VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Kvalita a měření softwarových procesů

Software Process Quality and Measurement

2020 Bc. Lukáš Stankovič

Zadání bakalářské práce

Jiří Dvorský

Ukázka sazby diplomové nebo bakalářské práce

Diploma Thesis Typesetting Demo

+++

Podpis vedoucího katedry

+++

Podpis děkana fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2016 .....

TODO

Abstrakt

Klíčová slova:

Abstract

Keywords:

Obsah

Seznam použitých zkratek a symbolů

ALM	–	Application lifecycle anagement
Automotive SPICE®	–	Automotive Software Process Improvement and Capability Determination
SIG	–	Automotive Special Interest Group
VDA QMC	–	Quality Management Center in the German Association of Automotive Industry
RUP	–	Rational Unified Process
BIRT	–	Business Intelligence and Reporting Tools
IDE	–	Integrated Development Environment
SVG	–	Scalable Vector Graphics
JDBC	–	Java Database Connectivity
XML	–	Extensible Markup Language
ODA	–	Open Data Access
SEI	–	Software Engineering Institute
CMM	–	Capability Maturity Model

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Při vývoji téměř jakéhokoli nejen softwarového produktu je potřeba se věnovat vývojovému procesu. Vývojový proces slouží k tomu, aby byl daný produkt vyvinut v určitých kvalitách a řízen správným směrem. Softwarový proces kromě standardizace řízení a plánování definuje také jednotlivé kroky k vytvoření softwarového produktu. [CT]. Vyvíjet s daným softwarovým procesem nám může zaručit, že produkt jež je vytvářen bude kvalitní a bude dokončen v definovaném termínu.

Během každého softwarového procesu je nutné jednotlivé kroky plánovat, monitorovat a na základě získaných dat vývoj uzpůsobovat. Monitorování procesu probíhá pomocí metrik, které mohou být reprezentovány buď pomocí tabulky nebo vizualizovány pomocí grafu. Veškeré sledování a měření daného procesu by bylo bez pomocných nástrojů velmi složité. Proto existuje nespočet nástrojů a aplikací, které v řízení pomůžou. Těmto aplikacím se dále v práci věnuji. V práci se zaměřuji především na metriky sběru požadavků.

Práce je rozdělena na teoretickou a praktickou část. V teoretické části se věnuji softwarovému inženýrství, softwarovým procesům a samotnému vývoji nejen softwaru, ale také v automobilovém vývoji pomocí standardu Automotive SPICE®. V práci se také věnuji jednotlivým typů procesu jako je vodopádový model nebo agilní přístupy. Dále rozebírám jednotlivé metriky a možnosti použití v různých procesech.

Cílem praktické části práce je analyzovat nástroje pro správu nasbíraných požadavků a jejich následnou vizualizaci pomocí metrik. V rámci práce jsem vypracoval vlastní aplikaci sloužící pro agregaci dat z IBM Jazz, které verzuji a následně vizualizuji pomocí uživatelsky definovaných metrik. Velkou výhodou je univerzálnost aplikace na zvoleném softwarovém procesu. Během vývoje aplikace jsem si vyzkoušel jednotlivé fáze vývoje. Věnoval jsem se například analýze požadavků a vytvořil jsem případy užití. Také jsem si pro jednotlivé části aplikace navrhl uživatelské rozhraní pomocí drátových modelů.

bude dopsáno + citace

2 Softwarový proces

Dle Iana Sommervilla [1] je softwarový proces sekvence souvisejících aktivit, které vedou k produkci softwarovécho produktu. Tyto aktivity mohou probíhat buď v rámci vývoj softwarového produkt od nuly nebo úpravě již existujícího softwarového produktu. Existuje nespočet různých softwarových procesů. Všechny softwarové procesy by měly obsahovat následující čtyři aktivity:

1. Specifikace - během této aktivity je nutné zjistit a definovat požadavky zákazníka na budoucí software.
2. Návrh a implementace softwaru - na základě specifikovaných zákaznických požadavků je software navrhnut a poté implementován.
3. Validace softwaru - je nutné software validovat, aby bylo zajištěno, že jeho funkcionalita odpovídá předem specifikovaným požadavkům.
4. Evoluce softwaru - software se během používání mění, aby vyhovoval novým potřebám zákazníka.

Softwarové procesy je nutné vždy používat na základě vyvíjeného produktu. V praxi neexistuje žádný softwarový proces, který by pokrýval a řešil veškeré problémy při vývoji softwaru. Pro zjednodušení softwarového procesu slouží různé softwarové modely. Ty se dělí na plánované, inkrementální a agilní. Při vývoji velkého systému se můžou jednotlivé modely softwarových procesů kombinovat. Vyspělost společnosti v používání softwarových procesů můžeme hodnotit podle stupnice SEI (Software Engineering Institute). Model hodnocení vyspělosti používání softwarového procesu dané společnosti se nazývá CMM (Capability Maturity Model), přičemž můžeme společnost hodnotit od 1 do 5 [2]:

1. Počáteční - společnost nepoužívá žádný softwarový proces. Jelikož žádný softwarový proces není definován tak každý projekt je vyvíjen případ od případu.
2. Opakovatelná - sledovaná společnost identifikovala ve svých projektech postupy, které se často opakují. Tyto prováděné opakované činnosti poté systémově opakuje v každém novém projektu.
3. Definovaná - používaný softwarový proces je definovaná a dokumentován na základě vysledovaných opakovatelných kroků.
4. Řízená - jakmile má daná společnost definovaný a dokumentovaný softwarový proces, je společnost schopna daný proces řídit a monitorovat.
5. Optimalizovaná - na základě dlouhodobě vysledovaných dat při používání softwarového procesu je společnost schopná daný softwarový proces optimalizovat a vylepšit tak, aby více vyhovoval celému vývoji.

2.1 Plánování

Je to jedna z částí softwarového procesu. Velikost a důkladnost plánování procesu a samotného projektu závisí na používaném modelu softwarového procesu. Například při použití vodopádového modelu je plánování velmi důležitá část procesu, jelikož na kvalitní specifikaci a plánování kompletního procesu stojí celý vývoj. Naopak při vývoji pomocí agilních způsobů je část plánování a specifikace softwaru minimální.

Během plánování probíhá kromě plánování samotného procesu také specifikace výsledného softwaru. Specifikace vyvíjeného softwaru je část procesu, během které dokážeme určit a definovat, které vlastnosti budou od systému požadovány. Vlastnosti a požadavky na systém vzniknou sběrem požadavků od zákazníka, který má na vznikající software určité nároky. Tento proces bývá také nazýván inženýring požadavků. Bývá jednou z nejkritičtějších částí, jelikož chybná specifikace jednotlivých požadavků v této fázi vedou k pozdějším problémům jak při samotném návrhu a implementaci, tak i při pozdější správě.

Cílem inženýringu požadavků je získat od zákazníka co nejvíce požadavků, které na budoucí software klade. Na základě nasbíraných požadavků vznikne dokument schválených požadavků pro implementaci a následnou dokumentaci celého softwaru. Dále na základě získaných požadavků je vytvořen plán celého vývoje softwaru. U vodopádového modelu inženýring požadavků probíhá na začátku vývoje a je velmi detailní. Při agilních a iterativních způsobech vývoje probíhá sběr požadavků vždy před každou iterací.

