Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	LS 2015/2016
Autoři	Robert Rojo (xrojr01) Jakub Krejcar (xkrej47) Tomáš Kazatel (kazt01) Jaroslav Svoboda (xsvoj66)
Téma	Dokumentace v agilním vývoji softwaru
Datum odevzdání	15. 5. 2016

Abstrakt

Semestrální práce se zabývá agilním vývojem a jeho dokumentací. Nejprve shrnuje poznatky o agilním vývoji, kde se detailněji zaměřuje na agilní manifest. Další částí je rozbor agilní dokumentace z pohledu významu, potřeby, udržitelnosti a problémy spojenými s agilní dokumentací. V neposlední řadě se věnuje kvalitě dokumentace a problematice regulovaného vládního prostředí.

Klíčová slova

Agilní vývoj softwaru, agilní dokumentace, agilní vývoj v regulovaném sektoru, agilní manifest.

Obsah

1 Úvod 5

1.1 Volba tématu 5

1.2 Cíl práce 5

2 Agilní vývoj softwaru 6

2.1 Tradiční versus agilní vývoj 6

2.2 Agilní manifest ve vztahu k dokumentaci 7

2.2.1 Jednotlivci a interakce 8

2.2.2 Fungující software 9

2.2.3 Spolupráce se zákazníkem 9

2.2.4 Reakce na změny 10

2.3 Principy stojící za Agilním Manifestem 10

2.3.1 Včasné a průběžné dodávání 10

2.3.2 Vítání změn v požadavcích 11

2.3.3 Pravidelně dodávaný fungující software 11

2.3.4 Spolupráce lidí z byznysu a vývoje 11

2.3.5 Důraz na samostatnost a motivaci 12

2.3.6 Důraz na osobní komunikaci 12

2.3.7 Důraz na fungující software 13

2.3.8 Dlouhodobě udržitelné tempo 13

2.3.9 Pozornost věnovaná technické výjimečnosti a dobrému designu 13

2.3.10 Jednoduchost 14

2.3.11 Samo-organizující se týmy 14

2.3.12 Zpětná vazba a zlepšování sebe sama 14

3 Agilní dokumentace 16

3.1 Význam dokumentace při vývoji softwaru 16

3.2 Problémy spojené s dokumentací 16

3.3 Kdy je dokumentace skutečně agilní 19

3.3.1 Pokud maximalizuje ROI 19
3.3.2 Zúčastněné strany jsou si vědomi TCO 19
3.3.3 Agilní dokumentace je štíhlá, ale dostatečná 19
3.3.4 Agilní dokumentace plní svůj účel 19
3.3.5 Agilní dokumentace popisuje „Co je dobré vědět“ 20
3.3.6 Agilní dokumentace má vždy svou cílovou skupinu 20
3.3.7 Agilní dokumentace je především dostatečná 20
3.4 Kvalita dokumentace v agilním vývoji 21
3.4.1 Typy dokumentů 21
3.4.2 Vliv dokumentace na úspěšnost projektu 22
3.4.3 Udržování dokumentace 23
3.4.4 Kdy je vhodné tvořit dokumentaci 24
3.4.5 Nástroje pro podporu kvality dokumentace 25
3.5 Agilní vývoj v regulovaných prostředích a vládním sektoru 26
4 Závěr 28
5 Literatura 29

1 Úvod

1.1 Volba tématu

V agilním vývoji často hrozí nepochopení agilního manifestu, který uvádí, že chceme-li vyvíjet agilně, měli bychom „upřednostňovat fungující software před vyčerpávající dokumentací“. Agilní týmy pak nekladou důraz na tvorbu a správu dokumentace, protože je to zpravidla první věc, na které lze ušetřit pracovní sílu. Význam a smysl nedokumentovaného kódu a interní postupy popisující například nasazení aplikace na produkčním prostředí jsou jen v hlavách jedinců, kteří nemusí být vždy dostupní a mohou z týmu odejít, odebírajíce ze společnosti hodnotu a způsobujíce ve výsledku finanční ztrátu.

1.2 Cíl práce

Hlavním cílem práce je charakterizovat agilní vývoj softwaru, uvést jeho klíčové principy a praktiky v souvislosti s dokumentací a popsat význam dokumentace spolu s možnými přístupy její tvorby a správy v agilním prostředí. Hlavní cíl práce lze rozdělit na dílčí cíle, které charakterizují jednotlivé části této semestrální práce. Dílčími cíli jsou:

1. Uvést rozdíly mezi tradičním a agilním přístupem s ohledem dokumentaci
2. Vysvětlit myšlenku Agilního manifestu v souvislosti s tvorbou a správou dokumentace
3. Popsat význam dokumentace a dopady při jejím zanedbávání
4. Seznámit čtenáře s možnými nástroji pro tvorbu a správu dokumentace vhodnými pro užití při agilním vývoji softwaru
5. Popsat možnosti agilního vývoje softwaru v regulovaných prostředích a vládním sektoru

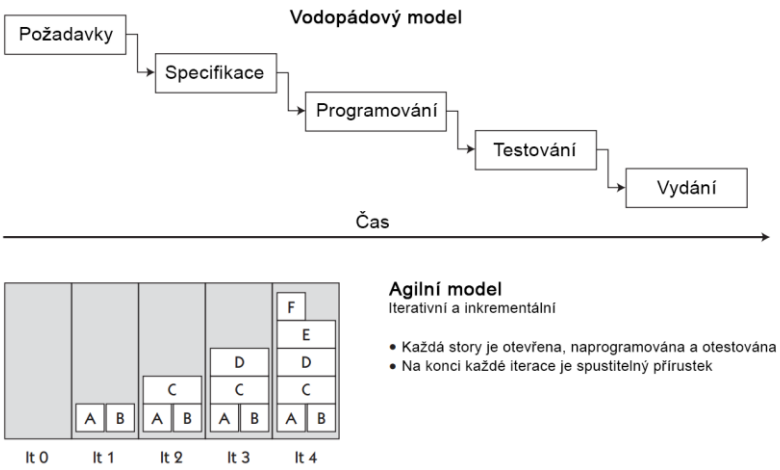
2 Agilní vývoj softwaru

Následující kapitola uvádí rysy agilního vývoje softwaru, stručně je srovnává s tradičními přístupy k vývoji a dále přibližuje myšlenku Manifestu Agilního vývoje softwaru v souvislosti s tvorbou a správou dokumentace, čímž uvozuje do problematiky této práce.

2.1 Tradiční versus agilní vývoj

K vývoji softwaru lze v zásadě přistupovat dvěma způsoby, tradičním a agilním. Tradiční přístup využívá vodopádový model životního cyklu, stanovuje požadavky na začátku vývoje, ještě před implementační fází. Už od počátku vývoje je jasné, jak má výsledek vypadat a fungovat. Všechny funkční i nefunkční požadavky jsou předem dány. Ve fázi analýzy a návrhu jsou detailně popsány dílčí části vyvíjeného systému, které jsou ve fázi implementace obvykle neměnné. V průběhu životního cyklu vývoje softwaru vzniká velké množství dokumentů popisujících vzhled i chování výsledného softwaru. Tyto dokumenty slouží jako podpůrné materiály, které mají za úkol buď stanovit předem pravidla vývoje vznikajícího softwaru, například v podobě diagramů případů užití, nebo jej zpětně popsat, například v podobě uživatelské dokumentace. Díky tomu, že jsou požadavky v průběhu vývoje neměnné, je i dokumentace stálá a není problém začít vytvářet uživatelskou dokumentaci už v raných fázích implementace. (pmdocuments.com, 2012)

