou fázi popř. fáze klade metodika největší důraz.

Kritérium	Vymezení Hodnota / významu kritéria
1. *„Typ řešení“ [2, s. 28]*	a) nové řešení, b) rozšíření stávajícího řešení, c) integrace řešení, d) inovace [2, s. 28]
2. *„Přístup k řešení“ [2, s. 28]*	a)strukturovaný, b) objektově-orientovaný, c) RAD – rychlý vývoj aplikací [2, s. 28]
3. Míra složitosti dané metodiky (propracovanost, obsáhlost)	a) velmi náročná na složitost b) složitá metodika b) střední náročnost c) jednoduchá metodika [2, s. 23 - 24]
4. Zaměření testů (typ testů)	a) Základní typy testů: white box a black box b) Fáze testu: unit testy, testy integrace, smoke testy funkční testování, uživatelsko akceptační testy c) ostatní testy
5. Rozsah (a hloubka) testování	a) testování na začátku vývoje (v počáteční fázi) před implementací b) průběžné testování v každé fázi po každé iteraci během celého vývoje c) testování na konci vývoje před implementací d) testování na konci vývoje v rámci fáze implementace
6. Zaměření metodiky na fázi vývoje	a) vyzdvihuje (klade důraz na) jednotlivou fázi : požadavky, návrh, implementace, testování atd. b) rovnoměrné rozložení fází

Tabulka 2: Kritéria

4.5 Metodiky podle stanovených kritérií

V tabulkové části je zpracován popis metodik podle kritérií. Pro jednotlivé metodiky byla každému kritériu porovnání přiřazena hodnota kritéria a číselné ohodnocení kritéria podle formulace zadání a vymezení projektu. Každá tabulka má kritérium, jeho hodnotu, co nejvíce vystihuje metodiku, a číselné hodnocení daných kritérií pro uvedené metodiky vývoje. Tyto metodiky byly vzájemně porovnávány na základě kritérií. Každá metodika byla číselně ohodnocena odborníky z praxe. Opět byl použit brainstorming, kde probíhal řízený rozhovor mezi odborníky z praxe. Opět jednotlivá kritéria jsou ohodnocena podle hodnotící stupnice od 1 do 10 (bodovací metoda). Minimální hodnota 1 značí, že metodika je nevyhovující požadavkům na projekt. Maximální hodnota 10 značí, že metodika vyhovuje nejlépe požadavkům na projekt.

RUP - Rational Unified Process		
Kritérium	**Hodnota kritéria**	**Hodnocení**
1. Typ řešení	spíše nové řešení (rozšíření zavedeného)	4
2. Přístup k řešení	objektově-orientovaná ve všech fázích	7
3. Míra složitosti metodiky	velmi náročná na složitost	8
4. Zaměření testování (typ testů)	black box testy, funkční testování, definuje FURPS	9
5. Rozsah (a hloubka) testování	průběžné testování v každé fázi (po každé iteraci během celého vývoje)	7
6. Zaměření metodiky na fázi vývoje	rovnoměrné rozložení fází	7

Tabulka 3: Hodnoty kritérií a jejich hodnocení metodiky RUP

XP – Extrémní programování		
Kritérium	**Hodnota kritéria**	**Hodnocení**
1. Typ řešení	rozšíření stávajícího řešení, integrace zavedeného	5
2. Přístup k řešení	strukturovaný, RAD	2
3. Míra složitosti metodiky	jednoduchá metodika	3
4. Zaměření testování (typ testů)	White box testování, unit testy, testy integrace, UAT (viz kapitola 5.2)	6
5. Rozsah (a hloubka) testování	testování na konci vývoje před implementací	5
6. Zaměření metodiky na fázi vývoje	Kódování (implementace)	6

Tabulka 4: Hodnoty kritérií a jejich hodnocení metodiky XP

FDD – Feature-Driven Development (vlastnostmi řízený vývoj)		
Kritérium	**Hodnota kritéria**	**Hodnocení**
1. Typ řešení	nové řešení	4
2. Přístup k řešení	objektově-orientovaná ve všech fázích	8
3. Míra složitosti metodiky	střední náročnost	6
4. Zaměření testování (typ testů)	nedefinováno	3
5. Rozsah (a hloubka) testování	testování na konci vývoje v rámci fáze implementace	4
6. Zaměření metodiky na fázi vývoje	Požadavky – seznam vlastností	5

