Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika
Obor: Informatika

Lean Software Development

BAKALÁŘSKÁ PRÁCE

Student : Lucie Hefnerová
Vedoucí : doc. Ing. Alena Buchalcevová, Ph.D.
Oponent : Mgr. Jakub Balada

2012

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně a že jsem uvedla všechny použité prameny a literaturu, ze které jsem čerpala.

V Praze dne 3. května 2012 . . . . . .

Lucie Hefnerová

Poděkování

Mé poděkování patří, vždy vstřícné vedoucí této bakalářské práce, doc. Ing. Aleně Buchalcevové, Ph.D., za svolení k vedení mé práce, nasměrování ke správnému výběru literatury, veškeré rady i připomínky. Dále bych chtěla poděkovat svým rodičům a trpělivým čtenářům – Ivaně a Jindřichovi, za neocenitelnou podporu nejen při psaní práce, ale v průběhu celého mého studia.

Abstrakt

Cílem této bakalářské práce je vznik uceleného, česky psaného materiálu o, v oblasti vývoje softwaru stále více skloňovaném, konceptu Lean Software Development. Dalším cílem práce je shrnout možné přístupy ke kategorizaci konceptu a shrnout možné přístupy k vymezení vztahu agilního a lean vývoje softwaru. Práce detailněji kategorizuje i nástroje využitelné k podpoře principů konceptu.

Práce nejprve uvádí čtenáře do problematiky vývoje softwaru a následně představuje původní principy Lean myšlení. Po tomto úvodním seznámení s kontextem konceptu následuje kapitola věnovaná jeho charakteristice (interpretace, kategorizace, myšlenkové směry). Práce rovněž shrnuje možné podoby výčtu principů konceptu a jejich charakteristiky, popisuje a kategorizuje vybrané nástroje pro podporu těchto principů. Na základě všech částí práce jsou shrnuta možná pozitiva a negativa spojená s jeho aplikací. Práce zahrnuje rovněž příklady reálného užití konceptu v praxi.

Klíčová slova

Lean Software Development, Lean myšlení, Agilní vývoj software, software, vývoj, iterace, funkcionalita

Abstract

The main goal of this bachelor thesis is the emergence of the clear Czech written material concerning the concept of Lean Software Development, which has been gaining significant attention in the field of software development, recently. Another goal of this thesis is to summarize the possible approaches of categorizing the concept and to summarize the possible approaches of defining the relationship between Lean and Agile software development. The detailed categorization of the tools potentially usable to support the principles is also covered.

First chapters are written for the introductory purposes of the context of the concept of Lean Software Development – the main approaches in the field of software development, history of the concept and Lean thinking are covered. Then, the main characteristics, categorization, schools of thought and principles of the concept are described. Following chapter deals with the both description and categorization of the tools usable to support the concept's principles. Finally, the advantages, disadvantages and real cases of the usage of Lean Software Development are introduced.

Keywords

Lean Software Development, Lean thinking, Agile Software Development, software, development, iteration, funkcionality

Obsah

1 Úvod

1.1 Téma práce

Práce se zabývá konceptem Lean Software Development, užívaným v oblasti vývoje softwaru. Téma pokrývá Lean Software Development od jeho historie, přes začlenění v systému konceptů a metodik vývoje softwaru obecně, různých pohledů na jeho interpretaci, jeho charakteristik, nástrojů pro podporu jeho principů, po příklady jeho užívání v praxi a možná omezení s jeho aplikací spojená.

Práce je určena všem čtenářům, kteří chtějí o tomto tématu získat ucelené a objektivní informace (tedy nejen pozitivní přínosy, ale i možná negativa spojená s použitím konceptu v praxi..

1.2 Volba tématu

Téma Lean Software Development jsem si vybrala z několika důvodů.

Jedním z nich je potenciál, který vidím v novějších konceptech a metodikách vývoje softwaru, jakožto alternativách k vývojově starším konceptům. Jedná se zejména o agilní metodiky, ke kterým se Lean Software Development nejvíce blíží. Tyto přístupy, při vhodné aplikaci, mohou být při vývoji softwaru velmi přínosné.

Druhým z důvodů výběru tohoto tématu, je existence velmi omezeného počtu česky psaných zdrojů, které se touto problematikou zabývají. I toto je impulsem k sepsání této práce – chuť podílet se na vyplnění mezery a přiblížit zájemcům tento koncept v češtině.

Posledním důvodem výběru tématu práce je koncept Lean Software Development samotný. Jeho základní kameny sahají až do doby 50. let 20. století, kdy se osvědčily v oblasti výroby, ve které dodnes nacházejí své uplatnění. Lean Software Development tak staví na historicky osvědčených principech a tím dle mého názoru roste jeho význam. Dále také fakt, že principy, původně používané ve výrobě, jsou (v modifikované podobě) aplikovány na vývoj softwaru, považuji za zajímavou konfrontaci dvou poměrně odlišných oblastí.

1.3 Cíl práce a vlastní přínos

Hlavním cílem práce a vlastním přínosem je vznik uceleného, česky psaného materiálu o konceptu Lean Software Development, který v budoucnu může napomoci lepší orientaci v této problematice, bez nutnosti znalosti cizího jazyka. Dalším cílem a vlastním přínosem je shrnutí možných přístupů ke kategorizaci tohoto konceptu a nalezení vazeb mezi konceptem Lean Software Developmenta existujícími agilními metodikami vývoje softwaru. Cílem je rovněž detailní kategorizace jednotlivých nástrojů, použitelných pro podporu principů konceptu.

Od splnění těchto cílů si slibuji větší přiblížení této problematiky české odborné veřejnosti a tím i její větší rozšíření. Nejen dostupná česky psaná literatura, ale i zastoupení konceptu Lean Software Development ve výuce na školách v České republice, je v současné době minimální. Jak ukazuje analýza provedená v [KOŠÁK, 2010], Lean Software Development se v době provádění analýzy vyučoval pouze na třech fakultách vysokých škol v ČR, z celkem 12 oslovených a to pouze v teoretické rovině.

Vzhledem k poměrně živelnému vývoji tohoto konceptu v rámci zejména zahraniční agilní/lean komunity, je cílem práce i shrnutí možných různých přístupů k jeho vnímání a definici.

1.4 Metoda dosažení cílů

Dosažení výše vytyčených cílů je realizováno studiem dostupné, převážně zahraniční, odborné literatury. Dále jsoustudovány názory a vývojové tendence napříč agilní/lean komunitou, ve snaze přiblížit a nastínit i možné alternativní přístupy. Zejména pečlivě provedené citace by pak v tomto ohledu měly zajistit přehlednost.

1.5 Předpoklady a omezení práce

Předpoklady práce žádné nejsou. Omezení spočívá pouze v nedostupnosti některých potenciálně zajímavých informačních zdrojů o dané problematice v České republice.

1.6 Struktura práce

Práce je členěna do jednotlivých kapitol, v jejichž úvodu je nastíněn jejich cíl. Na konci kapitoly čtenář nalezne krátké shrnutí, které velice stručně popisuje předešlý výklad. Kapitola 1 se věnuje tématu práce, formulaci cílů a metod jejich dosažení. Následuje kapitola 2, kterou je komentovaná rešerše prací na podobné téma. V kapitole 3 je čtenář seznámen se základní terminologií a koncepty v oblasti vývoje softwaru. Kapitola 4 popisuje historii a původní principy Lean myšlení. Kapitola 5 ukazuje různé pohledy na charakteristiku a zařazení konceptu Lean Software Development. V kapitole 6 jsou popsány principy Lean myšlení, modifikované pro oblast vývoje softwaru. Kapitola 7 se zaměřuje na výklad a přiblížení praktického využití jednotlivých nástrojů (tools) podporujících principy konceptu Lean Software Development, které jsou seskupovány do jednotlivých podkapitol dle toho, k jakému principu LeanSD se vztahují. Kapitola 8 vyústí v souhrn pozitiv a negativ spojených s využitím LeanSD v praxi a představí příklady, kdy byl koncept LeanSD použit. Kapitola 9 je věnována závěru práce – tedy shrnutí výsledků práce, přínosů autora a námětům vhodných k řešení v budoucnu.

1.7 Použitá terminologie

Navzdory tomu, že v mnoha zdrojích je ve spojení s Lean Software Development používáno označení „metodika“, používám ve své práci označení „koncept“. Je tomu tak s ohledem na možné různé dimenze pohledu na kategorizaci LeanSD. Tato problematika a její detailnější osvětlení následuje v kapitole 5.1. V práci je pro spojení Lean Software Development používána zkratka LeanSD.

2 Komentovaná rešerše

Kapitola shrnuje existující nejdůležitější zdroje, pojednávající o konceptu LeanSD a jemu příbuzných tématech formou komentované rešerše.

Výchozím zdrojem, použitým pro proniknutí do problematiky, je zdroj [BUCHALCEVOVÁ, 2009], který se zabývá metodikami budování informačních systémů a který věnuje konceptu LeanSD jednu ze subkapitol. Dalším zdrojem, pokrývajícím úvod do problematiky vývoje softwaru, je [KADLEC, 2004].

Dalšími zdroji z oblasti agilního vývoje softwaru jsou zahraniční [CHARETTE, 2003] a [HIGHSMITH, 2002]. [WOMACK, a další, 2003] se stal základním zdrojem pro proniknutí do Lean myšlení.

Pro bližší seznámení s konceptem LeanSD bylo použito zejména zahraničních zdrojů [POPPENDIECK, a další, 2003], [POPPENDIECK, a další, 2006] a [HIBBS, a další, 2009], které akcentují vazbu mezi konceptem Lean Software Development a agilním vývojem softwaru. [POPPENDIECK, a další, 2003] a [POPPENDIECK, a další, 2006] se věnují rovněž otázce nástrojů pro podporu principů LeanSD a jejich výkladu formou příběhů z praxe.

Žádný z těchto zdrojů se výrazně nezabývá kategorizací konceptu LeanSD, postavením LeanSD a AgileSD[1], ani srovnáváním různých směrů konceptu LeanSD a jejich principů. Rovněž popis nástrojů[2] formou příběhů z praxe může být, zejména pro méně zkušené čtenáře, těžko uchopitelný (je potřeba nalézat vazby mezi nástroji a konkrétním principem LeanSD, tedy ptát se „V jakém směru určitý nástroj podporuje konkrétní princip?“), proto je v kapitole 7 této bakalářské práce kladen důraz na detailnější kategorizaci jednotlivých nástrojů vzhledem k principům LeanSD.

Jedinými výraznými českými zdroji zabývajícími se konceptem Lean Software Development jsou [BUCHALCEVOVÁ, 2009] a [KADLEC, 2004], avšak nejedná se o zdroje primárně zaměřené na tento koncept.

V rámci bakalářských a diplomových prací byla dohledána pouze práce na téma „Zlepšení softwarových projektů podle Lean Software Development“[3], avšak kontakt s autorkou se bohužel nepodařilo navázat a protože práce není veřejně dostupná, bližší obsah práce není znám.

---

[1] AgileSD je zkratka pro Agile Software Development – agilní vývoj softwaru

[2] Nástroji se ve vztahu k LeanSD v této práci rozumí: praktiky, zásady, činnosti a techniky, které je možné použít k podpoření principů konceptu LeanSD

[3] Abstrakt k dispozici na adrese http://theses.cz/id/svy528/?lang=cs;furl=%2Fid%2Fsvy528%2F

3 Vývoj softwaru

Kapitola představuje úvod do konceptů a metodik vývoje softwaru. Podkapitola věnovaná agilním metodikám, ke kterým má Lean Software Development nejblíže, pak vytváří náhled na jejich obecné zásady, vyplývající z Manifestu agilního vývoje softwaru (The Agile Manifesto), představeného v roce 2001.

3.1 Základní terminologie

3.1.1 Software

Softwarem se rozumí *„Komplex programů, programových prostředků“* [GÁLA, a další, 2006 str. 468], využívaný pro podporu automatizace činností.

3.1.2 Metodika vývoje softwaru

Metodiky vývoje softwaru definuje [KADLEC, 2004] jako *„Oblast popisující, jakým způsobem by měl probíhat vývoj softwaru s cílem získat kvalitní, dobře fungující a přitom rychle a pokud možno levně vytvořený programový produkt.“*

[BUCHALCEVOVÁ, 2009 str. 17] definuje metodiky budování IS/ICT (a potažmo metodiky vývoje softwaru), podrobněji: *„Metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“*

3.1.3 Životní cyklus a model životního cyklu softwaru

„Životní cyklus softwaru je časový úsek, který začíná úmyslem vytvořit software a končí, když se software přestane používat.“ [BUCHALCEVOVÁ, 2009 str. 16]

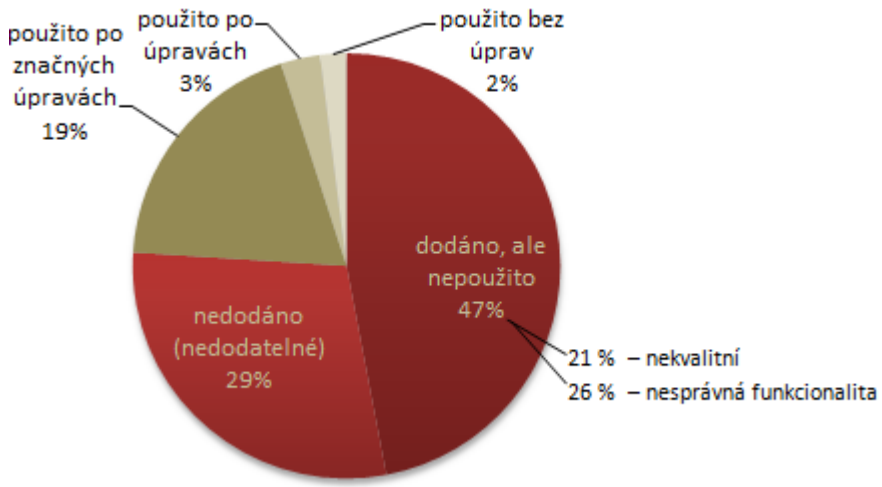
Modelem životního cyklu softwaru se rozumí *„rámec realizace procesů životního cyklu v časové posloupnosti“* [BUCHALCEVOVÁ, 2009 str. 16].

3.2 Důvody formování modelů a metodik pro vývoj softwaru

S narůstající potřebou automatizace, typickou zejména pro druhou polovinu 20. století, dochází k posunu vývoje softwaru od jednoduchých, izolovaných aplikací k robustnějším integrovaným řešením. Toto období (zejména přelom 60. a 70. let 20. století) však bylo typické neuspokojivou kvalitou vznikajícího softwaru a bývá tak nazýváno obdobím tzv. „softwarové krize“. Neuspokojivou kvalitu vznikajícího softwaru ilustruje Obrázek 1, který ukazuje, že pouze 24 % softwarových produktů bylo opravdu použito (z toho 22 % muselo projít dalšími úpravami).

[KADLEC, 2004] jako hlavní důvody vzniku softwarové krize uvádí:

- špatnou komunikaci na všech úrovních vývoje,
- nesprávný přístup osob k vývoji – snaha o seberealizaci převažující nad přáními zákazníka,
- nesprávné odhady, absence hlubších analýz (času, ceny, rozsahu, efektivity),
- špatné plánování,
- nízkou produktivita práce – nedostatečná koordinace práce programátorů,
- neznalost základních pravidel vývoje,
- podcenění hrozeb a rizik,
- nevhodně zvolené, neefektivně využívané technologie.



Obrázek 1: Ilustrace softwarové krize – neuspokojivá kvalita softwaru

Zdroj dat: [SAGE, a další, 1990 str. 20], graf: [autorka]

Metodiky vývoje softwaru a modely životního cyklu softwaru se tedy začínají formovat i z následujících důvodů:

- v reakci na výše popsané důvody, které vyústily v softwarovou krizi,
- ve snaze zajistit vývoj kvalitního softwaru i při neustále vzrůstajících požadavcích na tento vývoj (požadavky na rychlost, technologie a další, spjaté i s rozvojem webových aplikací).

I v současné době mnoho projektů vývoje softwaru končí fiaskem a zdá se, že neexistuje univerzální model či metodika, zajišťující úspěšnost. Proto se v otázce vývoje software stále objevují další možné koncepty a metodiky. Klíčovým faktorem pro jejich úspěšné použití je však citlivý výběr ve vztahu ke konkrétní situaci.

3.2.1 Vybrané modely životního cyklu softwaru

Podkapitola popisuje tři modely životního cyklu softwaru, přičemž tyto byly, pro účely co nejstručnějšího uvedení do oblasti, vybrány, jakožto nejzásadnější milníky. Tyto modely jsou zde popisovány ve snaze ilustrovat rozdíl mezi tradičními přístupy k vývoji softwaru (rigorózní metodiky), které nejčastěji stavějí na vodopádovém modelu a agilními metodikami vývoje softwaru, ke kterým má LeanSD nejblíže. Spirálový model přinesl iterace, které jsou pro LeanSD typické a proto zde bude stručně shrnut. Práce se, zejména v kapitole 7, na vodopádový model a iterace často odvolává, proto jsou zde stručně shrnuty.

Navzdory třem zde popisovaným modelům životního cyklu softwaru však existuje více a podrobněji se touto problematikou zabývá např. [BUCHALCEVOVÁ, 2009] nebo [KADLEC, 2004].

Model programuj a opravuj

Na počátku modelů, využívaných pro vývoj software, stál v 50. letech 20. století model programuj a opravuj (Code and Fix Model). Ten však je svou triviálností pro vývoj složitějšího softwaru nedostačující – je spojen pouze s minimálním počátečním návrhem a plánováním. Skládá se, mimo počáteční fáze specifikace požadavků, ze psaní kódu a z opravování chyb, které jsou v něm následně odhaleny.

Vodopádový model

Vodopádový model, který vznikl v 70. letech 20. století, přinesl, společně s modelem Stagewise z roku 1957, rozpad vývoje softwaru do více na sebe navazujících částí. Jedna z jeho možných podob je následující: specifikace požadavků, analýza, návrh, implementace, testování, zavedení, údržba [BUCHALCEVOVÁ, 2009]. Ve vodopádovém modelu existuje možnost návratu do předchozí fáze vývoje, avšak pouze pokud nebyla ukončena stávající fáze. Hlavní nevýhodou tohoto modelu je vyčlenění zákazníka z procesu vývoje (zákazník nemá možnost vstupovat do vývoje průběžně).

Spirálový model

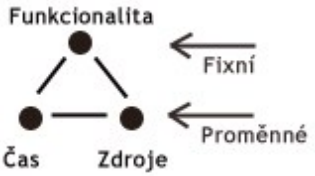
Spirálový model, který vznikl v 80. letech 20. století, přináší do životního cyklu iterativní přístup a analýzu rizik. Na rozdíl od vodopádového modelu pracuje s jinou posloupností jednotlivých fází. Zákazník je začleňován do vývoje – může měnit své požadavky, na jejichž základě se poté ve vývoji pokračuje dále. Vývoj je tedy realizován v iteracích (opakovaných krocích), ve kterých dochází ke zpřesňování požadavků [KADLEC, 2004]. Tento revoluční přístup pomáhá snížit náklady na vývoj software, protože případné nedostatky jsou odstraňovány průběžně (nikoliv zjištěny a případně odstraňovány až v konečné fázi vývoje).

3.3 Rigorózní metodiky

Rigorózní metodiky představují způsob vývoje, kdy jsou zákazníkovy požadavky na konečný software specifikovány na začátku životního cyklu a není prostor pro jejich modifikaci.

[KADLEC, 2004] Tyto metodiky bývají často založeny na výše představeném vodopádovém modelu. [BUCHALCEVOVÁ, 2009]

Jak ukazuje Obrázek 2, požadavky jsou tedy fixní, ale doba vývoje a zdroje (zejména finanční) jsou proměnné – těžko odhadnutelné. Avšak tyto metodiky dodnes nalézají své uplatnění u velkých projektů a projektů, kde není problém plně specifikovat požadavky předem.