2.2 Monitorování

Další důležitou částí každého softwarového procesu je jeho monitorování. Během monitorování zjišťujeme, zda proces probíhá tak jak jsme ho plánovali a také, zda neexistují možnosti, jak proces zlepšit a tím dosáhnout lepších výsledků. Monitorování je nepřetržitý proces, který také kromě zlepšování procesu zahrnuje sledování, aby bylo možné identifikovat potenciální problémy dostatečně včas, než způsobí závažný problém. Samotné monitorování je často prováděno současně s kontrolou a uzpůsobením daného procesu (viz kapitola 2.3).

Hlavním důvodem pro monitorování procesu je, že výkonnost a daného procesu je důkladně sledován. Díky tomu je možné rychle identifikovat odchylky od původního plánovaného plánu daného projektu. Během monitorování také sledujeme, zda se plní nadefinované požadavky a jak rychle. K tomuto účelu mohou sloužit metriky (viz kapitola 3).

2.3 Uzpůsobení

Proces bychom měli pravidelně uzpůsobovat potřebám, které se mohou v rámci vývoje produktu měnit. Podměty ke změnám daného softwarového procesu získáváme z monitorování procesu. V rámci uzpůsobení řešíme také, zda je projekt stihnutelný v definovaném termínu. Tuto informaci nejčastěji zjistíme pomocí vytvořených metrik.

Samotné uzpůsobení procesu může probíhat několikrát během celého vývoje. Dále během uzpůsobení provádíme validaci celého procesu a řešíme, zda není možné někde dělat nějakou část lépe a efektivněji. V rámci uzpůsobení se také validuje, zda v daném procesu postupujeme správně a zda probíhá vše tak, jak bylo v rámci procesu definováno. V rámci uzpůsobení a zlepšování procesu můžeme použít několik konceptů.

Zde si nejsem jistý jestli to zde ty koncepty můžu zařadit?

2.3.1 CMM a CMMI

2.3.2 PDCA

2.3.3 SEL/NASA

bude učesáno + ověřeno že to je pravda...

2.4 Sekvenční a iterativní přístupy

2.4.1 Vodopádový model

Jde o jeden z nejstarších vývojových procesů, který byl poprvé publikován v článku v roce 1970 Winstonem W. Roycem a vychází z obecných vývojových procesů. Vodopádovým modelem je nazýván, protože jednotlivé fáze procesu postupně kaskádovitě navazují na sebe a celý model poté vypadá jako vodopád. Veškeré kroky musí být pečlivě analyzovány a naplánovány. K další vývojové fázi se přistupuje až poté, co jsou veškeré předchozí kroky splněny, schváleny a uzavřeny. [5] Aby byl projekt podle modelu úspěšný je nutné věnovat prvním fázím dostatek času.

Základní fáze vodopádového modelu vychází ze standardních potřeb vývoje softwaru:

1. Specifikace požadavků - na základě jednotlivých požadavků zákazníka jsou požadavky detailně sepsány a analyzovány.

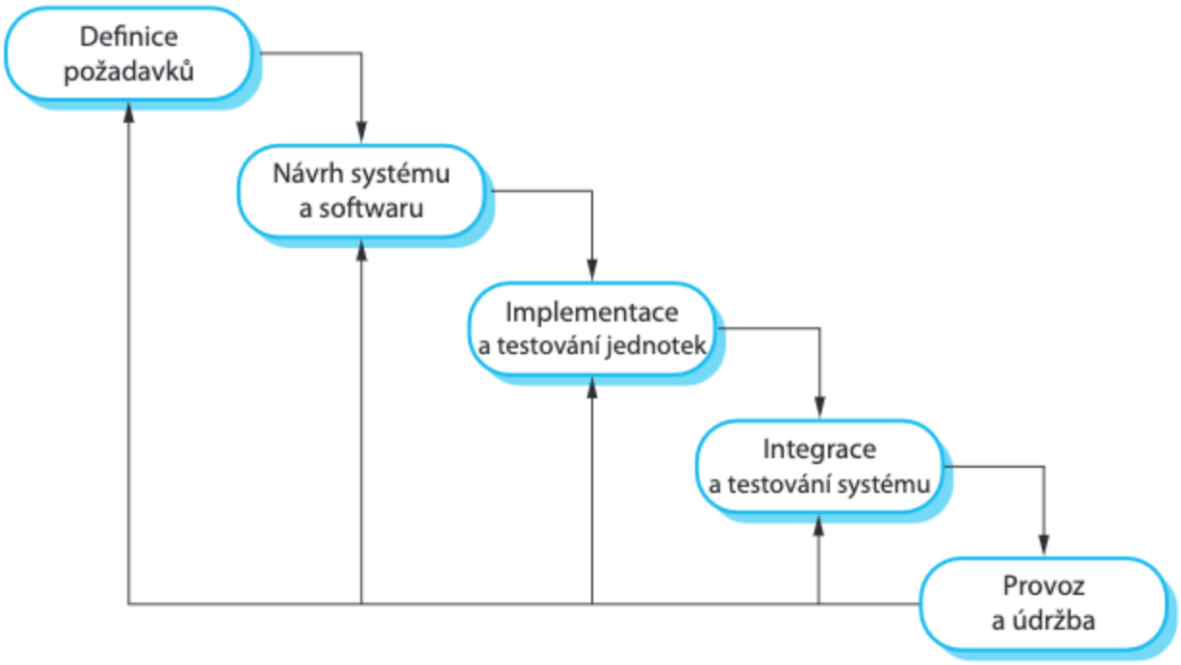
2. Systémový a softwarový návrh - jednotlivé požadavky zákazníka určují náročnost na systém. Návrh zahrnuje také celkovou architekturu systému.

3. Implementace a unit testování - v rámci této fáze se systém realizuje a každá jednotka je samostatně testována, zda splňuje určité požadavky.

4. Integrace a testování systému - jednotlivé vyvinuté části systému se integrují a otestují jako kompletní systém. Tímto je zajištěno, že celý systém splňuje definované požadavky. Jakmile je produkt kompletně otestován, je předán zákazníkovi.

5. Provoz a údržba - během této fáze se systém nasazuje do ostrého prostředí. Během fáze probíhá oprava chyb a vylepšování funkcí.

V praxi je složité jednotlivé fáze pečlivě oddělit a postupovat v další fáze pouze jakmile je předchozí fáze plně dokončena. Proto se většinou jednotlivé fáze postupně překrývají a vyměňují



Obrázek 1: Vodopádový model (převzato z [5])

si mezi sebou informace. Hlavní nevýhoda modelu je praktická nemožnost reagovat na měnící se požadavky zákazníka. Vodopádový model je vhodné zpravidla použít tam, kde víme, že se požadavky zásadně nebudou měnit.