Tabulka 5: Hodnoty kritérií a jejich hodnocení metodiky FDD

TDD – Teature-Driven Development (vlastnostmi řízený vývoj)		
Kritérium	**Hodnota kritéria**	**Hodnocení**
1. Typ řešení	rozšíření stávajícího, integrace zavedeného	5
2. Přístup k řešení	strukturovaný, RAD	3
3. Míra složitosti metodiky	jednoduchá metodika	3
4. Zaměření testování (typ testů)	white box testování, unit testy, testy integrace (viz kapitola 5.2)	7
5. Rozsah (a hloubka) testování	na začátku vývoje v počáteční fázi před implementací	6
6. Zaměření metodiky na fázi vývoje	testování	9

Tabulka 6: Hodnoty kritérií a jejich hodnocení metodiky TDD

5 Diskuze

5.1 Řešení podle FDD

Metodika FDD popisuje celkem 5 fází vývojového procesu: podrobný seznam vlastností, vytvoření celkového modelu, návrh podle vlastností, plánování podle vlastností a implementace dle vlastností. Nejdříve popíšu setup vývojového týmu a poté jednotlivé fáze.

Vývojový tým

Tým často má 6 členů, které dělíme na 6 rolí:

- Projektový manažer (a současně hlavní programátor)
- Analytik a Designer
- Programátor
- Databázový specialista
- Tester

Celkový počet členů pro všechny role je 10, v týmu ale vidíme jen 6 lidí, což znamená, že někteří členové týmu zastávají více rolí. Jeden z programátorů je také i databázový specialista, další programátor také testuje.

5.1.1 Část 1: Celkový model

Celý model by měl poskytovat náhled na systém jako jeden celek a demonstrovat základní účel. Často je reprezentován diagramem tříd. Model se žačne vytvářet po tom, co expert na doménu seznámí vývojářský tým s cílem projektu a s hlavními požadavky na systém.

Doménový expert ve spolupráci s hlavním architektem navrhl tento následující model zakreslený diagramem tříd.

K tomuto diagramu byl také vypracován následující popis všech objektů.

Člověk

Jelikož má systém sloužit k sledování externích lidských zdrojů, je hlavní a nejdůležitější třídou „člověk" v celém systému. Pracuje s lidma, kteří v systému figurují jakýmkoliv způsobem, jak aktivně nebo jen pasivně. Práva a úlohu člověka k systému

často určuje atribut „typ". Většina techto atributů uchovává různé kontakty na daného člověka, seznam techto lidí slouží pro zaměstnance jako adresář kontaktů.

Projekt

Zde se ukládají informace o všech projektech, která firma dělá, a na kterých pracují její zaměstnanci.

Milník

Milníky jsou „checkpoint“ body projektu, identifikované datem a názvem. Například to může být „dokončení prvního návrhu", „dodání designu od grafika",

„start testování", apod.

Úkol

Pojem úkol je nárazová, často krátkodobá, práce na některém z určených projektů, kterou ale nevykonává jeden z členů projektového týmu. Slouží ku příkladu k zadání adhoc úkolů konkrétním členům týmu, které není dobré dlouhodobě vést jako lidi projektového týmu. U úkolu sledujeme jeho „stav", který řešitel úkolu mění, a který nabývá hodnot „akceptován", „zadán" a „dokončen".

Profil

Je namysli časový profil, který patří danému člověku. Každý z techto lidí má 7 záznamů v profilu, pro každý den. Každý záznam poté obsahuje 3 hodnoty uvádějící optimální, minimální a maximální odpracovaný počet hodin.

Čas

Třída podobná třídě „profil", s rozdílem, že zde máme 3 výše zmíněné časové hodnoty vztažené už k určitému danému datu. Uchovává se tu časová dostupnost pro jednotlivé lidii.

Požadavek

Požadavek je krátká zpráva, kterou kterýkoliv člověk může zaslat jinému člověku. Projektový manažer může přes daný systém například poslat požadavek člověku, který se stará o lidské zdroje ve firmě, že kapacita jeho týmu nestačí pro všechny jeho zadané práce na projektu a požaduje rozšíření svého projektového týmu o další lidi. Stejně tak jako úkoly mají i požadavky svůj daný „stav", konkrétně jednu z hodnot „akceptován", „poslán" nebo „vyřízen". Stav požadavku může změnit pouze jeho příjemce. Odesílatel má poté přehled , jestli se požadavku adresát věnuje.

