VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE
Fakulta informatiky a statistiky
nám. W. Churchilla 4
130 67 Praha 3 – Žižkov

Lean Software Development

(pro předmět 4IT421 Zlepšování procesů budování IS)

Zpracoval: Jan Filipčík

datum: 13. 12. 2011

Obsah

1. Úvod

V úvodní kapitole popíši a definuji důvod výběru tématu, jednotlivé cíle seminární práce a v neposlední řadě charakter a strukturu práce.

1.1 Důvod výběru tématu

Toto téma jsem si vybral z důvodu, že o ostatních agilních metodikách existuje relativně velké množství materiálů nebo se jim v průběhu studia věnuje větší význam, a proto jsem si vybral právě téma, o kterém se zase tak významně nemluví, abych se o něm dozvěděl více informací a přiblížil tuto agilní metodiku ostatním čtenářům této práce. Dále bych rád poukázal na klady a zápory metodiky a také na to, že nejenom RUP a jiné známé a hojně využívané metodiky, ale také ty méně známější metodiky se hodí znát a umět je efektivně využívat pro vývoj software.

Důležitým faktem je také to, že počítačová technika jde neustále rychlým krokem dopředu, vznikají stále nové metody a postupy nebo se alespoň jednotlivé metodiky neustále upravují a vylepšují. S rostoucím vývojem informatiky a počítačové techniky vznikají ruku v ruce neustálé nové a nové možnosti pro vývoj software, což zvyšuje požadavky i na konkrétní finální produkt, který se pak může stávat více nákladným a to jak z hlediska cenového, tak zdrojového a v neposlední řadě také časového. V tomto ohledu je Lean Software Development velmi zajímavý, protože pracuje s třetinovou cenou, zdroji a časem. Jedná se o zajímavé téma a jednotlivé uvedené aspekty a důvody jsou pro mě hlavním důvodem výběru tématu.

1.2 Cíle

Ve své práci se zaměřím na zpracování metodiky Lean Software Development. Hlavním cílem bude popsat a analyzovat metodiku a konkrétně zmapovat historii metodiky, jednotlivé praktiky a principy, na kterých je metodika založena a jak konkrétní vývoj software pomocí této metodiky vypadá a co vše je potřeba brát při vývoji do úvahy. Cílem bude také zanalyzovat a popsat silné a slabé stránky metodiky, resp. její výhody a nevýhody.

1.3 Charakter práce

Práce je rozdělena do 7 hlavních částí včetně úvodu a závěru. V druhé části práce se zaměřím na historii, definici a vymezení metodiky. Ve třetí části popíšu jednotlivé znaky metodiky Lean Software development. Ve čtvrté části definuji jednotlivé praktiky. V páté kapitole se zaměřím na konkrétních 10 nejznámějších principů metody. V následující kapitole se zkusím zaměřit na konkrétní výhody a nevýhody Lean Software Development. A v poslední části závěr zanalyzuji a popíši zjištěné výsledky.

2. Historie a popis metody Lean Software Development

Termín Lean software development vznikl v Japonsku a do světa tento pojem pronikl v 80. letech 20. století. Do češtiny tento pojem můžeme volně přeložit jako lehký, či štíhlý softwarový vývoj, což v praxi znamená uplatnění a aplikování lean principů týkajících se vývoje software. Metodika v praxi vychází převážně z principů manufakturingu a total quality managementu. Metodika je převážně zaměřena na vývoj softwaru a analýzu rizik. Nemůžeme jí však chápat pouze jako řízení nebo rozvoj metodologie, ale jedná se převážně o principy, které usnadňují konkrétní vývoj a návrh software. Silný důraz je kladen na využívání zdrojů, času a rozpočtového plánu, kde konkrétně Lean software development pracuje s třetinovým časem, třetinovými zdroji a třetinovým rozpočtovým plánem. Výhodou metodiky také je, že se relativně snadno dá kombinovat s ostatními agilními metodikami. Snižování odpadu je převážně dosahováno tím, že se klade důraz na jednoznačné vymezení hrubých požadavků na software. V tomto je např. metodika Scrum slabší a často se tyto dvě metody v praxi spolu kombinují. Metodika Lean software development se také často kombinuje s metodikou Kanban, která se používá při plánování procesů a výroby v některých výrobních i nevýrobních podnicích.

Existuje velké množství způsobů, jak nejlépe definovat a popsat metodiku. Jedna z nejlepších definic Lean software development je popsána národním institutem pro standardy a vysvětluje metodiku jako systematický přístup k identifikaci a odstranění odpadu prostřednictvím neustálého zlepšování konkrétního procesu stejného výrobku pro zákazníka se snahou o dokonalost, přičemž se snaží také snižovat poruchy a doby trvání vývojového cyklu procesu a současně se snaží zajistit růst přidané hodnoty pro podnik. Existují další dva způsoby jak definovat a nahlížet na metodiku. Prvním způsobem je soustředit se na snížení objemu odpadu. Tento způsob vychází z předpokladu, že každý prostředek využívaný při vývoji a v jednotlivých procesech se spojeným s vývojem má potenciál být promarněný, proto je potřeba identifikovat oblast úniku zdrojů a zamezit opakovaného úniku zdrojů. Tento způsob vymezení je patrný prakticky ve všech oblastech a činnostech spojených s vývojem např. od získávání informací a požadavků od zákazníků, až po vytvoření produktu. Rozlišují se činnosti s efektem pro organizaci a zákazníka a činnosti, které žádný efekt v procesu nepřináší. Činnosti, které nepřináší žádný efekt a nejsou v souladu s konečným cílem a efektem pro zákazníka se většinou snaží podnik eliminovat a tyto činnosti se považují za odpad. Tímto stylem řízení lze dostat řízení zdrojů projektu vývoje pod kontrolu. Druhý způsob vymezení Leanu je, že dochází často k plýtvání se zdroji. Princip plýtvání zdrojů, času a rozpočtu je většinou důsledek neodborného vedení, špatně definovaných procesů, vysoké pravděpodobnosti nastání nějaké krizové situace. Všechny tyto aspekty mohou vést nakonec k vyšším nákladům a možnému nedodržení konkrétní slíbené částky a dodání softwaru v čas.

Lean software development má 10 základních principů, 7 praktik a 5 znaků.

Mezi jednotlivé principy metodiky patří odstranění odpadu, minimalizace zásob, maximalizace toků, zlepšování kvality, vytváření poznatků a učení se v týmu, rozhodovat se co nejpozději, zpětná vazba, doručit nejrychleji, jak je to možné, budovat integritu a optimalizace celku.

Metodika se dále skládá ze 7 praktik, mezi které patří vidění odpadu, mapování a zlepšování procesů pro vývoj, Set-Based development, pull systém, teorie hromadné obsluhy, motivace a metrika měření.