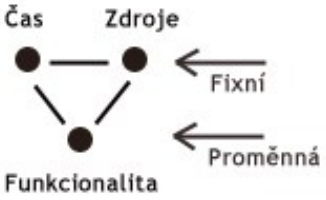
Obrázek 2: Ilustrace tradičních přístupů (zdroj: [KADLEC, 2004 str. 119]

Mezi tyto metodiky patří například Enterprise Unified Process (EUP), Rational Unified Process (RUP), OPEN a další.

3.4 Agilní metodiky

Agilní metodiky vývoje softwaru představují ve vztahu k právě popsaným, rigorózním metodikám, vývojově mladší, alternativní způsob vývoje softwaru. Dávají větší prostor zákazníkovi – jeho (měnícím se) požadavkům nejen na funkcionalitu, ale i na dobu trvání vývoje a zdroje. Flexibilita k zákazníkovým požadavkům je umožněna na, základě výše zmíněného, iterativního a inkrementálního přístupu k vývoji, který probíhá v malých, spolupracujících týmech.

Jak ukazuje Obrázek 3, v roli proměnné je v případě agilních metodik funkcionalita softwaru, která může být přizpůsobována (měnícím se) požadavkům zákazníka v průběhu vývoje. Naopak nejdelší možný čas (doba trvání vývoje) a zdroje (nejvyšší možné náklady) jsou fixní. [KADLEC, 2004]



Obrázek 3: Ilustrace agilních přístupů (zdroj: [KADLEC, 2004 str. 119]

Mezi agilní metodiky lze řadit např. Scrum, Extreme Programming, Feature Driven Development, nebo Crystal, které detailněji popisuje např. [BUCHALCEVOVÁ, 2009].

3.4.1 Manifest agilního vývoje softwaru

V agilním manifestu z roku 2001[4], sepsanému sedmnácti představiteli různých metodik vývoje softwaru, byly položeny čtyři hodnoty formující filozofické principy všech agilních metodik. Jeho základní znění je přeloženo v [BUCHALCEVOVÁ, 2009]:

„Odhalili jsme lepší způsob vývoje softwaru, sami jej používáme a chceme pomoci i ostatním, aby jej používali. Z tohoto pohledu dáváme přednost:

- *individualitám a komunikaci před procesy a nástroji,*
- *fungujícímu softwaru před podrobnou dokumentací,*
- *spolupráci se zákazníkem před sjednáváním kontraktu,*

[4] Plné znění k dispozici na http://agilemanifesto.org

- *reakci na změnu před plněním plánu.“*

3.5 Shrnutí

Společně s vyvíjejícími se a zvyšujícími se nároky na vývoj softwaru, přestává dostačovat nejmladší model životního cyklu vývoje softwaru – model programuj a opravuj. Vyskytuje se velké procento softwarových projektů, které končí neúspěchem a období přelomu 60. a 70. let 20. století je nazýváno obdobím softwarové krize.

Jako lék na důvody jejího vzniku se začínají formovat nové, propracovanější modely životního cyklu softwaru a metodiky vývoje softwaru. Mezi nejznámější modely životního cyklu softwaru patří vodopádový model, ve kterém se životní cyklus softwaru rozpadá do několika fází, avšak jeho flexibilita, vzhledem k měnícím se zákazníkovým požadavkům je prakticky nulová. Odpovědí na tyto nedostatky se stal spirálový model, který je založen na iterativním vývoji, kdy se do procesu vývoje softwaru má možnost zapojit i zákazník a může tak upravovat či zpřesňovat své požadavky.

Vodopádový model je častou základnou rigorózních metodik. Naopak iterace a inkrementální vývoj jsou prvky typické pro agilní metodiky vývoje softwaru. Velmi propracované srovnání agilních a rigorózních metodik je zpracováno v [BUCHALCEVOVÁ, 2009].

V Manifestu agilního vývoje softwaru jsou formulovány čtyři hodnoty, společné pro všechny agilní metodiky, ke kterým má Lean Software Development nejblíže a někdy mezi ně bývá i řazen (více v kap. 5.2).

4 Lean myšlení

Kapitola představuje pilíře Lean myšlení, historicky od dob jejich vzniku v oblasti výroby, pro pochopení vztahů a souvislostí, které následně umožní vnímat toto myšlení v obecné rovině a následně ho aplikovat na jiné oblasti, včetně vývoje softwaru.

Spojení Lean myšlení (Lean thinking) znamená filozofii, způsob uvažování a nazírání na okolní svět. Cílem Lean myšlení je co nejplynulejší a s minimálním plýtváním spojená tvorba hodnoty. Ačkoliv jeho základy byly položeny v japonských manažerských praktikách, spojení „Lean myšlení“ bylo představeno v roce 1990 v knize „The Machine That Changed the World: The Story of Lean Production“, ve snaze přiblížit tyto praktiky západnímu světu.

Lean myšlení je dle [WOMACK, a další, 2003] méně formálně popisováno jako cesta k tomu, aby člověk tvořil více (hodnoty) a aby při tom zároveň využíval co nejméně (zdrojů).

4.1 Toyota Production System

Historie Lean principů sahá do 50. let 20. století, kdy docházelo k jejich aplikaci v oblasti výroby. Japonská automobilka Toyota v této době čelila výzvě vyrovnat se produktivitou a výkonností sériové výrobě automobilového průmyslu Spojených států amerických, zejména pak General Motors – výrobci aut sídlícímu v Detroitu, který v té době ovládal světový trh (v roce 1955 se GM dokonce staly první firmou v USA, která vydělala 1 bilion dolarů za rok) [COHN, 2008].

Vzhledem k rozdílům japonského a amerického trhu nebylo v Toyotě možné jednoduše aplikovat americkou sériovou výrobu. K těmto rozdílům patří dle [HIBBS, a další, 2009] fakt, že japonský automobilový trh byl mnohem menší než americký a rovněž různorodost poptávky, která by sériovou výrobou nebyla uspokojena. Vznikla tedy potřeba najít jinou cestu výroby. Taiichi Ohno, viceprezident výroby v Toyotě, několikrát navštívil Detroit a vypozoroval ve výrobních procesech plýtvání (anglicky „waste“, japonsky „muda“). Na základě následně provedených experimentů pak stanovil 7 druhů tohoto plýtvání, které položily základ pro Toyota Production System (TPS):

- NADPRODUKCE
- PŘEPRAVA
- ČEKÁNÍ
- ZÁSOBY
- VADY
- ZPRACOVÁVÁNÍ
- POHYBY

Oblastmi plýtvání jsou pak obecně nazývány takové oblasti, které jsou součástí výrobního procesu, ale které přímo negenerují hodnotu pro zákazníka. Tato plýtvání by ve výrobním procesu měla být buď úplně eliminována, nebo alespoň snížena na co nejnižší úroveň.

Některá z výše identifikovaných plýtvání nejsou k nalezení striktně jen ve výrobě, ale jsou, s mírnou modifikací, vlastní i vývoji softwaru. Proto zde budou stručně shrnuta:

Nadprodukce (výroba toho, co není bezprostředně potřebné), vyžaduje skladovací prostory, což s sebou nutně přináší vznik dalších nákladů. ***Přeprava*** ve výrobním procesu, v podobě rozvozu z firmy i rozvozu uvnitř firmy, často roztrouseného díky uspořádání výrobních úseků uvnitř firmy, rovněž přináší náklady, zahrnující mimo jiné i takové položky, jako vysokozdvižné vozíky, dopravní pásy nebo paletové vozíky. ***Čekání*** přerušuje a zabraňuje pokračování výrobního procesu – jedná se například o prostoje ve schvalování (byrokratické úkony), nedostatečné množství materiálu nebo poruchy strojů. [TRILOGIQ, c2011]

Zásoby**,** které nejsou vyžadovány pro realizaci žádné aktuální výrobní objednávky, jsou plýtváním - nepřinášejí žádnou hodnotu pro zákazníka a generují další náklady.

Včasné odhalení ***závad a zmetků*** a zabránění jejich probublávání do dalších částí výrobního procesu, je dalším důležitým krokem, vedoucím k omezení plýtvání. Pokud vyráběný produkt vykazující zmetkovitost prostupuje výrobním procesem a v konečném důsledku není použitelný vůbec, nebo jen částečně, pak veškeré zdroje, použité na jeho výrobu jsou rovněž plýtváním. Plýtvání v oblasti ***zpracování*** pak plyne zejména z absence nebo nedostatečné standardizace činností, které jsou součástí výrobních procesů, popřípadě z jejich nepochopení.

Dále je plýtvání v této oblasti podporováno také odporem a negativním přístupem ke změnám, inovacím a k zlepšování procesů, často obhajovaným argumentem „Vždycky se to dělalo takhle!“

Pohyby, jejichž existence není v činnostech výrobního procesu nezbytně nutná, nebo pohyby nedostatečně efektivní, jsou plýtváním. Mezi tyto nadbytečné, nežádoucí pohyby se řadí i ty zdánlivě nepatrné, jako je například nutnost přinášet součástky a nástroje ze vzdálenějších míst na ta, ve kterých je s nimi nakládáno.

Norman Bodek, autor knih o Lean principech a Kaizenu[5], který byl jedním z prvních autorů, kteří přinesli informace o Japonském stylu managementu na západ [STRATEGOS, c2012], v průběhu let přidal další, neoficiální (není součástí koncepce Toyoty), osmou oblast plýtvání – **nedostatečné využití talentu a kreativity zaměstnanců**. Právě tento zaměstnanecký potenciál by se totiž jinak mohl podílet na zvyšování efektivity výrobního procesu.

[5] Kaizen je *„Opakovaná aktivita a změna způsobu myšlení, která musí vycházet z nejnižší operativní úrovně současně s podporou nejvyššího vedení.“* [PROCHÁZKA, 2010]

Z TPS se dále vyvinula *štíhlá výroba* (Lean manufacturing, Lean Production často také zkráceně zkráceně *Lean*) a proto mají podobné, nikoliv však identické, principy.

Dr. Jeffrey Liker v Toyota Way (koncepci TPS) uvádí, že pokud někdo chce použít Toyota Way, ve snaze „být lean“, pak je potřeba mít na paměti, že k tomu není nutně potřeba imitovat specifické nástroje užívané v Toyotě. Tyto nástroje vyžadují, aby byly vhodně aplikovány na konkrétní situaci. [KOCHNEV, 2007]

4.1.1 Podstata fungování konceptu Toyota Production System a štíhlé výroby

Cílem těchto konceptů je spokojenost zákazníka (realizovaná mimo jiné skrze flexibilní reakce na jeho požadavky), úspora nákladů a plynulost činností výrobního procesu. K dosahování těchto cílů je pak využíváno několik nástrojů a zásad. Patří mezi ně výše zmíněné omezování plýtvání, neustálé zlepšování fungování výrobních procesů, jidoka, JIT, pull princip, kanban, ale i například vizualizace (informace určené ke sdílení jsou jasně znázorňovány).

Štíhlá výroba a TPS některé tyto nástroje sdílejí a jsou rovněž vlastní konceptu Lean Software Development. Z tohoto důvodu a pro souvislé uvedení do Lean myšlení, pokládám za vhodné zde stručně shrnout alespoň následující:

JIT

JIT (Just In Time) je metoda pro plánování a řízení výroby [BASL, a další, 2008], orientovaná na uspokojení požadavků zákazníka (na včasnou dodávku správných položek objednávky). JIT je *tažný (pull) systém*, Dle JIT by, v rámci minimalizace zásob a větší flexibility výrobního procesu, mělo být vyráběno pouze to, co je v danou chvíli skutečně vyžadováno (co je skutečně potřeba vyrobit), za pomoci přesného časového plánování.

Kanban

V rámci JIT je pak využíváno Kanbanové signalizace – Kanbanových karet, jejichž prostřednictvím je vysílán požadavek na výrobu – výroba tak započne až ve chvíli, kdy některá z částí výrobního procesu signalizuje potřebu. Tyto karty mohou být realizovány elektronicky, nebo tradičním způsobem, kdy např. paleta se součástkami obsahuje Kanbanovou kartu s informacemi o těchto součástkách (dle [Lean company, c2010] se jedná např. o informace o produktu/materiálu, o způsobu zásobování, o místu skladování a další). Mnohdy také obsahují čárové kódy pro lepší strojovou zpracovatelnost). V případě, že dojde k vyčerpání množství těchto součástek na určitém pracovišti, je tato paleta, společně se svou kanbanovou kartou, odeslána na příslušnou část výroby, která dle informací na kanbanové kartě vyrobí nové součástky a nahradí původní paletu, která je, opět s přiloženou kanbanovou kartou, odeslána na pracoviště, které signalizovalo vyčerpání součástek na paletě.

Jidoka

Jidoka je koncept, který detekuje odchylky a defekty ve výrobě, čímž je na ně umožněno okamžitě reagovat. Při odhalení odchylky nebo defektu při činnosti ve výrobním procesu je tato činnost okamžitě (v případě stroje – automaticky) zastavena. Následně je tento problém signalizován, ve snaze zabránit případnému probublání vadného rozpracovaného výrobku do dalších stádií výroby. [HEFNEROVÁ, 2012]

„Operátor zjistil, že díl nepasuje na místo, kam má být namontován. Zatažením za lano signalizuje problém. Na světelné tabuli andon se ihned rozsvítí číslo pracovní pozice, kde k problému došlo a rozezní se zvukový signál.“ [Toyota Peugeot Citroën Automobile Czech, c2003-2007]

4.2 Pět principů Lean myšlení

[WOMACK, a další, 2003] shrnuje následujících pět principů Lean myšlení:

1. **DEFINICE HODNOTY**
2. **IDENTIFIKACE ČÁSTÍ HODNOTOVÉHO TOKU**
3. **TVORBA HODNOTOVÉHO TOKU**
4. **PRINCIP TAHU**
5. **DOSAHOVÁNÍ DOKONALOSTI**

Zavádění Lean myšlení musí začínat ***definováním hodnoty*** – je potřeba určit, co přesně přináší koncovému zákazníkovi hodnotu, z hlediska konkrétního produktu – tedy „misi“ [PROCHÁZKA, 2010] produktu. Přesná specifikace této hodnoty je rozhodujícím prvním krokem – vytvoření produktu nebo služby, který se neshoduje s potřebami zákazníka (nejen z hlediska funkcionality, ale i v otázce ceny, dostupnosti a dalších faktorů), byť správným, ve firmě zavedeným způsobem, je plýtváním. Aktuální potřeby akcionářů, finanční pohnutky vrcholových manažerů, stávající aktiva, nebo technologie by neměly být upřednostňovány před vytvářením hodnoty pro zákazníka – často je tedy třeba přehodnotit od základů výrobní proces, který bývá těmito prvky zatížen a zabraňuje zlepšování. [WOMACK, a další, 2003]

Hodnotový tok je soubor všech činností, které vedou k vytvoření konečného produktu nebo služby. Identifikace jednotlivých prvků tohoto toku přináší, dle [WOMACK, a další, 2003] následující tři jejich typy:

a) Prvky tvořící hodnotu

b) Prvky bezprostředně nevytvářející hodnotu, ale nevyhnutelné

c) Prvky nevytvářející hodnotu a okamžitě odstranitelné

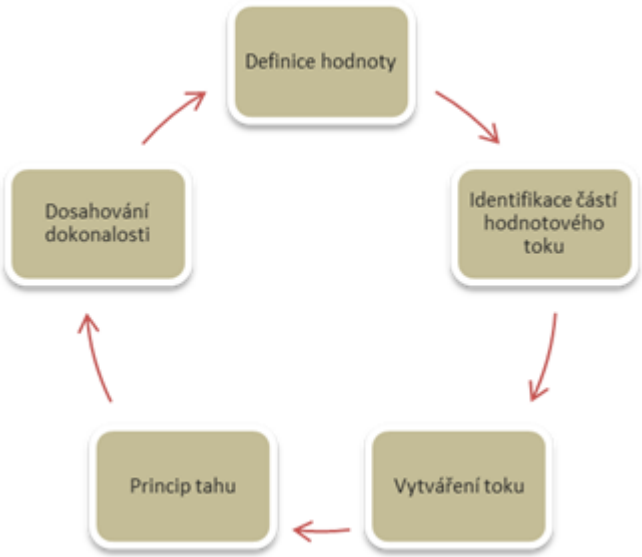
Následně je žádoucí eliminovat každou část, která nevytváří hodnotu (zamezit plýtvání) a která není ve výrobním procesu nevyhnutelná.

Tvorba toku spočívá v realizaci těch kroků, které tvoří hodnotu v těsných a integrovaných sekvencích tak, aby produkt plynule směřoval k zákazníkovi. Tradiční prostupování výrobků odděleními a jejich zpracovávání v dávkách nemusí být vždy nejefektivnějším způsobem, jelikož základním cílem Lean myšlení je vytvořit každý jednotlivý výrobek co nejrychleji s co nejmenším úsilím – při dávkovém zpracování výrobek ale čeká, až budou zpracovány i ostatní produkty v dávce. Až poté je možný přesun celé dávky do další části výroby, což značně zpomaluje výrobu konkrétního výrobku. Naproti tomu kontinuální vytváření jednoho výrobku je rychlejší, plynulejší a efektivnější.

Princip tahu byl stručně popsán v kapitole 4.1.1. Podstatou jeho zavedení je vytvářet výrobek pouze pokud ho zákazník opravdu chce.

Výrobek je tažen od zákazníka, na základě jeho potřeby, na rozdíl od principu tlaku, kdy jsou výrobky tlačeny směrem k zákazníkovi, aniž by je nutně vyžadoval. Zákazník je zde chápan jak v širším (zákazník určitého procesu) i užším (koncový zákazník) významu. [BASL, a další, 2008]

Snaha o ***dosažení dokonalosti*** v hodnotovém toku, skrze jeho neustálé zlepšování a pokračování v hledání a eliminaci plýtvání, uzavírá pět principů Lean myšlení a vytváří z nich uzavřený koloběh.



Obrázek 4: Koloběh pěti principů Lean myšlení [autorka]

4.3 Shrnutí

Lean myšlení i koncept řízení výroby – štíhlá výroba se vyvinuly a čerpají z japonského stylu řízení, který byl uplatňován v japonské automobilce Toyota v období po druhé světové válce. Lean myšlení je filozofie, jejíchž pět principů lze uplatňovat v mnoha oblastech, mezi které patří i vývoj softwaru.

Správná definice hodnoty napomáhá tomu, aby byl vyroben správný produkt, který uspokojí zákazníka. Aby bylo možné uspokojit potřeby koncového zákazníka, je nutné vědět kdo jím skutečně je a jaké jsou jeho potřeby. Aby bylo možné detekovat a eliminovat plýtvání nejen ve výrobním procesu, ale i v procesu vývoje softwaru, je potřeba důkladně znát hodnotový tok (jednotlivé kroky a jejich provádění) tvorby hodnoty. Pro zajištění plynulosti a efektivity je často potřeba změnit provádění částí hodnotového toku z tradičního dávkového přístupu na kontinuální zpracování konkrétního výrobku. Princip tahu zásadně omezuje plýtvání, jinak spouštěné nadprodukcí. Neustálé zlepšování a snaha o dosažení dokonalosti spojuje všech pět principů v cyklus.

Zatímco první z principů se zaměřuje na to, aby byl vyroben správný výrobek, ostatní čtyři se věnují tomu, jak docílit, aby byl výrobek vyroben správně. [MIDDLETON, a další, 2005]

5 Lean Software Development

Tato kapitola se věnuje obecné charakteristice LeanSD a jeho zařazení v systému přístupů k vývoji softwaru.

Lean (česky štíhlý) v názvu konceptu LeanSD obecně (nejen v otázce vývoje softwaru, ale např. i ve výrobě a dalších oblastech) symbolizuje zaměření na odstranění plýtvání skrze efektivní využívání zdrojů, ve snaze o minimalizaci nákladů a o co nejrychlejší a nejflexibilnější uspokojení zákazníkových potřeb.

5.1 Interpretace konceptu Lean Software development

S jistotou lze napsat, že Lean Software Development aplikuje Lean myšlení na vývoj softwaru. Jednoznačnou odpověď na otázku „Co je to Lean Software Development?“ je ale poměrně těžké najít.

Existují dva výrazné tábory v otázce definice LeanSD – zjednodušeně lze říci, že první tábor nazývá LeanSD metodikou vývoje softwaru a druhý tábor označení „metodika“ odmítá a LeanSD pokládá za myšlenkový směr, jenž na vývoj softwaru aplikuje množinu principů a známých nástrojů.