Příloha

Zanechává přílohy, které projektový manažer může uložit do systému k danému úkolu nebo projektu. Mohou to být různé podklady a dokumenty, například specifikační dokument, návrh grafického designu Aplikace, apod. Všichni členové týmu je díky tomu mají pořád k dispozici ke stažení. Příloha Specializace třídy „Příloha", která určuje některý z konkrétních úkolů, ke kterému je právě příloha určena.

Celý tento model ještě může být podroben inspekci a schválen zadavatelem, ale tento zadaný model je natolik jednoduchý, že stačí kontrola mezi jednotlivými členy vývojářského týmu.

5.1.2 Část 2: Seznam vlastností

Nalezení a sestrojení seznamu vlastností má na starost vedoucí programátorů (chief programmer), který se podílí na sestavování celkového modelu v první fázi. V tomto projektu je vedoucí programátor pouze jeden, ale u větších projektů jich může být více.

Seznam daných vlastností je rozdělen do skupin, které podle toho spolu logicky souvisejí. V tomto případě tento seznam může být poměrně dlouhý a u větších projektů může čítat i stovky položek. Proto zde uvedu jen jeho menší část. Vezměme například část systému zaměřený na nastavování času a jeho alokací v daných projekty.

Nalezní skupiny vlastností:

- Nastavení profilu času
- Časové nastavení
- Alokace daného času na projekt

Nalezení a popis jednotliveých vlastností:

- Nastavení profilu času
- Vypsání časového profilu na obrazovku
- Uložení změněného profilu do databáze

Nastavení času:

- Vypsání aktuálního času do formuláře
- Uložení času do databáze

Alokace času na projektu:

- Vypsání vybraného projektu na obrazovku
- Zobrazení formuláře alokace času s možností výběru měsíce
- Uložení alokací do databáze

Podobným způsobem můžeme sestavit kompletní seznam vlastností.

5.1.3 Část 3: Plánování dle vlastností

Úkolem této fáze je správně seřadit činnosti do daných posloupností, v jaké budou jednotlivé vlastnosti naimplementovány, předběžně správně rozvrhnout práce, společně se stanovením datumu ukončení vývoje. Dále se musí jednotlivým programátorům přiřadit třídy, za jejichž dokončení budou odpovídat. Tým pro plánování stanovil posloupnost vlastností a přiřadil ke každé vlastnosti její prioritu.

Vysoká priorita znamená to, že daná vlastnost je důležitá k fungování systému a měla by být naimplementována před jeho nasazením. Vlastnosti, které se týkají časového profilu, budou naimplementovány až na závěr, mají nižší prioritu, protože nejsou pro funkčnost daného systému nezbytně nutné. Ve specifikačním dokumentu je profil času uveden jako konkrétní požadavek, tato funkce bude také naimplementována, ale tým správně odhadl, že odložení těchto prací na této části daného systému neohrozí jeho dokončení.

5.1.4 Část 4: Návrh dle vlastností

Tato fáze je již zaměřena na jednotlivou vlastnost. Její výběr uskuteční hlavní programátor ze daného seznamu vlastností. Vybírá je podle priorit a vzájemných závislostí. Jediným výstupem by mohl být sekvenční diagram a nebo diagram spolupráce. V tomto případě mohl být pro tuto skupinu vlastností uvedenou v druhé části sestaven

sekvenční diagram. Vykresluje komunikaci tříd při zadávání času a alokace na konkrétní projekt.

Nesmíme ještě zapomínat ověřit celkový model narozdíl od detailního návrhu vlastnosti. Může se stát, že díky tomu se bude muset upravit celkový model vytvořený v první fázi. V takovém případě se vrátíme do první části, model upravíme, a projdeme znovu část druhou a třetí, abychom zpracovali všechny vzniklé změny.

V tomto daném případě k žádným konkrétním změnám při návrhu podle vlastností nedošlo, můžeme tedy globální model ponechat beze změny.

5.1.5 Část 5: Implementace dle vlastností

Nastává poslední fáze, ve které se tým pouští do funkcí potřebných pro naimplementování vybrané vlastnosti. Metodika Future Driven Development specifikuje jak naprogramovat, jak provést inspekci, otestovat dané jednotky, integrovat a sestavit. Jak konkrétně a jak za použití některých nástrojů by dané činnosti měly být provedeny už metodika nespecifikuje. Volba je vždy na hlavním programátorovi, který má mít dostatek zkušeností, aby pro daný vývojářský tým zvolil správné prostředky a postupy.