2.4.2 Rational Unified Process

Rational Unified Process (RUP) je iterativní model vývoje softwaru vytvořený firmou Rational Software Corporation. RUP byl vytvořen a odvozen z osvědčených praktik při vývoji softwaru. Mezi tyto praktiky například patří řízení změn, komponentová architektura, aktivní správa požadavků nebo například ověřování kvality software. Model obsahuje čtyři fáze, které se iterují dle potřeby. Jednotlivé fáze oproti vodopádovému modelu nesouvisejí s technickými hledisky ale spíše se soustředí na podniková hlediska. [6] Fáze jsou následující:

1. Zahájení - cíl této fáze je vytvořit podnikový případ systému. V rámci této fáze je potřeba identifikovat veškeré osoby a další systémy které budou s daným systémem pracovat či komunikovat.

2. Příprava - během této fáze se vytvoří architektura systému. Dále se vytvoří celkový plán projektu a identifikují se klíčová rizika, která mohou na projektu nastat. Během fáze vznikne například plán vývoje, UML diagramy nebo popis systému.

3. Konstrukce - tato fáze obsahuje návrh systému, samotné programování a testování, přičemž se zároveň jednotlivé části systému integrují. Dále během této fáze vzniká dokumentace. Na konci fáze je hotový funkční produkt připravený na dodání zákazníkovi.

4. Předávání - v závěrečné fázi je funkční produkt spolu s dokumentací předán zákazníkovi. Produkt je nasazen do ostrého prostředí a zprovozněn.

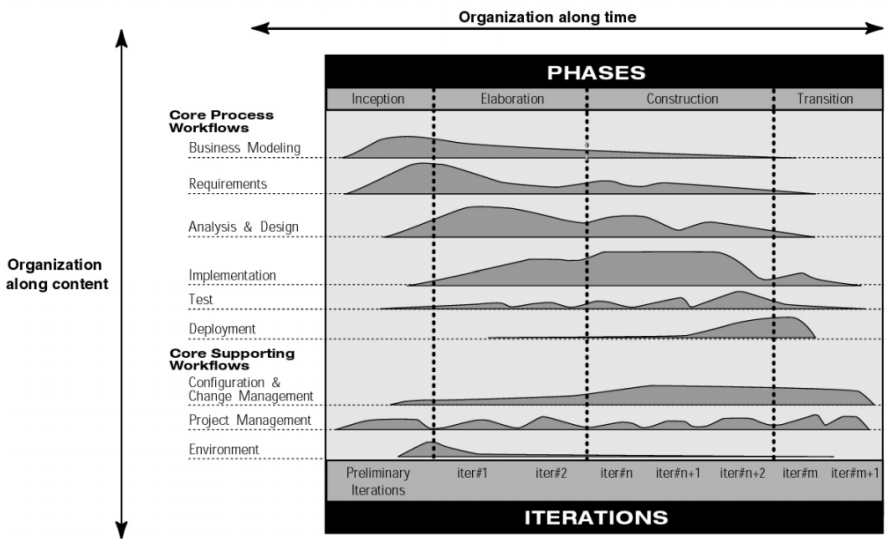
U modelu RUP je možné provádět inkrementaci jednak samostatným opakováním jednotlivých fází ale i celého procesu. Kromě čtyřech fází RUP definuje jednotlivé disciplíny, které jsou v rámci fázích prováděny. Velkou výhodou je, že tyto disciplíny nejsou navázány na jednotlivé fáze. Disciplíny jsou činnosti, prováděné během celého vývoje produktu. Disciplíny jsou následující:

- Tvorba modelu
- Správa požadavků
- Analýza a návrh
- Implementace
- Testování
- Nasazení
- Správa konfigurace a změn
- Řízení projektu
- Správa prostředí

Výše popsané fáze společně s disciplínami je možné vidět na obrázku 2. Vidíme, že různé disciplíny mají v jednotlivých fázích vyšší důležitost než jiné.

2.5 Agilní přístupy

Aktuální trh se softwarem se rychle mění a proto je potřeba reagovat na měnící se prostředí. Agilní přístupy patří k inkrementálním způsobům vývoje. Software vytvářený pomocí vodopádového modelu může být v době vydání již velmi zastaralý. Z tohoto důvodu jsou agilní přístupy vývoje v poslední době velmi oblíbené. Takto vyvíjený software je většinou dodán rychleji, ale za to může být dodán s horší kvalitou. Takový přístup se ovšem nehodí u kritických systémů, u kterých je kladen velký důraz například na bezpečnost a spolehlivost. Software se během agilního vývoje vytváří ve verzích, kdy každá nová verze obsahuje nové funkcionality. Jednotlivé vývojové procesy se prolínají a většinou neexistuje podrobná specifikace a dokumentace systému. V dokumentaci obvykle bývají pouze základní a nejdůležitější prvky systému. Tento nedostatek se může ovšem rychle vymstít při následné údržbě daného softwaru.



Obrázek 2: Grafické znázornění fází a disciplín RUP (převzato z [7])

Filozofie agilních metodik vychází z manifestu, na kterém se shodli vývojáři těchto vývojových metod. Manifest popisuje čtyři základní hodnoty, kterých si autoři cení [12]:

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

Každý vývojový proces má svá pozitiva a svá negativa. Mezi pozitiva při agilním vývoji patří například rychlejší vývoj, větší flexibilita vývoje, kdy je možné upravovat směr dalšího vývoje dle potřeby zákazníka nebo například zapojení zákazníka do procesu vývoje. Mezi negativa patří nekompletní dokumentace, závislost na vývojovém týmu, kdy při rozpadu týmu může dojít k problémům. Další problém může být, že v dnešní době chce každá firma vyvíjet agilně, ikdyž na to není personál proškolený a nikdo z dané firmy pořádně neví co to znamená být agilní.

Podle Iana Sommervilla [13] jsou agilní metody úspěšné pouze pro určité typy softwarových systémů:

1. Vývoj malého nebo středně velkého systému, který je určený k prodeji.
2. Vývoj systémů pro zákazníka, kde si zákazník uvědomuje nutnost zapojit se do procesu vývoje a kde neexistuje mnoho pravidel, které mají vliv na vývoj softwaru.

2.5.1 Extrémní programování

Dle Kenta Becka [8] je extrémní programování agilní metoda, která spočívá o změně myšlení a zvyků při vývoji. Extrémní programování se zaměřuje na excelentní programovací schopnosti, zlepšení komunikaci mezi členy týmu a týmovou práci. Extrémní programování se zaměřuje na zextrémění a zefektivění těchto úkonů, které se při vývoji osvědčily. Je důležité z extrémního programování do procesu zařadit pouze ty aktivity, které mají hodnotu pro vývojový tým a které přinášejí benefit zákazníkovi. Můžeme zde například zařadit párové programování, kdy dva programátoři vyvíjejí společně jednu funkčnost, přičemž jeden programátor pozoruje druhého při programování. Tyto role si pravidelně střídají. Nevýhoda tohoto přístupu je, že takto vyvíjený software zabere více času. Výhoda je naopak v tom, že software obsahuje méně chyb díky vzájemné revizi kódu.