Posledním rozdělením je na znaky metody. Ta se skládá z 5 znaků a to z celulárního programování, tlačením rozvrhem, total quality managementem, rapid setupu a týmového vývoje. Jednotlivé principy, praktiky a znaky více specifikuji v samostatných kapitolách. (1) a (2)

3. Znaky metodiky Lean Software Development

Prvním znakem metodiky je **celulární programování**. Celulární programování se využívá prakticky po celou dobu vývoje popřípadě i údržby a provozu software a je zaměřeno na řešení problémů se software. Jedná se o rozdělení programátorů do týmů, které řeší jednotlivé vzniklé problémy se software odděleně.

Druhým znakem Lean Software Development je **tlačení rozvrhem**, což prakticky znamená důraz na dodržení jednotlivých termínů, rozvržení prací, způsob řešení problémů. Při vývoji software pomocí této metodiky jsou velmi detailně a jednoznačně naplánované jednotlivé etapy a jejich rozvržení, role, funkce, které jsou jednoznačně sepsány v rozvrhu a plánu projektu, a klade se důraz na jejich dodržování.

Třetím neméně významným znakem je **total quality management**. Jedná se o přístup, kterým organizace zlepšují kvalitu svých vnitřních procesů s účelem zvýšení spokojenosti zákazníka. Jedná se o filozofii neustálého zlepšování všech činností v podniku, konkrétně v metodice lean software development se klade důraz na neustálé zlepšování, efektivitu a rychlejší vývoj software. Když je total quality management dobře prováděn tak to vede ke snížení nákladů a to hlavně nižších nákladů na případnou nápravnou nebo preventivní údržbu, současně dochází také ke zvýšení výkonu a spokojenosti jednotlivých zákazníků. Proto, aby mohl být total quality management dobře řízen musí být splněno následujících 7 kritérií. Musí existovat a být definováno řízení kvality; ne lidé, ale většinou špatně definovaný proces způsobuje problémy; neřešit pouze základní příznaky problémů, ale vyřešit problém do hloubky a vytvořit opatření vůči znovu nastání problému; každý zaměstnanec musí nést určitou zodpovědnost za kvalitu; kvalita musí být měřitelná; zlepšování kvality musí být plynulé a kvalita se musí stát dlouhodobou investicí. (3)

Posledním znakem metodiky je **rapid setup**. Rapid setup v podstatě znamená, co nejrychleji si připravit podklady, zdroje, informace a další věci ke konkrétní etapě, práci, či např. k procesu vývoje SW. Pomocí rapid setupu jsme schopni velmi rychle optimalizovat jednotlivé vnitropodnikové procesy. V lean metodice dochází nejčastěji ke třem případům, které zpomalují nebo úplně přerušují lidi nebo konkrétní procesy. Mezi ně patří pozdní příprava konkrétních materiálů a pozdní start procesu, či činnosti; malá podpora IT oddělení a většina individuálních žádostí má zpoždění, protože lidé raději pracují ve skupinách a dává se přednost kolektivním žádostem.

Metoda rapid setup se skládá ze 4 kroků, jejichž cílem je úprava konkrétního procesu a jeho zeštíhlení do té podoby, aby přinášela, co největší přidanou hodnotu organizaci a finálnímu zákazníkovi. Prvním krokem je zploštění jakéhokoli procesu, který by způsoboval jednu z těchto čtyř činností: zpoždění již při startu, způsobení přerušení procesu přidanou hodnotou nebo činností do procesu, proces začínají ze snížené ceny výstupu, proces obsahující částečně shodnou nebo kompletně duplicitní část procesu. Následující krok 2 se zaměřuje, jestli konkrétní zpožďující činnost procesu může být z procesu odstraněna nebo je na procesu přímo závislá a z procesu odstranit nelze. Předposledním krokem je pokusit se automatizovat a optimalizovat všechny problematické činnosti a úlohy, které nemohou být z procesu odstraněny. A posledním je krok 4, ve kterém se bere nově optimalizovaný proces pod statickou kontrolu a vedení.

Posledním znakem metodiky je **týmový vývoj**, což znamená vývoj SW a jeho komponent v několika členných týmech. (4)

4. Praktiky a nástroje Lean Software Development

Lean Software Development obsahuje celkem 7praktik a nástrojů.

4.1 Vidět odpad (seing waste)

Prvním praktikou je **vidění odpadu (seing waste)**, tato praktika je hodně taktéž spojená s prvním principem metodiky a to odstraňování odpadu (eliminate waste). Z hlediska důležitosti je potřeba umět rozeznávat, jaké zdroje, činnosti a procesy jsou důležité, resp. které přináší nějakou užitnou hodnotu podniku nebo zvyšují spokojenost zákazníka, a které naopak nenesou užitnou hodnotu nebo nemají vliv na spokojenost zákazníka a je potřeba je odstranit nebo částečně eliminovat. Většina agilních metod se snaží eliminovat odpad. Hlavní odpad u metodiky vzniká hlavně při vývoji software, protože ne všechny činnosti a etapy procesů mají přidanou hodnotu pro zákazníka. Pan Shigeo Shingo definoval 7 druhů výrobních odpadů, které usnadnilo definování toho, co odpad je a toho, co naopak odpadem není při vývoji software. Mezi těchto sedm druhů odpadů týkajících se vývoje softwaru patří pouze částečně vykonané práce, extra procesy, zvláštní vlastnosti, přepínání úloh, pohyb, čekání a vady.

Částečně vykonané práce resp. částečně hotový software, nebo software s jehož vývojem jsou problémy, a nepracuje se na něm, se může stát zastaralý a to z důvodu, že se nakonec vůbec nedodělá nebo se mezitím dá přednost vývoji úplně jiného software a původní se tedy může stát odpadem z hlediska, že už ho zákazník nebude chtít nebo se software stane zastaralým.

Extra procesy je myšleno takové činnosti a věci, které sice s vývojem software souvisí, ale jsou dělány samostatně. Mezi ně patří například papírování a tvorba dokumentace software. I papírování stojí velké množství peněz a času a ne vždy přináší dokumentace přidanou hodnotu pro zákazníka, a proto je dopředu potřeba zvážit, zda zákazník nutně konkrétní dokumentaci chce a zda je tvorba dokumentace opravdu nutná a potřebná vytvořit. Pro některé zákazníky se totiž tato dokumentace stává odpadem a pro organizaci jsou to pak jen zbytečně vynaložené náklady s dokumentací na tvorbu a čas.

Třetím odpadem je tvorba **zvláštních vlastností.** Každý programátor by rád projevil iniciativu a dodal softwaru nějaké nové funkce, které ani zákazník nespecifikoval a nezamýšlel. Prvotní idea se může zdát jako dobrý nápad, ale každý kus kódu v systému musí být sledován, sestaven, integrován a otestován a všechny tyto činnosti zvyšují čas na testování, dokumentaci, zdrojový kód a vzniká tak nový odpad v podobě ztraceného času, zbytečného testování a většího rizika chyb softwaru s přibývajícím kódem. Každý kus kódu zvyšuje složitost a potenciální selhání v určitém bodě. Existuje také reálně velká možnost, že kód bude navíc zastaralý dříve, než bude použit, proto je potřeba odolat pokušení navrhovat větší funkčnost než zákazník požaduje.