5.2 Řešení podle SCRUM

SCRUM je metodika, která přesně definuje co ve které fázi vývoje dělat, kdo co má na starosti a co má být správným výsledkem. Tady je ovšem všechen dostupný popis u konce. Dozvíme se co je potřebné udělat, ale už nevíme, jak toho dosáhnout. Konkrétní danou formu realizace určitých kroků musí určit Scrum Master. To je ovšem na znalostech a zkušenostech, které on sám má. Metodika jako taková mu nijak nepomůže. Na toto lze nahlížet dvěma pohledy. Buď to bereme jako nedostatek a poté nám může tato metodika připadat nekompletní, a nebo na to můžeme nahlížet jako na příležitost, jak do metodiky vnést nějaké své vlastní vývojové procesy, postupy a možnost, jak přizpůsobit metodiku vlastním potřebám. V takovém případě nám SCRUM může dávát volnou ruku a nesvazovat nás žádnými pravidly.

Vývojářský tým

Ze strany zadavatele je v daném týmu osoba označována jako zákazník (Customer). Na řešitelově straně je nejdůležitější Scrum Master. Ten se stará o to, aby vývoj proběhl podle daných praktik a zásad SCRUM metodiky. Vlastník (Product owner) zodpoveídná za projekt jako takový. Spolupracuje se Scrum Masterem na sestavení Product Backlogu. Při vybírání, které z vlastností budou mít určitou prioritu a kolik času se bude věnovat jejich implementaci. Pokud není stanoven samostatný člověk na tuto roli, pak roli vykonává Scrum Master. Toto je náš případ. Posledním článkem je vždy Scrum Team, tým podílející se na sestavení určitého Sprint Backlogu , na návrhu a implementaci jednotlivých daných vlastností. Detailnější rozdělení týmu metodika nikde neuvádí, jen dodává. Scrum Team by měl být samoorganizující s dobrou komunikací. Měl by být schopen se dohodnout, kdo bude odpovědný za testování nových naimplementovaných vlastností, kdo se bude starat o zakreslování návrhů nově vzniklích vlastností v UML, apod. Vývojářský tým se tedy skládá ze 7 lidí. Jeden je zákazník, další je role Scrum Mastera a je zároveň rolí Vlastníka produktu. Zbývající členuové týmu tvoří Scrum Team.

Vývoj dle SCRUMU probíhá ve třech navazujících fázích, které nazýváme předehra, hra a dohra. První a třetí fáze bývají lineární, prostřední fáze bývá iterační.

5.2.1 Část 1: Předehra

Formálně se tato konkrétní fáze skládá ze 2 určitých kroků: Plánování, druhým pak Architektura a návrh. Při plánování má Scrum Master nejvíce úkolů. Sestavuje harmonogram, zajist´uje potřebné zdroje, zvolí vývojové nástroje a provede analýzu rizik. Nejdůležitějším dokumentem, který vznikne v této fázi je Product Backlog. Udržuje ho Vlastník produktu (společně se Scrum Masterem) po celou jeho dobu vývoje dané aplikace. Většinou to bývá ve formě listu v některém tabulkovém procesoru například OpenOffice.org Microsoft Excel nebo Calc. Pro ukázku stačí jeho část, v celé verzi by Product Backlog měl by obsahovat co největší počet požadavků vycházejících ze specifikace. Product Backlog se bude v průběhu vývoje podle daných situací postupně doplňovat a narůstat, to má často na starost Scrum Master.

Každé položce je dán přiřazena priorita, očekávaná doba implementace a člověk, který je zodpovědný za implementaci. Předpokládaná doba trvání je pouze orientační

a musí se počítat s tím, že členové týmu nejsou na začátku vývoje schopni přesněji určit, jak dlouho vývoj dané položky bude trvat. Scrum Master toto musí mít na paměti a musí brát tyto hodnoty s rezervou. Používají se jen pro hrubé plánování.

V kroku Architektura a návrh se navržená architektura často může modifikovat ku příklad podle analýzy rizik v průběhu Plánování. Dále zde můžeme provádět doménové analýzy, tzn. podrobně zkoumat prostředí zadavatele, kde vyvíjený software bude nasazen, se všemi obchodními procesy, kterých se z některých daných aplikací týká. S tím také souvisí i studium dostupných a relevantních dokumentů, často poskytovaných vývojovému týmu zadavatel.