Na druhé straně stojí agilní vývoj, v němž tým na straně dodavatele přímo komunikuje se zákazníkem a zaměřuje se na vytvoření a včasné dodání tzv. MVP (Minimum Viable Product), čili nejnutnějšího minima, které zákazník potřebuje. Potřeba zákazníka je obvykle zachycena pomocí uživatelských příběhů, tzv. User Stories, které pocházejí z Extrémního programování. Jednotlivé soběstačné inkrementy, obvykle odpovídající těmto příběhům poté tvoří potenciálně spustitelný produkt (v angličtině je používán výraz Potentially Shippable Product). Ten je zákazníkovi dodáván v pravidelných iteracích – v případě metodiky SCRUM v tzv. „sprintech“ o délce 2 až 4 týdnů. Každý z inkrementů je (až na výjimky) nezávislý na ostatních a projde v rámci iterace celým životním cyklem. Na konci iterace je tedy každý přírůstek otestovaný a schopný provozu v produkčním prostředí. Základní rozdíl v životním cyklu vývoje u tradičního a agilního přístupu ilustruje Obr. 1



Obr. 1 Srovnání tradičního a agilního modelu (Cripsin, a další, 2009)

Protože se požadavky v agilním vývoji rychle a často mění, je nutné na ně stejně rychle reagovat úpravou vyvíjeného softwaru. To způsobuje neustálou potřebu aktualizovat většinu dokumentace, což je časově a finančně nevýhodné, a navíc v mnoha případech zbytečné. Z toho důvodu jsou obvykle dokumentovány pouze nejdůležitější části vyvíjeného systému. Problém nastává v momentě, kdy není dokumentováno ani nejnutnější minimum. V ten okamžik může opět růst finanční a časové zatížení z důvodu nedostupnosti klíčových informací.

2.2 Agilní manifest ve vztahu k dokumentaci

Myšlenka agilního vývoje je zachycena v Manifestu Agilního vývoje softwaru, jenž byl v roce 2001 sepsán 17 autory, mezi které patří například Jim Highsmith, Martin Fowler a Alistair Cockburn. Ti tvrdí, že má-li být vývoj agilní, měly by být v jeho průběhu zastávány hodnoty, které manifest definuje. Mezi ně patří upřednostňování:

- **jednotlivců a interakcí** před *procesy a nástroji*,
- **fungujícího softwaru** před *vyčerpávající dokumentací*,
- **spolupráce se zákazníkem** před *vyjednáváním o smlouvě*,
- **reakcí na změny** před *dodržováním plánu*. (agilemanifesto.org, 2001)

Myšlenku manifestu si může vyložit každý jinak, což může způsobovat problém (nejen) při tvorbě a správě dokumentace. To komentuje i Ambler, podle kterého si manažeři na seniorních pozicích si myslí, že jim jsou zásady dané manifestem jasné, ale obvykle se ukáže, že tomu tak není. Podle něj tito manažeři tvrdí, že jsou jejich zaměstnanci tím nejdůležitějším ve společnosti, ale přitom po nich chtějí, aby se řídili normami ISO řady 9000, a ve skutečnosti se k zaměstnancům chovají jako k běžným nahraditelným aktivům. Uvádí také, že management často trvá na tom, aby vývojový tým dodržoval jasně stanovené procesy, ale neposkytne jim k tomu dostatek zdrojů. Dále říká, že se všichni shodnou na tom, že cílem vývoje softwaru je

jeho tvorba, ale přesto mají potřebu zabývat se několik měsíců tvorbou dokumentace, která popisuje, jak bude software budován, místo toho, aby si jednoduše „vyhrnuli rukávy“ a začali pracovat na samotném vzniku softwaru. Ambler přímo kritizuje i jistou část komunity vývojářů, která tvrdí, že vyvíjí agilně, protože nevytváří dokumentaci, což vede k nárůstu negativních ohlasů ze strany vyznavačů tradičních přístupů. (Ambler, 2002)

Z uvedených čtyř pravidel Agilního Manifestu se dokumentace přímo týká pouze to druhé, a to „upřednostňování fungujícího softwaru před vyčerpávající dokumentací“. Ostatní pravidla přímo o dokumentaci nepojednávají, nicméně je i přesto lze na zkoumanou oblast aplikovat. Následující podkapitoly popisují a komentují jednotlivá pravidla při jejich aplikaci na problematiku tvorby a správy dokumentace.

2.2.1 Jednotlivci a interakce

„Upřednostňování jednotlivců a interakcí před procesy a nástroji“

Ambler zde především zdůrazňuje sílu týmové práce. Říká, že nezkušeným a nespolupracujícím lidem nepomohou ani ty nejlepší a nejdražší nástroje. Tým by se měl zaměřit nejprve na možnosti efektivní týmové spolupráce a teprve poté se zabývat výběrem a zaváděním procesů a pomocných nástrojů. (Ambler, 2002)

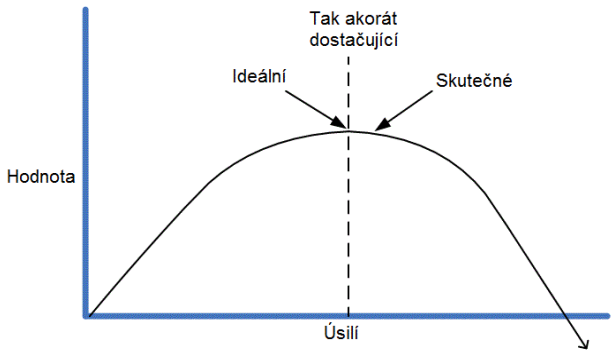
Toto pravidlo by si někteří mohli vyložit jako úplné zavržení procesu tvorby a udržování dokumentace. Za předpokladu, že si zákazník přímo nevyžádá uživatelský manuál, nebude mu dodán. Podobný princip platí pro technickou dokumentaci. Protože agilní vývoj předpokládá, že členové týmu jsou zkušení a soběstační, může nastat situace, ve které si programátoři vytvoří pouze nejnutnější modely (například diagram tříd a stavový diagram). Do okamžiku napsání kódu tak není jasné, jak budou některé, zpravidla nekritické části softwaru fungovat. Po napsání kódu je fungování jasné jen jeho autorovi. Dle manifestu se autora může kdokoliv kdykoliv zeptat. Dokud je programátor v týmu dostupný, může na dotazy odpovídat. Odejde-li však z agilního týmu, odejdou spolu s ním i jeho znalosti o napsaném kódu. Z toho důvodu by měla být alespoň nezbytná část softwaru zdokumentována. O tom, jak určit, kterou část dokumentace udržovat a kterou vypustit, pojednávají kapitoly 3.4.3 Udržování dokumentace a 3.4.4 Za jakých podmínek vytvářet dokumentaci. (Ambler, 2002)
Rovněž by nemělo být zavrhováno použití specializovaných nástrojů pro tvorbu a správu dokumentace. Některé nástroje pro správu dokumentace mohou naopak mnohem více přispívat k pružnosti vývoje. Nástroje blíže popisuje kapitola 3.4.5 Nástroje pro podporu kvality dokumentace.