Další obor, který může extrémní programování řešit extrémnější testování softwaru, kdy samotní programátoři programují jednotkové (unit) testy, ale zároveň testuje také zákazník pomocí akceptačních testů. Extrémní programování takto řeší prakticky každou oblast vývoje, která se při vývoji osvědčila a může být zefektivněna. Jednotlivé zadání úkolů jsou řešena stejně jako v metodě Scrum pomocí uživatelských příběhů (takzvané *User stories*). Další možná praktika, kterou extrémní programování řeší je například zakázník na pracovišti, vydávání malých verzí softwaru, průběžná integrace anebo neustálá refaktorizace kódu.

2.5.2 Scrum

Dle Josefa Myslína [9] je to agilní technika, při které se více než technické procesy řeší projektové procesy. Vymysleli ji Ken Schwaber a Jeff Sutherland na začátku 90. let 20. století. Týmy, které ve Srumu pracují jsou samoorganizované, přičemž vývojový tým nemá přímého nadřízeného. Tým obsahuje několik klíčových rolí, mezi které patří *Product owner* (vlastník produktu), který komunikuje se zákazníkem a případně prioritizuje práci, dále *Scrum master*, který má na starost samotný proces a interakci s týmem a samotný vývojový tým. Scrum stojí na třech základních pilířích:

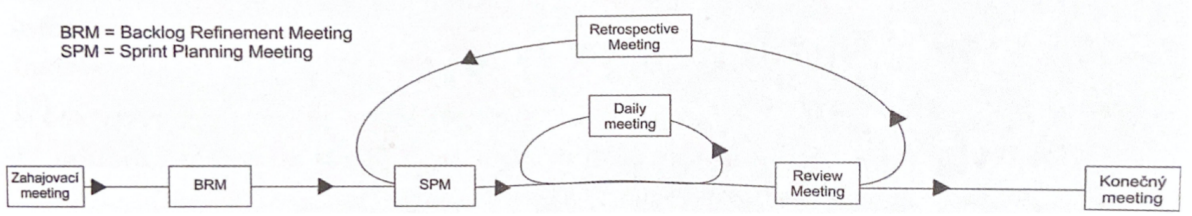
1. Transparentnost - všichni v týmu, by měli mít přehled o veškerém dění v rámci projektu. Každý člen týmu tedy například ví v jakém stavu je určitý úkol a proč existuje.

2. Kontrola - pravidelně se v rámci celého týmu dělá kontrola, zda procesy a fungování týmu jsou vyhovující.

3. Adaptace - na základě zjištěných požadavků v rámci pravidelných kontrol se provádějí změny, aby byl daný tým spokojený a doručoval dále výsledky.

Veškerý vývoj produktu probíhá v pravidelných iteracích nazývané *Sprinty*. Tyto iterace mají nejčastěji trvání od dvou týdnů po měsíc. Na konci každého sprintu je dodána nová verze funkčního softwaru.

Jak je možné vidět na obrázku 3 na začátku každého sprintu probíhá seznámení s úkoly na daný sprint a plánování, které úkoly musí tým daný sprint doručit. Během sprintu probíhají různé druhy schůzek. Každý den probíhá rychlá schůzka *Standup*, během které v rychlosti členové týmu představí co budou daný den řešit. Na konci každého sprintu probíhá pravidelná kontrola, která se nazývá retrospektiva a dále samotné zhodnocení sprintu. Na této schůzce je řešeno jaké adaptace by měl tým provést, aby následující sprint proběhl lépe.



Obrázek 3: Vývojový cyklus řízený pomocí metodiky Scrum (převzato z [10])

Jednotlivé schůzky by se poté daly rozdělit do tří základních skupin:

- Plánovací – tento typ schůzek předchází jakékoli aktivitě, která po schůzce probíhá. Během těchto schůzek může být plánován buď celý projekt nebo naplánován sprint.
- Hodnotící – tyto schůzky probíhají buď na konci sprintu, nebo na konci projektu. Během tohoto typu schůzek se hodnotí, zda to co proběhlo, proběhlo v pořádku a případně, co bylo špatně.
- Plánovací i hodnotící – do této skupiny spadá pouze denní schůzka zvaná *Standup*. Během schůzky se jednak hodnotí předchozí den a plánuje se co se bude dále daný den dělat. Během schůzky mají všichni členové týmu možnost vidět, jak se úkol hýbe, nebo nehýbe.

Jednotlivé zadání úkolů jsou psány pomocí uživatelských příběhů (takzvané *User stories*), přičemž většinou obsahují jen minimum technických informací. Jednotlivé uživatelské příběhy nepopisují interakci ze systémem. Uživatelské příběhy mají sjednocenou podobu například ve tvaru: „Jako *role* chci *cíl* tak, aby *užitek*.“ Tedy například: „Jako uživatel chci vidět seznam zboží v objednávce, tak aby si uživatel mohl zboží zkontrolovat.“ [11]

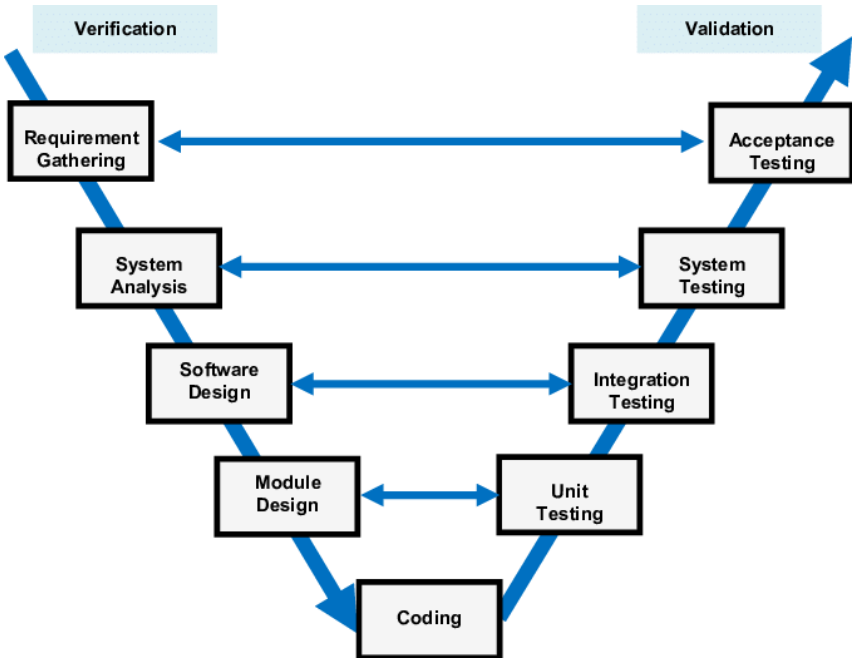
2.6 V-Model

V-model je grafické znázornění vývojového cyklu prakticky jakéhokoliv vývoje u kterého je důležité nepřetržité fungování a bezchybnost systému. Model je tedy využíván prakticky v každém odvětví, nejen v automobilovém průmyslu nebo softwarovém vývoji kritických systémů. Tento model zdědil po vodopádovém modelu jistou kaskádovitost, kdy každý krok je proveden až po předchozím kroku. Za rozdílnost V-Modelu lze považovat to, že je oproti agilním přístupům zaměřen na velmi důkladnou kontrolu a testování systému. Kontrola a testování probíhá již v počátečních etapách návrhu systému. Jednotlivé testování systému se provádí ve stejné době

jako určitá část vývoje. Tedy například jednotkové testy se píšou během implementační části nebo například integrační testování probíhá při integraci jednotlivých komponent systému. Je vhodný zejména pro projekty, kde je většina požadavků jasná a předem daná. Grafické znázornění V-Modelu je možné vidět na obrázku 4.