5.2.2 Část 2: Hra

Několik sprintů tvoří herní fázi (hru). Počet a délka sprintů je závislá na druhu a složitosti aplikace, ale také na tom, jak velký je vývojový tým a na řadě dalších paramerů. Vedoucí projektu vybere z každého sprintu "skupinu položek Product Backlogu," které se budou v rámci téhož sprintu vyvíjet. Této skupině říkáme Sprint Backlog. Následuje důkladné rozebrání jednotlivých položek Sprint Backlogu vývojovým týmem a stanovení plánu vývoje. V tomto případě nemůžeme hovořit o složitém projektu, proto určíme délku trvání sprintu 14 dní. Backlog udržuje Scrum Master v průběhu celého sprintu, zaznamenává do něho *„aktualizace času odpracovaného na jednotlivých úkolech a případně doplňuje nově vzniklé úkoly."* Další ústřední činností pro Scrum je vyjma zachování backlogu také organizonání denních setkání vývojářů (Daily Scrum Meetings). Jejich organizování a vedení má zodpovědnost také Scrum Master.

5.2.3 Část 3: Dohra

Dohra znamená, že je již opuštěn iterační cyklus. Vývoj dalších vlastností a funkcí již není potřebný a pokračuje se přímočaře. *„Výsledný produkt často prochází integrací a testováním, jakým způsobem se bude testovat metodika nepopisuje, záleží to na vývojovém týmu. Jako poslední bod se vytváří dokumentace, například uživatelské manuály, apod".*

6 Závěr

Hlavním cílem Bakalářské práce bylo zhodnotit Agilní metodiky vývoje SW. Byly vybrány tři nejvýznámější metodiky, u nichž byly popsány jejich výhody a nevýhody. Bylo dosaženo závěru, že tradiční vývojové metody jsou dnes již na ústupu a více se používají Agilní metodiky.

V Teoretické části byl také podrobně rozpracován postup a praktiky metodik Extrémní programování, Lean Development, SCRUM Development Process, Feature-Driven Development a Metodiku Crystal. Další částí byl rozpracován rozbor vybraných vývojových nástrojů. Byly vyzdviženy jejich nejdůležitější vlastnosti k vývoji SW. Každá agilní metodika a nástroj má své pozitivní i negativní praktiky při vývoji.

V Praktické části byly zhodnoceny vybrané a nejvíce používané metodiky a nástroje k vývoji SW. U všech porovnávaných metodik je zřetelná jedna společná věc a to, že největší důraz je kladen vždy na jednu konkrétní činnost.

U metodik Extremní progamovaní, Feature Driven Development je více zaměřeno na jeden daný aspekt vývoje. U Metodiky Feature Driven Development je kladen velký důraz na disciplínu modelování. Výhodou metodiky jménem Rational Unified Process (RUP) je považována dokumentace vývojového procesu, která je velice důležitá pro případné rozšiřování systému. Další předností této metodiky, oproti ostatním, je především v bohatém rozsahu testů. Žádná z výše porovnávaných metodik není tak bohatá na testování jako právě metodika Rational Unified Process (RUP). Podle zjištení během porovnání je ale v dnešní době nejvíce používána metodika SCRUM. Výhodou metodiky jsou krátkodobé sprinty, po kterých jsou změny nasazeny na produkční prostředí. Dalšími výhodami jsou denní Stand Upy, kde se konkrétně komunikuje co je v jakém stavu v aktuálním sprintu.

Při dobře složeném a kompaktně fungujícím týmu, který má mezi sebou smysluplnou spolupráci, bylo zjištěno, že nejvhodnější metodiku, kterou můžeme doporučit k vývoji SW v menších IT firmách je právě SCRUM. Byly charakterizovány metodiky, podrobnější zhodnocení Agilní metodik vývoje SW a v praktické části bylo

nastíněno fungováni různých metodik, podle kterých by se mohl vyvíjet produkt na nějakém daném projektu.

7 Seznam použitých zdrojů

[1] **Astels, David**. Test Driven Development: A Practical Guide, Prentice Hall 2003. ISBN-13: 978-0131016491.