2.2.2 Fungující software

„Upřednostňování fungujícího softwaru před vyčerpávající dokumentací“

Toto pravidlo se přímo týká dokumentace a Ambler jej vysvětluje názorně a jednoduše. Tým by se měl zamyslet nad tím, zda preferuje mít pouze 50stránkový dokument o tom, co je potřeba vytvořit, nebo už mít hotový fungující software. Dle něj si fungující software zvolí 99 lidí ze 100. Z toho vyplývá, že by se měl tým zaměřit na postupné budování výsledného softwaru, zejména proto, aby bylo možné zákazníkovi a koncovým uživatelům přinést včas to, co potřebují. Zákazník i uživatelé jistě software lépe pochopí, když si jej budou moci vyzkoušet, než kdyby si měli pročítat rozsáhlé diagramy popisující vnitřní chování. Dle Amblera má dokumentace místo spíše tam, kde má za úkol popsat, proč vůbec vyvíjený software vznikl, jak byl vybudován a jak s ním pracovat, což je obvyklý případ uživatelské dokumentace nebo nějakého provozního manuálu. Za každou cenu však musí zůstat tvorba softwaru tím hlavním cílem vývoje. (Ambler, 2002)

Z toho důvodu je nutné udržovat dokumentaci alespoň na takové úrovni, aby posloužila svému účelu nevytvářet jí zbytečně moc. Ambler pro tento stav využívá označení Just Barely Good Enough (JBGE), což lze volně přeložit jako „tak akorát dostačující“. Vztah mezi hodnotou dokumentace a vynaloženým úsilím znázorňuje Obr. 2.



Obr. 2 Vztah mezi vynaloženým úsilím a hodnotou dokumentace (Ambler, 2005)

2.2.3 Spolupráce se zákazníkem

Při spolupráci se zákazníkem mohou kromě úvodního stručného soupisu požadavků vznikat také další dokumenty. Například když si zákazník si přeje vidět aktuální stav rozpracovaná věci, ale věc ještě není nasaditelná na testovací prostředí. Zpětná vazba od zákazníka je také určitou formou dokumentace, zákazník požadavky v průběhu vývoje upřesňuje právě na základě toho, co má již k dispozici. Agilní vývoj preferuje osobní komunikaci (ideálně by měl být zákazník v jedné místnosti s vývojáři, což nemusí být vždy dosažitelné).

2.2.4 Reakce na změny

Podobně jako při spolupráci i v případě reakcí na změny vznikají alespoň krátké poznámky, které nejsou formální dokumentací, ale slouží členům týmu k lepší orientaci v požadavcích. Změny je nutno komunikovat rychle, proto je opět preferována osobní komunikace s případnými krátkými poznámkami. Spíše se v takovém případě upravuje původní zadání, které již někdo sepsal dříve.

2.3 Principy stojící za Agilním Manifestem

Pro lepší pochopení smyslu agilního vývoje softwaru dále autoři Agile Alliance uvádějí 12 principů. I přes to, že uvedené principy přímo nezmiňují dopady na dokumentaci softwaru, lze z jejich povahy vztah k dokumentaci nalézt. Těmito principy jsou:

2.3.1 Včasné a průběžné dodávání

„Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.“

Tento princip říká, že problémem spousty lidí v IT je jejich mylná představa o tom, že vše musí být už definováno ještě před samotným začátkem budování softwaru a že postupné rozšiřování a nabalování funkcionalit funguje mnohem lépe. (Ambler, 2002)

Včasnost ale může zapříčinit zanedbání dokumentace. I když předpokládáme-li tvorbu „tak akorát dostačující“ dokumentace, která by neměla agilní tým nijak zatěžovat, nelze vyloučit situaci, kdy zkrátka není dostatek času nebo lidských zdrojů na tvorbu hodnotné dokumentace. Příkladem může být tvorba testovacích scénářů. Co když zákazník nestojí o nějaké tabulky s testovacími případy a popisem jednotlivých kroků? Zákazník si zaplatil za výsledek a ten chce také dostat. Zákazníka zajímá to, že bude moci co nejdříve objednaný software využívat, neboť jej pravděpodobně potřebuje ke svému podnikání. Protože dal někomu nemalé peníze, očekává naproti tomu kvalitní a bezchybnou aplikaci. I když je zákazník ten, kdo řešení akceptuje a testuje dílčí funkcionality, nelze automaticky předpokládat, že má zájem o nějaké výstupní dokumenty týkající se testování. To je problém pro testery i programátory, protože při dlouhodobém rozvoji vyvíjené aplikace je nutné znát, jak mají všechny existující součásti fungovat.

2.3.2 Vítání změn v požadavcích

„Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.“

Autoři říkají, že požadavky se budou při vývoji měnit vždy. V tradičním vývoji softwaru je prý řízení změn spíše nástrojem, jak změnám zamezovat, kdyžto agilní vývojáři změny vítají. (Ambler, 2002)

Otázkou je, zda vítají změnu i ti, kdo jsou odpovědní za udržování dokumentace. Je-li v jeden okamžik software dostatečně dobře zdokumentován, neznamená to z hlediska teoreticky nekonečného agilního vývoje vůbec nic. Za několik sprintů se spousta věcí přidá, některé ubydou a značná část se změní. S přibývajícími součástmi roste i objem dokumentace, kterou je však nutno průběžně aktualizovat.

2.3.3 Pravidelně dodávaný fungující software

„Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.“

Autoři prosazují kratší intervaly, neboť to napomáhá vidět napříč celým projektem a tvoří to zpětnou vazbu pro všechny zainteresované strany, které mohou vývojový tým co nejdříve nasměrovat požadovaným směrem. (Ambler, 2002)

Vyžaduje-li zákazník nebo samotná povaha vyvíjeného softwaru tvorbu dokumentace ve větším rozsahu, např. zákazník požaduje kvalitní uživatelskou dokumentaci (např. včetně instruktážních videí) nebo jsou kladeny vysoké bezpečnostní nároky na výsledný software. Proto nemusí být z hlediska tvorby a údržby dokumentace krátké intervaly příliš vhodné. Cílem totiž přestává být dodání pouhého fungujícího softwaru.

2.3.4 Spolupráce lidí z byznysu a vývoje

„Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.“

Agilní vývoj zde preferuje aktivní spolupráci všech zainteresovaných, ať už se jedná o zákazníka, který by měl být mimo jiné denně k dispozici, ideálně by měl být ve stejné místnosti, nebo o ostatní zainteresované strany, mezi které patří i koncoví uživatelé či manažeři. (Ambler, 2002)

Z pohledu dokumentace lze zmínit, že stejně jako u dodávaného softwaru, může zákazník na dokumentaci nahlížet a vznášet případné dotazy. Opět zejména v případě uživatelské dokumentace.

2.3.5 Důraz na samostatnost a motivaci

„Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.“

Spousta organizací má dle autorů vizi, že zaměstnají velké množství relativně nezkušených lidí, dají jim k dispozici popisy procesů dle CMMI a ISO norem a oni budou s úspěchem vyvíjet software. To však v praxi nefunguje. Do agilních týmů je důležité získat takové členy, kteří spolu budou ochotni spolupracovat, učit se od sebe navzájem a budou si uvědomovat, že právě lidé jsou tím hlavním faktorem úspěšnosti softwaru. (Ambler, 2002)

V některých případech mohou být normy a metodiky užitečné, ba dokonce nezbytné pro vývoj, a to zvláště v případě, kdy agilní tým působí v regulovaném prostředí, kde jsou kladeny vyšší nároky na množství a kvalitu dokumentace. Agilní vývoj v regulovaných prostředích je popsán v kapitole 3.5 Agilní vývoj v regulovaných prostředích a vládním sektoru.

2.3.6 Důraz na osobní komunikaci

„Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.“

Aby mohl být vývojový tým úspěšný, musí jeho členové umět efektivně komunikovat a spolupracovat. Možností, jak komunikovat je mnoho. Autoři uvádějí, že nejefektivnějším způsobem je komunikace „z očí do očí“ spojená s tužkou a papírem či fixou a tabulí. (Ambler, 2002)

Problémem osobní komunikace je její dočasnost, to částečně řeší krátké poznámky a diagramy. Zápisky na tabuli jsou užitečné pouze krátkodobě a obvykle jsou vytvářeny za účelem lepší představy a doplňují ústně vysvětlovanou problematiku; nejsou tedy samy o sobě dostatečné. Potřebuje-li člen týmu k zápiskům přistoupit později, v řádu několika let, nemusí se v nich orientovat.