Dalším problémem a odpadem je **přepínání úloh.** Není vhodné, aby člen projektu vývoje softwaru byl členem více týmů a různých projektů. Přepínáním mezi projekty a úkoly se člen týmu plně nesoustředí na vývoj ani jednoho z nich a vývoj software je tak pomalejší a více náchylnější na vzniknutí chyby. Se špatně odvedenou prací může klesat i užitná hodnota pro zákazníka a software se může stávat zastaralý a konkrétní pracovník pro tým naprosto neužitečný.

Pohyb souvisí s alokací zdrojů, resp. jestli jsou jednotliví členové týmu dostupní a je možné jim pokládat otázky a aktuálně měnit požadavky. Jedná se převážně o to, zda se zákazník pohybuje na pracovišti vývojáře softwaru a zodpovídá na doplňující otázky, jestli existuje dobrá a správná komunikace mezi lidmi a jestli jsou efektivně a optimálně využívány zdroje pro vývoj projektu.

Nezanedbatelným a častým odpadem jsou **vady.** Je důležité dbát na vyvarování se vad, každá vada a její odstranění zvyšuje náklady projektu a také délku vývoje. Objevená vada do tří hodin je relativně rychle opravitelná a její odstranění nezabere tolik času a peněz k její nápravě, než vada, která je objevena např. až ke konci projektu vývoje a její odstranění může znamenat velké problémy a zásahy do vývoje a může být ohroženo dokončení vývoje SW.

Jeden z největších odpadů ve vývoji softwaru je **čekání** na to, co se stane, důvodem může být čekání při zahájení projektu, zpoždění v personálním obsazení, zpoždění kvůli nadměrným požadavkům dokumentace, zpoždění v hodnocení a schvalování, zpoždění v testování, a zpoždění při zavádění, všechna tato čekání se stávají odpadem. Zpoždění jsou běžné ve většině procesů. (5)

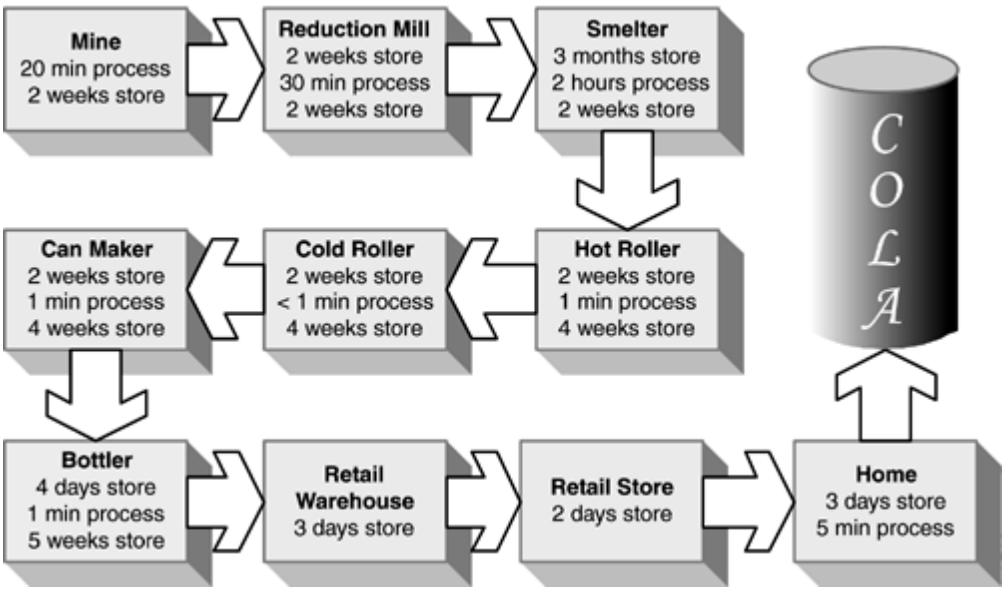
4.2 Mapování (Value Stream Mapping)

Druhou praktikou je **hodnota postupného mapování a tvorba procesů a činností** v organizaci s cílem zlepšit jejich efektivitu a optimalizaci**.** Mapování se v Lean Software Development používá jako prostředí pro analýzu a design toků na úrovni systému. Mapování se spíše spojuje s plánováním výrobních procesů, ale je možné je využít i pro vývoj softwarového produktu. Klíčovou metrikou splnění mapování je dodací lhůta, kdy má být software vytvořen.

Tvorba map a grafů jednotlivých procesů je dobrým způsobem, jak efektivně objevovat odpad vznikají při vývoji softwaru. Pomocí zmapování také rychle a snadno zjistíme efektivitu využívaného procesu a jakou hodnotu daný proces tvoří pro zákazníka. K simulaci procesů není potřeba pořizovat drahé programy, které zmapují jednotlivé procesy, ale zmapovat, jak probíhají jednotlivé procesy pro analýzu a návrh softwaru lze i pomocí tužky a papíru konkrétním zaměstnancem a to například tak když zaměstnanec simuluje roli zákazníka ve firmě, kdy jeden ze zaměstnanců bude simulovat jeho činnosti od objednání vývoje softwaru a tvorby požadavků, přes konkrétní vývoj a testování až do předání vyhotoveného softwaru zákazníkovi. Jednotlivá mapa je také charakteristická dvěma typy os. Tyto osy se nacházejí ve spodní části mapy, jedna ukazuje, kolik času se stráví v žádosti na přidanou hodnotou činností, a druhá kolik času se stráví v různých typech čekání, zdržení a také u činností, které nepřináší žádnou přidanou hodnotu organizaci a zákazníkovi.

Tvorba a implementace mapy se skládá z Identifikování cílových produktů nebo služeb. Druhým krokem je zakreslení pomocí mapy současného stavu procesu. Následně se provádí analýza současného stavu a definování, které činnosti jsou odpadové a bez přidané hodnoty. Po té se vypracovává nová mapa procesu a hledají se nepotřebné činnosti, které je možné odstranit. A posledním krokem je vypracovaná mapa budoucího stavu procesu a zavádění optimalizovaného procesu.

Snahou je docílit urychlení, zefektivnění, zkrácení a zrychlení jednotlivých činností a v neposlední řadě mít možnost relativně rychlých reakcí na a možnosti změn jednotlivých požadavků zákazníků a rychlou reakci na poptávku zákazníků. Dalším cílem je dodat produkt včas a předejít problémům a jednotlivým vadám, které by bez dobře zmapovaných procesů pro vývoj software mohly nastat.