V-Model je graficky definován do tvaru velkého písmene V. Na levé straně modelu se nachází fáze spojené s verifikací. První fází v rámci verifikační části je sběr požadavků, ve kterém jsou od zákazníka získány požadavky. Tyto požadavky jsou následně analyzovány. V další fázi se provede na základě těchto nasbíraných požadavků návrh systému. Dále následuje vytvoření architektury systému a návrh jednotlivých modulů. Poté je na základě veškeré analýzy provedena implementace.

Na pravé straně jsou fáze validační, ve kterých se produkt testuje a validuje jeho funkčnost. Jednotlivé fáze jsou vzájemně propojené. Na nejnižší vrstvě probíhá jednotkové testování, které testuje chování malých modulů. Dále probíhá integrační testování, které je prováděno proti návrhu architektury. Jakmile je úspěšně provedeno integrační testování, tak je testován návrh systému systémovými testy. Na nejvyšší úrovní probíhá akceptační testování, které ověřuje, zda jsou správně naimplementovány veškeré definované požadavky zákazníka. Každý přechod do jiné fáze probíhá jakmile je kompletně splněna předchozí fáze.



Obrázek 4: Grafické znázornění V-Modelu (převzato z [14])

3 Metriky

Jelikož v rámci praktické části vytvářím nástroj pro metriky, je nutné se nejprve seznámit se samotnou teorií metrik a měření vývoje. Metriky se můžou používat nejen při samotném vývoji požadavků, ale i při vývoji systému.

BUDE DOPSÁNO

3.1 Reprezentace a vizualizace metrik

Bude napsáno o tom jak můžou metriky vypadat (tabulka, graf, ...) a jak je lze vizualizovat (birt + odkaz na praktickou část)

3.1.1 BIRT

BIRT je open source softwarový projekt vyvíjený konsorciem Eclipse Foundation a původně sponzorovaná společností Actuate s pomocí společnosti IBM, který poskytuje platformu pro vytváření vizualizací dat. Je určený zejména pro webové a klientské aplikace, které jsou založených na prostředí Java a Java EE. BIRT se skládá ze dvou hlavních komponent [3]:

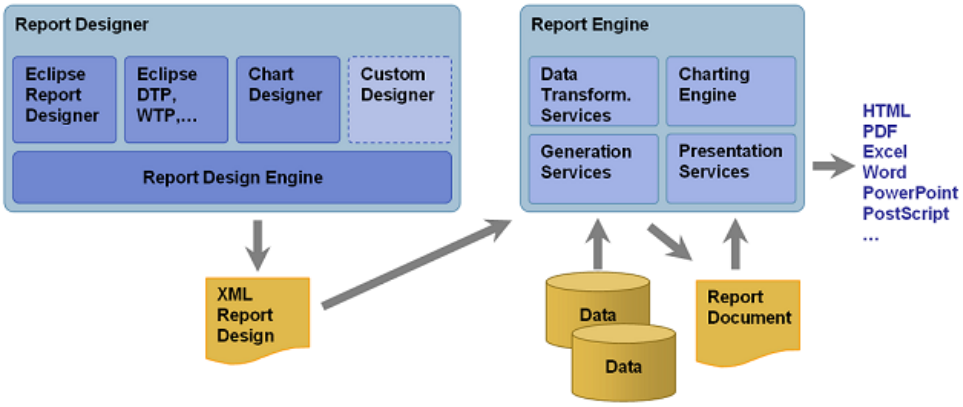
- vizuální návrhář sestav v IDE Eclipse,
- komponenta pro generování navržených sestav, které lze nasadit do libovolného Java prostředí.

Projekt BIRT také obsahuje mapovací modul, který je plně integrován do návrháře sestav a může být použit samostatně k integraci grafů do aplikace. Pomocí technologie BIRT je výstupy možné generovat kromě grafů ve formátu SVG, také jako seznamy dat (například pomocí tabulky) nebo jako textové dokumenty. Sestavy BIRT se skládají ze čtyř hlavních částí [3]:

- Data - poskytuje podporu pro přístup k datům pomocí JDBC, XML, webových služeb prostých databázových souborů. Vytvořená sestava může dále zobrazovat data z libovolného počtu zdrojů dat. BIRT využívá ODA (Open Data Access), který umožňuje komukoli vytvořit novou sestavu pro jakýkoli druh tabulkových dat.
- Transformace dat - vygenerované reporty reprezentují shrnutá, seřazená, filtrovaná a seskupená data podle definovaných potřeb uživatele. Tato data jsou poté připravena pro vizualizaci.
- Obchodní logika - jelikož jsou data v reálném světě málokdy strukturovaná přesně tak, aby vyhovovaly z byznysového hlediska pro jednotlivé reporty, je nutné je nejprve strukturovat tak, aby pro transformaci a vizualizaci vyhovovala. Tato strukturalizace je možná buď přímo pomocí BIRT v jazyku JavaScript, nebo externě pomocí zdrojového kódu v programovacím jazyku Java.

- Vizualizace - Jakmile jsou data pro vizualizaci připraveny je možné je vizualizovat například pomocí grafů nebo tabulek. Jedna připravená sada dat může být vizualizována více způsoby.

Architektura BIRT se skládá z *Report Designeru*, který zajišťuje vytváření reportů, které vygeneruje do formátu XML. Dále se skládá z *Report enginu*, který vygeneruje jednotlivé vizualizace dat do definovaných formátů, jako například PDF, HTML, Excel, Word a další. Celkovou architekturu systému BIRT je možné vidět na obrázku 5.



Obrázek 5: Architektura BIRT (převzato z [3])

4 Automotive SPICE

Automotive Software Process Improvement and Capability Determination (zkráceně ASPICE) je standard, používaný v automobilovém průmyslu pro vývoj mechatronických a softwarových systémů v automobilovém průmyslu. Vychází z obecného standardu SPICE (ISO/IEC 15504), který se věnuje obecnému vývoji softwaru. Od vytvoření v roce 2005 je Automotive SPICE® oborovou variantou této normy. Standard je volně přístupný na webových stránkách v několika světových jazycích [15]. Poslední dostupná verze je verze 3.1 z roku 2017. Tato verze obsahuje opravu drobných jazykových chyb a vychází z verze 3.0 [16].