2.3.7 Důraz na fungující software

„Hlavním měřítkem pokroku je fungující software.“

Hlavním měřítkem by dle autorů měla být dodávka fungujícího softwaru, který splňuje požadavky všech zainteresovaných stran. Měřítkem by naopak neměla být získaná hodnota měřená na základě toho, kolik bylo předáno dokumentace. (Ambler, 2002)

Ač je pravdou, že cílem je dodat software a ne dokumentaci, je potřeba myslet na budoucí rozvoj a existující součásti systému dostatečně dobře dokumentovat. Jak autoři uvádějí, je klíčové, aby úspěch nebyl primárně odvozován od množství dokumentace, nýbrž od její kvality.

2.3.8 Dlouhodobě udržitelné tempo

„Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.“

Zde přirovnávají autoři vývoj k maratonu. Stejně jako v maratonu nemůže běžec sprintovat po celou dobu, nemohou ani vývojáři pracovat nad rámec běžné pracovní doby po domu několika měsíců. Z Amblerových zkušeností je člověk schopen vykonávat intelektuálně náročnou práci maximálně 5 až 6 hodin denně. Intelektuálně náročnou prací zde označuje skutečný čas strávený vývojem. Říká také, že pracuje-li člověk například 12 hodin denně na plný výkon po delší dobu, je brzy vyčerpán a dosahuje poté pouze průměrných výsledků. (Ambler, 2002)

Dokumentování v agilním vývoji probíhá zpravidla co nejpozději, kdy bylo zamezeno zbytečnému přepisování dokumentace. To může vyvolat potřebu vytvořit dokumentaci o větším objemu za relativně krátkou dobu. Protože se ale nejedná o každodenní činnost, není nárazová tvorba dokumentace na konci iterace v tomto ohledu v rozporu s tímto principem.

2.3.9 Pozornost věnovaná technické výjimečnosti a dobrému designu

„Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.“

Pracovat s vysoce kvalitním kódem je mnohem snazší než pracovat s kódem nekvalitním, proto se v agilním vývoji musí začínat u kvality kódu. Tu je možné podpořit refaktoringem a vývojem řízeným testy. Díky automatizovaným testům je možné chyby ihned odhalit, což ve výsledku ušetří čas i peníze. (Ambler, 2002)

Automatizované testy mohou sloužit, stejně jako sady běžných testovacích scénářů, jako vhodná alternativa detailních případů užití a dalších dokumentů specifikujících interakce mezi uživatelem a aplikací a mezi dvěma aplikacemi nebo komponentami vyvíjeného systému.

Kvalitní a okomentovaný kód nevyžaduje další popis, stejně jako dobře navržený systém; odpadá tak nutnost časté komunikace mezi programátory, což vede k vyšší efektivitě práce.

2.3.10 Jednoduchost

„Jednoduchost – umění maximalizovat množství nevykonané práce – je klíčová.“

Agilní vývojáři se dle Amblera mají zaměřovat jen na činnosti s vysokou hodnotou, které přinášení návratnost investic do IT. To znamená, že by měli automatizovat vše, co automatizovat lze. (Ambler, 2002)

Stejně jako v předchozím případě, i zde je zmíněna automatizace. Kromě testování lze automatizovat i sestavení výsledných dokumentů pomocí vhodných nástrojů.

2.3.11 Samo-organizující se týmy

„Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.“

Ambler tento bod považuje za jeden z nejradikálnější principů agilního vývoje. Princip se týká využití Testy řízeného návrhu (TDD – Test Driven Design) a Agilního modelově řízeného vývoje (AMDD – Agile Model Driven Development). Tyto dvě metody jím považovány za hlavní přístupy k zajištění vzniku efektivních architektur, požadavků a návrhů v rámci agilní komunity. (Ambler, 2002)

Zmíněné metody opět prosazují myšlenku, že bychom neměli vytvářet více dokumentace než je opravdu nezbytné.

2.3.12 Zpětná vazba a zlepšování sebe sama

„Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.“

Ambler zde zdůrazňuje výhodu EUP (Enterprise Unified Process), kterou je disciplína SPI (Software Process Improvement), která cíli na zlepšování procesů organizace. Pro agilní vývoj pak uvádí užití DAD rámce (Disciplined Agile Delivery, nově označováno jako Disciplined Agile 2.0) pro podporu v rozhodování. (Ambler, 2002)

Zpětnou vazbu lze shledat i při tvorbě a správě dokumentace. Nevidí-li tým smysl v udržování určitého dokumentu či celé skupiny dokumentů, nemá smysl v jejich údržbě pokračovat. Naopak, cítí-li některý z členů týmu potřebu mít nějaké informace pohromadě a s vysvětlením,

měl by tým zvážit, zda by nebylo vhodné potřebný dokument vytvořit. Podobně lze s postupem času nalézt optimální využití nástrojů a potřebné množství vytvářené dokumentace.

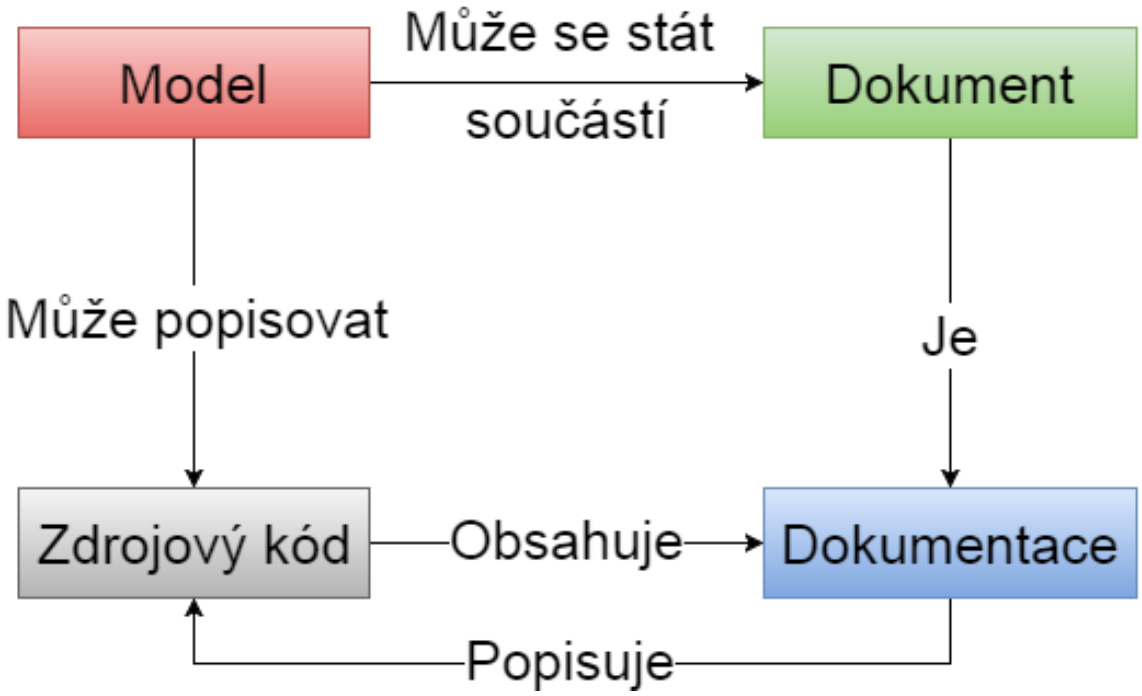
3 Agilní dokumentace

3.1 Význam dokumentace při vývoji softwaru

Začněme s popisem vztahů mezi modely, dokumenty, zdrojovým kódem a samotnou dokumentací. Dokument je jakýkoliv artefakt vně zdrojového kódu, který trvale uchovává informace, čímž se liší od modelu, který je abstrakcí, která má za úkol popsat jeden nebo více aspektů problému a jeho potencionálního řešení.