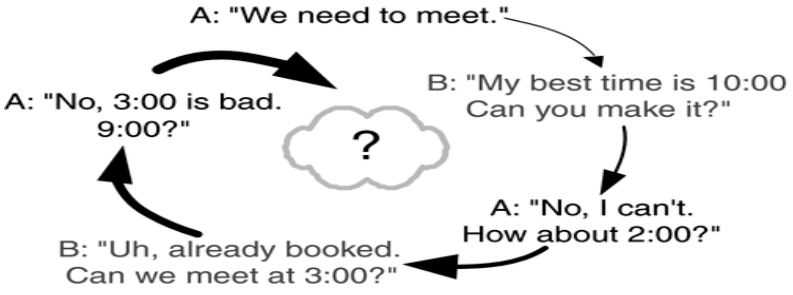


Obr. 1 Mapování procesu, zdroj: (5)

Na obrázku je možné vidět zmapovaný proces tvorby Koly od úplného začátku až po finální produkt. Zajímavostí je, že celý projekt trvá 319 dnů a z těchto dnů jsou pouze 3 hodiny, resp. 0,04 % přidaná hodnota pro zákazníka. (5) a (6)

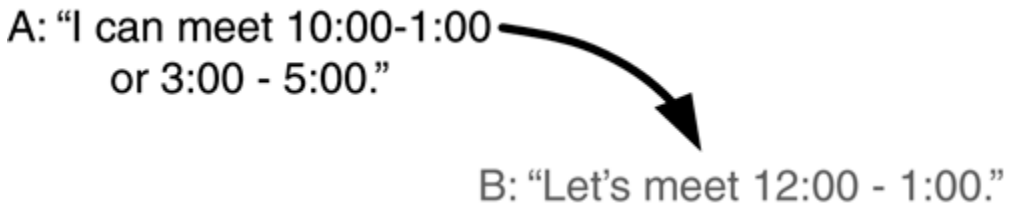
4.3 Set-Based Development

Existují dva způsoby plánování, jak naplánovat konkrétní činnost. Prvním způsobem je, že se vybere konkrétní čas činnosti, jak se to dotyčnému nejlépe hodí a ten předloží tento návrh druhému (Point-based). Problémem, ale je, že když někdo nastaví nebo navrhne konkrétní čas, nemusí být všechny zdroje a lidi být v daný okamžik k dispozici, aby daná činnost mohla proběhnout, a proto se konkrétní čas neustále dolaďuje a probíhá někdy i několik iterací, aby se zvolil konkrétní čas, který vyhovuje všem. Toto řešení je velmi náročné na komunikaci mezi lidmi a může docházet k neefektivnímu rozložení času, který je vyčleňován na neustále řešení, kdy danou činnost provést. Druhý způsob je definování všech omezení konkrétních zdrojů a lidí, kdy je navrženo omezení od kdy do kdy může konkrétní člověk nebo zdroj být k dispozici a na základě těchto omezení se najde v intervalu tohoto omezení čas, kdy danou činnost uskutečnit (Set-based). Obě řešení lze vidět na následujících dvou obrázcích.



Obr. 2Point-Based, zdroj: (5)

Na obrázku 2 je vidět příklad, kdy jeden člověk navrhne pro něj nejlepší čas schůzky a zeptá se druhého, zda mu tento čas vyhovuje, a když mu čas nevyhovuje, neustále se domlouvají a mění čas do té doby, dokud nenajdou rozumné řešení. Toto řešení je poněkud zdlouhavé a probíhá tak někdy velké množství iterací, než se dohodne konkrétní čas.



Obr. 3 Set-Based, zdroj: (5)

Na obrázku 3 je možné vidět návrh podle Set-Based, kdy konkrétní člověk, navrhne omezení, kdy může provést konkrétní činnost a druhý se v rámci zvolených omezení vyjádří, kdy ve zvolených omezeních prvního člověka má čas. Toto řešení je daleko rychlejší a naplánování proběhne hned pouze v rámci první iterace. Na rozdíl od prvního typu řešení, máme návrh rychleji a nepotřebujeme velké množství komunikace, jako tomu bylo v předchozím případě.

Pro vývoj softwaru v rámci Lean software development se využívá převážně Set-based development. Jedná se převážně o řešení složitých problémů a zaznamenávání vzniklých problémů a jejich řešení, kdyby se podobný problém objevil znovu u jiného vývoje SW. Když existuje složitý problém, je dobré zaznamenávat vyzkoušená a aplikovaná řešení z dřívějších vývojů a definovat, jak dobře se řešení konkrétního řešení osvědčila a či nedochází v momentální situaci k podobným problémům. Popřípadě, je li problém podobný je možné využít a sloučit ty nejlepší vlastnosti původního řešení nebo zvolit úplně jiný postup řešení. Někdy je zbytečné vyvíjet více řešení stejného problému, protož již podobně vyřešený problém může vést k lepšímu řešení rychleji. (5)

4.4 Pull systémy

Pull systémy jsou nástroje a systémy, které kladou velký důraz na dodání softwaru v čas a za stanovenou cenu. S aplikací těchto systémů je v organizaci potřeba znát, aby každý člověk věděl, co má kdy, kde a proč dělat a byly optimalizované jednotlivé zdroje. Jakmile lidé v týmu nevědí, co mají dělat, ztrácí se drahocenný čas a doručit vyvíjený software ve stanovenou dobu nebude možné. Existují dva způsoby, jak zajistit, aby pracovníci co nejefektivněji využili svého času. Buď je možné říci, co konkrétně mají dělat a direktivně je řídit, nebo jim dát volnou ruku s dodržováním dohledu nad prováděním jejich činností.

Cílem je zjistit, co, kde a kdy by měl každý zdroj a každý člověk udělat, aby bylo možné maximalizovat užitnou hodnotu pro zákazníka, splnit všechny požadavky a dodat software ve

stanovené domluvené lhůtě za konkrétní smluvenou částku. Ve výrobních podnicích se často kombinuje Lean s Kambanem a využívají se různé ERP nebo MRP systémy, které slouží pro plánování výroby.

Výchozím bodem pro pull system ve vývoji softwaru jsou krátké iterace založené na podnětech zákazníků na začátku každé iterace. Na začátku iterace, zákazník nebo zástupce zákazníka napíše popis funkcí a požadavky na konkrétní software na Indexovací karty. Existuje mnoho jiných způsobů, jak dokumentovat, co zákazníci chtějí, ale indexovací karty jsou hodně podobné kartě používané u Kanbanu a lze tak relativně snadno kombinovat Lean Software Development s postupem Kanbanu. U každé indexovací karty se snaží vývojář zanalyzovat jednotlivé požadavky a odhadnout, jak bude dlouho asi práce trvat, na základě došlých indexovacích karet a podle vytvořených plánovaných termínů se rozhodne, kterou kartu bude brát dříve a kterou později. Vývoj softwaru probíhá následně v týmu, který by se měl přibližně každý den scházet zhruba na 15 minut, kde by se mělo sdělit, co je potřeba udělat, co je již hotové a otestované, případně jaké jsou překážky projektu vývoje. (5)

4.5 Teorie hromadné obsluhy

Jedná se o teorii umožňující analýzy několika souvisejících procesů a to jak na konci fronty, uprostřed fronty anebo při výstupu z fronty. Tato teorie umožňuje odvozování a výpočty několika různých opatření jako např. průměrná čekací doba ve frontě, očekávaný počet čekání, očekávaný počet přijímaných služeb aj.