Standard Automotive SPICE® byl vyvinut v rámci spolupráce evropských automobilových společností. Ty se spojily do sdružení Automotive Special Interest Group (SIG). Dále je standard aktualizován skupinou Quality Management Center in the German Association of Automotive Industry (VDA QMC). Standard je tímto sdružením pravidelně aktualizován a upravován, aby vyhovoval nejnovějším požadavkům. Mezi evropské automobilky spojené do sdružení SIG patří například:

- BMW Group,
- Daimler AG,
- Jaguar,
- Volkswagen AG,
- Volvo Car Corporation a další.

V současné době se automobily skládají z velkého množství součástí. Automobily se již neskládají pouze z mechanických součástí. V poslední době roste množství používaného softwaru nejen při vývoji ale i v samotných automobilech. Většina softwaru v automobilu slouží také pro zajištění nejen bezpečnostních funkcí, ale také pro zajištění životně důležitých funkcí automobilu včetně brzd a samotného řízení. Nesmí se tedy stát, že software v automobilu selže a například zablokuje brzdy a způsobí havárii. Dodržování tohoto standardu je rozhodující kritérium při volbě dodavatelů dané automobilové společnosti, jelikož zajišťuje že dodavatel splňuje určité standardy vývoje.

V rámci vývoje v automobilovém průmyslu je většina systémů vyvíjena pomocí V-Modelu. Každá část vývojového procesu musí mít definovány cíle. Aby bylo možné určit, zda jsou cíle splněny jsou používány metriky (viz kapitola 3).

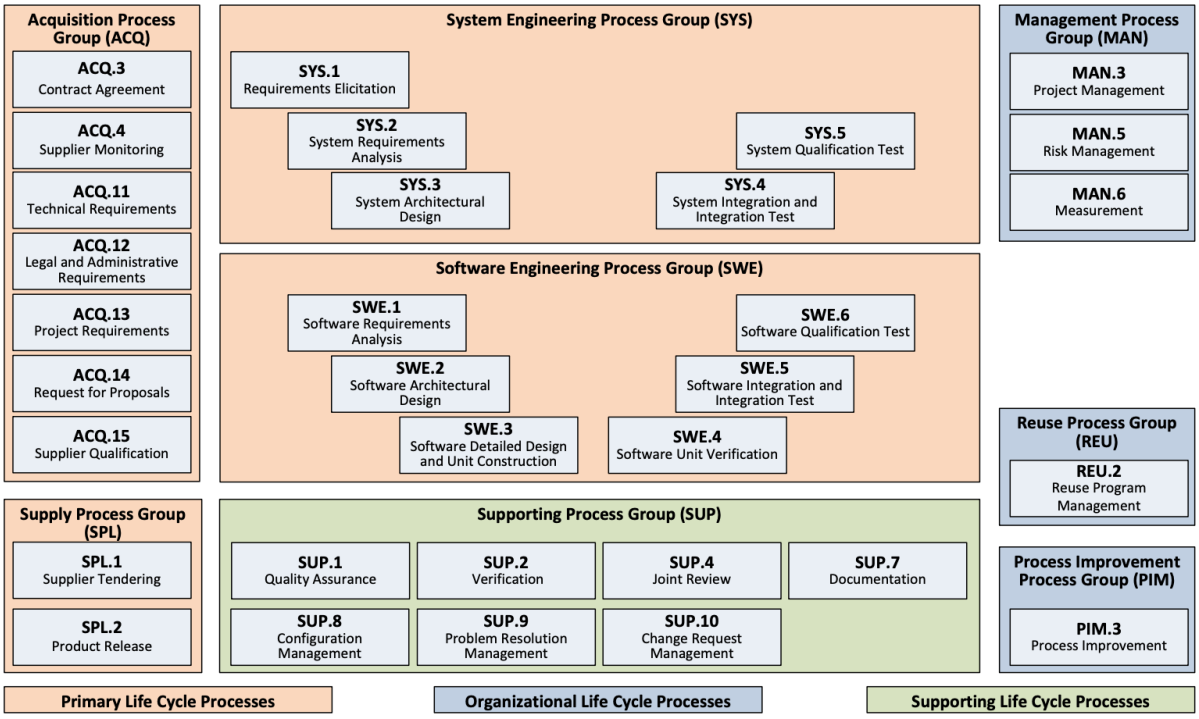
4.1 Referenční procesní model

Je to jeden z procesních modelů v rámci standardu Automotive SPICE®. Každý proces obsahuje kromě popisu také funkční cíle v daném prostředí a seznam výsledků a výstupů, které má proces dosáhnout. [17]

Referenční procesní model definuje několik procesů ve třech základních kategoriích:

- primární procesy,
- podpůrné procesy,
- organizační procesy.

Jednotlivé kategorie procesů a samotné procesy jsou detailně probrány v kapitole 4.2 a je možné je graficky seskupené vidět na obrázku 7. V rámci každé základní kategorie existují podkategorie procesů, dle toho jakým okruhem činností se zaobírají. Procesy jsou pojmenovány pomocí třímístného identifikátoru, která určuje podkategorii procesu. Dále název procesu obsahuje číslo, které identifikuje proces v rámci své podkategorie.



Obrázek 6: Referenční procesní model (převzato z [18])

4.2 Procesy Automotive SPICE

Zde bude obecný popis a úvod do procesů.

4.2.1 Primární procesy

Zde budou popsány: ACQ, SPL, SYS, SWE

4.2.2 Podpůrné procesy

Zde budou popsány: SUP

4.2.3 Organizační procesy

Zde budou popsány: MAN, PIM, REU

4.3 Hodnotící framework

Hodnotící framework poskytuje nezbytné požadavky a pravidla pro zjištění jakým způsobem se dané procesy v rámci projektu používají. Poskytuje schéma a návod, které umožňuje hodnotiteli určit úroveň schopnosti procesu, který daná firma u projektu používá. Tyto úrovně jsou definovány v rámci hodnotícího frameworku pomocí procesních atributů. Jednotlivým úrovním se věnuji v následujících kapitolách. Dané procesní atributy jsou přiřazené jednotlivým úrovním. Míra dosažení určitého procesního atributu je reprezentována pomocí hodnocení na základě definované hodnotící stupnice. Konečnou úroveň rozhodne na základě pravidel hodnotitel. Pravidla jsou reprezentována jako atributy které musí daný proces obsahovat. Automotive SPICE® pro toto používá standard ISO/IEC 33020. [19]

4.3.1 Úrovně způsobilosti procesu a atributy procesu

Jak již bylo zmíněno v předchozí kapitole, Automotive SPICE® používá pro úrovně způsobilosti standard ISO/IEC 3302. Procesní atributy jsou vlastnosti procesu, které lze hodnotit na stupnici 0 až 5 a poskytují měřítko schopnosti procesu. Úrovně způsobilosti jsou použitelné pro všechny druhy procesů. Jednotlivé dodržení úrovně způsobilosti hodnotí hodnotitel na základě dodržení definovaných procesních atributů pro každou úroveň. Každá vyšší úroveň zároveň obsahuje podmínku pro 100% splnění předchozí úrovně.