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

[3] **BEEDLE, M**. **-- SCHWABER, K**. Agile Software Development with Scrum. Upper Saddle River: Prentice Hall, 2002. ISBN 0-13-067634-9.

[4] **Highsmith, Jim**. Agile Software Development Ecosystems, Addison-Wesley 2002. ISBN-13: 978-0201760439.

[5]**PALMER, S R**. -- FELSING, J M. A practical guide to feature-driven development. Upper Saddle River, NJ: Prentice Hall PTR, 2002. ISBN 0130676152.

[6] **Beck, Kent**. Extrémní programování. [překl.] Pavel Makovec. Praha : Gradapublishing, 2002. str. 160. ISBN 80-247-0300-9.

[7] **Agilní metodiky řízení vývoje software (Agile software development methodologies)** [online]. [cit. 2018-11-09]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html

[8] **BUCHALCEVOVÁ, Alena**. Metodiky vývoje a údržby informačních systémů. 1. vyd. Praha : Grada, 2005. 163 s. Management v informační společnosti. ISBN 80-247-1075-7.

[9] **Marry, Poppendick**. Lean Software Development. místo neznámé : Addison Wesley, 2003. ISBN 0-321-15078-3.

[10] **Manifest Agilního vývoje software** [online]. [cit. 2018-12-20]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html

[11] **Návrh vývojové metodiky pro webové aplikace** [online]. [cit. 2019-02-04]. Dostupné z: http://docplayer.cz/3551036-Navrh-vyvojove-metodiky-pro-webove aplikace.html

[12] **Spirálový model** [online]. [cit. 2019-02-05]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu softwaru/spiralovy-model/

[13] **Metodika řízení projektů podle A ...** [online]. [cit. 2019-02-05]. Dostupné z: http://slideplayer.cz/slide/2283123/

[14] **Programujte agilně, ...** [online]. [cit. 2019-02-08]. Dostupné z: http://www.zive.cz/clanky/programujte-agilne-nic-jineho-vam-nezbyva-a-nebo-ano/sc-3-a-111219/default.aspx

[15] **Extrémní programování pod drobnohledem** [online]. [cit. 2019-02-09]. Dostupné z:https://www.zive.cz/clanky/extremni-programovani-pod-drobnohledem/sc-3-a-111952/default.aspx

[16] **RUP – Rational Unified Process** [online]. [cit. 2019-02-11]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/rup/

[17] **Agilní vývoj: Scrum** [online]. [cit. 2019-02-11]. Dostupné z: https://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

[18] **Méně práce s řízením organizace** [online]. [cit. 2019-02-11]. Dostupné z: http://www.pdqm.cz/blog/manifest-agilniho-programovani

[19] **Sparx. Enterprise Architect** [online]. [cit. 2019-02-13]. Enterprise Architect - UML. Dostupné z www: http://www.sparxsystems.es/New/products/ea.html.

[20] **SVOBODA, Kamil**. Modelovací nástroj Enterprise Architect. Hradec Králové, 2005. 21 s. Seminární práce. Univerzita Hradec Králové. [cit. 2019-02-13].

[21] **MYJIRA** Základní popis. Myjira.cz [online]. [cit. 2019-02-13]. Dostupné z: http://www.myjira.cz/produkty/project-tracking-tools/jira.html

[22] **Vodopádový model** [online]. [cit. 2019-02-20]. Dostupné z: http://slideplayer.cz/slide/2505269/

[23] **Spirálivý model** [online]. [cit. 2019-02-20]. Dostupné z: http://slideplayer.cz/slide/2913967/

[24] **RUP model** [online]. [cit. 2019-02-20].Dostupné z: http://en.wikipedia.org/wiki/RUP_hump

[25] **Agilní projekty z pohledu zákazníka** [online]. [cit. 2019-02-20]. Dostupné z: http://m.web-integration.info/cs/blog/agilni-projekty-zpohledu-zakaznika/

[25] **Metodika Crystal** [online]. [cit. 2019-03-06]. Dostupné z: http://www.adapma.com/page11/styled-8/index.html

[26] **Jira nástroj** [online]. [cit. 2019-03-06]. Dostupné z: http://www.g2crowd.com/products/jira/reviews