Některé modely se mohou stát dokumenty, nebo alespoň součástí některých dokumentů, ale mnohé z nich jsou jednoduše zahozeny ve chvíli, kdy naplnili svůj účel a dále nejsou potřeba.

Zdrojový kód je pak souborem instrukcí zahrnujícím i komentáře k těmto instrukcím. Přestože je zdrojový kód pouhou, byť podrobnou abstrakcí, nejedná se o model, už z důvodu potřeby tyto dva prvky od sebe odlišit. Pro další účely je také dobré si připomenout, že dokumentace zahrnuje jak samotné dokumenty, tak i komentáře přímo v kódu.



3.2 Problémy spojené s dokumentací

Hlavním problémem dokumentace jsou její autoři, samotní vývojáři. Přesto, že smyslem dokumentace je pomáhat ostatním, nejen vývojářům, sami její autoři v ní nevidí žádnou hodnotu. A to i přesto, že psaní dokumentace může vést k uspořádání autorových myšlenek a možnému

lepšímu porozumění problematice. Tento negativní pohled na dokumentaci je často doplněn citací Agilního manifestu: „working software over comprehensive documentation“. Avšak tyto dvě entity se navzájem nevylučují.

V ostatních disciplínách se považuje dokumentace za eticky nutnou, možná někdy ne příliš příjemnou avšak je vnímána jako nezbytnost pro vyšší dobro. Je nezbytně nutná pro každého zvenčí, kdo se chce se systémem seznámit a pochopit jeho logiku a pravidla. Dokumentace však může sloužit nejen samotným vývojářům. I uživatelé, testeři, projektoví manažeři či dokonce obchodníci mohou potřebovat informace, jaký je princip jejich softwaru. Další variantou vedle dokumentace je zdlouhavé a vpravdě zbytečné zkoumání kódu často metodou pokus omyl. Navíc takto získané informace jsou nekompletní, nekonzistentní a hlavně neefektivně nalezené. Pokud vezmeme v potaz, že moderní software rychle nabývá na komplexitě, jsou často revidovány a úklid kódu je přenechávána zaměstnancům na junior pozicích, kteří s ním nemusí být plně seznámeni, je nutnost dokumentace nevyvratitelná.

Problematická je údržba dokumentace kvůli dynamickému vývoji software. Náklady s tímto spojené se mohou zdát příliš vysoké, protože by tento lidský kapitál mohl vytvářet nový kód. Někteří dokonce argumentují, že kód je sám o sobě dokumentací, avšak každý, kdo přišel do styku se stavem, jaký panuje ve většině firem, ví, že existují kusy starého kódu, který se všichni bojí pozměnit, protože ne přesně chápou jeho funkčnost. Problémem je propast mezi sémantikou moderních programovacích jazyků, které řeší daný problém a jeho reprezentaci v reálném světě. Navíc je zde fakt, že struktura programovacího jazyka nebyla vytvořena pro čtení, ale například s ohledem na kompilaci, takže čtenář musí pro porozumění kódu replikovat funkcionalitu kompilátoru. Argument Agilního manifestu, tak platí pouze v případě složitých specifikací, které nepřežijí první kontakt s realitou, protože ne všechny problémy lze předjímat během designové fáze.

Podstatou dokumentace však není sáhodlouhý dokument obsahující komentáře na úrovni kódu, nýbrž popis vyšší úrovně abstrahující od zbytečných technicií, ale obsahující princip designu architektury včetně zavržených alternativ. Takový dokument nejen pomáhá získat rychlý přehled nad funkční podstatou skrývající se ve stohu kódu, ale zároveň pomáhá zajistit soudržnost při rozhodování o hrubších zásazích. Mluvíme tedy o agilní dokumentaci designu, která dokáže držet krok s vývojem.

Jeden z nadějných přístupů je založen na používání modelovacích jazyků vyšší úrovně, který umožňuje rychlý vývoj architektury. Ta je pak zhodnocena klasickými agilními metodami. Tím se elegantně vyhneme vysokým režijním nákladům programování komplexních struktur klasickými programovacími jazyky. Je tedy důležité neupouštět od praktik, které prokazatelně fungují a

pomáhají, nýbrž je naší povinností je přizpůsobit současným procesům. Podle Amblera lze shrnout problémy s dokumentací do 9 bodů:

1. **Vývoj SW versus vývoj dokumentace.** Projekt stále balancuje mezi dvěma extrémy. Buďto je projekt pouze se softwarem nebo existuje jen dokumentace. Primárním cílem je vývoj software, tj. tvorba hodnoty pro své zákazníky, což jde ale přímo proti sekundárnímu cíli a to psaní dokumentace.

2. **Spustitelná specifikace je mnohem cennější nežli statická dokumentace.** Soustava testů, které specifikují většinu požadavků je velmi cenná, protože nejenže specifikuje klientovy požadavky, ale zároveň je i testuje jejich implementaci.

3. **Vývojáři mají znalosti, techničtí spisovatelé řemeslo.** Dokumentace psaná vývojářem nikdy nedosáhne kvality technických spisovatelů, proto je vhodné napsat pouze základní dokumentaci a tu poté předat spisovateli.

4. **Co je požadováno během vývoje je mnohdy odlišné od toho, co je požadováno po něm.** Během vývoje jsou důležité prototypy, testovací sestavení atd., které jsou zachyceny v dokumentaci. Při vydání finální verze je potřeba tuto dokumentaci aktualizovat podle současného stavu kódu.

5. **Dokumentovat souběžně s prací nebo až po skončení?** Jednou možností je psát dokumentaci souběžně s kódem, kdy autor nejlépe zachytí jeho princip, avšak při refaktorizaci je ji pokaždé nutno aktualizovat. Druhou možností je psát ji až po dokončení projektu, ale je zde nebezpečí opomenutí či zapomenutí fungování kódu nebo nedostupnost jeho autorů.

6. **Počkat až se informace vytříbí?** Pokud je projekt ve stavu kdy se stále mění jeho specifikace, je lepší počkat s dokumentací na dobu, kdy se tyto informace stabilizují, jinak hrozí riziko jejího rozsáhlého přepisování.

7. **Dokumentace kódu versus dokumentace kódem.** Pokud jsou cílovou skupinou dokumentace vývojáři, pak pro ně bude nejpřijatelnější dokumentace kódem, avšak nejen tato skupina lidí těží z dokumentace. Uživatelé, vysoký management a pracovníci operací také potřebují využívat dokumentaci a to takovou, která je přístupná a pro tuto skupinu lidí srozumitelná.

8. **Projektová versus celofiremní dokumentace.** Ne všechna dokumentace, která je v projektu napsána se týká pouze interních záležitostí. Někdy je potřeba vytvořit

dokumentaci pro část nebo i všechny ve firmě, kdy je nutné využít již existující artefakty, zvyklosti a styl s jakým je psaná celofiremní dokumentace.

9. **Kvantita versus kvalita.** Dalekosáhlá dokumentace pravděpodobně bude obsahovat velké množství chyb, i když zachycuje projekt do detailu. Kratší dokumentace vyšší úrovně může poskytnout přehled projektu s méně chybami.

3.3 Kdy je dokumentace skutečně agilní

3.3.1 Pokud maximalizuje ROI

Pokud maximalizuje návratnost investic do ní (a potažmo I celého projektu) vložených. Mělo by tedy platit, že užitek plynoucí z agilní dokumentace by měl být vyšší než náklady na její pořízení a údržbu a v nejoptimálnějším případě bylo investování do ní první nejlepší možností pro využití užitých zdrojů. Jinými slovy, dokument musí poskytnout co nejvyšší přidanou hodnotu. Pokud je například návratnost zhotovení dokumentace 50% a návratnost nového školení programátorů 80%, je lepší zdroje využít k proškolení programátorů.