- Úroveň 0: Neúplný proces – proces na této úrovni není vůbec implementován a definován nebo nedosahuje cíle. Patří zde také procesy, které dodržují procesní atributy, které ASPICE definuje nejvýše na úroveň *částečně dosaženo* (P) (viz kapitola 4.3.2).
- Úroveň 1: Vykonávaný proces – implementovaný proces na této úrovni plní svůj procesní účel. Během procesu projde vývoj celým V modelem, ale budou v něm mezery a nedokonalosti. K dosažení této úrovně musí být splněn atribut PA 1.1 – výkonnostní atribut procesu.
- Úroveň 2: Řízený proces – Vykonávaný proces je nyní implementován v řízeném prostředí. Proces je plánován, monitorován a uzpůsoben dle požadavků. Vyvinuté produkty jsou náležitě zavedeny, kontrolovány a udržovány. V rámci této úrovně musí být splněny procesní atributy PA 2.1 – řízení výkonu a PA 2.2 - řízení pracovního produktu.

- Úroveň 3: Zavedený proces – Dříve popsaný řízený proces je nyní implementován pomocí definovaného procesu, který je schopen dosáhnout svých výsledků procesu. Společnost, která má proces na této úrovní má centralizované standardy, jak pracuje, a daný projekt se těmito standardy řídí. Na úrovni 3 je nutné splnit procesní atributy PA 3.1 – definice procesu a PA 3.2 – procesní nasazení.
- Úroveň 4: Předvídatelný proces – Dříve popsaný zavedený proces nyní pracuje prediktivně v definovaných mezích, aby dosáhl svých procesních výsledků. Úroveň 4 zahrnuje procesní atributy PA 4.1 – kvantitativní analýza a PA 4.2 – kvantitativní kontrola.
- Úroveň 5: Optimalizovaný proces – Výše popsaný předvídatelný proces je nyní neustále vylepšován, aby reagoval na změny v dané společnosti. Pátá úroveň zahrnuje dva procesní atributy PA 5.1 – inovace procesu a PA 5.2 – implementace procesní inovace.

4.3.2 Hodnocení procesních atributů

Aby mohl proškolený hodnotitel hodnotit úroveň způsobilosti procesu v určité společnosti, určuje Automotive SPICE® hodnotící stupnici, která pochází ze standardu ISO/IEC 33020. Definované procesní atributy jsou hodnoceny na základě splnění podmínek, které určuje daný proces. Rozšířená stupnice pro hodnocení procesních je možné vidět v tabulce 1.

4.3.3 Model úrovně způsobilosti procesu

Úroveň způsobilosti procesu určí hodnotitel na základě hodnocení procesních atributů. Model definuje pravidla, na základě kterých je úroveň určena. Proces může úrovně dosáhnout pouze pokud jsou procesní atributy dané úrovně ohodnoceny alespoň L (téměř dosaženo) a zároveň pokud jsou veškeré nižší úrovně procesních atributů splněny na F (plně dosaženo).

Hodnocení		Úroveň dosažení	Popis
N	Nedosaženo	dosažení 0 % až 15 %	V projektu existují malé nebo žádné důkazy o dosažení definovaných procesních atributů v posuzovaném procesu.
P-	Částečně nedosaženo	dosažení 15 % až 32,5 %	V rámci projektu existuje několik důkazů o dosažení procesního atributu v posuzovaném procesu. Mnoho aspektů k dosažení procesního atributu můžou být nepředvídatelné.
P+	Částečně nedosaženo	dosažení 32,5 % až 50 %	V rámci projektu existuje několik důkazů o dosažení procesního atributu v posuzovaném procesu. Některé aspekty k dosažení procesního atributu můžou být nepředvídatelné.
L-	Téměř dosaženo	dosažení 50 % až 67,5 %	V projektu se systematicky přistupuje k definovanému procesnímu atributu a jeho významném dosažení. V posuzovaném procesu může existovat mnoho slabin souvisejících s tímto procesním atributem.
L+	Téměř dosaženo	dosažení 67,5 % až 85 %	V projektu se systematicky přistupuje k definovanému procesnímu atributu a jeho významném dosažení. V posuzovaném procesu mohou existovat některé slabiny související s tímto procesním atributem.
F	Plně dosaženo	dosažení 85 % až 100 %	V projektu se úplně a systematicky přistupuje k definovanému procesnímu atributu a jeho úplném dosažení v posuzovaném procesu. V posuzovaném procesu neexistují žádné významné slabiny související s tímto procesním atributem.

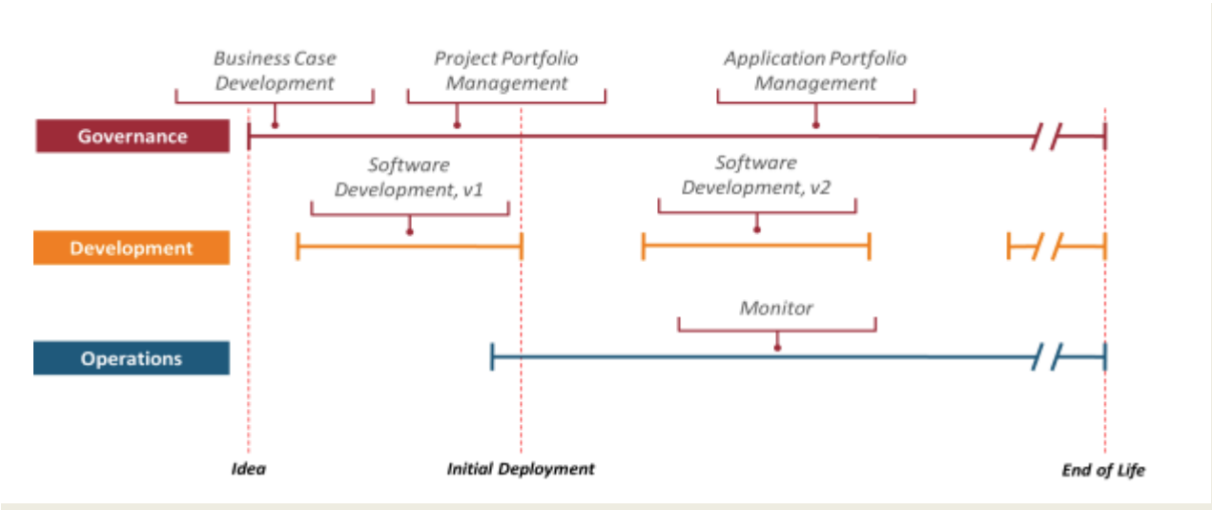
Tabulka 1: Rozšířená stupnice hodnocení procesu v ASPICE dle ISO/IEC 33020 [19]

5 Application Lifecycle Management

Application Lifecycle Management není pouze proces samotného vývoje a programování softwaru. Je to možné přirovnat k životě člověka, kdy jde o kompletní proces, začínajíc od první myšlenky a požadavku zákazníka zákazníka, přes samotný vývoj softwaru a nasazení do produkčního prostředí. Application Lifecycle Management končí až jakmile končí samotný software, kdy například přestane dávat z byznysového hlediska smysl. [4]

Application Lifecycle Management je možné rozdělit na tři základní aspekty, které je možné vidět na obrázku 7:

- řízení,
- vývoj,
- provoz.