Tato teorie a analýzy se konkrétně ve vývoji softwaru týkají testování vyvíjeného softwaru a dělají se konkrétní analýzy a opatření pro fungující kód programu. Často se dělají testy vývojářů, zákaznické texty a komplexní testy, např. junit testy. (5)

4.6 Motivace

Proto, aby tým byl schopen dosáhnout vysokých a vynikajících výsledků je potřeba, aby byl řádně motivován a všichni jeho členové v dostatečné psychické pohodě. Vášeň a kamarádství a dobré pracovní podmínky vytvářejí dobrou atmosféru v týmu. Pracovníci, kteří tvoří přesvědčivé myšlenky, nápady mají tendenci inspirovat ostatní členy týmu. Práci a formování činností v týmu sleduje sponzor a manažer, kteří hlídají, aby se projekt nevychýlil mimo naplánovanou osu a byly splněny veškeré požadavky zákazníka v čas. Sponzoři a manažeři mohou pomoci členům týmu, přijímat a zjednodušovat přístup k materiálům a potřebným informacím.

Typy jak udržet motivaci v týmu:

- začít s jasným a přesvědčivým účelem
- Ujistit se, že cíle je možné dosáhnout.

• Dát týmu přístup k zákazníkům. Možnost mluvit na skutečné, živé zákazníky je skvělý způsob, jak členové týmu pochopí účel a podstatu toho, co dělají.
• Nechat tým pracovat podle sebe.
• Vysoce motivovaný tým nemusí mít řečeno, co má dělat, popřípadě je pouze v nesnázích řízen pomocí rad od managementu.
• Udržet skeptické myšlení pryč od týmu. Nic nepotopí více projekt, než někdo, kdo pořád tvrdí, že se projekt nestihne nebo, že lidé netuší, jak projekt vytvořit nebo dokončit. (5)

4.7 Měření a metriky

Většinou máme tendenci rozložit velký úkol na menší úkoly. Celkový výsledek je pak získán optimalizací dílčích výsledků jednotlivých pracovních míst. Optimalizovat každý úkol je často velmi špatná strategie. Je potřeba vyčlenit důležité úkoly, které přináší člověku nějakou užitou hodnotu a ty je potřeba v první řadě optimalizovat, od těch méně důležitých úkolů.

Ekonomický přínos rychlého toku procesu se těžko měří, ale zvyšování průtoku, je skvělý způsob, jak identifikovat a odstranit odpad vzniklý v sub optimalizovaném měření. Typický projekt měření nákladů a kontroly časového harmonogramu se často provádí ze zvyku také. Lidé většinou věří, že jsou nejdůležitější měření úspěšnosti projektu, ale naopak je důležité měřit výkon a efektivitu jednotlivých procesů pro vývoj a měřit splňování projektu více metrikami po celou dobu jeho vývoje, jen tak lépe odhalíme chyby a výsledná efektivita úspěšnosti procesu a projektu bude dobrá. (5)

5. Principy Lean Software Development

Lean software Development obsahuje celkem 10 principů, z nichž 7 je hlavních, a další 3 jsou vedlejší. Mezi 7 hlavních principů metody patří likvidace odpadu, zvyšování kvality, vytváření poznatků, rozhodování se co nejpozději, doručování tak rychle, jak je to možné, budování integrity a optimalizace celku.

Mezi tři vedlejší principy patří minimalizace zásob, maximalizace toku a zpětná vazba. (7) a (8)

5.1 Eliminace odpadu

Tento princip je založen na likvidaci odpadu, resp. na likvidaci takových částí procesů a úloh ve vývoji software, které nezvyšují kvalitu kódu, ale zvyšují nároky na čas, zdroje a náklady. Jinými slovy se dá říci, že se jedná o každou neefektivní činnost, která je v procesu zabudována a snažíme se jí odstranit, aniž by došlo ke snížení kvality procesu, ale naopak aby došlo k lepší optimalizaci a zeštíhlení procesu s nižší pracností a menšími náklady. Důležité

pro eliminaci těchto odpadových činností je potřeba mít schopnost rozpoznávat, co odpadem je a co naopak odpadem není.

Mezi nejčastější odpad při vývoji softwaru patří nadbytečný kód a nadbytečné funkce, zpožďování procesu vývoje software, nejasné požadavky na software, byrokracie, pomalá interní komunikace a nedostatečné testování, které může zapříčinit vznik škod a špatnou funkčnost vyvíjené aplikace. Aby bylo možné snížit množství odpadu, je důležité, aby se vývojové týmy mohly samostatně organizovat a provozovat tak, aby splňovaly konkrétní úlohy a práci, kterou je potřeba dosáhnout.

Velkou roli také hrají aktivity managementu, ty přímo netvoří přidanou hodnotu aplikace pro zákazníky, ale taktéž mají vliv na vznikající odpad. Sledování projektu a kontrolní systémy také nemají přidanou hodnotu, a další, mohou být údaje o příliš velkém množství práce v systému. Naučit se vidět odpad je dlouhodobý proces, který mění způsob, jakým člověk myslí a přemýšlí o tom, co je opravdu nutné zachovat a co ne. Jeden způsob, jak zjistit, co odpadem je, je přemýšlet o tom, co by měli lidé hodit přes palubu, pokud by se měli zbavit všech problémů zpožďující projekt. Nástroje a praktiky, které se používají pro eliminaci odpadu, jsou vidět odpad a mapování. (1), (5), (6) a (9)

5.2 Zvyšování kvality

V organizaci je neustále potřeba zlepšovat a nacházet chyby v jednotlivých interních procesech firmy a zvyšovat jejich kvalitu. Důležité je u zvyšování kvality provádět synchronizaci, automatizaci a refaktorizaci. Cílem je, aby jednotlivé procesy neobsahovaly žádné defekty a problémy, pokud tomu tak není, je potřeba provézt hloubkovou analýzu konkrétních problémů a daný problém odstranit a po té provést refaktorizaci procesu, či software.

Proto, aby vývoj softwaru byl v co v nejlepší kvalitě, je potřeba se o konkrétní vývoj softwaru starat již daleko dříve než těsně před implementací a před vznikem prvních řádků kódu. Synchronizace by pak v jiném případě mohla být velmi nákladná a náročná.

Druhým zmíněním prvkem je automatizace. Vhodné je provádět automatizaci u jednotlivých testů, zdrojů, zkrátka u všech činností v procesu, které jsou rutinní a neustále se opakují a je možné je automatizovat.