3.3.2 Zúčastněné strany jsou si vědomi TCO

Stakeholdeří znají a chápou hodnotu nákladů na vlastnictví dokumentace a byli ochotni do jejího vytváření a udržování investovat.

3.3.3 Agilní dokumentace je štíhlá, ale dostatečná

Agilní dokumentace by měla obsahovat právě tolik informací, aby plnila svůj účel, je tedy zjednodušena na nejvyšší únosnou mez. Část dokumentace může být kupříkladu psána pouze v bodech, namísto souvislého textu, ale i tak zachycovat dostatečnou míru potřebných informací, aniž by autor musel investovat čas nad vizuálním pojetím dokumentace. Obsah je v tomto případě důležitější než samotná forma reprezentace informací. U agilní dokumentace se navíc často stává, že jedena část dokumentace odkazuje na druhou, namísto redundantního zápisu. Při psaní agilního dokumentu je tady třeba vždy pamatovat na co nejvyšší jednoduchost.

3.3.4 Agilní dokumentace plní svůj účel

Agilní dokumentace by měla splňovat pravidlo soudržnosti. Měla by tedy plnit přesně definovaný účel. Pokud není jisté, za jakým účelem je dokumentace vytvářena, nebo pokud je smysl jejího vytváření rozporuplný, měla by se práce na tomto dokumentu pozastavit a znovu promyslet, zdali má opravdu smysl tento dokument vytvářet.

3.3.5 Agilní dokumentace popisuje „Co je dobré vědět“

V agilní dokumentaci jsou zachyceny důležité informace. Informace, které nejsou patrné na první pohled, například zdůvodnění použitého designu, použité procedury atd. V agilní dokumentaci naopak nejsou zachyceny informace, které jsou na první pohled každému jasné. Pokud například část kódu využívá pole F_NAME tabulky ZÁKAZNÍK, není třeba zdlouhavě popisovat, že právě v tomto poli je uloženo jméno zákazníka (First Name), naopak informace, proč je část databáze vedena na jiném úložišti, už je navýsost vhodná.

3.3.6 Agilní dokumentace má vždy svou cílovou skupinu

Každý agilní dokument by měl být určený konkrétnímu zákazníkovi, či skupině zákazníků, které má přinášet hodnotu a usnadňovat další práci. Například systémová dokumentace je obvykle pro maintenance developery, kterým poskytuje přehled nad architekturou celého systému, klíčovými požadavky na systém a použitými návrhovými vzory. Oproti tomu uživatelská dokumentace je psána srozumitelnější (méně technickou) formou tak, aby běžnému uživateli poskytla informace, jak se systémem pracovat. Každý „zákazník“ vyžaduje jiný přístup i formu dokumentace a je nutné na něj brát v těchto směrech ohledy. Agilní dokumentace by se vždy měla svým účelem setkat s potřebami daného zákazníka. Dobrým způsobem, jak toto zajistit, je jeho zapojení. Jeho zpětná vazba je rychlým a efektivním způsobem validace napsané dokumentace. Zákazník si je většinou moc dobře vědom, co mu usnadní další práci a co je pro nějak čitelné a co naopak není k užitku.

3.3.7 Agilní dokumentace je především dostatečná

Jako dobrý příklad by mohla sloužit tato modelová situace. Určitě se mnohým stalo, že se učili se softwarovým nástrojem z příručky, která nebyla pro aktuální verzi SW, ale pro verzi starší. Podle této dokumentace je rozhodně možné se s daným nástrojem naučit. Není to nejideálnější, nepokrývá to sto procent nové funkcionality, ale přesto je zákazník schopný se z dokumentu dozvědět téměř vše podstatné. Určitě si v této situaci umíte představit, zda byste byli ochotni zaplatit například 400 Kč za novou příručku pro aktuální verzi, když už vlastníte jejího předchůdce. Agilní dokumentace nemusí být dokonalá, musí být zkrátka dostatečná.

3.4 Kvalita dokumentace v agilním vývoji

3.4.1 Typy dokumentů

Obecně je vžit spíše názor, že dokumentace v agilním přístupu nedává smysl. Nejčastějším názorem je, že v agilním přístupu k vývoji software je dostatečnou dokumentací samotný kód. Přes všechna tato fakta byly odborníky a některými zastánci dokumentace stanoveny rámce, které vymezují, jaké dokumenty je možné v agilních přístupech použít nebo vytvářet. Dokumenty lze definovat následovně (Ambler, 2012):

Modely smluv - dokument je tvořen pro ostatní týmy, které jsou součástí společnosti, a popisuje technické rozhraní k již existujícím systémům ve společnosti, které jsou aktuálně využívané.

Návrhová rozhodnutí - cílová skupina tohoto dokumentu jsou projektanti, servisní vývojáři a projektoví manažeři. Dokument obsahuje souhrn kritických rozhodnutí týkajících se designu a architektury, které by mohl vývojový tým udělat během samotného vývoje. Souhrn musí být dostatečně specifický a měly by zde být uvedeny návrhy pro odstranění těchto kritických bodů.

Ustanovení vizí a řízení - určení dokumentu je pro vrcholový management, správu uživatelů a vedení celého projektu. Definují se zde především vize, které by měl nově vyvíjený systém naplňovat. Současně by měl obsahovat aktuální odhad nákladů, přínosů, rizik nebo personálního zabezpečení. Důležitou součástí dokumentu je rovněž nastavení milníků, ke kterým musí projekt bezpodmínečně mířit. Může obsahovat i seznamy zainteresovaných skupin.

Operační dokumentace - dokument je především určen operativním zaměstnancům. Jeho součástí jsou indikátory, které představují různá propojení s další infrastrukturou společnosti. Především pak interakce s databázemi, soubory nebo typ zálohování. Souhrn požadavků na dostupnost, spolehlivost a případném možném zatížení nového software. Poslední částí dokumentu jsou pokyny k řešení problému a výjimečných situací, které by mohly v rámci nového systému nastat.

Přehled projektu - cílem dokumentu je informovat vývojáře, manažery, servisní vývojáře i provozní personál o důležitých a úplných informacích o projektu. Součástí dokumentu je soupis uživatelských kontaktů, technologií a nástrojů používaných k sestavení a vývoji nového systému. Rovněž poskytuje soupis kritických projektových artefaktů, jako je například zdrojový kód (především je důležitá podpora dostatečné dokumentace kódu samotného, aby byl později snáze editovatelný). Stanovuje, jaké dokumenty se k projektu vztahují a jakou plní roli v dokumentaci.

Dokumentace požadavků - dokument je určen především pro projektanty, servisní vývojáře, uživatele a uživatelský management. Dokument definuje, co přesně systém dělá a shrnuje artefakty. Definuje obchodní pravidla, use-cases, user stories nebo prototypy uživatelského rozhraní. Částí, které by tento dokument mohl obsahovat je mnoho a tak jsou zde jmenované jen ty nejdůležitější a nejčastější.

Servisní dokumentace - je určena především servisnímu personálu. Dokument obsahuje školící materiály se specifickým oddělením pro zaměstnance a uživatele. Je používán jako průvodce při odstraňování chyb. Jsou zde popsány základní postupy jak řešit chyby, problémy a další výjimečné situace. Současně dokument obsahuje kontakty na osoby, které dokumentaci tvořili a jsou zodpovědně za servis aplikace.