Pokud uvažujeme následující definici pojmu metodika:

„Metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“ [BUCHALCEVOVÁ, 2009 str. 17]

A následně provedeme srovnání této definice s konceptem LeanSD, nedá se říci, že by LeanSD definoval principy, procesy, role, techniky, nástroje ani produkty. Lze však uvažovat nad tím, že definuje jakýsi rámec praktik, v podobě sedmi principů, v jejichž duchu by měl vývoj softwaru probíhat.

„Praktiky, na rozdíl od procesů, více akcentují realitu a primárně se zaměřují na lidi. Zatímco procesy vyjadřují explicitní (popsanou) znalost, praktiky zapouzdřují interní „tacit“ znalosti.“ [BUCHALCEVOVÁ, 2009 str. 56]

Byť jsou v kapitole 7 popisovány nástroje LeanSD, koncept sám o sobě žádné nástroje nedefinuje. Avšak mnoho nástrojů, typických zejména pro již existující agilní metodiky, lze využít pro podporu principů konceptu LeanSD.

Vzhledem k tomu, že tedy neexistuje jednotná definice konceptu LeanSD, budou v této části shrnuty možné pohledy na interpretaci LeanSD, které se vyskytují napříč různými zdroji:

a) **Sada principů, aplikovaných na jiné metodiky**

LeanSD je z tohoto pohledu vnímán jako sada principů, které jsou aplikovány jako podpůrné principy na známé, již existující metodiky vývoje softwaru. [HIBBS, a další, 2009] pak analogicky uvádí, že LeanSD pokládá praktiky agilních metodik za validní a podpůrné.

LeanSD je přizpůsobivé pojetí vývoje, které se řídí obecným souborem principů, hodnot a praktik – jedná se tedy spíše o filozofii a soubor principů, než o proces vývoje definovaný krok po kroku či metodiku. [PANNONE, 2010] [COHEN, a další, 2004]

[HIBBS, a další, 2009] uvádí, že při zavádění LeanSD je běžné jako startovní bod použít některou z agilních metodik, např. Scrum.

Nástroje, které při vývoji využívá (např. mapování hodnotového toku a další), se týkají spíše zlepšování procesu vývoje. LeanSD tedy dle [KELLY, 2009] není ani tak metodika vývoje, jakožto spíše metoda pro zlepšování stávajících postupů vývoje.

b) **Samostatná metodika vývoje softwaru**

LeanSD je z tohoto pohledu vnímán jako samostatná metodika vývoje softwaru, řazená do skupiny agilních metodik. [HIGHSMITH, 2002], [Agile Sherpa, 2010], [KADLEC, 2004] Avšak, jak už bylo výše řečeno, LeanSD se tradičním agilním metodikám vymyká – nelze ho jednoduše připodobnit k ostatním agilním metodikám, ani stávajícím životním cyklům vývoje.

5.1.1 Směry konceptu Lean Software Development

[LADAS, 2009] a [ANDERSON, 2011] dělí přístupy k LeanSD do myšlenkových směrů, kterými jsou: a) metaforický směr, b) směr Lean Software and Systems Consortium (LeanSSC)[6]. Hlavní rozdíl mezi jednotlivými směry spočívá v „šířce“ využití LeanSD. I přesto, že oba směry v rámci svých principů akcentují vnímání celku, metaforický směr se orientuje primárně na oblast vývoje softwaru (doporučuje základní principy pro vývoj softwaru). Směr LeanSSC více vnímá, mimo vývoje softwaru, i hledisko řízení a byznysu, ve kterém je vývoj zasazen. Směr LeanSSC je oproti metaforickému směru více spojen s pojmem „Lean podnik“.

1. **Metaforický směr LeanSD**

Metaforický směr získal svůj název díky přenášení a modifikaci původních principů Lean myšlení ve výrobě na oblast vývoje softwaru. Zastánci metaforické větve

[6] Nezisková organizace na podporu podniků závislých na vývoji softwaru. [Lean Software and Systems Consortium , 2012]

přenášejí Lean principy na původní metodiky vývoje softwaru – při využívání metodik jako je Scrum či Extrémní programování jsou využívány principy LeanSD, a někdy je na jejich základě konkrétní metodika odpovídajícím způsobem modifikována. Za průkopníky tohoto přístupu jsou považováni manželé Mary a Tom Poppendieck, kteří upravili původních 5 Lean principů, tak aby odpovídaly potřebám vývoje softwaru. Tento směr je oproti směru LeanSSC, popsanému níže, více soustředěn na oblast vývoje softwaru.

V souvislosti s tímto směrem se také stále častěji objevuje pojem Lean-Agile development.

2. **Směr Lean Software & Systems Consortium**

Tento směr více prorůstá mimo oblast vývoje softwaru do ostatních částí podniku, zaměřuje se na integraci vývoje softwaru do „Lean podniku“. [SCOTLAND, 2009]

Zatímco metaforický směr, jakožto kombinace Lean principů a většinou agilních nástrojů se zaměřuje primárně na vývoj software a vývojářské týmy, směr LeanSSC pokrývá hlouběji i oblast organizační úrovně a podniku jako celku. [SCOTLAND, 2010] Směr LeanSSC má tedy širší záběr – zaměřuje se více na byznys kontext vývoje a management. [ANDERSON, 2010]

Pokud jsou uvažovány následující faktory, kterými se zabývá softwarové inženýrství, pak směr LeanSSC klade oproti metaforickému směru větší důraz na druhý bod:

- *„technické aspekty zahrnující počítačovou infrastrukturu a softwarové vybavení;*
- ***netechnické aspekty dané organizační strukturou organizace vyvíjející daný produkt a jejími ekonomickými možnostmi;***
- *znalostmi z oblasti specifikace požadavků na softwarový produkt, jeho analýzy, návrhu, implementace, testování a na konec také instalace u zákazníka;*
- *lidské zdroje schopné aplikovat výše uvedené znalosti a uplatnit je tak při realizaci softwarového systému;*
- *řízení spjaté s vývojem samotného produktu umožňující efektivní využití všech výše uvedených faktorů s cílem vytvořit produkt požadované kvality.“* [VONDRÁK, 2002]

Směr LeanSSC definuje vlastní základní principy, které se mírně liší od principů metaforického směru a které jsou blíže popsány v kapitole 6.2.

Mezi zástupce této větve patří např. Donald Reinertsen, Peter Middleton nebo James Sutton.

5.2 Postavení konceptů Lean Software Development a Agile Software Development

Rovněž v otázce hierarchie LeanSD a AgileSD (popsaného v kapitole 3.4), se zdroje často rozcházejí. V předchozí kapitole bylo vysvětleno, že TPS byl základem pro formování pěti principů Lean myšlení. Principy Lean myšlení mají ale mnoho společného s principy agilního vývoje softwaru, čímž vzniká značná provázanost a na hierarchii LeanSD a AgileSD tak existuje více možných pohledů, přičemž se ale tyto pohledy nemusí nutně vylučovat a nelze jednoznačně tvrdit, že některý z nich je nesprávný.

LeanSD může být vnímán jako:

a) **Samostatná disciplína.**

   LeanSD a AgileSD jako dva, byť provázané, ale vedle sebe stojící koncepty, sdílející určité hodnoty ale neshodující se zcela. Detailnější popis rozdílů popisuje subkapitola 5.2.1.

b) **Podmnožina agilního vývoje softwaru.**

   LeanSD může být vnímán jako koncept, který se sice vyvinul z principů Lean myšlení, ale mezistupněm tohoto vývoje byl právě agilní vývoj software – jeho vývoj částečně staví na agilním manifestu.

   Objevují se ale i názory, že vnímání LeanSD, jakožto jednu z agilních metodik odporuje Lean myšlení, protože LeanSD pak může být nesprávně vnímán jako proces vývoje softwaru. [BELLWARE, 2009]

5.2.1 Lean Software Development jako samostatná disciplína

Dle tohoto přístupu jsou LeanSD a Agile dva velice úzce provázané zastřešující pojmy pro principy a praktiky vývoje softwaru. Nejsou však zcela identické. LeanSD se zaměřuje na dosažení efektivity prostřednictvím eliminace plýtvání a respektování ostatních (zefektivnění celého hodnotového toku → uspokojení zákazníka), kdežto agilní vývoj softwaru si klade za cíl vyhovět měnícím se potřebám zákazníka prostřednictvím rychlého dodání softwaru (větší důraz na rychlé uspokojení zákazníka a jeho zapojení, než na samotný hodnotový tok).

Agilita (ang. Agility, česky také mrštnost, hbitost) je pak vedlejším produktem LeanSD, vznikající díky rychlým cyklům vývoje, které jsou nutné k identifikaci a eliminaci plýtvání. [BECK, 2009]

LeanSD také, dle [HIBBS, a další, 2009] oproti AgileSD, více vnímá i byznys okolí, ve kterém je software vyvíjen – principy LeanSD mohou být využívány nejen v částech podniku věnujících se vývoji softwaru, ale i v dalších částech a dokonce i za hranicemi podniku.

Hlavním rozdílem mezi AgileSD a LeanSD je dle [CAWLEY , a další, 2010] ten, že AgileSD uplatňuje přístup zdola-nahoru, kdežto LeanSD uplatňuje přístup shora-dolů. Tento rozdíl upřesňuje [PROCHÁZKA, 2010]: *„Agilní přístupy nám pomáhají zdola nahoru (motivace*

lidí, samo řízení, agilní inženýrské praktiky), kdežto Lean spíše shora dolů (definice hodnoty – vize, mise, služeb, produktů – a snaha o její doručení; změna kultury řízení, nastavení prostředí pro tuto změnu).“

[SIMS, 2008] naproti tomu uvádí, že mnoho lidí, kteří se podíleli na formování dnešních agilních metodik, bylo ovlivněno právě původními principy Lean myšlení (ilustruje Obrázek 5). Od tohoto tvrzení se pak odvíjí fakt, že Agile a Lean Software Development mají mnoho společných prvků a tím pádem Lean Software Development je vždy trochu agilní a Agile Software Development je vždy trochu Lean. Vyvrací tak tedy teorii, že Lean a Agile jsou dva koncepty, stojící izolovaně vedle sebe a odmítá smysluplnost otázky, zda se v konkrétním projektu rozhodnout pro agilní nebo Lean cestu vývoje softwaru.

Obrázek 5: Formování agilních metodik bylo ovlivněno Lean myšlením. [autorka], inspirováno [ABILLA , 2007]

5.3 Shrnutí

LeanSD je aplikací principů Lean myšlení na oblast vývoje softwaru. LeanSD je někdy charakterizován, jako agilní metodika vývoje softwaru, avšak od ostatních agilních metodik se poměrně odlišuje. Proto bývá někdy definován spíše jako filozofie, či soubor principů pro vývoj softwaru, než jako metodika.

Ve vztahu k LeanSD se objevují dva základní směry, které se liší svým zaměřením. Metaforický směr LeanSD je primárně zaměřen na oblast vývoje softwaru, kdežto směr LeanSSC se více zabývá i stránkou managementu v rámci celého podniku a byznys okolím, ve kterém je tento vývoj zasazen.

S pojmem LeanSD úzce souvisí AgileSD – oba koncepty mají podobné cíle a prvky, avšak nejedná se o zcela identické koncepty. Na LeanSD lze nahlížet jako na podmnožinu AgileSD, jím ovlivněnou, nebo jako na samostatnou disciplínu, přičemž platí, že oba koncepty se mohou kombinovat.

6 Principy konceptu Lean Software Development

Kapitola popisuje principy konceptu LeanSD, které pro účely vývoje softwaru vznikly modifikací původních principů Lean myšlení, užívaných ve výrobě (popsaných v kapitole 4.2). Nástroje, použitelné pro podporu těchto principů v praxi, jsou popsány později, v kapitole 7.2.

6.1 7 principů – metaforický směr

Sedm principů konceptu LeanSD bylo nejdříve shrnuto v [POPPENDIECK, a další, 2003] a následně mírně upraveno v [POPPENDIECK, a další, 2006]. Těmito principy jsou následující:

- **ELIMINOVAT PLÝTVÁNÍ**
- **VČLEŇOVAT KVALITU DO VÝVOJE**
- **VYTVÁŘET ZNALOSTI**
- **ODKLÁDAT ZÁVAZKY**
- **DODÁVAT CO NEJRYCHLEJI**
- **DŮVĚRA A RESPEKT K LIDEM PODÍLEJÍCÍM SE NA VÝVOJI**
- **OPTIMALIZOVAT CELEK**

[POPPENDIECK, a další, c2010] uvádí ještě méně známou (je obsažena pouze v tomto zdroji), novější podobu principů – princip „Vytvářet znalosti“ se mění na „Zlepšovat se“, princip „Odkládat závazky“ se mění na „Neustále se učit“ a „Důvěra a respekt k lidem podílejícím se na vývoji“ se mění na „Zapojit všechny“, avšak obsahově se popis principů příliš nemění.

6.1.1 Princip 1 – Eliminovat plýtvání

Plýtvání v oblasti vývoje softwaru lze, podobně jako plýtvání ve výrobě (popsané v kapitole 4.1), rozdělit do následujících sedmi oblastí, které shrnuje Tabulka 1. Každá oblast je následně stručně vysvětlena (v čem plýtvání spočívá), přičemž nástroje pro eliminaci plýtvání v těchto oblastech jsou představeny v kapitole 7.2.

Tabulka 1: Druhy plýtvání, vytvořeno autorkou dle [POPPENDIECK, a další, 2006]

VÝROBA	VÝVOJ SOFTWARU
Nadprodukce	Funkcionalita
Přeprava	Předávky artefaktů mezi rolemi
Zásoby	Neúplná, nedokončená práce
Zpracovávání	Nutnost znovu se učit zapomenuté
Pohyby	Střídání úkolů
Čekání	Prodlevy
Vady	Chyby

Funkcionalita

Typickou oblastí plýtvání v oblasti softwaru je (zákazníkem) nevyžádaná nebo nevyužívaná funkcionalita, která vzniká nejčastěji z následujících důvodů:

- Přidávání funkcionality plynoucí z přesvědčení, že „toto bude zákazník zcela určitě také potřebovat (i když si o to bezprostředně neřekl)“, tedy přístup Just-In-Case [ABRAHAMSSON, a další, 2009 str. 265] namísto přístupu Just-In-Time.
- Vývoj řízený životopisem – Resume Driven Development (RDD) – způsob vývoje, kdy manažeři či programátoři prosazují při vývoji využívání určitých nástrojů, technik nebo technologií, aby obohatili kolonku zkušeností ve svém životopise, namísto toho, aby efektivně řešili skutečný zákazníkův problém, k čemuž jsou často dostačující metody již v podniku zavedené. [SHAH, 2007]
- Příkaz k začlenění funkcionality přišel „shora“ – nereflektuje potřeby zákazníka, ale interní potřeby firmy, případně zájmy různých zainteresovaných stran.

Takováto nevyžádaná funkcionalita je plýtváním, protože každý řádek jejího kódu navyšuje náklady a čas potřebný na vývoj a údržbu. Rovněž je potenciálně spojena i s náklady a časem potřebným na její případné odstranění. Také nevyužívaná funkcionalita pro zákazníka nepředstavuje hodnotu – pokud neexistuje jasná představa o využití funkcionality, pak by neměla být do projektu začleňována.

Předávky artefaktů mezi rolemi

Každá předávka artefaktu mezi rolemi (typicky řetězec Analytici-Návrháři-Programátoři-Testeři) s sebou přináší ztrátu tzv. „tacitní znalosti“ – vnitřní znalosti, která je těžko popsatelná a přenositelná. [POPPENDIECK, a další, 2006] uvádí, že s každou předávkou mezi rolemi se ztrácí asi 50 % tacitní znalosti, což by při třech předávkách znamenalo, že k poslednímu článku řetězce se dostane pouhých 12 % tacitní znalosti.

Neúplná, nedokončená práce

Neúplná, nedokončená práce (typicky nezdokumentovaná, neotestovaná či nedodaná funkcionalita) je spojena s rizikem – čím déle setrvává nedokončená, tím je náchylnější k tomu stát se zastaralou (ve chvíli, kdy je vývoj ukončen už nemusí reflektovat aktuální zákazníkovu situaci). Namísto hromadění takové práce je potřeba soustředit se na kontinuální vývoj.

Nutnost znovu se učit zapomenuté

V kapitole 6.1 bylo řečeno, že jedním z principů konceptu Lean Software Development je tvorba znalostí. Pokud však byla určitá znalost vytvořena a následně zapomenuta – protože např. nebyla vhodně či vůbec zdokumentována (při snaze ji využít, vznikla potřeba se ji znovu naučit), pak původní první naučení se, ve skutečnosti nepřineslo očekávanou hodnotu.

Střídání úkolů

V případě, kdy je určitý jedinec součástí několika různých úkolů, které je potřeba plnit ve stejném časovém období, vzniká plýtvání spojené s nutností „přepínat“ se z jednoho úkolu do druhého. Jedná se v tomto případě jednak o plýtvání v podobě času, který negeneruje hodnotu, ale který je potřebný k opětovnému zorientování se v úkolu, ale také o plýtvání spojené s potenciálním celkovým prodloužením dob vývoje těchto paralelních úkolů, plynoucí ze sdílení stejných zdrojů.

[POPPENDIECK, a další, 2006] demonstruje tento druh plýtvání na existenci tří různých úloh vývoje softwaru (úloha A, B a C), přičemž na realizaci každé z nich je potřeba týden času. V případě, že realizace další úlohy začíná až po úplném skončení realizace úlohy předešlé, jsou všechny úlohy hotové za tři týdny – v každém týdnu je dokončena jedna z nich a po každém jednotlivém týdnu je tak z pohledu zákazníka generována hodnota.

V opačném případě, kdy práce na těchto úlohách běží současně a jsou při ní využívány stejné zdroje a dochází ke střídání úkolů, nebude po třech týdnech hotová žádná z úloh, a tedy z pohledu zákazníka, není generována hodnota.

Prodlevy

Prodlevy v realizaci úloh plynou především z přirozené potřeby konat při nich rozhodnutí. Pokud vývojář nemá dostatek informací nutných k učinění určitého rozhodnutí, pak vzniká plýtvání – hrozí, že rozhodnutí bude učiněno i bez potřebných informací, že se rozhodnutí odloží a vývojář se přepne do jiné úlohy (vznik nedokončené práce a střídání úkolů), nebo že se vývojář bude snažit chybějící informace zjistit.

Pokud ke zjišťování takovýchto informací nemá vhodné podmínky – zjištění je např. závislé na dalších osobách, dokumentech, organizačních místech, které jsou obtížně dostupné, pak vzniká plýtvání.

Chyby

Chyby, které se vyskytnou v procesu vývoje softwaru, jsou tím nebezpečnější, čím později jsou odhaleny – jsou spojeny s plýtváním plynoucím z jejich oprav.

6.1.2 Princip 2 – Včleňovat kvalitu do vývoje

Kvalita by do kódu měla být včleňována od začátku – přístup, kdy chyby a nedostatky prostupují procesem vývoje a jsou opravovány až na jeho konci, ústí v rozsáhlé napravování těchto vad a tím v prodlužování doby vývoje. Ideální je vadám předcházet – avšak, pokud se nějaká objeví, pak je potřeba ji opravit co nejrychleji a udělat vše pro to, aby se neopakovala.

6.1.3 Princip 3 – Vytvářet znalosti

Důležitým principem LeanSD je vytváření a zaznamenávání znalostí. Klasickým přístupem, typickým pro vodopádový model životního cyklu softwaru, je sepsání požadavků a potažmo vytvoření návrhu vyvíjeného softwaru ještě před započetím samotného vývoje.

Vývoj softwaru je ale zasazen v proměnlivém prostředí, ve kterém se v průběhu vývoje otevírají různé cesty jak postupovat, přičemž možnost výběru té cesty, byť lišící se od té předem definované v požadavcích, může přinést lepší výsledek. Může tak docházet k plýtvání, protože např. požadavky se mohou (a pravděpodobně budou) ještě měnit. Pokud je vytvořena nová znalost (došlo k rozhodnutí vydat se určitou cestou), je potřeba zaznamenat jaká a na základě jakých důvodu. Zároveň, ve snaze eliminovat plýtvání v oblasti prostojů, je vhodné umisťovat informace o vytvořených znalostech co nejblíže ke zdroji (ideálně v podobě komentářů do kódu).