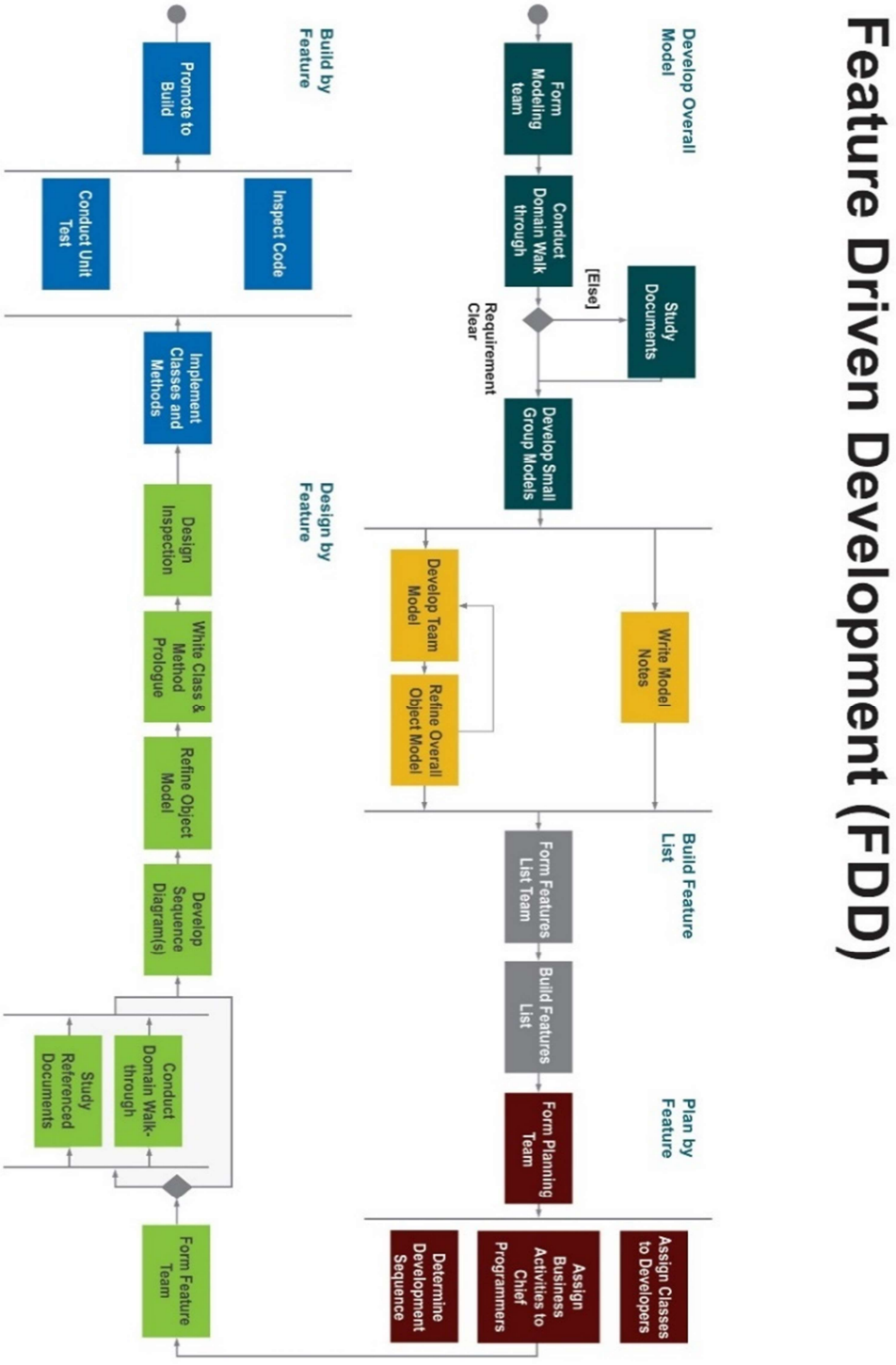
[27] **Feature Driven development Flow** [online]. [cit. 2019-03-06]. Dostupné z: https://mejorimagen.eu/feature-driven-design-fdd.html