Posledním, ale velmi významným krokem je provádění refaktorizace. Refaktorizací je myšleno odstraňování duplicit kódu, které mohou při vývoji software nastat. Po každé dojde-li k objevení určité duplicitní části kódu je potřeba provést refaktorizaci. Důležité je také co nejvíce snížit složitost testování a dokumentace. Je potřeba vzít v úvahu, že vnitřní struktura systému bude vyžadovat neustálé zlepšování a neustále nový vývoj. Stejně jako jsou výrobní procesy neustále zlepšovány výrobními dělníky, tak podobně na stejném principu je softwarový systém neustále zlepšován vývojáři. Ve skutečnosti organizace používá mnoho

softwarových produktů, které proto, aby nezestárly a nezačaly být neefektivní, prošly několika verzemi a byly několikrát refaktorovány. Bez neustálého zlepšování bude každý softwarový systém postupně více a více zastaralý a v překvapivě krátké době, přestane být pro zákazníka užitečný a také přestane odpovídat konkrétním měnícím se požadavkům, což povede od ustoupení zákazníka od používání daného software, a proto je velmi důležitá role refactoringu, který zajišťuje neustálé zlepšování kvality jednotlivých vývojových procesů, která zahrnuje, neustále vylepšování procesu, udržování jeho kvality a eliminaci jeho zastarávání. Nástroje a praktiky, které se používají pro zvyšování kvality, jsou měření a metriky. (8) a (9)

5.3 Vytváření nových poznatků, zkušeností a učení

Jedním z nejdůležitějších a nejužitečnějších věcí pro získávání poznatků je plánování a modelování. Plánování tvoří základ veškerého dalšího vývoje software, procesu, či např. strategie. Plánování také pomáhá týmům učit se a odhalovat to, co opravdu zákazníci chtějí. Důležité je také pravidelně reflektovat to, co tým po celou dobu dělá a pomocí zpětné vazby od zákazníka pracovat na zlepšení svých dovedností. (9)

Proto, aby bylo možné vytvářet nové poznatky a zkušenosti a zpětně se učit z činností, které byly již realizovány je potřeba evidovat a zaznamenávat detailně konkrétní postupy týmu. Důležitá je taktéž role vedoucího týmu, který musí mít přirozenou autoritu a jednotliví členové ho musí respektovat a plně poslouchat. Důležitým úkolem vedoucího týmu je vytvořit dobré přátelské a kulturní prostředí a budovat přátelské vztahy mezi jednotlivými členy v týmu. (8)

Vývoj softwaru je nepřetržitý proces učení. Nejlepší přístup ke zlepšení prostředí vývoje softwaru je zesílit učení. Proces shromažďování požadavků uživatelů je možné zjednodušit tím, že se představí obrazovky pro koncového uživatele a jejich jednotlivé vstupy, výstupy a přechody mezi obrazovkami. Ke zlepšování a vytváření nových poznatků a zkušeností je potřeba provádět zpětnou vazbu se zákazníkem a zjišťovat, co bylo dobře a co naopak špatně a poučit se do budoucna z vytvořených chyb. Nástroje a praktiky, které se používají pro vytváření nových poznatků, jsou zpětná vazba, iterace, synchronizace, Set-based development. (6)

5.4 Rozhodovat se, co nejpozději

Jako většina agilních metodik i lean software development neklade přímý důraz na absolutní specifikaci všech požadavků pro vývoj software zákazníka. Důležité je mít určitou představu, jakým směrem by se řešení mělo ubírat a pracovat pouze s hrubou představou projektu. Konkrétní detailnější požadavky vznikají při vývoji a při další analýze a implementaci software. Je potřeba však dbát na to, aby se vývoj řídil pořád správným směrem a byly naplněny konkrétní požadavky zákazníka. Vývoj jednotlivých komponent daného software by

měl být ve velké míře na sobě nezávislý a měla by být garantována funkčnost konkrétní samostatné komponenty, i když ještě není dokončený kompletní vývoj celého software. Dobrá, ale ne vždy levná záležitost je vyvinout více řešení pro všechny kritické oblasti a rozhodnutí, které nastanou při vývoji, a postupně zjistit, které řešení funguje nejlépe. To může být někdy velmi těžké, drahé a časově náročné, ale je potřeba se rozhodnout, co je z hlediska dlouhodobého období levnější. Důležité je mít zdrojový kód v takové formě, aby bylo možné rychle reagovat na změny požadavků zákazníků a nebylo potřeba kvůli jedné oznámené změně změnit a programovat znovu celý software. Obecně platí, pokud si nejsme jisti rozhodnutím, je vhodné ho odložit a poradit se, než zbytečně investovat náklady, zdroje a čas do možného špatného řešení. (10)

5.5 Doručit, jak nejrychleji je to možné

V dnešní době rychlého rozvoje informačních technologií nehraje roli velikost a robustnost systému, ale rychlost jeho vývoje. Pokud proběhnout všechny testy v pořádku a software je přijat bez výhrad a žádných vad je výhodou rychlého vývoje možnost přijmout relativně rychle zpětnou odezvu a zpětnou vazbu od zákazníka a případně provést další rychlé iterace a dovést software k úplné dokonalosti. Čím kratší jsou jednotlivé iterace, tím lépe tým nabývá nové zkušenosti a poznatky a zvyšuje svojí kvalitu, ale také se zlepšuje komunikaci mezi jednotlivými členy týmu. Pro rychlý vývoj software se používá v Lean Software Development převážně metoda just-in-time nebo Kanban. Rychlý vývoj zabezpečuje převážně také to, že se software nestává rychle zastaralým a nepoužitelným.

Jednotlivé nové a upřesňující požadavky při vývoji SW se zvyšují nelineárně s rostoucím časem. U typických projektů, které trvají zhruba 9-12 měsíců vzniká zhruba 25% změn v požadavcích při vývoji. Nicméně, množství požadavků na změny v průběhu jednoho měsíce jsou v průměru pouze 1-2%.

Proto, aby bylo možné doručit a vyvinout software, co nejrychleji je potřeba rozvrhnout práci do malých dávek, resp. jedná se převážně o snížení velikosti projektů, zkrácení cyklu vydávání, stabilizování pracovního prostředí, opakování toho, co se osvědčilo a bylo dobré a odstraňování postupů, které vytvářejí překážky. Dále je také potřeba určit jednotlivé limity pracovních kapacit a v neposlední řadě zkracovat cyklus celého vývoje. Nástroje využívané pro rychlý vývoj SW jsou teorie hromadné obsluhy a pull systémy. (1), (6) a (8)

5.6 Budovat Integritu

Budování integrity se skládá z vnímané integrity, koncepční integrity z testování a integrace. Vnímaná integrita se určuje podle zákazníka a znamená, jak zákazník vnímá konkrétní vyvíjený software a jaké jsou jeho zkušenosti s tímto software. Koncepční integrita je vnímaná z hlediska systému, jak systém a jednotlivé komponenty spolu dokážou spolupracovat jako celek. Snaha je udržet rovnováhu mezi flexibilitou, udržovatelností,

efektivností a schopností reagovat rychle na změny požadavků. Velmi důležitý je zde tok informací, ten musí probíhat obousměrně, tedy nejen od zákazníka k vývojáři, ale také opačně. I v rámci tohoto principu se velmi často používá refactoring. Na konci by měla být ověřena integrita pomocí důkladného testování, čímž se zajistí, že systém dělá to, co zákazník opravdu očekává. Automatizované testy bývají součást každého procesu, ale ne vždy mají přidanou hodnotu pro zákazníka, a proto by nemělo být automatizované testování cílem, ale pouze prostředkem k dosažení cíle, které slouží k převážně k eliminaci a snížení vzniku vad při vývoji software. (1) a (6)