6.1.4 Princip 4 – Odkládat závazky

Odkládání závazků (závazných rozhodnutí) na nejpozdější možný moment, umožňuje dělat taková rozhodnutí, která budou více odrážet aktuální realitu – je vhodnější vyvíjet na základě opravdových, aktuálních informací, než na základě nejistých předpovědí. Zároveň tak lze získat čas pro vyzkoušení více alternativních cest, kterými se lze vydat, což opět umožňuje získat více informací o tom, která varianta je tou nejvýhodnější.

6.1.5 Princip 5 – Dodávat co nejrychleji

Rychlé dodávky (podpořené vývojem v častých a krátkých iteracích) umožňují lepší propojení se zákazníkovými skutečnými potřebami. Iterativní a inkrementální vývoj je výhodou zejména z toho důvodu, že zákazník nemusí pracovat s představami a domněnkami, ale namísto toho s reálně fungující částí celku, na základě které snadněji vyhodnotí, zda vývoj směřuje správným směrem, nebo zda je potřeba upravit požadavky – lze tak včas předejít plýtvání plynoucímu z dodání nechtěné, či nepoužívané funkcionality.

6.1.6 Princip 6 – Důvěra a respekt k lidem podílejícím se na vývoji

Vzhledem ke kreativní povaze profese vývojáře, je potřeba, aby měl pocit, že pracuje v prostředí, kde je tato kreativita a její rozvoj podporován. Je potřeba, aby těmto lidem bylo naslouchánо a v oprávněných případech důvěřováno (svěřit jim pravomoc činit rozhodnutí).

6.1.7 Princip 7 – Optimalizovat celek

Z hlediska zákazníka je nutné, aby jednotlivé části softwaru fungovaly jako celek – pokud jsou jednotlivé části funkční, ale nefungují jako celek ideálně, pak převážily lokální optimalizace nad optimalizací celku. Je tedy potřeba se lokálních optimalizací vyvarovat.

6.2 Principy směru Lean Software & Systems Consortium

Za zakladatele konceptu LeanSD je považován Robert Charette, který v 90. letech definoval a v [CHARETTE, 2003] shrnul 12 principů LeanSD. Po založení organizace a potažmo i směru Lean Software & Systems Consortium (LeanSSC) v roce 2009, mezi jejíž reprezentanty Robert Charette patří, bylo touto organizací stanoveno 8 principů, kterými jsou, dle [Lean Software and Systems Consortium, 2012], následující:

Tabulka 2: Osm principů LeanSSC (vytvořeno autorkou dle [ANDERSON, 2011])

PRINCIP	PODSTATA
Myslet a navrhovat systémově	Ekvivalent k principu „Optimalizovat celek“ – vyvarovat se lokálním optimalizacím, vnímat celek.
Adaptabilní vývojové procesy	Vývoj softwaru není deterministická činnost – proces, který není adaptabilní, nebude schopen adekvátně reagovat na nepředvídané události.
Respekt k lidem podílejícím se na vývoji	Lidé, kteří se na vývoji skutečně reálně podílejí, vědí o své práci nejvíce (např. v porovnání s top managementem) – proto by mělo být naslouchánо jejich názorům a důvěřováno jejich rozhodnutím (v oprávněných případech).
Pro zlepšování používat vědecké metody	Používat metody jako je spektrální analýza, teorie omezení (TOC) a další. Na základě nasbíraných kvantitativních dat odhadovat, jaké důsledky bude mít pro systém určitá změna.
Podporovat spoluúčast pracovníků na vedení	Dát pracovníkům napříč organizací možnost podílet se na vedení (přispívat k vizi, strategii, taktice a dalším složkám).
Vizualizované a jasně popsané pracovní procesy	Pokud není zřetelné, jakým způsobem určitý proces funguje, pak je prakticky nemožné ho řídit či zlepšovat. Je potřeba vědět a vizualizovat, jak a s jakými zásadami je určitá práce prováděna a jak prostupuje procesem vývoje.

PRINCIP	PODSTATA
Minimalizovat celkový čas toku	Sledování a měření času potřebného k prostupu práce procesem vývoje, má podobné důvody jako oblast plýtvání „Neúplná, nedokončená práce“ – kratší vývojové cykly dříve odhalí chyby a zamezí, či eliminují plýtvání vznikající z důvodu nutnosti odstraňování těchto chyb. Kratší vývojové cykly adresují rovněž „byznys“ důvody, protože hodnota má být generována v tu pravou chvíli a odrážet aktuální potřeby zákazníka.
Eliminovat plýtvání	Poměr práce generující z pohledu zákazníka hodnotu a celkové práce (včetně práce hodnotu negenerující), určuje úroveň efektivity procesu vývoje. Za maximálně efektivní je považován vývoj vykazující nulové transakční náklady, nulové náklady na koordinaci a žádná selhání (chyby).

LeanSSC však, navzdory definici principů vlastních, vnímá principy metaforického směru jako validní. [Net Objectives, 2010]

6.3 Shrnutí

Kapitola popsala různé kategorizace Lean principů, ve snaze utřídit možnosti podob jejich výskytu napříč zdroji.

Původně bylo pro koncept LeanSD Robertem Charettem definováno 12 principů, které posloužily jako základ pro novějších 8 principů, definovaných organizací Lean Software & Systems Consortium (LeanSSC). Výrazněji se tak formuje směr LeanSSC. Metaforický směr konceptu LeanSD, reprezentovaný zejména manželi Poppendieckovými, definuje 7 principů.

Ačkoliv se některé principy metaforického a LeanSSC směru shodují (eliminace plýtvání, respekt k lidem a optimalizace celku), jak už bylo řečeno v kapitole 5.1.1, směr LeanSSC je více spjat s managementem a akcentuje prorůstání Lean principů do celé organizace.

Další části práce, věnující se popisu nástrojů užívaných konceptem LeanSD, budou již popisovány z hlediska metaforické větve.

7 Nástroje pro podporu principů konceptu Lean Software Development

Jak už bylo řečeno v kapitole 5.1, LeanSD nepředepisuje žádná závazná pravidla, ani nedefinuje určitý postup vývoje. Cílem je, aby se vývoj nesl v duchu Lean principů. Jelikož je tato bakalářská práce, touto částí počínaje, primárně zaměřena na koncept LeanSD z pohledu metaforické větve, pak se jedná o sedm principů, popsaných v kapitole 6.1.

Nástroji se pro účely této kapitoly rozumí praktiky, zásady, činnosti a techniky, které je možné použít k podpoření principů metaforického směru konceptu LeanSD – jejich popis a kategorizaci si tato kapitola klade za cíl. Některé nástroje podporují více principů současně – neexistují zde pevné hranice – tabulka v příloze B proto nastiňuje možná další přiřazení nástrojů k jednotlivým principům.

Každá subkapitola úvodem obsahuje tabulku, která stručně zobrazuje, které nástroje jsou v subkapitole popisovány a v jakém směru daný princip podporuje.

7.1 Eliminace plýtvání – nástroje

Tabulka 3: Přehled v kapitole popisovaných nástrojů pro eliminaci plýtvání [autorka]

ÚČEL	NÁSTROJ
Identifikace hodnoty	Kano model
Identifikace plýtvání	Mapa hodnotového toku
Eliminace plýtvání	Sada zásad pro každou oblast plýtvání

Aby bylo možné eliminovat plýtvání, je potřeba:

- Identifikovat, co generuje hodnotu pro zákazníka.

  Podobně jako ve výrobním procesu, je základem pochopit, co zákazníkovi z hlediska vývoje přináší hodnotu. Veškeré kroky, které tuto hodnotu bezprostředně negenerují a nejsou v procesu vývoje nevyhnutelné, by měly být eliminovány.

- Dobře znát hodnotový tok vývoje softwaru.

  Vědět, z jakých kroků se vývoj skládá a jakým způsobem jsou tyto kroky realizovány.

7.1.1 Identifikace hodnoty

Prvním krokem při snaze o eliminaci plýtvání v hodnotovém toku (tzn. odstranění kroků, které negenerují hodnotu a přitom nejsou v procesu vývoje softwaru nevyhnutelné), je vlastní identifikace hodnoty. Jinými slovy – je potřeba porozumět tomu, co přesně pro zákazníka

představuje hodnotu a na základě toho umět odlišit, které kroky v procesu vývoje se na generování této hodnoty podílejí a které nikoliv.

Kano model

Jedním z nástrojů jak docílit spokojenosti zákazníka a vytvoření produktu, který pro něj bude představovat hodnotu, je pochopení Kano modelu, jehož autorem je Japonský profesor Noriaki Kano. Tento model poukazuje na skutečnost, že dodání dobrého (softwarového) produktu se neodvíjí pouze od toho, co si zákazník žádá – je nutné zákazníkovi porozumět hlouběji – objevit jeho nevyřčené nenaplněné potřeby a překvapit ho řešením, které tyto bude zahrnovat. Dle modelu existují tři typy parametrů produktu:

a) **Parametry základní potřeby** – zákazník automaticky předpokládá jejich splnění, většinou je ani explicitně nejmenuje – jejich splnění nevede k výrazné spokojenosti, avšak nesplnění se setká s nespokojeností.

b) **Vyslovené požadavky** – čím více těchto požadavků splníme, tím bude zákazník spokojenější.

c) **Parametry zákazníkem neočekávané, ale vítané** – vzbuzují zákazníkovo nadšení, vyprovokují exponenciální růst spokojenosti.

Opět je však třeba akcentovat skutečnost, že parametry zákazníkem neočekávané, ale vítané, musí pramenit z opravdu důkladného poznání zákazníkových potřeb. V opačném případě by se přidávání neočekávaných parametrů dostalo do konfliktu s eliminací plýtvání v oblasti nevyžádané funkcionality.

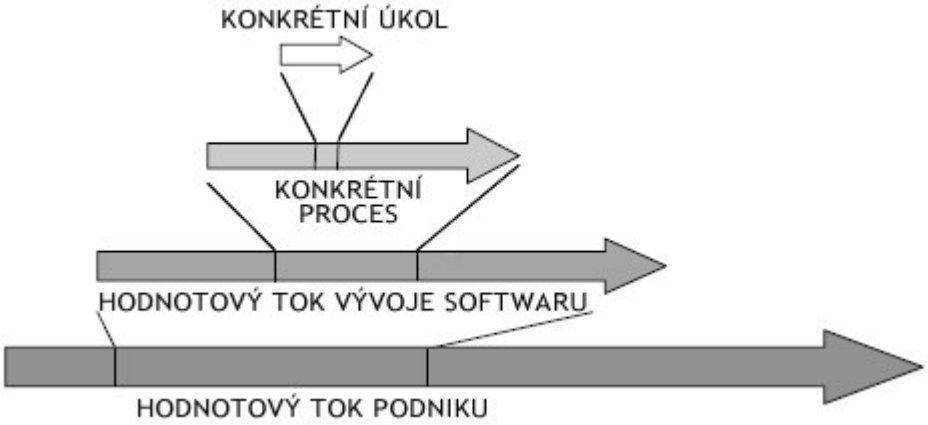
Obrázek 6: Kano model (zpracováno autorkou dle [BURIÁNEK, c2007])

7.1.2 Identifikace plýtvání v hodnotovém toku

Mapování hodnotového toku (Value Stream Mapping – VSM)

Mapování hodnotového toku je nástroj pro vizualizaci a vytvoření přehledu o průběhu celého procesu vývoje softwaru – napomáhá odhalení úzkých míst a prostojů. Stává se tak vodítkem k tomu, na jaká místa se zaměřit při eliminaci plýtvání. [POPPENDIECK, a další, 2003] uvádí, že k tvorbě hodnotové mapy je dostačující tužka a papír.

Nejprve je třeba učinit rozhodnutí ohledně toho, jaká úroveň hodnotového toku bude mapována (ilustruje Obrázek 7). Dle [POPPENDIECK, a další, 2006] je ideální mapovat proces, nebo pro firmu typický projekt.



Obrázek 7: Různé úrovně hodnotového toku (zdroj: [McMANUS, 2005 str. 9]upraveno autorkou pro účely práce)

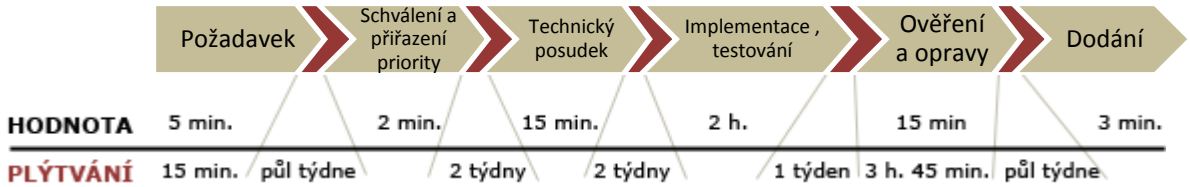
Mapa hodnotového toku by pak měla obsahovat:

- jednotlivé prvky hodnotového toku a jejich návaznost,
- průměrný čas nutný ke splnění každého kroku,
- časové prostoje mezi jednotlivými kroky.

Všechny tyto části mapy hodnotového toku musí reflektovat skutečný stav – i za cenu toho, že nebudou odpovídat oficiální dokumentaci podniku.

Mapování hodnotového toku slouží rovněž jako podpora principu „Optimalizovat celek“.

Při tvorbě hodnotové mapy je však rovněž potřeba osvětlit (zejména managementu) nutnost koncentrace na změnu – nastolit přátelské podmínky, aby se pracovníci nebáli sdělit skutečné časy plnění a prostojů z obavy, že budou potrestání v případě, že mapa odhalí znepokojivé výsledky.



Obrázek 8: Ukázka mapy hodnotového toku, vytvořeno autorkou dle [POPPENDIECK, a další, 2006]

[McMANUS, 2005] uvádí několik zásad pro tvorbu map hodnotového toku pro konkrétní proces:

- Sledovat průběh procesu.
  - Pokud možno sledovat skutečný proces v akci (avšak uvažovat, že provádění činností může být tímto sledováním ovlivněno – zaměstnanci se chovají jinak, když nejsou sledováni).
  - Pokud nelze proces sledovat v akci, pak ho lze sledovat v abstraktní rovině:
    - pokládáním otázek „Co následuje?“ (postup od začátku),
    - pokládáním otázek „Z čeho toto vzešlo?“ (postup od konce).

    V praxi je potřeba sledovat tok oběma směry (zejména u toků obsahující iterace a různé závislosti).
- Získávat informace přímo.
  - Získávat informace přímo od lidí, kteří se na práci podílejí, přímo v průběhu. Popřípadě komunikovat s těmito lidmi alespoň telefonicky (preferovat přímé získávání informací před např. elektronickou poštou.
  - Vždy zjišťovat, jak je práce opravdu prováděna, namísto předložených oficiálních dokumentací.
- Využívat i již existující nasbíraná data.
  - Tato data mohou někdy být cenná z rozličných důvodů – např. reporty (zprávy) mohou obsahovat data, pomáhající následně při výpočtu času, za který je proveden cyklus a další.
- Mapovat „tužkou“.
  - Do podoby map není potřeba příliš investovat – je nutné, aby byly snadno pochopitelné, tedy co nejjednodušší. K tomu postačí např. arch papíru, nebo velká tabule.

- Mapovat celý hodnotový tok.
    - Prvotním krokem by mělo být mapování celého hodnotového toku, namísto jeho jednotlivých částí odděleně – je potřeba primárně zjistit, jak je hodnota postupně vytvářena a vnímat při tom kontext této tvorby.

7.1.3 Eliminace plýtvání

Pro vlastní eliminaci plýtvání může posloužit jako nástroj sada zásad pro každou jednotlivou oblast plýtvání, který shrnuje tabulka Tabulka 4.

Tabulka 4: Zásady pro eliminaci plýtvání [autorka]

OBLAST PLÝTVÁNÍ	ZÁSADA
Funkcionalita	Funkcionalitu přidávat, pouze pokud je k tomu opravdu důvod (existuje reálná představa o využití funkcionality v budoucnu)
Předávky artefaktů mezi rolemi	Omezit předávky artefaktů mezi rolemi na minimum Podpořit přímý kontakt členů vývojářského týmu Častá zpětná vazba
Neúplná, nedokončená práce[7]	Nevytvářet detailní dokumentaci požadavků, potažmo návrh s přílišným předstihem[8] Testovat Co nejčastější synchronizace paralelně vznikajícího kódu Tvorbu dokumentace softwaru neoddělovat od implementace Dodávat co nejčastěji
Nutnost znovu se učit zapomenuté	Dokumentovat důvody určitého rozhodnutí na vhodné a snadno dostupné místo, co nejblíže ke zdroji (nejlépe přímo do kódu)
Střídání úkolů	Preferovat postupnou realizaci úkolů využívajících stejné lidské zdroje před jejich paralelním během
Prodlevy	Umožnit co nejsnadnější a nerychlejší získávání informací a zpětné vazby – spolupracující týmy a jejich členy, nejlépe lokalizované co nejblíže u sebe

[7] Kód musí být integrován, otestován a schválen, jinak ho nelze považovat za hotový. Netestovaný software je jedním z největších zdrojů chyb a tím pádem nedokončené práce.

[8] Požadavky by neměly být sepisovány s přílišným předstihem. Zákazník by neměl být nucen sumarizovat veškeré požadavky v předstihu s vědomím, že nebude mít možnost je později měnit – tendence vzniku enormního seznamu požadavků, které nemusejí odpovídat reálným potřebám.

OBLAST PLÝTVÁNÍ	ZÁSADA
Chyby	Preferovat prevenci chyb (tvořit testy) Testovat kód co nejdříve po jeho napsání

7.2 Včleňovat kvalitu do vývoje – nástroje

Tabulka 5: Přehled v kapitole popisovaných nástrojů pro podporu včleňování kvality do vývoje [autorka]

ÚČEL	NÁSTROJ
Zpětná vazba	Iterativní vývoj Testování
Prevence vzniku chyb	Vývoj řízený testy Automatizace Refaktorizace Nepřetržitá integrace
Disciplína při vývoji	Sada principů 5S

Vodopádový model (popsaný v kapitole 3.1.3) se na hledisko kvality zaměřuje většinou až v pozdějších stádiích vývoje (kvalita nebývá dostatečně včleňována v průběhu jednotlivých jeho fází [De ZOYSA, 2011 str. 26], zejména díky nedostatečně časté zpětné vazbě). Naproti tomu, LeanSD klade důraz na včleňování kvality do vyvíjeného softwaru od samého počátku.

7.2.1 Prevence vzniku chyb

Automatizace

Jedním z nejzásadnějších nástrojů pro prevenci vzniku chyb je automatizace běžných úkonů, jako je např. testování nebo vytváření buildů[9].

Vývoj řízený testy (Test Driven Development – TDD)

Jedná se o přístup k vývoji softwaru, který probíhá tak, že jednotkový test je napsán ještě před napsáním vlastního kódu. Kód může být tedy testován okamžitě po napsání.

Nepřetržitá integrace (Continuous Integration – CI)

Jednotlivé buildy, vytvářené různými vývojáři, je potřeba často integrovat, z důvodu získání co nejrychlejší zpětné vazby. Na rozdíl od privátních buildů, které vytváří konkrétní vývojář

[9] Buildy jsou verze softwaru, vznikající v průběhu jeho vývoje [PC Magazine Encyclopedia, c1981-2012]

na své pracovní stanici, by integrační buildy měly být, pro potřeby časté a neustálé integrace, vytvářeny *automaticky*, pomocí k tomu určených nástrojů.