Systémová dokumentace - cílem této dokumentace je poskytnout vývojářům a servisním vývojářům ucelený přehled o celém systému. Hlavními částmi dokumentu jsou definice technické a obchodní architektury. Dále jsou zde definovány ostatní architektury systému, design databázových modelů včetně odkazů na místa kde jsou umístěny. Dokumentace napomáhá vyhnutí se odchylkám od původně navrženého řešení architektur a struktur.

Uživatelská dokumentace - je určena především pro samotné uživatele vytvořeného systému a poté pro uživatelský management. Uživatelská dokumentace může vystupovat jako referenční příručka, průvodce, podpora nebo jako výukový materiál, pomocí něhož mohou uživatelé získat potřebné znalosti. Výše jmenované jsou jednotlivé různé příručky, které na sebe navazují a jsou různě dlouhé s různou úrovní hloubky řešených problémů

Jak je možné vidět, tak dokumentů, které je možné a doporučené při agilním vývoji sepisovat je celá řada. Dostáváme se tedy k otázce, zda nemizí základní výhoda agilního vývoje, kterou je absence papírování a dokumentování. Odpověď zní ne! Především proto, že žádný vývojový tým není povinen sepisovat žádný z předchozích dokumentů, aby byly agilní metodiky funkční. Jedná se jen o různé druhy pomoci a pomyslných berliček, které mají zajistit maximální zaměření na výsledek a neodchýlení se od požadovaného výsledku.

3.4.2 Vliv dokumentace na úspěšnost projektu

Vliv dokumentace na úspěšnost projektu je velmi kontroverzním tématem. Odborníci se v tomto nemůžou shodnout především z důvodu, že neexistují žádná exaktní data, která by podporovala hypotézu ovlivnění psaní komplexní dokumentace na úspěšnost celého projektu. Nelze ani předpokládat, že čím více dokumentace ať už v jakékoliv kvalitě by mělo jakýkoliv vliv na úspěšnost agilního projektu. Podle Amblera (Ambler, 2012) i když budou psány dokonalé

dokumentace s perfektní připraveností, stejně nebude zajištěn úspěch projektu. Projekt tak může skončit stejně špatně jako ten, kde dokumentace chybí úplně.

Většina dokumentace je psána pro vedení ať už projektu nebo produktu (Stakeholders). Ti si jí nechávají převážně sepisovat z důvodu udržitelnosti, plánovatelnosti, použitelnosti nebo z ekonomických důvodů, které ovlivňují celý projektové řízení. Dokumentace jim napomáhá zjistit, zda jsou jejich investice do projektu návratné (ROI).

Z toho důvodu je dobré objasnit, proč si lidé v době, kdy se všechny projekty, společnosti i korporátní společnosti zbavují zbytečné dokumentace a byrokracie stále myslí, že potřeba dokumentace je tak důležitá. Dle Amblera (Ambler, 2012) je tato až posedlost dokumentováním a sepisováním všech "důležitých" informací do samostatných a mnohdy zbytečně složitých dokumentů pozůstatkem z osmdesátých až devadesátých let, kdy byl obrovský tlak na tvorbu takové dokumentace. V tuto dobu přecházely společnosti od mírně nekontrolovaného stavu "programuj a fixuj" do stavu maximální potřeby dokumentace při vodopádovém vývoji výsledného produktu. To patrně zůstalo ve společnosti silně zakořeněno dodnes.

Bohužel tyto stereotypy je nutné ve společnosti pomalými kroky a zaváděním nových a inovovaných metodologií odstranit jako nemoc. Ne všechny metodologie a standarty jsou totiž obecně prospěšné. Například metodika CMMI je velmi nešťastnou, jelikož nutí jejich uživatele k zbytečně obsáhlé dokumentaci, která ale nemá vůbec žádný vliv na výsledný produkt. Ambler rovněž pokládá otázky, zda by nebylo lepší přemýšlet nad tím, jak zefektivnit samotný vývoj, kde jako příklad zlepšení uvádí testy řízený vývoj, refaktoring kódu nebo řízení se agilním modelem vývoje než psát obsáhlé dokumentace. Samotný kód a produkt by měl být dostatečnou dokumentací, a pokud tak není, nejedná se o dobrý kód. Je nutné ho refaktorovat, nebo jinak upravit, aby dokumentace nebyla potřebná a mohlo se přistoupit k dalšímu rozvoji.

3.4.3 Udržování dokumentace

Pohledy na důležitost a frekvenci udržování agilní dokumentace jsou v podstatě věci dva. Ten první tvrdí, že potřeba úpravy dokumentace nastává jen tehdy, pokud je to pro projekt nezbytně nutné. Druhý zastává názor, že úprava dokumentace by měla být kontinuální a naprosto přirozená součást agilního vývoje, která by neměla být nijak zvláštním nebo bolestivým úkonem.

Zastáncem teorie o změně dokumentace až v momentě bolestivosti problému je i Ambler (Amber, 2012). Ten tvrdí, že starší dokumenty u vývoje software příliš nevadí. Ukazuje to na příkladu využití JDK (Java Develop Kit), kde lze využít dokumentaci verze 1.2.x k verzi 1.3.x. Není to sice ideální varianta, ale není to natolik bolestivé, aby se musela kompletně měnit, nebo jakkoliv celá přepisovat. Zároveň ale uvádí, že použití staré dokumentace je částečně

kontraproduktivní a ztrátové na poli produktivity. Z tohoto názoru lze odvodit, že neaktualizovaná dokumentace není na škodu, ale pokud dokumentaci aktualizujeme pravidelně, bude produktivita a správné využití vyvíjeného software mnohonásobně vyšší. Dále Ambler uvádí, v jakých případech doporučuje k změně nebo úpravě dokumentace přistoupit (Ambler, 2012):

1. při výrazné změně smluv, služby o proti předchozím modelům, které byly naplánovány,
2. uživatelskou dokumentaci je potřeba aktualizovat vždy při změně systému, aby jeho využívání bylo správné a úplné,
3. pokud to skutečně bolí, například kdyby měl být zákazník nebo uživatel produktu poškozen při neaktualizaci dokumentace.

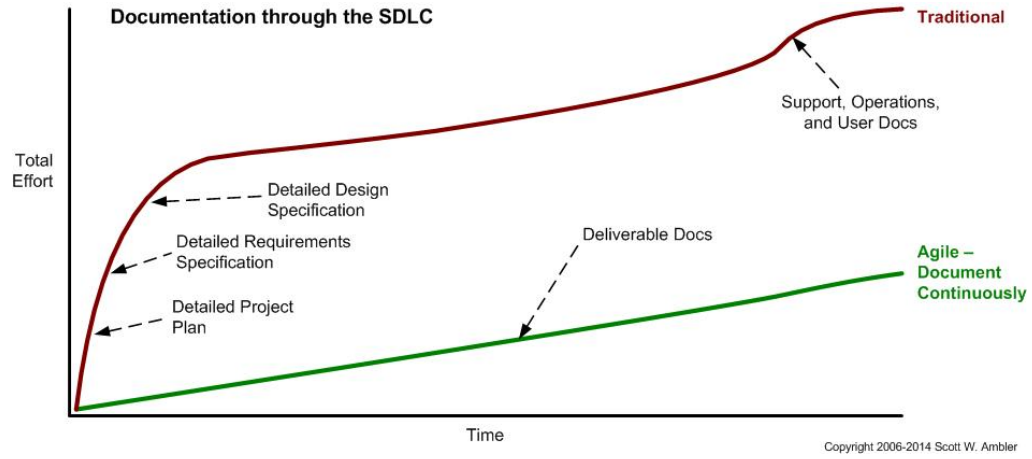
Z předchozích třech bodů můžeme odvodit následující fakta. Každý typ dokumentace má jinou dobu "trvanlivosti". Například uživatelská dokumentace bude vyžadovat častější údržbu než návrhová dokumentace. Ta se bude měnit, jen pokud to bude nezbytně nutné. Určení doby kdy už je dokumentace neaktuální a je potřeba jí opravit může být velmi frustrující a matoucí. Je dobré mít jasně stanovená pravidla určující rozpoznání neaktuální dokumentace. Pokud budou tato pravidla dobře stanovena, odpadá frustrace z údržby a chuť k úpravám se prohloubí. Účastníci totiž budou přesně vědět, kdy už je nezbytné dokumentaci upravit (neboli kdy už to opravdu bolí tzv.: pain neck).