5.7 Optimalizace celku

Pokud chce být člověk nebo tým efektivní musí na organizaci pohlížet z vrchu jako na úplný celek, protože i organizace je jakýsi zastřešovací celek a jednotlivé procesy, které v ní probíhají, spolu přes určité vazby souvisí. Tým musí pochopit na vysoké úrovni jednotlivé business procesy organizace, provázanost jednotlivých procesů a úloh, z nichž některé jednotlivé procesy jsou součástí více systémů, aby lépe pochopil jednotlivé požadavky na vývoj software od zákazníka. Ve své podstatě to při vývoji software znamená, zanalyzovat chod firmy a pochopit její myšlení a chod organizace. Konkrétně při vývoji různých typů software, když dochází k neustálým stejným nebo podobným problémům, je snaha jim předejít i do budoucna. To znamená, že je potřeba vyřešit problém nejenom v rámci konkrétního vývoje daného software, ale v rámci celé organizace upravit, optimalizovat jednotlivé procesy a činnosti postupu vývoje software, tak, aby smluvené termíny, náklady a zdroje byly dodrženy a nedocházelo k neustále se opakujícím problémům, které každý vývoj SW zdržovaly. Těmto problémům a vadám se nejčastěji předchází rozložením vývoje do menších úkolů a iterací, které jsou dokončeny a vytvořeny samostatně a pak se postupně jednotlivá navržená řešení slučují. Výhodou těchto rychlých a malých iterací a úkolů, je, že jsou podstatně jednodušší, než vytvářet jednu velkou iteraci celého systému.

5.8 Minimalizace zásob

Jak již bylo zmíněno Lean Software development pracuje s třetinovými zdroji, časem a rozpočtem. Tedy je snaha, co nejvíce minimalizovat plýtvání zdroji, časem a vynaloženými náklady na vývoj. Nejčastěji je minimalizování zásob dosahováno rychlým vývojem SW, kdy není potřeba využívat zdroje delší dobu než je nutné a v rámci malých, ale rychlých iterací se ušetří i podstatné množství času. Druhým aspektem, který přispívá k minimalizaci zásob je využívání v Lean Software Development při vývoji prvky just-in-time a prvky Kanbanu. Kdy konkrétně v Kanbanu zajišťování zásob začíná až předáním indexovací karty s požadavky na software od zákazníka k vývojáři. Vývojář pak stanoví přibližný čas tvorby vývoje software a kolik a jaké zdroje, či profese lidí budou potřeba k danému software být dostupné. Není potřeba evidovat žádné zásoby, ty se efektivně vytváří a určují, až po předání konkrétních

požadavků zákazníkem. Zásoby se tedy konkrétně vážou vždy na konkrétní indexovací kartu, tím je zajištěné, že nedochází ke zbytečnému čerpání nebo skladování zásob. (5)

5.9 Maximalizace toku a odstranění lokální optimalizace

Ve vývoji software je aktivní myšlenkou maximalizace toku informací a zvyšování hodnoty pro zákazníka a organizaci. V Lean Software Development se klade důraz na vývojový tým softwaru, který získává od kvalifikovaného zákazníka potřebné informace a maximalizuje tok a následně v malých a rychlých krocích provádí iterace a zákazníkem jsou určovány priority a případně měněny požadavky na funkčnost softwaru pomocí zpětné vazby. (11)

5.10 Zpětná vazba

Jak již bylo zmíněno Lean Software Development se snaží vyvíjet software v co nejkratším termínu a v co nejrychlejších iteracích. Lean jako agilní metodika se snaží přizpůsobovat okamžitým měnícím se podmínkám na trhu.

Problémem koncepce specifikování detailních požadavků bez možnosti změn hned v detailní analýze návrhu je, že zákazník sám mnohdy neví a nemá představu o detailní funkcionalitě a nemá konkrétní a detailní představu úplně detailního řešení finální aplikace, a proto, definování jednotlivých detailních požadavků hned z počátku se může stávat statické a systém nemusí splňovat vše, co by zákazník původně požadoval. Protože se požadavky na změnu se často po celou dobu životnosti většiny systémů mění a nemohou být u takto navržených systémů často dostatečně splněny a vykonány určité změny, které by zákazník dodatečně požadoval, případně pokud je to možné, je jejich realizace většinou velice nákladná a časově náročná. Takový systém rychle může přestat splňovat požadavky zákazníka a stát se tak rychle zastaralým, protože není schopný reagovat na požadované změny od zákazníka a ani na měnící se požadavky na trhu.

Proto agilní metodiky a metodika lean software development chtějí po zákazníkovi specifikaci pouze jednotlivých hrubých a základních požadavků, principů a popis funkčnosti, který by měl systém splňovat. Mnohdy zákazník, ani vývojář nemají představu o konkrétní podobě systému a ta je vždy blíže specifikována při aktuální fázi vývoje. Proto, aby byly splněny požadavky zákazníka je potřeba provádět neustálou komunikaci se zákazníkem a je potřeba provádět zpětnou vazbu se zákazníkem. Tedy je potřeba provádět různá setkání týmu a zástupci zákazníka a představit již např. část fungujícího kódu nebo aplikace a specifikovat a vyjednat se zákazníkem detailnější požadavky, popřípadě zaimplementovat změny ve stávajícím řešení, které zákazník požaduje. Tím je zaručeno, že tým vývojářů nesejde z cesty a vyvine systém takový, který bude odpovídat konkrétním požadavkům zákazníka.

Na zpětnou vazbu od zákazníka je možné reagovat dvěma způsoby, buď vyvinout novou komponentu a dopsat, specifikovat a otestovat konkrétní funkčnost nebo provést refactoring. Refactoring je často jednou z nejrychlejších metod, jak rychle zareagovat na změny požadavků od zákazníka, je k tomu ale potřeba mít vytvořený přehledný a ucelený kód, aby bylo možné, provést rychle a efektivně změnu v systému. Tato změna by se měla dotknout např. jen určité komponenty nebo části systému, refactoring by měl zajišťovat, že ostatní části systému, pokud se do nich nebude zasahovat, by měly být funkční a provozu schopné, nezávisle na změnách v jiných částech systému nebo komponentách. (11)

6. Výhody a nevýhody Lean Software Development

6.1 Výhody Lean Software Development

První výhodou Lean Software Development je **odstraňování odpadu**. Odstranění odpadu vede k celkové účinnosti procesu vývoje software. Současně je urychlován proces vývoje software, který snižuje čas a náklady projektu, což je v dnešním světě naprosto nezbytné.