Refaktorizace

Refaktorizací se rozumí: *„Změna struktury systému beze změny jeho chování. Jejím cílem je zjednodušení, zpřehlednění a zajištění flexibility.“* [BUCHALCEVOVÁ, 2009 str. 151]. Refaktorizace spočívá v realizaci různých úprav kódu, kterými jsou dle [PROCHÁZKA, a další, 2010] typicky:

- *„Úpravy metod,*
- *přesouvání elementů mezi objekty,*
- *organizace dat,*
- *zjednodušení podmíněných příkazů,*
- *zjednodušení volání metod,*
- *generalizace.“*

Refaktorizace se může jevit jako nástroj podporující plýtvání (jednou vykonaná práce je předělávána), avšak refaktorizace výrazně koriguje komplexitu vyvíjeného softwaru, usnadňuje jeho údržbu, čímž plýtvání spíše předchází.

7.2.2 Zpětná vazba

Iterativní vývoj

Iterativní vývoj, byl stručně popsán v kapitole 3. Každá jeho iterace přináší inkrement (přírustek) – funkční část finálního softwaru, což umožňuje získávání objektivní zpětné vazby přímo od zákazníka, který tak nemusí uvažovat v abstraktní rovině, ale může hodnotit konkrétní část vyvíjeného softwaru a na základě tohoto zhodnocení vyjádřit potřebu změny svých požadavků. Potřeba změny je tak odhalena v průběhu vývoje, nikoliv až na jeho konci a tím se výrazně snižuje dopad realizace její změny.

Krátké iterace a tedy častá zpětná vazba, napomáhají rovněž synchronizaci jednotlivých týmů, podílejících se na vývoji. Potenciální problémy jsou tak detekovány brzy a jejich napravování má menší negativní dopad na celý systém.

Testování

Testování (tedy kontrola, zda části vyvíjeného softwaru fungují jednotlivě i vzájemně, tak jak bylo zamýšleno) je důležitým nástrojem pro zajištění zpětné vazby a tím i pro kontrolu kvality. Důležitou podmínkou jsou ale co nejkratší časové prostoje mezi psaním kódu, jeho testováním a případnými opravami. Čím déle totiž zůstává neopravený problém v procesu vývoje daného softwaru, tím se zvyšují náklady na jeho odstraňování.

[POPPENDIECK, a další, 2006] dělí testy následovně:

a) **Programátorské testy** – poskytují zpětnou vazbu v otázce, zda se kód chová tak, jak vývojář zamýšlel.

   Včleňování kvality v této kategorii napomáhá mimo jednotkové a integrační testování i např. regresní testování, které se provádí při přidávání nové funkcionality, nebo při provádění oprav kódu.

   Regresní testování probíhá po změně, případně přidaní, nové funkce. Testovány jsou nejen nové funkce, ale znovu i nezměněné oblasti kódu. [HOLDBERH, 2011]

b) **Zákaznické testy** – někdy nazývány také „akceptačními testy" – poskytují zpětnou vazbu v otázce, zda se systém pracuje dle zákazníkových potřeb.

c) **Testy použitelnosti** – testy, při kterých je software testován skutečnými lidmi (testují se např. vstupy, hraniční hodnoty a další).

d) **Testy dalších vlastností** – testují se požadavky, které se netýkají požadované funkcionality – např. čas odezvy, robustnost a další.

7.2.3 Disciplína při vývoji

Sada principů 5s

Další z lean nástrojů, sada principů 5s, je pojmenovaný podle názvů jednotlivých principů, ze kterých je složen. Každý z nich v angličtině i v japonštině začíná na písmeno „s": sort – seiri – vytřídit, systematize – seiton – zorganizovat, shine – seiso – uklidit, standardize – seiketsu – standardizovat, sustain – Shitsuke – udržovat. Nástroj lze aplikovat v jakékoliv oblasti – tabulka Tabulka 6: Přehled principů nástroje 5s – vytvořeno dle [POPPENDIECK, a další, 2006] ukazuje aplikaci nástroje na pracovní prostředí a na kód psaný v jazyku Java, jehož kvalita může být použitím nástroje pozitivně ovlivněna již v průběhu jeho psaní.

Tabulka 6: Přehled principů nástroje 5s – vytvořeno dle [POPPENDIECK, a další, 2006]

PRINCIP	PRACOVNÍ PROSTŘEDÍ	JAZYK JAVA
Vytřídit Sort (Seiri)	Vytřídit pracovní stanice a servery – odstranit staré verze softwaru, složky a další položky, které již nejsou používány	Zmenšit velikost kódu - odstranit nevyužívané části jako např.: nepoužívaný kód, importy, proměnné, metody a třídy. Refaktorizace
Zorganizovat Systematize (Seiton)	Soubory organizovat logicky, aby byly snadno dohledatelné. Pracovní místa sdílená více osobami by měla být organizována dle dohodnutých pravidel.	Logicky organizovat projekty a balíčky (packages).

PRINCIP	PRACOVNÍ PROSTŘEDÍ	JAZYK JAVA
Uklidit Shine (Seiso)	Odstranit nepořádek na pracovišti, smazat tabule a další.	Vyřešit oblasti, které neprošly jednotkovými testy. Vylepšit jednotkové testy a zvýšit jejich pokrytí. Vyřešit varování, která hlásí např. Checkstyle[10], PMD[11], Javadoc[12] a další.
Standardizovat Standardize (Seiketsu)	Automatizace a standardy pro pravidelné zálohování Automatizované zajišťování vždy nejnovější verze nástrojů na každé pracovní stanici.	Vyvarovat se přílišné složitosti, která znepříjemňuje údržbu.
Udržovat Sustain (Shitsuke)	Udržovat stav dosažený předchozími zásadami.	Udržovat stav dosažený předchozími zásadami, dodržovat standardy.

[10] Checkstyle je nástroj, který kontroluje, zda je Java kód psán dle standardu. [SourceForge.net, 2011]

[11] PMD je doplněk vývojářského prostředí, který prochází kód a hledá potenciální problémy (chyby, nepoužívané části kódu, příliš složité výrazy, duplicitní kód a další). [SourceForge.net, 2012]

[12] Javadoc je nástroj, který analyzuje kód a vytváří dokumentaci popisující třídy, rozhraní, konstruktory, metody a pole. [Oracle, c2011]

7.3 Vytvářet znalosti – nástroje

Tabulka 7: Přehled v kapitole popisovaných nástrojů pro vytváření znalostí [autorka]

ÚČEL	NÁSTROJ
Podpora tvorby a sdílení znalostí	Komunikace Vědecká metoda Metoda A3 Vývoj založený na souboru možností Speciální pravidla Párové programování
Zachycování znalostí	Dokumentace Wiki systémy

Aby bylo možné znalosti vytvářet, je potřeba vytvořit podmínky pro podporu jejich tvorby. Jakmile je znalost vytvořena, měla by být zachycena tak, aby byla snadno dostupná. Vytvořené a zachycené znalosti pak lze využívat k rozhodování, či řešení problémů.

7.3.1 Podpora tvorby a sdílení znalostí

Tvorba a zachycování znalostí napomáhá lepší flexibilitě vývoje – umožňuje rychlejší rozhodování, protože vývojář, nebo vývojářský tým při rozhodování může disponovat těmito zaznamenanými informacemi. Aby k tvorbě znalostí docházelo, musí k tomu být vytvořeny podmínky – vývojáři by měli dostávat prostor ke sdílení a tvorbě těchto znalostí.

Komunikace

Pravidelné schůzky, na kterých probíhá diskuze o tom, na čem aktuálně probíhá práce, jsou základem efektivní komunikace. Zlepšuje se tak přehled všech členů vývojářského týmu o celkovém stavu vývoje daného projektu a zároveň napomáhají sdílení informací a tvorbě nových znalostí. Schůzky by měly být samozřejmostí po dokončení každé iterace, za účelem zhodnocení jejích výsledků a selekce úkolů pro iteraci následující.

Vědecká metoda (Scientific method)

Výhoda strukturovaného přístupu k řešení problémů spočívá v explicitním zachycení jednotlivých jeho kroků, zabránění ztrátě detailů a komunikaci úvah vedoucích k vyvozování určitých důsledků s ostatními členy vývojářského týmu. Napomáhá tedy k rozvoji a sdílení znalostí.

Jednou z metod pro strukturované řešení problémů je tzv. Vědecká metoda, která se skládá z několika kroků [POPPENDIECK, a další, 2006]:

1. DEFINICE PROBLÉMU
2. ANALÝZA PROBLÉMU
3. VYTVOŘENÍ HYPOTÉZY
4. PROVEDENÍ EXPERIMENTŮ
5. OVĚŘENÍ VÝSLEDKŮ
6. OPAKOVÁNÍ CELÉHO PROCESU METODY / VYTVOŘENÍ STANDARDU

Při aplikaci metody v oblasti vývoje softwaru, se lze ve fázi definice problému tázat například „Čeho chceme dosáhnout? Co chceme zlepšit?“. Ve fázi analýzy problému probíhá sběr potřebných informací o problému (pozorování chování a sběr relevantních dat). Následně je vytvořena hypotéza – předpoklad jak se, ve vztahu k pozorovanému problému, projeví určitá změna. Hypotéza je následně testována prováděním experimentů a výsledky experimentů jsou ověřovány. Nově vytvořené znalosti následně vedou k vyvození důsledků, opakování celého procesu, či standardizaci.

Tabulka 8: Názorný příklad strukturovaného přístupu k řešení problému. Vytvořeno autorkou dle [CLINE, c2012] a [POPPENDIECK, a další, 2006]

FÁZE METODY	BĚŽNÁ ŽIVOTNÍ SITUACE	VÝVOJ SOFTWARU
Definice problému	Selhal pokus o rozsvícení světla.	„Chceme zlepšit reakci na požadavky zákazníků.“
Analýza	Ostatní elektronická zařízení jsou funkční – nejedná se o problém výpadku elektřiny.	Existuje fronta požadavků, ignorovaných ve prospěch reakce na požadavky nově příchozí.
Hypotéza	Selhání je způsobeno vadným kontaktem.	Pokud dojde ke zmenšení fronty požadavků a dojde k mírnému zrychlení tempa realizace požadavků, zlepší se reakce na požadavky.
Experiment	Opakovaný pokus o rozsvícení. Neúspěšný výsledek.	Emailové ověřování, zda zájem o požadavky starší 8 týdnů stále trvá. 99 % požadavků starších než 8 týdnů již nebylo vyžadováno 77 % požadavků starších 8 týdnů již nebylo vyžadováno

FÁZE METODY	BĚŽNÁ ŽIVOTNÍ SITUACE	VÝVOJ SOFTWARU
Ověření	Opakovaný pokus o rozsvícení. Neúspěšný výsledek.	V průběhu několika týdnů došlo k poklesu možných požadavků ve frontě na zhruba 50 požadavků, díky kombinaci rušení starších, neaktuálních požadavků a zrychlení dokončování požadavků s vyšší prioritou.
Opakování / Standardizace	Experiment správnost hypotézy nepotvrdil → opakování jednotlivých kroků metody, tentokrát s hypotézou „Praskla žárovka“.	Fronta bude obsahovat pouze 50 požadavků.

Metoda A3

Metoda A3 je další metodou, která napomáhá strukturovanému řešení problémů (lze ji využít i např. pro standardizaci psaní reportů). Metoda, která byla využívána i ve firmě Toyota, svůj název dostala dle velikosti formátu papíru, na který se její realizace musí vejít. Její struktura se skládá z několika částí, pro které bývá užívána zkratka PDCA:

1. PLÁNUJ (Plan) – popis problému a činností k jeho řešení, cíl.
2. PROVEĎ (Do) – zavedení popsaných činností.
3. OVĚŘ (Check) – sledování výsledků zavedení a porovnání s plánem.
4. JEDNEJ (Act) – v případě shody výsledků s plánem standardizovat a řídit se provedenými změnami. V případě neshody najít a odstranit příčinu (fáze se propojí v cyklus).

Zásadou pro užívání metody A3 je nutnost vyjadřovat se stručně, jasně a objektivně. Ukázka jedné z možných podob A3 šablony pro oblast vývoje softwaru je k nalezení v Příloze A.

Vývoj založený na souboru možností (Set-based development)

Znalosti a jejich tvorba je úzce propojena i s rozhodováním. Čím déle je rozhodnutí oddalováno, tím jsou získávány aktuálnější informace a vytvářeny znalosti využitelné k potenciálnímu výběru toho nejlepšího řešení. Vývoj založený na souboru možností je detailněji popsán v podkapitole 6.1.4.

Speciální pravidla

Zajímavý příklad, jak lze podpořit vytváření znalostí, popisuje [POPPENDIECK, a další, 2006]: Po dokončené iteraci je ve firmě Rally Software Development vyhlášen tzv. „hack-a-thon“(z anglických slov „marathon“ a „hack“) – týden, ve kterém nedochází ke psaní dalšího kódu, ale který je vyčleněn na podporu tvorby znalostí a kreativity – vývojáři této firmy v tomto týdnu mohou zkoušet cokoliv, co je napadne. Pokud však zákazník ohlásí problém odhalený v právě dokončené iteraci, je hack-a-thon okamžitě přerušen a tým se věnuje

nápravě ohlášeného problému. Jedním z pozitivních aspektů, je fakt, že vývojáři si při iteracích dávají více záležet, aby žádný potenciální problém nenarušil následující týden hack-a-thonu.

Párové programování

Párové programování je významnou praktikou pro sdílení znalostí a okamžitou zpětnou vazbu, která je typická pro agilní metodiku Extrémní programování. Programování je při této praktice realizováno v párech u jednoho počítače, kdy jeden z vývojářů píše vlastní kód a druhý z páru mezitím o kódu přemýšlí (jaké testy je potřeba napsat, jak kód zjednodušovat). [BUCHALCEVOVÁ, 2009] Programování v páru přináší nespornou výhodu v předávání znalostí – programátoři komunikují přímo, bez mezistupňů a tak je minimalizována ztráta tacitních znalostí.

7.3.2 Zachycování znalostí

V kapitole 6.1.1 bylo popsáno plýtvání plynoucí z vytváření chyb. Pokud se taková chyba ve vývoji jednou vyskytla, pak je potřeba, po jejím okamžitém odstranění dále zajistit, aby se už neopakovala. Poučit se z nezdaru, vytvořit a v rámci prevence zachytit znalost o vzniklém problému a jeho řešení.

Dokumentace

Mylnou domněnkou bývá fakt, že znalost je nutně vytvářena tvorbou dokumentace. Pokud však takové dokumenty a znalosti v nich zaznamenané nejsou nikým využívány, pak je jejich tvorba plýtváním. Znalosti by měly být zachycovány v takové podobě a dostupnosti, aby plnily účel své tvorby.

Existuje dokumentace mnoha různých typů, které blíže popisuje např. [ČERNÁ, 2011] – mezi typy dokumentace patří i dokumentace v podobě komentářů přímo ve zdrojovém kódu, které umožňují umisťování důležitých informací přímo ke zdroji.

Wiki systémy

Wiki je prostor (nejčastěji ve formě webové stránky), který umožňuje sdílení a zachycování znalostí, na kterém se mohou členové vývojářského týmu podílet současně.

7.4 Odkládat závazky – nástroje

Tabulka 9: Přehled v kapitole popisovaných nástrojů pro podporu principu Odkládat závazky [autorka]

ÚČEL	NÁSTROJ
Vytvoření tolerance ke změnám	Iterativní vývoj Minimalizace závislostí
Odkládání závazných a nevratných rozhodnutí	Vývoj založený na souboru možností Neoddělovat tvorbu návrhu od implementace

Základem tohoto principu je myšlenka, že **nevratná a závazná rozhodnutí by měla být odkládána až na nejpozdější možný moment**. Definovat, který moment tímto momentem je (v případě, že moment rozhodnutí není explicitně daný), je poměrně obtížné a schopnost ho správně identifikovat souvisí se zkušenostmi. Obecně lze říci, že je to moment, který je pro učinění rozhodnutí nejvhodnější, z hlediska množství informací nutných k učinění správného rozhodnutí (je vhodnější rozhodovat se na základě reálných informací, namísto rozhodování na základě spekulací).

Existují ale rozhodnutí jako např. rozhodnutí o užitém programovacím jazyku, která by odkládána být neměla, protože představují základní omezení, od kterých se celý vývoj odvíjí. [POPPENDIECK, a další, 2003]

7.4.1 Vytvoření tolerance ke změnám

Aby bylo možné odkládat závazná a nevratná rozhodnutí až na nejpozdější možný moment, je nutné, aby pro to vývoj softwaru byl uzpůsoben – tedy, aby při něm existovala podpora (tolerance) změn v jeho průběhu. [POPPENDIECK, a další, 2003] dokonce poukazuje na novou, osmou oblast plýtvání – software, který je netolerantní ke změnám.

Iterativní vývoj

Jedním z nejdůležitějších nástrojů pro podporu změn v průběhu vývoje softwaru, je iterativní vývoj, který byl blíže popsán již v kapitole 3 a podrobněji v kap. 7.2.2, která se věnuje principu včleňování kvality do vývoje. Požadavky na funkcionalitu se v průběhu vývoje založeného na iteracích mohou měnit (vývoj není realizován na základě předem daných, neměnných závazků – požadavků a návrhu).

Minimalizace závislostí

Odkládání závazných a nevratných rozhodnutí je podporováno i minimalizací počtu implementačních závislostí. Čím méně takových závislostí existuje, tím se zvyšuje flexibilita vzhledem ke změnám.

7.4.2 Odkládání závazných a nevratných rozhodnutí

Vývoj založený na souboru možností (Set-based development)

Rozhodování je úzce propojeno se znalostmi a jejich tvorbou. Čím déle je rozhodnutí oddalováno, tím jsou získávány aktuálnější informace a vytvářeny znalosti využitelné k potenciálnímu výběru toho nejlepšího řešení.

Vývoj založený na souboru možností spočívá v realizaci více alternativních řešení, kdy v posledním možném momentu (kdy je nashromážděno nejvíce informací) dochází k výběru jednoho z nich.

Příklad uvádí [POPPENDIECK, a další, 2006]: Jedná se o situaci, kdy je vyvíjeno softwarové řešení za dvou podmínek – a) termín dodání nelze měnit, b) dodávka se musí uskutečnit za každých okolností. Tvorbou alternativních řešení jsou pověřeny tři různé týmy:

- Tým A vyvíjí méně optimální řešení, které ale s jistotou bude hotové v daném termínu.
- Tým B vyvíjí vhodnější řešení, snaží se stihnout termín, ale zároveň má jistotu, že je jištěn řešením týmu A, v případě nedodržení termínu.
- Tým C vyvíjí nejvhodnější řešení, s nejmenší pravděpodobností, že stihne termín, avšak s jistotou, že může být použito řešení týmu A nebo B.

Za každých okolností tedy bude existovat hotové řešení a bude zároveň nejlepším řešením pro daný moment uskutečnitelným. Vývoj založený na souboru možností nenahrazuje iterativní vývoj, ale pouze přidává novou dimenzi – v prvních iteracích je vytvářena klíčová funkcionalita jednotlivých alternativních řešení. V pozdějších iteracích jsou řešení sloučena, nebo postupně omezována v jedno vítězné. [POPPENDIECK, a další, 2003]

Neoddělovat tvorbu návrhu od implementace

[POPPENDIECK, 2011] pokládá za návrh systému detailní požadavky, veškeré seznamy funkcionalit a případy užití. Tvorba takového návrhu by měla být vytvářena zkušenými návrháři, architekty a inženýry z vývojářského týmu (nikoliv přímo business analytiky nebo Product Ownerem[13], avšak při vytváření návrhu přijímat jejich vstupy). Oddělování tvorby návrhu od implementace přenáší odpovědnost za vytvoření vhodného produktu na stranu lidí, kteří nejsou přímou součástí vývojářského týmu. Vývojářský tým v takovém případě musí postupovat dle návrhu vytvořeného těmito lidmi, namísto využívání svých znalostí pro vymýšlení vhodných řešení.

Pro tradiční přístupy k vývoji softwaru, založené na vodopádovém modelu životního cyklu, je typická specifikace požadavků s předstihem, na jejichž základě probíhá následující vývoj. Takovýto přístup ale často neodpovídá realitě – vývoj softwaru se běžně odehrává

[13] Product Owner je jednou z rolí v metodice Scrum

v proměnném prostředí, kde se jak zákazníkovy požadavky, tak možná řešení konkrétního problému, často mění.