Obrázek 7: Aspkety ALM a jejich fáze [4]

Proces, který probíhá během celého aplikačního cyklu je řízení. Vzniká počáteční myšlenkou zákazníka a pokračuje až do konce života softwaru. Během této činnosti jsou prováděny rozhodovací procesy, plánování, sběr požadavků a samotné řízení projektu. Je to nejdůležitější aspekt celého aplikačního cyklu. Pakliže by tento proces nebyl správně dodržován, nemusí být naplněna byznysová hodnota daného softwaru. [4] První fází, kterou proces začíná je *Business Case Development*, ve kterém je řešen sběr požadavků zákazníka a vytvoření základní koncepce vývoje. Poté se v rámci fáze *Project Portfolio Managmentu* řeší procesy spojené s organizací vývoje a nasazení do produkčního prostředí. V rámci poslední fáze, která probíhá po nasazení do produkčního prostředí má manažer na starosti kontrolovat, zda daný software splňuje požadavky zákazníka.

Proces vývoje začíná jakmile je sběr požadavků dokončen a funkčnosti jsou specifikované. Hlavní část procesu probíhá až do nasazení softwaru do produkčního prostředí. Po nasazení do produkčního prostředí vývoj probíhá již jen v menších fázích, ve kterých má zákazník další nové požadavky, které si přeje dodělat. Nové požadavky na funkčnost mohou vzniknout například proto, aby daný software ustál v konkurenčním prostředí. [4]. V rámci jednotlivých fázích vývoje tedy probíhá postupně nejprve základní první verze daného softwaru. Po nasazení do produkčního prostředí probíhá jednak údržba systému a také vývoj nových funkcionalit (viz Obrázek 7).

Samotný provoz softwaru začíná jakmile je vývoj dokončen. Provoz probíhá až do úplného ukončení činnosti softwaru. Během tohoto procesu probíhá práce spojené s během a udržováním aplikace. Během tohoto aspektu je software nasazován do produkčního prostředí a monitorován jeho běh. [4]

Každá část životního aplikačního cyklu se liší dle použitého vývojového procesu. Rozdílně vývoj probíhá při agilním procesu nebo při vodopádovém modelu. Jednotlivým vývojovým procesům se věnuji v rámci kapitoly 2.

5.1 codeBeamer

5.2 Siemens Polarion

5.3 IBM Jazz

5.3.1 Sběr požadavků

5.3.2 Monitorování procesů

5.3.3 Export dat

5.4 Porovnání aplikací

6 Business intelligence aplikace

TODO

Zde budou popsány BI aplikace + porovnání.

6.1 Qlik

6.2 Microsoft Power BI

6.3 Porovnání aplikací

7 Vlastní aplikace

TODO - praktická část.

Aplikace je realizována jako webová aplikace.

7.1 Architektura a použité technologie

7.1.1 Symfony

7.1.2 MongoDB

7.1.3 MySQL

7.2 Import dat z IBM Jazz

popsat jednotlivé druhy napojení na jazz, Automaticky pomocí cronu, rest api

7.3 Uložení dat

Kontroly importů duplicitních, přidávání dat k importu, GUI práce nad daty, promazávání apod.

7.4 Zpracování dat

předpříprava dat query language

7.5 Definice metrik

report builder

7.6 Zobrazení metrik

pomocí js

7.7 Výstupní reporty

Pdf, excel. Definování vlastního reportu, automatické ukládáni na projekt atd.

8 Závěr

TODO

Literatura

[1] SOMMERVILLE, Ian. *Software engineering*. 9th edition. Boston: Addison-Wesley, 2011, s. 28-29. ISBN 978-0-13-703515-1.

[2] *Úvod do softwarového inženýrství* [online]. In: VONDRÁK, Ivo. Ostrava, 2002, s. 3 [cit. 2020-11-30]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

[3] BIRT Design Overview. *BIRT* [online]. Ottawa: The Eclipse Foundation, 2014, 2014 [cit. 2020-11-28]. Dostupné z: https://www.eclipse.org/birt/about/

[4] *What is Application Lifecycle Management?* [online]. 2014, , 2-4 [cit. 2020-11-15]. Dostupné z: http://davidchappell.com/writing/white_papers/What-is-ALM–Chappell.pdf

[5] SOMMERVILLE, Ian. *Software engineering*. 9th edition. Boston: Addison-Wesley, 2011, s. 29-32. ISBN 978-0-13-703515-1.

[6] *Rational Unified Process: Best Practices for Software Development Teams* [online]. In: . Rev 11/01. Cupertino: Rational Software, 1998, s. 1 [cit. 2020-11-21]. Dostupné z: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[7] *Rational Unified Process: Best Practices for Software Development Teams* [online]. In: . Rev 11/01. Cupertino: Rational Software, 1998, s. 3 [cit. 2020-11-21]. Dostupné z: https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

[8] BECK, Kent a Cynthia ANDRES. *Extrémní programování: embrace change*. 2nd ed. Praha: Grada, 2002, s. 1—7. Moderní programování. ISBN 80-247-0300-9.

[9] MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.

[10] MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016 s. 85. ISBN 978-80-251-4650-7.

[11] MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016 s. 89. ISBN 978-80-251-4650-7.

[12] *Manifest Agilního vývoje software* [online]. 2001 [cit. 2020-11-21]. Dostupné z: https://agilemanifesto.org/iso/cs/manifesto.html

[13] SOMMERVILLE, Ian. *Software engineering*. 9th edition. Boston: Addison-Wesley, 2011, s. 59. ISBN 978-0-13-703515-1.

[14] TIERNO, Antonio, Max M. SANTOS, Benedito A. ARRUDA a Joao N. H. DA ROSA. Open issues for the automotive software testing. *2016 12th IEEE International Conference on Industry Applications (INDUSCON)* [online]. IEEE, 2016, 2016, , 1-8 [cit. 2020-11-27]. ISBN 978-1-5090-5127-4. Dostupné z: doi:10.1109/INDUSCON.2016.7874609

[15] Automotive SPICE | Download. *Automotive SPICE* [online]. 2005, 2018 [cit. 2020-11-17]. Dostupné z: http://www.automotivespice.com/download/

[16] *Automotive SPICE Process Reference and Assessment Model - Release 3.1* [online]. **1 November 2017**, s. 4 [cit. 2020-11-17]. Dostupné z: http://www.automotivespice.com/download/

[17] *Automotive SPICE Process Reference and Assessment Model - Release 3.1* [online]. **1 November 2017**, s. 8 [cit. 2020-11-17]. Dostupné z: http://www.automotivespice.com/download/

[18] *Automotive SPICE Process Reference and Assessment Model - Release 3.1* [online]. **1 November 2017**, s. 12-15 [cit. 2020-11-17]. Dostupné z: http://www.automotivespice.com/download/

[19] *Automotive SPICE Process Reference and Assessment Model - Release 3.1* [online]. **1 November 2017**, s. 15-23 [cit. 2020-11-18]. Dostupné z: http://www.automotivespice.com/download/