Velice populární a oblíbený je u zákazníků **rychlý vývoj software**, který metodika nabízí, rychlý vývoj software je druhou výhodou metodiky. Tým vývojářů je schopný zajistit velké množství funkcí softwaru za relativně krátké časové období.

Třetí výhodou je většinou **silný tým vývojářů** a jejich relativně nezávislá činnost při vývoji software. Důležitá je tvorba jednotlivých rozhodnutí, jakým postupem se vývoj bude ubírat. Pokud si tým není dostatečně jistý, je dobré rozhodnutí odložit a získat a více specifikovat jednotlivé informace a požadavky, aby nebyl zbytečně promrhán čas a náklady projektu. (12)

6.2 Nevýhody Lean Software Development

První nevýhodou je, že projekt a **vývoj je velmi závislý na soudržnosti celého týmu** a na individuálních závazcích jednotlivých členů týmu. Pokud celý tým mezi s sebou nespolupracuje a někteří členové si lidově řečeno házejí klacky pod nohy, může se konkrétní projekt zpozdit nebo prodražit. Ve finále jsou někteří členové nuceni pak dělat dlouhé přesčasy a pod velkou tíhou stresu a únavy mohou dělat ve své práci chyby.

Druhou nevýhodou je potřeba mít tým, **kde každý má své určité dokonalé a individuální znalosti v konkrétní oblasti** vývoje software a jednotlivý členové se mezi s sebou vzájemně doplňují, není-li tomu tak a je-li tým sestaven z neodborníků, vzniká při vývoji software velké množství problémů.

Problémem a třetí nevýhodou je, že sponzoři a klienti by rádi věděli, jakým způsobem se bude vývoj software vyvíjet. Cílem je vytvářet taková rozhodnutí, aby byl vývoj software

rychlejší a efektivnější, ale někdy dochází k velmi velkým prodlevám a strachu učinit konkrétní rozhodnutí, což má za následek ztrátu času a neefektivní využívání lidí a zdrojů.

Čtvrtou nevýhodou nebo spíše hrozbou je **volná specifikace požadavků** na software. Flexibilita je skvělá, ale vše, co vede k rychlému vývoji, může ztratit určitý zřetel a správnou cestu a může se odchýlit od původních cílů a plánů a tak vývoj a cyklus může trvat velmi dlouho a v některých případech nikdy neskončí a dojde k opuštění a nedokončení projektu, jak z hlediska nedostatku zdrojů, nebo vysoké ceny, tak např. i nedostatku času konkrétních členů týmu. (12)

7. Závěr

Cílem mojí práce bylo popsat a analyzovat agilní metodiku Lean Software Development. V práci jsem představil a popsal její historii a rozebral podstatu metodiky. Dále se zaměřil na konkrétní znaky metodiky, mezi nejdůležitější znaky patří total quality management a rapid setup. Následně jsem v práci definoval jednotlivé praktiky, principy a v neposlední řadě také výhody a nevýhody metodiky.

Závěrem je potřeba říci, že ve světě má tato metodika budoucnost a to hned z několika důvodů. Jedním z důvodů je, že je šetrná na hlavní aspekty vývoje, kterými jsou čas, náklady a zdroje. Dalšími důležitými přednostmi metodiky jsou její možnost kombinace s jinými agilními metodikami, převážně se Lean Software Development kombinuje se Scrumem. Ale metodika se neváže často jenom s agilními metodikami, ale také s některými metodikami podporující výrobní procesy jako je Just-in-time a Kanban. Tudíž lze říci, že využívá ze všech metod ty nejlepší vlastnosti, které využívá pro efektivní vývoj software.

Dalším důležitým faktem a výhodou Lean Software Development je rychlý vývoj software a využívání zpětné vazby se zákazníkem. Důležité je neopomenout, že Lean Software Development vytváří produkt v rámci malých a krátkých iterací, a proto může relativně rychle a snadno reagovat na nové a upřesňující požadavky zákazníka, aniž by musel měnit a zasahovat do celého vyvíjeného systému. Rychlý vývoj, jak již bylo zmíněno, šetří čas, ale také minimalizuje náklady a přináší užitnou hodnotu pro zákazníka, který má výsledný produkt vyvinutý brzy a v rámci zpětné vazby je schopný navrhnout a specifikovat problémy a další upřesňující požadavky.

Proto, ale aby byl vývoj efektivní a bez problému je v první řadě potřeba mít optimalizované procesy a úlohy pro vývoj software. Je potřeba efektivně a rychle provádět rozhodování, které eliminuje jednotlivé problémy, vady, a které odhalí i jednotlivé nedostatky vývojových procesů. Proto, aby mohl být vývoj úspěšný je potřeba celý proces vývoje zmapovat a nepřistupovat hned k implementaci. Analýza je mnohdy nejdůležitější součást vývoje softwaru.

8. Bibliografie

1. **Norton, Darell.** Lean Software Development Overview. *Code better.* [Online] [Citace: 2011. 11 1.] http://codebetter.com/darrellnorton/2005/02/02/lean-software-development-overview/.

2. Being agile with Lean software development. *The server side.* [Online] [Citace: 1. 11 2011.] http://www.theserverside.com/feature/Being-Agile-with-Lean-Software-Development.

3. Total Quality Management. [Online] [Citace: 9. 11 2011.] http://managementhelp.org/quality/total-quality-management.htm.

4. Reducing Delays in Service Processes with Rapid Setup . [Online] [Citace: 9. 11 2011.] http://www.isixsigma.com/index.php?option=com_k2&view=item&id=75:reducing-delays-in-service-processes-with-rapid-setup&Itemid=156.

5. **Mary Poppendieck, Tom Poppendieck.** *Lean Software Development: An Agile Toolkit.* Upper Saddle River : Pearson Education Corporate Sales Division, 2003. ISBN: 0-321-15078-3.

6. Lean Software Development. *Wikipedia.* [Online] [Citace: 1. 11 2011.] http://en.wikipedia.org/wiki/Lean_software_development.

7. **Waters, Kelly.** 7 Principles of Lean Software Development. *All about agile.* [Online] [Citace: 1. 11 2011.] http://www.allaboutagile.com/7-key-principles-of-lean-software-development-2/.

8. **Bielicki, Przemyslav.** Seven principles of Lean software development. *Agile Software development.* [Online] [Citace: 2011. 11 1.] http://agilesoftwaredevelopment.com/leanprinciples.

9. The Principles of Lean Software Development. *IBM.* [Online] [Citace: 2011. 11 1.] https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/entry/principles_lean_software_development?lang=en.

10. **Bielicki, Przemysław.** Seven Principles of Lean Software Development - Defer Commitment. [Online] [Citace: 11. 11 2011.] http://agilesoftwaredevelopment.com/blog/pbielicki/seven-principles-lean-software-development-defer-commitment.

11. **Kumar, Dasari Ravi.** Lean Software Development. [Online] [Citace: 1. 11 2011.] http://www.projectperfect.com.au/info_lean_development.php.

12. The advantages and disadvantages of Lean Software Development. [Online] [Citace: 2011. 11 1.] http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-lean-software-development.html.