7.5 Dodávat co nejrychleji – nástroje

Tabulka 10: Přehled v kapitole popisovaných nástrojů pro podporu principu Dodávat co nejrychleji [autorka]

ÚČEL	NÁSTROJ
Dodávat co nejrychleji	Krátké iterace, malé dávky

Rychlé dodávky znamenají rychlou reakci na aktuální zákazníkovy požadavky – čím déle trvá vývoj (jednotlivých požadavků i celku), tím náchylnější jsou tyto požadavky k tomu, aby se staly zastaralými a neaktuálními, protože dlouhými vývojovými cykly vzniká plýtvání v oblasti nedokončené práce.

Čím rychleji je realizována reakce na zákazníkovy požadavky, tím je menší pravděpodobnost, že se požadavky změní (jedním z důvodů změny může být např. i změna situace na trhu). Čím rychleji je realizována reakce na zákazníkovy požadavky, tím dříve dochází k obdržení cenné zpětné vazby, která napomůže brzkému odhalení problémů a chyb.

Rychlým dodávkám typicky brání:

- nízká produktivita,
- netolerance ke změnám,
- nedokončené úkoly,
- chyby.

Výskyt a řešení těchto problémů se projeví na průměrné době nutné k úspěšné realizaci klíčového procesu firmy – v oblasti vývoje softwaru, se jedná o dobu potřebnou k tomu, aby byly zákazníkovy potřeby přeměněny na software a je tedy potřeba je eliminovat.

7.5.1 Krátké iterace, malé dávky

Vývoj založený na krátkých iteracích, je jedním z nejdůležitějších nástrojů umožňujících co nejrychlejší dodávky. Náplň jednotlivých iterací je však nutno plánovat s rozmyslem, tedy zhodnotit úkoly k implementaci a na základě odhadu průměrných dob nutných k jejich realizaci rozhodnout o výběru jejich rozumného počtu pro konkrétní iteraci. V případě, že je pro iteraci určeno příliš mnoho úkolů, hrozí, že se ve frontě začnou hromadit nedokončené úkoly, které představují plýtvání. Signalizační systém Kanban, popsaný v kapitole 7.6.2, umožňuje vizualizaci stavu úkolů (pokroků) dané iterace.

7.6 Důvěra a respekt k lidem podílejícím se na vývoji – nástroje

Tabulka 11: Přehled v kapitole popisovaných nástrojů pro podporu principu Důvěra a respekt k lidem, podílejícím se na vývoji [autorka]

ÚČEL	NÁSTROJ
Pravomocní pracovníci	Samořízené týmy Motivace Komunikace
Podpora týmové práce	Kanban Andon Tvorba přehledů

Lean myšlení vnímá pracovníky, podílející se na tvorbě hodnoty, jako cenné zdroje, které je potřeba vnímat jako jednu z priorit. Tito pracovníci, často vysoké odbornosti, znají povahu své práce lépe, než kdokoliv jiný – proto by k nim mělo být dle toho přistupováno. Měli by mít důvěru a možnost rozhodovat v relevantních případech, namísto toho, aby jim bylo nařizováno, jak mají svou práci správně dělat.

7.6.1 Pravomocní pracovníci

Možnost konat rozhodnutí a rozhodovat o tom, jak by měla být práce určitého pracovníka prováděna, by tedy měla být soustředěna na co nejnižší stupně hierarchie. Pracovníci by současně měli být podporováni v neustálém zdokonalování, aby získávali potřebné znalosti a zkušenosti ke tvorbě dobrých rozhodnutí.

Samořízené týmy

Klíčové je tvoření týmů, které jsou samořízené a jejichž členové jsou oddáni plnění cílů týmu. Hodnocení je třeba odvíjet od výkonu týmu jako celku – nehodnotit individuální výkony, ale hodnotit dle míry přispění ke společnému cíli. Tým by měl být ideálně složen ze členů různých kvalifikací, jež se mohou vzájemně doplňovat. Týmy samy rozhodují o plánování a náplni jednotlivých iterací.

Motivace

Nutností je, aby členové týmu byli dostatečně motivováni k odvádění dobrých výsledků. Členové týmu musí vědět, co dělají a proč to dělají (jaký je smysl toho, co dělají). Vedoucí týmu (dle [ŠOCHOVÁ, 2012] typicky např. Scrum Master[14]) by měl členům naslouchat, zajišťovat, aby měli dostatek zdrojů potřebných k práci a motivovat je. Avšak, je nutné

[14] Scrum Master je jednou z rolí v metodice Scrum

podněcovat i určitou disciplínu – [POPPENDIECK, a další, 2003] jmenuje např. užívání správy verzí, konvencí při psaní kódu, automatizované testování a další.

Komunikace

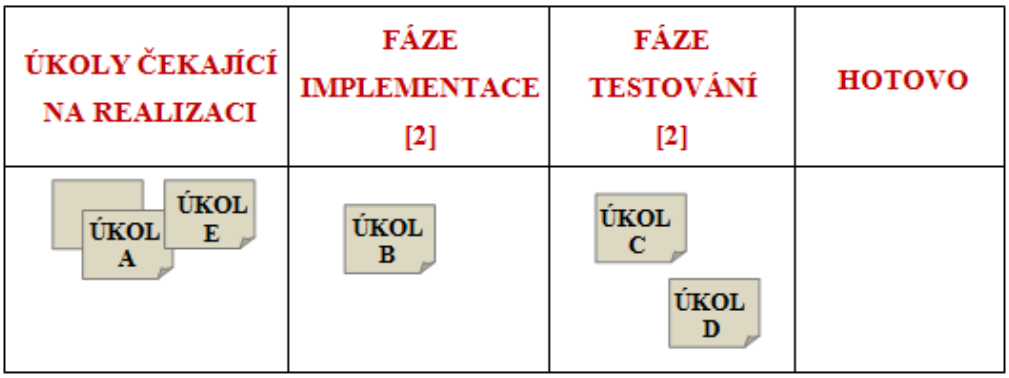
Efektivní komunikace a sdílení znalostí, blíže popsaná v kapitole 7.3, je jedním z nejdůležitějších nástrojů LeanSD. Pracovníci by pro komunikaci měli dostávat dostatečný prostor a podporu. [POPPENDIECK, a další, 2003] uvádí typický příklad nedůvěry manažerů na firmě Xerox – mnoho problémů bylo vyřešeno díky neformálním setkáním pracovníků, kteří si na nich vyměňovali zkušenosti a rady. Na základě těchto setkání se pracovníci rozhodli vytvořit databázi tipů, avšak narazili na odpor manažerů, kteří nevěřili, že si pracovníci budou předávat dostatečně odborné a hodnotné tipy (jedinou cestou jak zajistit kvalitu, byla dle manažerů cesta skrze standardizované pracovní procesy). Navzdory tomu, pracovníci vytvořili neoficiální databázi, která za dva měsíce fungování zvýšila produktivitu o 10 %.

7.6.2 Podpora týmové práce

Kanban

Kanban je ve své podstatě signalizační systém, který byl, ve vztahu k výrobě, stručně popsán v kapitole 4.1.1. V aplikaci na oblast vývoje softwaru se jedná o nástroj k plánování jednotlivých iterací. Existují různé podoby Kanbanu – např. elektronická, nebo běžně reprezentace pomocí malých lepících papírků, či kartiček, lepených na zeď, symbolizujících různé části celku (typicky jednotlivé úkoly).

Při plánování průběhu řešení určitého problému bývají papírky, reprezentující jednotlivé úkoly, které jsou s řešením problému spojeny, indikátorem počtu požadavků – pokud je papírků, symbolizujících úkoly čekající na vyřešení příliš mnoho, pak signalizují potřebu jejich omezení. [POPPENDIECK, a další, 2006] Na zdi, či tabuli bývají znázorněny jednotlivé fáze vývoje a papírky jsou do nich postupně přiřazovány. Často bývá omezen počet současně rozdělaných úkolů (papírků) pro jednotlivé fáze (Obrázek 9 znázorňuje toto omezení pomocí čísla v hranatých závorkách). Úkoly (papírky) mohou mít různé barvy např. dle důležitosti, které usnadňují okamžitý přehled o stavu konkrétního projektu. Příklad použití je popsán např. v kapitole 8.3.1 a konkrétněji v [MIDDLETON, a další, 2011].



Obrázek 9: Zjednodušená ukázka nástroje Kanban [autorka]

Andon

Andon, stručně popsaný v kapitole 4.1.1, ve výrobě signalizoval přerušení určité činnosti, z důvodu výskytu defektu. Přerušení probíhalo automatizovaně v případě stroje a v případě člověka např. zatáhnutím za spouštěč. Signál se vizuálně projevil na andonové tabuli a sloužil jako indikátor toho, že je něco v nepořádku a je potřeba okamžitá náprava.

V běžném životě je možné se s andonovými tabulemi setkat např. na vlakových nádražích, kde jsou jimi komunikovány informace o příjezdech a odjezdech vlaků. V případě vývoje softwaru je jimi signalizován např. výsledek buildu. Vlastní signalizace může mít mnoho různých podob – od emailů, po elektronickou tabuli. [POPPENDIECK, a další, 2006]

Tvorba přehledů

Pokroky týmu a projektu by měly být jasně a přehledně vizualizovány na viditelném a snadno dostupném místě.

7.7 Optimalizovat celek – nástroje

Tabulka 12: Přehled v kapitole popisovaných nástrojů pro podporu principu Optimalizovat celek [autorka]

ÚČEL	NÁSTROJ
Systémové myšlení	Metoda pěti „Proč?“ Vnímání širšího kontextu vývoje Metriky
Neustálé zlepšování	Kaizen

7.7.1 Systémové myšlení

Během aplikace všech šesti předchozích principů LeanSD je nutné neustále vnímat chování celku jakožto systému. *„Systém je komplex prvků,* ***nacházejících se ve vzájemné interakci.****“* [ROSICKÝ, 2009 str. 25] Jedná se tedy zejména o nutnost vyvarovat se optimalizaci jednotlivých částí (procesů, týmů, úkolů) hodnotového toku – lokální optimalizaci a namísto toho optimalizovat tok jako celek, protože, jak uvádí např. [POPPENDIECK, a další, 2003], úspěšnost systému se odvíjí od toho, jak jeho jednotlivé části fungují společně – nikoliv pouze od toho, jak fungují odděleně. Lokální optimalizace jsou rovněž často spojeny s lokálními metrikami, které mohou negativně ovlivnit chování celku.

Metoda pěti „Proč?“

Metoda pěti „Proč?“ je triviálním nástrojem, který však může často napomoci k odhalení skrytých existujících vztahů, které vyústily v určitý problém. Tato metoda, která byla hojně

využívána ve firmě Toyota, je jedním z nástrojů konceptu Six Sigma[15] a její podstata spočívá v definici problému a následnému odhalování důvodů jeho vzniku, za pomoci několika (ne nutně pěti) otázek „Proč?“. Příklad uvádí např. [POPPENDIECK, a další, 2003]:

- Problém: Zvyšuje se počet chyb.

1. Proč se zvyšuje počet chyb?
    - Protože byl přidán nový modul.
2. Proč nový modul zapříčiňuje chyby v jiných modulech?
    - Protože nebyl otestován.
3. Proč nebyl tento modul otestován?
    - Vývojáři byli pod tlakem nesplnitelného termínu.
4. Proč byl určen nesplnitelný termín?
    - Protože manažer má obavy z nedodržení plánů.

- Řešením by tedy měla být konverzace s manažerem, která osvětlí nepříznivé důsledky zadávání nesplnitelných termínů.

Vnímání širšího kontextu vývoje softwaru

V rámci optimalizace celku je třeba vnímat i širší kontext vývoje softwaru a firmy – nejedná se pouze o koncentraci na hodnotu pro zákazníka, ale i o další prvky, spjaté s hodnotovým tokem – dle [PROCHÁZKA, 2010] se jedná např. o dodavatele (uzavírat vhodný typ kontraktu, který rozptýlí rizika i výhody mezi jednotlivé strany), společnost, trh a další.

Metriky

[POPPENDIECK, a další, 2003] uvádí, že pouze méně než 20 % chyb je zapříčiněno přímo pracovníkem – zbytek je zakořeněn v systémech a praktikách, které většinou ovlivňuje management, nikoliv pracovník. Chyby by tedy neměly být spojovány s konkrétním pracovníkem – namísto toho je nutné odhalovat skutečné důvody jejich vzniku.

7.7.2 Neustále zlepšování

LeanSD se netýká pouze jednorázového zlepšení, ale reflektuje původní princip Lean myšlení – dosahování dokonalosti, popsaný v kapitole 4.2, je tedy nutné neustálé, kontinuální zlepšování a dosahování ideálního stavu.

Kaizen

Kaizen je jedním z nástrojů pro neustálé zlepšování. Podstatou nástroje Kaizen je stanovení určitého cíle, kterého je, za současného vnímání kontextu tohoto cíle, poté po malých, ale

[15] Six Sigma je koncept pro zajišťování kvality a snižování počtu defektů [BREYFOGLE, 2003], původně spjatý s výrobou, avšak existuje rovněž modifikace pro oblast vývoje [POPPENDIECK, a další, 2003] (softwaru).

usilovných a pravidelných krocích dosahováno. Dle [PROCHÁZKA, 2010] se jedná o: *„Opakovanou aktivitu a změnu způsobu myšlení, která musí vycházet z nejnižší operativní úrovně současně s podporou nejvyššího vedení.“*

V souvislosti s nástroji Kaizen a Kanban, bývá v mnoha zdrojích zmiňován i nástroj Kaiaku. Kaiaku je v porovnání s nástrojem Kaizen spojen s realizací razantnější a rychlejší změny. [SCHIFFER, 2011]

7.8 Shrnutí

Kapitola popsala nástroje pro podporu sedmi principů metaforického směru konceptu LeanSD, představených v kapitole 6.1. Nástroje byly kategorizovány dle principů, které podporují, avšak některé nástroje podporují více principů zároveň a neexistují zde jednoznačné hranice jejich přiřazení. V příloze B je nastíněna provázanost jednotlivých nástrojů a principů v tabulkovém přehledu.

8 Využití konceptu Lean Software Development

První dvě subkapitoly shrnují výhody a nevýhody použití konceptu LeanSD, vyplývající z předchozích částí práce. V dalších subkapitolách jsou představeny dva konkrétní případy užití LeanSD v praxi, které ústí ve shrnutí nejdůležitějších výhod / nevýhod spojených s tímto konkrétním použitím.

8.1 Přínosy použití konceptu Lean Software Development

Na základě předchozích částí této práce, definuji jako nejvýraznější výhody použití konceptu LeanSD následující:

- zefektivnění hodnotového toku (prostřednictvím eliminace plýtvání),
- efektivnější a rychlejší reakce na měnící se zákazníkovy požadavky,
- usnadnění celkového procesu vývoje a dodávání co nejlepšího možného řešení (zejména skrze podporu tvorby znalostí a jejich sdílení),
- snížení rizika spojeného s hrozbou dodání nesprávné funkcionality,
- snížení nákladů vznikajících v případě potenciální nutnosti odstranit nesprávnou funkcionalitu – podpořeno zejména iterativním vývojem s krátkými iteracemi, který rovněž umožňuje odkládat rozhodnutí na nejpozdější možný moment (moment kdy je k učinění závazného rozhodnutí nashromážděno dostatek informací); taková rozhodnutí lépe reflektují aktuální potřeby a situaci zákazníka,
- rychlé dodávky, které jsou spojeny s konkurenční výhodou,
- respekt k pracovníkům vede k jejich větší motivovanosti a chuti podílet se na neustálém zlepšování své i týmové práce – výsledkem může být i zvyšování produktivity,
- minimalizace lokálních optimalizací a dodávání produktu, jehož jednotlivé části dobře fungují jako jeden celek.

8.2 Nevýhody použití konceptu Lean Software Development

Na základě předchozích částí práce definuji jako příklady možných nevýhod spojených s použitím konceptu LeanSD zejména následující:

- aplikace Lean principů může narazit na nechuť ke změnám – pokud se jedná o snahu aplikovat Lean principy na celý podnik, pak je třeba často zásadně proměnit stávající firemní kulturu; je potřeba aby management (zejména projektový) týmům věřil a fungovat spíše jako podpora a kouč než jako prostředek uplatňující direktivní přístup,

- pro úspěšnou aplikaci Lean principů je základem dostatečná motivace lidí podílejících se na vývoji, která zajistí, aby týmy byly schopné samořízení a aby jejich členové odváděli kvalitní práci; této motivaci je nutné věnovat dostatek pozornosti; [AMBLER, a další, 2007] dále uvádí, že je v první řadě nutné, aby lidé v týmu byli vůbec samořízení schopni a aby byli ochotní přebírat zodpovědnost za svá rozhodnutí,
- kvalitní práce, odváděná jednotlivými týmy, je podmíněna vhodným složením týmů; členy je vhodné vybírat citlivě tak, aby měli co nejvhodnější kvalifikaci k výkonu práce (výběr zkušených členů) a aby se členové mohli vzájemně doplňovat,
- mimo kvalifikaci a motivaci hraje v aplikaci Lean principů důležitou roli také (těžko ovlivnitelná) disciplína jednotlivých pracovníků,
- správná aplikace Lean principů je spojena s jejich správným pochopením a s pochopením podstaty Lean myšlení – není možné pouze adoptovat fungující model zavedení Lean principů, který se osvědčil v určité firmě, bez pochopení kontextu a uzpůsobení pro podmínky konkrétní firmy.

8.3 Příklady užití konceptu Lean Software Development v praxi

Úvodem subkapitoly je třeba zmínit, že vzhledem k provázanosti konceptů AgileSD a LeanSD není jednoduché vypozorovat a vybrat případy, kdy byl jednoznačně použit koncept LeanSD. Proto byla tato část práce pojata formou souhrnů již existujících případových studií.

8.3.1 Případová studie: BBC Worldwide

Stručná charakteristika této případové studie byla vytvořena na základě zahraničního zdroje [MIDDLETON, a další, 2011][16], ve snaze demonstrovat využití konceptu Lean Software Development v praxi.

Tabulka 13: Informace o analyzovaném případu – BBC Worldwide [autorka], zpracováno dle [MIDDLETON, a další, 2011]

Analyzovaný podnik:	**BBC Worldwide** dceřiná společnost firmy British Broadcasting Corporation (BBC)
Časové období provádění studie:	duben 2008–říjen 2009

[16] K dispozici na: http://leanandkanban.files.wordpress.com/2011/04/lean-software-management-bbc-worldwide-case-study-feb-2011.pdf

Nulová hypotéza:	„Aplikace Lean přístupu bude mít buď žádný, nebo negativní vliv na proces vývoje softwaru.“
Alternativní hypotéza:	„Aplikace Lean přístupu zlepší proces vývoje softwaru.“
Použitá metoda:	zkušený řešitel pozoruje a zaznamenává realizaci softwarových procesů oddělení Webových médií společnosti BBC Worldwide
Frekvence pozorování:	sedm návštěv firmy (2-3 dny dlouhé), doplňováno emaily a telefonáty

Úvod

Případová studie se zabývá implementací Lean myšlení v rámci týmu pro vývoj softwaru (nikoliv aplikací Lean myšlení na celou organizaci). Tento tým se skládal z devíti členů: projektový manažer, business analytik, návrhář (Software Architect), tester, hlavní vývojář (Lead Developer), tři vývojáři a pomocný vývojář. Implementace začala v dubnu, avšak z důvodu přizpůsobení se změnám spojeným s implementací Lean myšlení a princpů, začalo shromažďování relevantních dat až v srpnu 2008.

Aplikace Lean myšlení

Hlavním cílem aplikace Lean myšlení byla identifikace nejdůležitější funkcionality z pohledu zákazníka a realizace jednotlivých částí vyvíjeného produktu v co nejmenších jednotkách, tedy dodávat co nejrychleji co nejvíce hodnoty.

V dubnu 2008 začala implementace kroků pro zlepšení používaných praktik:

- vývoj řízený testy,
- automatizované akceptační testování,
- software pro správu verzí,
- software pro sledování chyb (Bug-Tracking software),
- vyřazení či údržba zastaralého softwaru,
- představení konceptu Minimální obchodovatelné funkcionality[17].