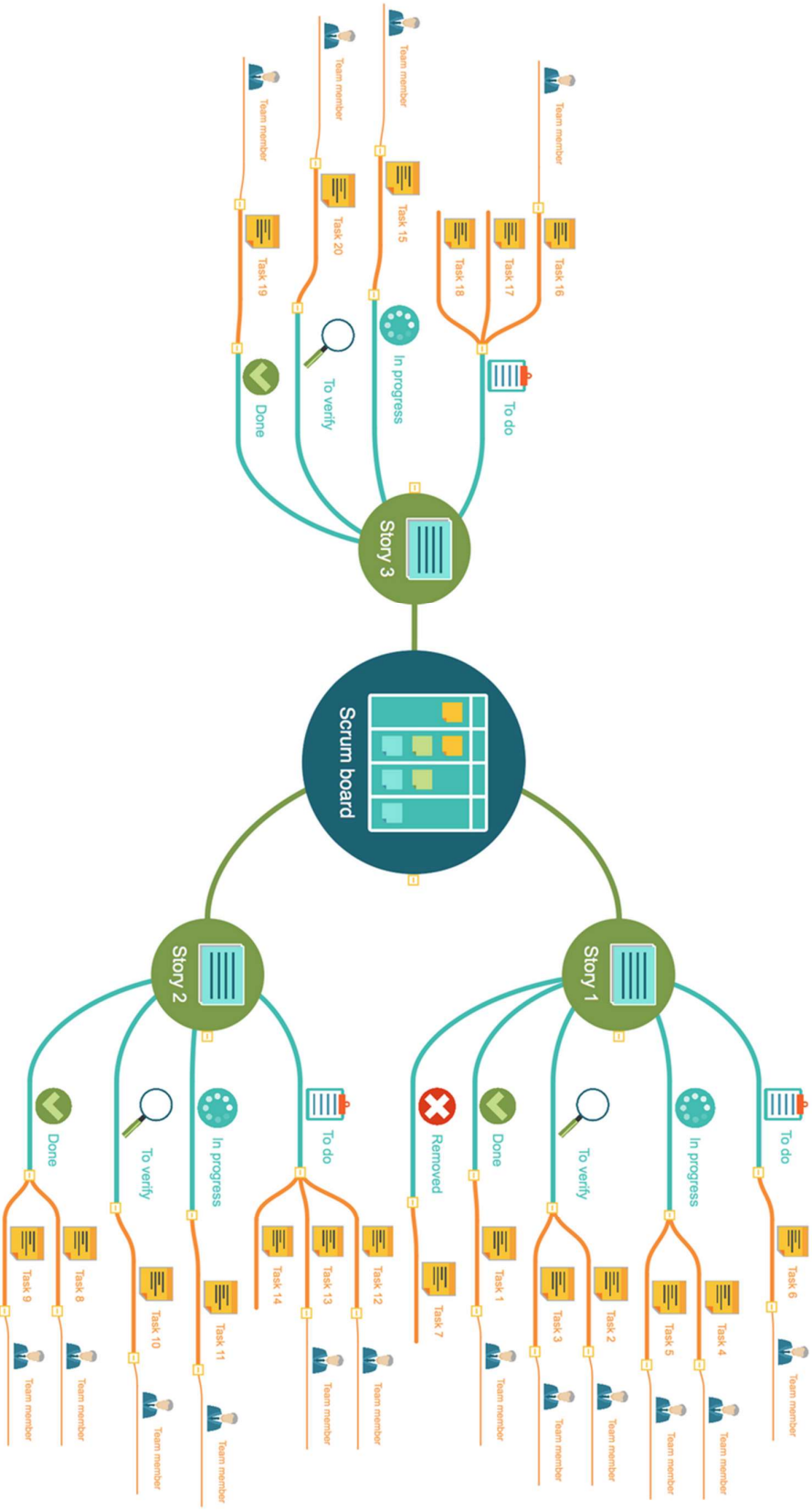
[28] **SCRUM Flow** [online]. [cit. 2019-03-06]. Dostupné z: https://www.conceptdraw.com/solution-park/project-management-scrum-workflow

8 Přílohy

8.1 Feature Driven Development Flow[27]

8.2 SCRUM Flow[28]




Scrum board
Story 1
To do
Task 6
Team member
In progress
Task 4
Team member
Task 5
Team member
To verify
Task 2
Team member
Task 3
Team member
Done
Task 1
Team member
Removed
Task 7
Story 2
To do
Task 12
Team member
Task 13
Team member
Task 14
In progress
Task 11
Team member
To verify
Task 10
Team member
Done
Task 8
Team member
Task 9
Team member
Story 3
To do
Task 16
Team member
Task 17
Task 18
In progress
Task 15
Team member
To verify
Task 20
Team member
Done
Task 19
Team member