Dále byla vytvořena mapa hodnotového toku a jeho části byly znázorněny na Kanbanových tabulích. Jednotlivé činnosti prováděné v těchto částech hodnotového toku byly zaznamenány na Kanbanové kartičky a přiřazeny příslušným tabulím. Výsledky těchto aktivit poukázaly na velké množství úzkých míst v hodnotovém toku a na velké množství nedokončené práce (Work In Progress – WIP). Byla proto zavedena příslušná omezení povoleného množství

[17] Minimální obchodovatelná funkcionalita (Minimal Marketable Feature – MMF) je část funkcionality, která reprezentuje podmnožinu zákazníkových požadavků a která, pokud je dodána jako samostatná jednotka, je schopna tvořit pro zákazníka hodnotu. [MIDDLETON, a další, 2011]

nedokončené práce v jednotlivých částech hodnotového toku. Osvojení konceptu Minimální obchodovatelné funkcionality umožnilo rozpad práce do malých jednotek, které lze dodávat rychleji.

Pro podporu vizualizace přehledu o celku byly kolem pracovního stanoviště rozmístěny výše zmíněné tabule. V případě, že má tým dostatečné kapacity pro započetí práce na dalším úkolu (přijetím dalšího úkolu nebude porušeno maximální povolené množství nedokončené práce), putuje Kanbanová kartička tohoto úkolu na další příslušnou tabuli. Dále byla zavedena každodenní 15 minutová setkání všech členů týmu, za cílem zhodnocení aktuálního stavu projektu, kontroly správnosti informací na tabulích, atd. Vymizela potřeba vytváření reportů jednotlivými členy týmu.

Měřená kvalitativní a kvantitativní data

Monitorováno bylo následující:

- celkový čas od přijetí zákazníkova požadavku na tvorbu softwaru po jeho dodání,
- doba trvání vývoje, měřeno v pracovních dnech,
- počet dodávek (dodaných položek) za měsíc,
- počet chyb za týden (chyby v průběhu týdne nahlášené zákazníkem a chyby dosud nevyřešené),
- zlepšování za měsíc – každodenní setkání týmu zajišťují identifikaci a eliminaci všeho, co brání pokroku projektu. Zlepšování je měřeno počtem dní, kdy je plnění některého úkolu (potažmo další postup) blokováno jinými úkoly.

Výsledky studie

Celkový čas potřebný pro dodání softwaru

Konzistentní proces vývoje softwaru je dle studie ten, který v otázce celkového času potřebného pro dodání softwaru vykazuje nízkou odchylku od průměru – cílem by tedy mělo být kontinuální snižování odchylky.

- Odchylku se podařilo snížit o 47 %.
- Software je v průměru dodáván o 37 % rychleji (z 22,8 pracovních dnů na 14,4), což umožňuje lepší reakci na aktuální potřeby zákazníka.

Doba trvání vývoje

- Odchylka v dodacích časech se snížila o 78 % (z 30,5 dnů na 6,8).
- Průměrná doba dodávky (spojeno s rozpadem práce na menší jednotky) se snížila o 73 % (z 9,2 pracovních dnů na 2,5 – v průběhu devíti měsíců).

Počet dodávek za měsíc

- Počet dodávek se zvýšil osminásobně – z původních dvou na šestnáct.

POČET CHYB ZA TÝDEN

- Snížení odchylky od průměru, chyby ubylo a jsou opravovány rychleji (zřejmě díky zkvalitnění struktury kódu), rovněž se mírně snížil počet nevyřešených chyb.
- Tým samotný přisuzuje snížení chybovosti i tomu, že byl vyčleněn čas na to, aby mohli zlepšovat kvalitu svého kódu.

ZLEPŠOVÁNÍ ZA MĚSÍC

- Průměrný počet dní, ve kterých byl další postup projektu blokován, se výrazně snížil, konkrétně o 81 % (z původních 25,8 dnů na 4,9 dnů měsíčně).

Závěr

Alternativní hypotéza studie byla následující: „Aplikace Lean přístupu zlepší proces vývoje softwaru.“ Vzhledem k nashromážděným kvantitativním a kvalitativním datům, byla hypotéza touto studií podpořena.

8.3.2 Zpráva popisující zavádění Lean principů ve firmě Capital One

Stručná charakteristika této případové studie byla vytvořena na základě zahraničního zdroje [PARNELL-KLABO, 2006][18], ve snaze demonstrovat využití konceptu Lean Software Development v praxi.

Tabulka 14: Informace o analyzovaném případu – Capital One [autorka], zpracováno dle [PARNELL-KLABO, 2006]

Analyzovaný podnik:	**Capital One** Společnost poskytující finanční služby V rámci společnosti existuje část zabývající se vývojem softwaru
Časové období provádění studie:	12 týdnů roku 2004

Úvod

Zpráva se věnuje popisu procesu zavádění zkušebního produktu, který započal v roce 2004. Pro zavedení tohoto produktu byla zvolena kombinace agilních metodik a Lean principů. Hlavním cílem bylo zkrácení doby dodávek. Pracovníci, jsou obvykle přiřazeni ke třem až čtyřem projektům současně.

Aplikace Lean principů

Pro účely řešení problému dlouhé doby dodávek, byl sestaven tzv. „Hlavní tým“. Dále byly provedeny následující kroky:

- důraz na definování hodnoty z pohledu zákazníka,
- zlepšování procesů pomocí metody DMAIC[19] – probíhalo 12. týdnů,
- tvorba mapy hodnotového toku a rozčlenění jednotlivých částí na činnosti: a) tvořící hodnotu (z pohledu zákazníka), b) tvořící byznys hodnotu c) netvořící žádnou hodnotu z předchozích,
- kategorizace činností do jednotlivých oblasti plýtvání,
- hledání zdrojů těchto plýtvání za pomoci metody pěti „Proč?“,
- výběr agilní metodiky Scrum pro doplnění Lean principů.

Výsledky zprávy

- Ukázalo se, že samotná implementace tvoří pouze 10 % celkové doby vývoje softwaru.

---

[18] Dostupné na adrese http://www.agilexp.com/Agile200xPapers/Agile2006-ExperienceReports/XR12-ParnellklaboIntroducingLean_revised%20final.pdf

[19] DMAIC – Define, Measure, Analyze, Improve, Control – Definovat, Měřit, Analyzovat, Zlepšit, Sledovat (metoda je součástí konceptu Six Sigma) [PARNELL-KLABO, 2006]

- V průběhu vývoje byla zaznamenána redundantní dokumentace.
- Konkrétní části toku mají přehled o návaznosti a průběhu svých jednotlivých kroků, avšak jednotlivé části nemají přehled o tom, jak fungují části ostatní (důsledkem jsou zbytečné časové prostoje spojené s nejistotou, koho je vlastně následně potřeba kontaktovat).

Problémy při zavádění

- Nedostatek prostor pro nastolení větší kolaborace členů jednotlivých týmů. Problém byl komunikován s managementem a úspěšně vyřešen.
- Nutnost získat podporu představenstva – vyřešeno díky přímé komunikaci, rozhovorům.
- Vztah firemní kultury a změn – vyřešeno díky komunikaci a díky tomu, že zainteresované strany svolily k otevřeným rozhovorům a realizovaly návštěvy pracoviště.

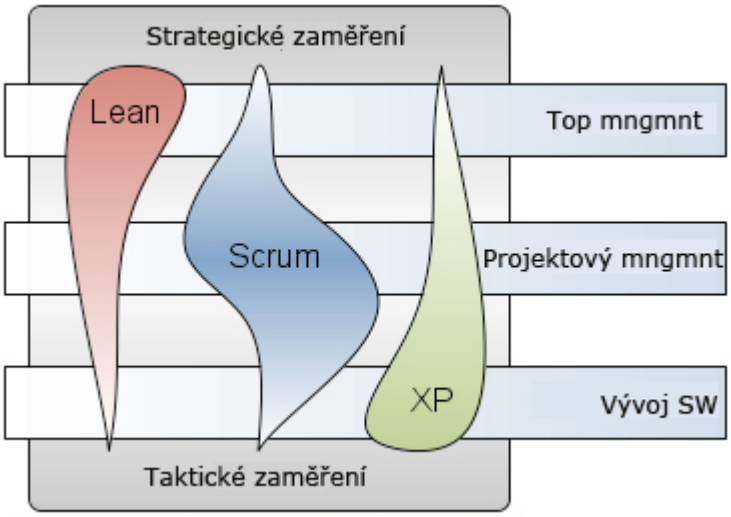
Přínosy zavedení Lean principů

- Počet úkolů v procesu vývoje softwaru byl snížen o 50 %.
- Došlo ke zkrácení doby dodávek.
- Doba celkového vývoje se zkrátila o 40 % (v porovnání s původním vodopádovým přístupem).
- Doba psaní kódu a testování se snížila o 30 %.
- Náklady spojené s využitými zdroji byly o 10 % menší než přidělené finanční zdroje.
- Nedošlo k navýšení nákladů ani ke snížení kvality.

9 Závěr

Cílem práce bylo představit koncept Lean Software Development českému čtenáři – popsat historii konceptu, jeho začlenění v systému konceptů a metodik vývoje softwaru obecně, různé pohledy na jeho interpretaci, principy konceptu (a jejich různé podoby), nástroje použitelné pro podporu těchto principů, příklady využití konceptu v praxi a možná omezení s jeho aplikací spojená. Tento cíl považuji za splněný.

Koncept Lean Software Development přináší do oblasti vývoje softwaru důležitou dimenzi v podobě vnímání celku (kontextu, do kterého je vlastní vývoj zasazen). Jak ukazuje Obrázek 10, LeanSD je spíše strategicky zaměřený koncept. Tento obrázek také demonstruje, že LeanSD vhodně doplňuje stávající agilní, převážně taktické, přístupy k vývoji softwaru.



Obrázek 10: Znázornění vztahu LeanSD, metodiky Scrum a Extrémního programování, přeloženo autorkou dle [BOURNE, 2010]

Vzhledem k netradiční historii konceptu, sahající ke konceptům pro řízení výroby, byly v kapitole 4 shrnuty nejdůležitější principy a praktiky Lean přístupů v oblasti výroby. Pro čtenáře této práce tak může být zajímavé srovnávat a zjišťovat některé překvapivě shodné znaky, ačkoliv se obecně jedná o oblasti vzájemně poměrně odlišné.

Dále se práce v 5 věnovala kategorizaci konceptu, jeho postavení vůči agilním metodikám vývoje softwaru a různým myšlenkovým směrům LeanSD.

V průběhu rešerše zdrojů pro psaní práce jsem narazila na nekonzistenci v oblasti formulace principů konceptu LeanSD (charakteristických pro konkrétní myšlenkový směr) a proto byly tyto jednotlivé přístupy byly shrnuty v kapitole 6.

V kapitole 7, věnované popisu nástrojů pro podporu principů LeanSD se práce zabývala, kromě tradičních nástrojů převzatých z oblasti výroby, zejména otázkou využití stávajících agilních přístupů k vývoji software pro podporu principů LeanSD. V kapitole se pak hypotéza

z kapitoly 5.1, že některé praktiky a nástroje agilních metodik vývoje softwaru jsou validními podpůrnými nástroji principů LeanSD, dle mého názoru potvrdila. Konkrétními metodikami, jejichž nástroje byly v tomto ohledu popsány jsou Scrum, Extrémní programování, nebo Vývoj řízený testy.

9.1 Přínos k řešené problematice

Vlastní přínos této práce k řešené problematice se týká zejména:

- vytvoření uceleného česky psaného materiálu o konceptu LeanSD,
- kategorizace konceptu LeanSD ve vztahu k agilním metodikám vývoje softwaru,
- souhrnu jednotlivých směrů v oblasti LeanSD,
- přiblížení možných pohledů na definici konceptu LeanSD,
- souhrnu možných podob výčtů principů LeanSD,
- detailnější kategorizace jednotlivých nástrojů pro podporu principů LeanSD, tak aby byl jasně zřetelný jejich vztah vzhledem k danému principu,
- pro účely demonstrace provázanosti nástrojů a principů LeanSD byl rovněž vytvořen tabulkový souhrn (viz Příloha B).

9.2 Další náměty pro řešení v této oblasti

Jako případné vhodné rozšíření práce navrhuji provedení aplikace konceptu LeanSD na konkrétním případě, popis průběhu jeho zavádění a následné zhodnocení výsledků (tzn. tvorba vlastní případové studie). Dalším možným námětem by mohl být výběr a zhodnocení vhodných softwarových nástrojů pro podporu principů LeanSD. Tato témata považuji za zajímavá rozšíření, která tato bakalářská práce, z důvodu požadavků na její rozsah, již nemohla pokrýt.

10 Terminologický slovník

Tabulka 15: Terminologický slovník

TERMÍN	ZKRATKA	VÝZNAM [ZDROJ]
Agile Software Development	AgileSD	Vývoj softwaru, který je: „Rychlý, hbitý, čilý, aktivní a svižný, který nepostává zdlouhavě u jednotlivých fází a jehož touhou je co nejrychleji postupovat k vytyčenému cíli – k dodání fungující aplikace.“ [KADLEC, 2004]
Build		Buildy jsou verze softwaru, vznikající v průběhu jeho vývoje [PC Magazine Encyclopedia, c1981-2012]
	DMAIC	Metoda DMAIC (Define, Measure, Analyze, Improve, Control) – Definovat, Měřit, Analyzovat, Zlepšit, Sledovat (metoda je součástí konceptu Six Sigma) [PARNELL-KLABO, 2006]
General Motors	GM	Americký výrobce aut.
Checkstyle		Nástroj, který kontroluje, zda je Java kód psán dle standardu. [SourceForge.net, 2011]
	IS/ICT	„Informační systémy a informační a komunikační technologie“ [GÁLA, a další, 2006 str. 23]
Javadoc		Nástroj, který analyzuje kód a vytváří dokumentaci popisující třídy, rozhraní, konstruktory, metody a pole. [Oracle, c2011]
Kaizen		„Opakovaná aktivita a změna způsobu myšlení, která musí vycházet z nejnižší operativní úrovně současně s podporou nejvyššího vedení.“ [PROCHÁZKA, 2010]
koncept		pojetí
Lean Software Development	LeanSD	Koncept vývoje softwaru [autorka]
Lean Software & Systems Consortium	LeanSSC	Nezisková organizace na podporu podniků závislých na vývoji softwaru. [Lean Software and Systems Consortium , 2012]
Metodika vývoje softwaru		„Oblast popisující, jakým způsobem by měl probíhat vývoj softwaru s cílem získat kvalitní, dobře fungující a přitom rychle a pokud možno levně vytvořený programový produkt.“ [KADLEC, 2004]

TERMÍN	ZKRATKA	VÝZNAM [ZDROJ]
Minimální obchodovatelná funkcionalita		Minimální obchodovatelná funkcionalita (Minimal Marketable Feature – MMF) je část funkcionality, která reprezentuje podmnožinu zákazníkových požadavků a která, pokud je dodána jako samostatná jednotka, je schopna tvořit pro zákazníka hodnotu. [MIDDLETON, a další, 2011]
	PMD	Doplněk vývojářského prostředí, který prochází kód a hledá potenciální problémy (chyby, nepoužívané části kódu, příliš složité výrazy, duplicitní kód a další). [SourceForge.net, 2012]
Product Owner		Jedna z rolí v metodice Scrum.[autorka]
Refaktorizace		„Změna struktury systému beze změny jeho chování. Jejím cílem je zjednodušení, zpřehlednění a zajištění flexibility.“ [BUCHALCEVOVÁ, 2009 str. 151]
Scrum Master		Jedna z rolí v metodice Scrum.[autorka]
Six Sigma		Koncept pro zajišťování kvality a snižování počtu defektů [BREYFOGLE, 2003], původně spjatý s výrobou, avšak existuje rovněž modifikace pro oblast vývoje [POPPENDIECK, a další, 2003] (softwaru).
Software	SW	„Komplex programů, programových prostředků“ [GÁLA, a další, 2006 str. 468]
Systém		„Systém je komplex prvků, nacházejících se ve vzájemné interakci.“ [ROSICKÝ, 2009 str. 25]
Toyota Production Systém	TPS	Koncept řízení výroby v japonské automobilce Toyota

11 Seznam použité literatury

[ABRAHAMSSON, a další, 2009] ABRAHAMSSON, Pekka., MARCHESI, Michele a Frank MAURER. Agile *Processes in Software Engineering and Extreme Programming: 10th International Conference, XP 2009, Pula, Sardinia, Italy, May 25-29, 2009, Proceedings*. Berlin : Springer, 2009. ISBN 3642018521.

[Agile Sherpa, 2010] Agile Sherpa**.** Agile methodologies. Agile Sherpa. [Online] 29. červenec 2010. [Citace: 15. duben 2012.] Dostupný z <http://www.agilesherpa.org/intro_to_agile/agile_methodologies/>.

[AMBLER, a další, 2007] AMBLER, Scott W., KROLL, Per. Best practices for lean development governance, Part III: Roles and policies. Developer Works. [Online] 15. srpen 2007. [Citace: 24. duben 2012.] Dostupný z: <http://www.ibm.com/developerworks/rational/library/aug07/ambler_kroll/>.

[ANDERSON, 2010] ANDERSON, David J**.** David Anderson Talks Kanban, Agile and the Lean Software and Systems Consortium. InfoQ. [Online] 9. červen 2010. [Citace: 10. duben 2012.] Dostupný z: <http://www.infoq.com/interviews/anderson-kanban-agile-leanssc>.

[ANDERSON, 2011] Lean Software Development. MSDN. [Online] Listopad 2011. [Citace: 12. březen 2012.] Dostupný z: <http://msdn.microsoft.com/en-us/library/hh533841%28v=vs.110%29.aspx>.

[BASL, a další, 2008] BASL, Josef., BLAŽÍČEK, Roman**.** *Podnikové informační systémy : Podnik v informační praxi : 2., výrazně přepracované a rozšířené vydání.* Praha : Grada Publishing a.s., 2008. ISBN 978-80-247-2279-5.

[BECK, 2009] BECK, Kent**.** Agile and Lean Software Development – an Oxymoron? Agile Blog. [Online] Rally Software, 22. Červen 2009. [Citace: 17. březen 2012.] Dostupný z: <http://www.rallydev.com/agileblog/2009/06/agile-and-lean-software-development-an-oxymoron/#comment-3507>.

[BELLWARE, 2009] BELLWARE, Scott**.** Decade of Agile, Dawn of Lean. Scott Belvare. [Online] 23. únor 2009. [Citace: 16. Březen 2012.] Dostupný z: <http://blog.scottbellware.com/2009/02/decade-of-agile-dawn-of-lean.html>.

[BOURNE, 2010] BOURNE, Geoffrey**.** The Marriage of Lean, Scrum and Extreme Programming (XP): How to Align Agile Across an Organization. Agile Journal. [Online] 10. květen 2010. [Citace: 28. duben 2012.] Dostupný z:<http://www.agilejournal.com/articles/columns/column-articles/2878-the-marriage-of-lean-scrum-and-extreme-programming-xp-how-to-align-agile-across-an-organization>.

[BREYFOGLE, 2003] BREYFOGLE, Forrest W**.** *Implementing Six Sigma: Smarter Solutions Using Statistical Methods*. New York : John Wiley & Sons, 2003. ISBN: 0471265721.

[BUCHALCEVOVÁ, 2009] BUCHALCEVOVÁ, Alena**.** *Metodiky budování informačních systémů*. Praha : Oeconomica, 2009. ISBN 978-80-245-1540-3.

[BURIÁNEK, c2007] BURIÁNEK, Martin**.** Kano model – kvalita služeb a výrobků není jednorozměrná. Interquality, spol. s r.o. [Online] c2007. [Citace: 2. duben 2012.] Dostupný z: <http://www.interquality.cz/%C4%8CL%C3%81NKY/tabid/67/ItemId/30/View/Details/AMID/431/Default.aspx>.

[CAWLEY, a další, 2010] CAWLEY , Oisín., WANG , Xiaofeng a Ita RICHARDSON**.** Lean/Agile Software Development Methodologies In Regulated Environments – State of the art. University of Limerick. [Online] 2010. [Citace: 12. duben 2012.] Dostupný z: <http://ulir.ul.ie/bitstream/handle/10344/683/2010-Cawley-%20Lean-Agile.pdf?sequence=2>.

[CLINE, c2012] CLINE, Austin**.** What is the Scientific Method? About.com. [Online] c2012. [Citace: 16. duben 2012.] Dostupný z: <http://atheism.about.com/od/philosophyofscience/a/ScientificMethod.htm>.

[COHEN, a další, 2004] COHEN, David., LINDVALL, Mikael a Patricia COSTA. An Introduction to Agile Methods. [Online] Elsevier Inc., 2004. [Citace: 12. březen 2012.] Dostupný z: <http://www.cs.umd.edu/~mvz/cmsc435-s09/pdf/agile-paper.pdf>.

[COHN, 2008] COHN, Meredith**.** GM loses $1 billion, will cut 25,000 jobs: Company plans to boost efficiency and quality, build fewer vehicles. The Baltimore Sun. [Online] 8. červen 2008. [Citace: 10. Březen 2012.] Dostupný z: <http://www.baltimoresun.com/business/bal-te.bz.gm08jun08,0,2480020.story>.

[ČERNÁ, 2011] ČERNÁ, Martina. *Dokumentace při vývoji softwaru*. Praha, 2011. Bakalářská práce. Vysoká škola ekonomická.

[De ZOYSA, 2011] De ZOYSA, Lakmali**.** Software Quality Assurance in Agile and Waterfall Software Development Methodologies: A Gap Analysis. [Online] University Of Colombo, Únor 2011. [Citace: 12. březen 2012.] Dostupný z: <http://archive.cmb.ac.lk/research/bitstream/70130/1335/1/Viva%20Corrected%20Thesis_sent.pdf>.

[GÁLA, a další, 2006] GÁLA, Libor., TOMAN, Prokop a Jan POUR. *Podniková informatika*. Praha : Grada Publishing a.s., 2006. ISBN 8024712784.

[HEFNEROVÁ, 2012] HEFNEROVÁ, Lucie**.** *Koncepty řízení výroby – štíhlá výroba a MRP II*. Praha, 2012. Semestrální práce. Vysoká škola ekonomická.

[HIBBS, a další, 2009] HIBBS, Curt., JEWETT, Steve. a Mike SULLIVAN. *The Art of Lean Software Development. Sebastopol*: O'Reilly Media, Inc., 2009. ISBN 978-0-596-51731-1.

[HIGHSMITH, 2002] HIGHSMITH, Jim. What Is Agile Software Development? CrossTalk: The Journal of Defense Software Engineering. [Online] 2002. [Citace: 12. březen 2012.] Dostupný z: <http://agilesweden.com/doc/oct02.pdf>.

[HOLDBERH, 2011] HOLDBERH, Raman**.** *Automatizované testování webových aplikací*. Praha, 2011. Bakalářská práce. Vysoká škola ekonomická.

[CHARETTE, 2003] CHARETTE, Robert N. Challenging the Fundamental Notions of Software Development. The IT Metrics and Productivity Institute. [Online] Computer Aid, Inc., 2003. [Citace: 1. duben 2012.] Dostupný z: <http://itmpi.org/assets/base/images/itmpi/privaterooms/robertcharette/ChallengingtheFundamentalNotions.pdf>.

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

[KELLY, 2009] KELLY, Allan**.** Agile Demystified (v.2). DevelopMentor. [Online] Říjen 2009. [Citace: 16. březen 2012.] Dostupný z: <http://www.develop.com/agiledemystified>.

[KOCHNEV, 2007] KOCHNEV, Ivan. What, if any, are the differences between the Toyota Production System and Lean? : A research paper. [Online] 2007. [Citace: 9. březen 2012.] Dostupný z: <http://innovationlighthouse.com/TPSversusLean.aspx>.

[KOŠÁK, 2010] KOŠÁK, Vojtěch. *Analýza výuky agilních metod vývoje softwaru na vysokých školách v ČR*. Praha, 2010. Bakalářská práce. Vysoká škola ekonomická.

[LADAS, 2009] LADAS, Corey**.** Introduction to Lean Software Development. Shaping Software. [Online] 15. Červen 2009. [Citace: 12. březen 2012.] Dostupný z: <http://shapingsoftware.com/2009/06/15/introduction-to-lean-software-development/>.

[Lean company, c2010] Lean company**.** LEAN slovník. Lean company. [Online] c2010. [Citace: 14. Březen 2012.] Dostupný z: <http://www.leancompany.cz/leanslovnik.html>.

[Lean Software and Systems Consortium, 2012] Lean Software and Systems Consortium**.** Vision. Lean Software and Systems Consortium. [Online] Lean Software and Systems Consortium, 2012. [Citace: 1. duben 2012.] Dostupný z: <http://www.leanssc.org>/.

[McMANUS, 2005] McMANUS, Hugh L. Product Development Value Stream Mapping (PDVSM) Manual. Lean Advancement Initiative. [Online] Massachusetts Institute of Technology, ZZáří 2005. [Citace: 2. duben 2012.] Dostupný z: <http://lean.mit.edu/component/docman/doc_download/1090-product-development-value-stream-mapping-manual>.

[MIDDLETON, a další, 2011] MIDDLETON, Peter., JOYCE , David**.** Lean Software Management: BBC Worldwide. Systems Thinking, Lean and Kanban. [Online] únor 2011. [Citace: 26. duben 2012.] Dostupný z:

<http://leanandkanban.files.wordpress.com/2011/04/lean-software-management-bbc-worldwide-case-study-feb-2011.pdf>.

[MIDDLETON, a další, 2005] MIDDLETON, Peter., SUTTON, James. *Lean Software Strategies: Proven Techniques for Managers and Developers*. New York : Productivity Press, 2005. ISBN 1563273055.

[Net Objectives, 2010] Net Objectives**.** LeanSSC Wiki . LeanModel. [Online] 16. prosinec 2010. [Citace: 22. březen 2012.] Dostupný z: <http://www.leanssc.org/wiki/index.php?title=LeanModel>.

[Oracle, c2011] Oracle**.** Javadoc Technology. Oracle : Java SE Documentation. [Online] c2011. [Citace: 12. duben 2012.] Dostupný z: <http://docs.oracle.com/javase/6/docs/technotes/guides/javadoc/index.html>.

[PANNONE, 2010] PANNONE, Russell. Is "Agile Methodology" an Oxymoron and Counterintuitive to Agile-Lean Product Development and Craftsmanship? Agile Journal. [Online] 7. Březen 2010. [Citace: 12. Březen 2012.] Dostupný z: <http://www.agilejournal.com/articles/columns/column-articles/3226-is-agile-methodology-an-oxymoron-and-counterintuitive-to-agile-lean-product-development-and-craftsmanship>.

[PARNELL-KLABO, 2006] PARNELL-KLABO, Emma. Introducing Lean Principles with Agile Practices at a Fortune 500 Company. Agile Coaching Experience. [Online] 2006. [Citace: 28. duben 2012.] Dostupný z: <http://www.agilexp.com/Agile200xPapers/Agile2006-ExperienceReports/XR12-ParnellklaboIntroducingLean_revised%20final.pdf>.

[PC Magazine Encyclopedia, c1981-2012] PC Magazine Encyclopedia. Definition of: build. PC Magazine Encyclopedia. [Online] c1981-2012. [Citace: 15. Duben 2012.] Dostupný z: <http://www.pcmag.com/encyclopedia_term/0,2542,t=build&i=39031,00.asp>.

[POPPENDIECK, a další, 2006] POPPENDIECK, Mary, POPPENDIECK, Tom. *Implementing Lean Software Development From Concept to Cash*. Boston : Addison-Wesley, 2006. ISBN 0321437381.

[POPPENDIECK, a další, 2003]. POPPENDIECK, Mary, POPPENDIECK, Tom. *Lean Software Development : An Agile Toolkit*. Boston : Addison-Wesley, 2003. ISBN 0-321-15078-3.

[POPPENDIECK, a další, 2010]. c2010. POPPENDIECK, Mary, POPPENDIECK, Tom. Principles. Lean Software Development. [Online] Poppendieck.LLC, c2010. [Citace: 20. duben 2013.] Dostupný z: <http://www.poppendieck.com/>.

[POPPENDIECK, 2011] POPPENDIECK, Mary**.** Don't Separate Design from Implementation . LeanEssays. [Online] 19. srpen 2011. [Citace: 21. duben 2012.] Dostupný z: <http://www.leanessays.com/2011/08/dont-separate-design-from.html>.

[PROCHÁZKA, 2010] PROCHÁZKA, Jarek**.** Agile samotný nestačí, je čas pro Lean. Differ.cz. [Online] 23. Listopad 2010. [Citace: 12. Březen 2012.] Dostupný z: <http://www.differ.cz/?p=189>.

[PROCHÁZKA, 2010]. PROCHÁZKA, Jarek**.** Kaizen workshop. Differ.cz. [Online] 7. listopad 2010. [Citace: 20. duben 2013.] Dostupný z: <http://www.differ.cz/?page_id=185>.

[PROCHÁZKA, a další, 2010] PROCHÁZKA, Jaroslav, VAJGL, Marek a Cyril KLIMEŠ**.** Informační systémy 2 : Učební text. [Online] Ostravská univerzita v Ostravě, 2010. [Citace: 12. Duben 2012.] Dostupný z : <http://www1.osu.cz/~prochazka/infs2/INFS2_skripta.pdf>.

[ROSICKÝ, 2009] ROSICKÝ, Antonín. *Informace a systémy : Základy teorie pro úspěšnou praxi*. Praha : Oeconomica, 2009. ISBN 978-80-245-1629-5.

[SAGE, a salší, 1990] SAGE, Andrew P., PALMER, James D. *Software Systems Engineering*. New York : John Wiley & Sons, 1990. ISBN 047161758X.

[SCOTLAND, 2009] SCOTLAND, Karl. Announcing the Formation of the Lean Software & Systems Consortium. Avail Agility. [Online] 7. květen 2009. [Citace: 22. duben 2012.] Dostupný z: <http://availagility.co.uk/2009/05/07/announcing-the-formation-of-the-lean-software-systems-consortium/>.

[SCOTLAND, 2010]. SCOTLAND, Karl**.** Introducing Karl Scotland. Lean Software and Systems Consortium. [Online] 19. březen 2010. [Citace: 10. duben 2012.] Dostupný z: <http://www.leanssc.org/2010/03/introducing-karl-scotland/>.

[SHAH, 2007] SHAH, Shahid**.** Resume Driven Development (RDD). The Healthcare IT Guy. [Online] 19. leden 2007. [Citace: 4. duben 2012.] Dostupný z: <http://www.healthcareguy.com/2007/01/19/resume-driven-development-rdd/>.

[SCHIFFER, 2011] SCHIFFER, Bernd. Kaizen and Kaikaku Regardless of Scrum and Kanban. Agile Trail. [Online] 12. říjen 2011. [Citace: 20. duben 2012.] Dostupný : <http://agiletrail.com/2011/10/12/kaizen-and-kaikaku-regardless-of-scrum-and-kanban/>.

[SIMS, 2008] SIMS, Chris**.** Fowler: Agile Vs. Lean Misses the Point. InfoQ. [Online] 1. Září 2008. [Citace: 17. březen 2012.] Dostupný z : <http://www.infoq.com/news/2008/09/Not-Agile-Vs-Lean>.

[SOBEK, 2010] SOBEK, Durward K. Toyota Style Problem solving A3 Reports. Lean Enterprise Institute. [Online] 16. leden 2010. [Citace: 23. duben 2012.] Dostupný z: <http://www.lean.org/admin/km/documents/c0db04df-5d64-416b-921d-9471b9726d3d-A3_template.pdf>.

[SourceForge.net, 2011] SourceForge.net**.** Checkstyle. SourceForge.net. [Online] 11. Květen 2011. [Citace: 12. duben 2012.] Dostupný z: <http://checkstyle.sourceforge.net/>.

[SourceForge.net, 2012]. SourceForge.net**.** Welcome to PMD. SourceForge.net. [Online] 31. Leden 2012. [Citace: 12. Duben 2012.] Dostupný z: <http://pmd.sourceforge.net/>.

[STRATEGOS, c2012] STRATEGOS, Inc. Pioneers of Lean Manufacturing — Taiichi Ohno & Shigeo Shingo: Featuring an Interview With Norman Bodek. STRATEGOS, Inc. [Online] c2012. [Citace: 12. březen 2012.] Dostupný z: <http://www.strategosinc.com/downloads/lean_pioneers-dl1.pdf>.

[ŠOCHOVÁ, 2012] ŠOCHOVÁ, Zuzana. Motivace.Zuzi's blog. [Online] 23. únor 2012. [Citace: 16. duben 2012.] Dostupný z : <http://soch.cz/blog/management/motivace/>.

[Toyota Peugeot Citroën Automobile Czech, c2003-2007] Toyota Peugeot Citroën Automobile Czech**.** Jidoka. Toyota Peugeot Citroën Automobile Czech. [Online] c2003-2007. [Citace: 16. březen 2012.] Dostupný z: <http://www.tpca.cz/cz/vyrobni-system-toyota/vyroba/jidoka>.

[TRILOGIQ, c2011] TRILOGIQ**.** 7 druhů plýtvání (muda). [Online] c2011. [Citace: 14. březen 2012.] Dostupný z: <http://trilogiq.cz/filosofie-stihle-vyroby/7-druhu-plytvani-muda/>.

[VONDRÁK, 2002] VONDRÁK, Ivo**.** Úvod do softwarového inženýrství. [Online] 2002. [Citace: 12. duben 2012.] Dostupný z: <http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf>.

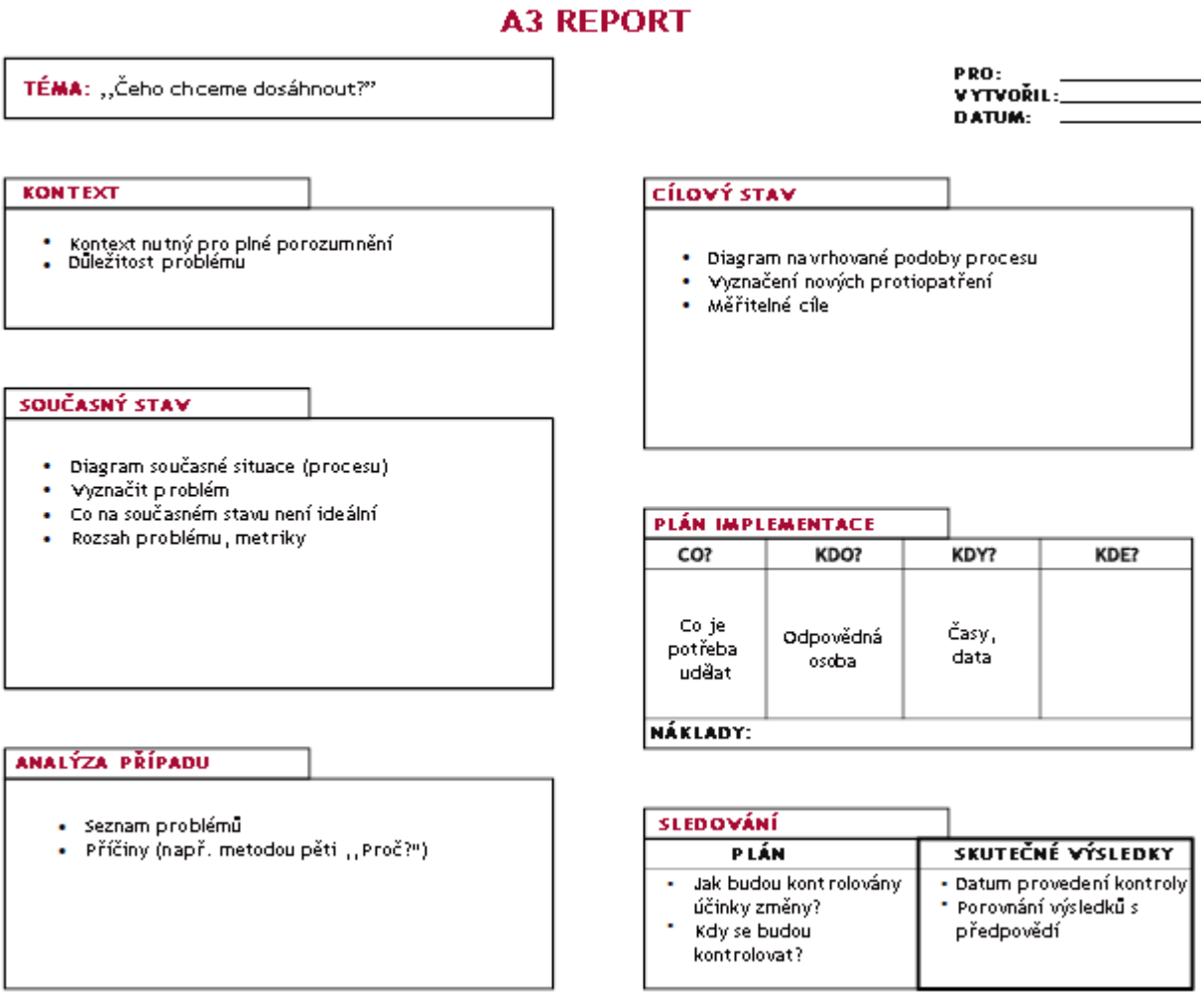
[WOMACK, a další, 2003] WOMACK, James P., JONES, Daniel T.*Lean Thinking: Banish Waste and Create Wealth In Your Corporation*. New York : Free Press, 2003. ISBN 0-7432-4927-5.

12 Seznam obrázků a tabulek

12.1 Seznam obrázků

12.2 Seznam tabulek

Příloha A: Ukázka šablony pro metodu A3



Obrázek 11: A3 report, vytvořeno autorkou dle [SOBEK, 2010]

Příloha B: Tabulka nástrojů a principů

Tabulka nastiňuje provázanost jednotlivých nástrojů a jimi podporovaných principů. Tabulka byla zpracována autorkou na základě předchozích částí této práce a je jednou z možných variant přisouzení provázaností nástrojů a principů.

Tabulka 16: Vizualizace podpory principů konceptu LeanSD jednotlivými nástroji [autorka]

	ELIMINOVAT PLÝTVÁNÍ	VČLEŇOVAT KVALITU DO VÝVOJE	VYTVÁŘET ZNALOSTI	ODKLÁDAT ZÁVAZKY	DODÁVAT CO NEJRYCHLEJI	DŮVĚRA A RESPEKT K LIDEM PODÍLEJÍCÍM SE NA VÝVOJI	OPTIMALIZOVAT CELEK
Kano model	●	●					
Mapa hodnotového toku	●				●		●
Sada zásad pro každou oblast plýtvání	●	●					●
Efektivní komunikace se zákazníkem	●	●	●		●		
Iterativní vývoj	●	●	●	●	●	●	●
Testování	●	●	●				
Vývoj řízený testy	●	●					
Automatizace	●	●			●		

	Eliminovat plýtvání	Včleňovat kvalitu do vývoje	Vytvářet znalosti	Odkládat závazky	Dodávat co nejrychleji	Důvěra a respekt k lidem podílejícím se na vývoji	Optimalizovat celek
Párové programování	●	●	●				
Refaktorizace	●	●	●		●		●
Nepřetržitá integrace		●	●		●		●
Sada principů 5s	●	●	●				●
Komunikace	●	●	●			●	●
Vědecká metoda		●	●				
Metoda A3			●				
Vývoj založený na souboru možností		●	●	●			
Speciální pravidla			●			●	
Dokumentace		●	●				
Wiki systémy		●	●			●	
Minimalizace závislostí	●	●		●			●

	Eliminovat plýtvání	Včleňovat kvalitu do vývoje	Vytvářet znalosti	Odkládat závazky	Dodávat co nejrychleji	Důvěra a respekt k lidem podílejícím se na vývoji	Optimalizovat celek
Neoddělovat návrh od implementace	●	●	●	●			●
Krátké iterace, malé dávky	●	●	●	●	●		●
Samořízené týmy		●	●			●	
Motivace		●				●	
Kanban			●	●		●	●
Andon			●			●	
Tvorba přehledů			●			●	●
Systémové myšlení		●				●	●
Neustále zlepšování	●	●	●			●	●