3.4.4 Kdy je vhodné tvořit dokumentaci

Nejprve je vhodné definovat některá nezbytná pravidla pro tvorbu dokumentace, která předcházejí samotné otázce, kdy tvořit dokumentaci. Doporučení jsou následovná (Ambler, 2012):

1. udržení jednoho informačního zdroje,
2. tvořit dokumentaci průběžně a ne v poslední fázi projektu,
3. zdůrazňovat použitelnou specifikaci před statickou,
4. dokumentovat finální koncepty nikoliv neúplné nebo spekulativní.

Na základě předchozích doporučení je možné vidět, že pokud je dokumentace vytvářena po celou dobu projektu, je vytvářena správně. Dokumentace tím získá kontinuálního charakteru, který zabezpečí kompletní a důkladnou dokumentaci. Na následujícím obrázku je vidět, jak a kdy by měly být tvořeny různé dokumenty v optimálním případě, kdy je dokumentace tvořena včas.



Obr. 3. - Dokumentation trough the SDLC (Ambler 2012)

Autor obrázku předpokládá, že je možné na konci každé iterace dodat nový funkční produkt, který pokud je dokumentace tvořena dodána současně. Velkou nevýhodou průběžné tvorby dokumentace může být nesoulad a časté úpravy dokumentace v případě změny předchozího produktu, v důsledku oprav nebo jiných výjimečných situací. Nabízí se možnost prodloužit iterace tak aby dokumentace vznikala rovnou kompletní. Ani to však není nejlepším řešením a tak musí vzniknout kompromis mezi délkou iterací a časem tvorby dokumentace samotné.

3.4.5 Nástroje pro podporu kvality dokumentace

Nástrojů pro podporu kvality dokumentace je nepřeberné množství. Z toho důvodů jsme hledali nějaký relevantní průzkum mezi projekty a společnostmi, které se věnují vývoji. Tomuto výzkumu se věnovali Andrew Forward a Timothy C. Lethbridge z University of Ottawa, který sumarizovali v článku The Relevance of Software Documentation, Tools and Technologies: A Survey (Forward, Leithbride, 2002). Z výzkumu vyšly jako nejčastěji užívané nástroje pro tvorbu a podporu kvality dokumentace nástroje uvedené v následující tabulce.

Tabulka 1.: procento užití nástrojů při dokumentaci (Forward, Leithbride, 2002)

Nástroj	Procento užití
MS Word (ostatní textové editory)	36.5 %
Javadoc, Doxygen, Doc++	35 %
Textové editory	15 %
Rational Rose	8.5 %
Control Center, IDE	5%

Na základě těchto výsledků lze rozdělit nástroje do dvou oddělených skupin:

1. Textové editory, kde je dokumentace vytvářena automaticky
2. Automatické nástroje, které dokumentaci dokážou generovat například ze správného dokumentování kódu

Všechny výše jmenované nástroje podporují kvalitu dokumentace i jejího samotného vytváření. Pokud půjdeme do důsledků, je téměř jedno, za pomoci jakého nástroje je dokumentace vytvářena. Důležité je, zda vůbec a jak kvalitní je dokumentový výstup.

3.5 Agilní vývoj v regulovaných prostředích a vládním sektoru

Vlády se snaží zavádět agilní praktiky do vývoje svého softwaru. Jelikož je však prostředí veřejné správy silně regulováno, je agilnímu přístupu kladeno mnoho překážek pod nohy. Jelikož agilní přístup upřednostňuje fungující software před úplností dokumentace, může jeho využívání být viděno negativně, hlavně z pozice strážců demokracie. Například nespočet neziskových organizacím zabývajícím se veřejným prostorem, zákonodárcům, novinářům či čistě jen široké veřejnosti chybějící či neúplná dokumentace snižuje jejich možnosti kontroly a zvyšuje netransparentnost fungování státu. Stát je přitom vázán zástupem nařízení, které se snaží tomuto zabránit, a vyžadují po tvůrcích určitou úroveň zdokumentování kódu. Tuto činnost zastává technický zapisovatel, jehož hlavní výzvou je, aby dokumentace byla dostatečně komplexní a zároveň stručná aby byl tým schopen pružného fungování.

Z dokumentace by mělo být jasné, zdali projekt splňuje požadavky týkající se například bezpečnosti a dalších pravidel a standardů. Toto je pak prověřováno audity. Častým problémem je i nutnost vytváření ad-hoc dokumentace.

Velkým problémem agilních metod je absence pravidel bezpečnosti. V době kdy roste agilně vyvíjených projektů webových aplikací je nutnost zahrnout do procesu metody bezpečnostního inženýrství. Zvýšit bezpečnost je také možno párovým programováním, které však zvyšuje náklady na vývoj a ve veřejné správě na něj může být nahlíženo jako na plýtvání prostředky, zvláště v dobách snižování počtu státem zaměstnaných.

Z důvodu nutnosti úplné dokumentace by o ní měly vývojáři přemýšlet stejně jako o fungujícím softwaru. V tomto odvětví je to jeho neoddělitelná součást. Best practices think-tanku ZapThink zahrnují tento přístup:

1. Týmy by si měly ujasnit, jaké jsou požadavky na dokumentaci, zapsat je jako user stories a přidat je do backlogu.

2. Větší updaty dokumentace by měly být v epicu obsahujícím více user stories, kterým se budou věnovat jednotlivé sprinty.
3. Jakmile jsou epic a user stories vytvořeny, tým může upravit stories tak, aby identifikoval priority story pro zákazníka, definoval akceptační kritéria a odhadl potřebnou práci, která by neměla přesahovat délku sprintu.
4. Jakmile story obsahuje kompletní informace, tým může přistoupit k typickým procesům výběru story z backlogu pro vytvoření backlogu sprintu pro nadcházející sprint meeting.
5. Jakmile je story v backlogu sprintu, může dedikovaný programátor začít pracovat na story. To musí být připraveno na zhodnocení na konci sprintu.

Vývoj v regulovaných prostředích není sice k agilním metodám přívětivý, ale s výše zmíněnými přístupy je jejich aplikace možná.

4 Závěr

V rámci semestrální práce jsme se snažili nahlédnout na dokumentaci při agilním vývoji z několika různých pohledů a úhlů. V úvodu jsme rozebrali rozdíl mezi tradičním a agilním vývojem. Následovalo shrnutí agilního manifestu, kde byl rozebrán vztah k agilní dokumentaci a principy, které za agilním manifestem stojí.

V další části jsme rozebrali agilní dokumentaci podrobněji. Zabývali jsme se především významem, problémy, rozlišením kdy skutečně je dokumentace agilní, kvalitou dokumentace, typy dokumentů, udržitelností a nástroji, pomocí nichž lze tvořit kvalitní dokumentaci. V poslední části jsme se zaměřili na agilní vývoj a dokumentaci v regulovaném prostředí a vládním sektoru.

Po důkladném prozkoumání literatury a vyhodnocení získaných dat jsme došli k následujícím poznatkům. Agilní vývoj dokumentaci nezavrhuje, jen se snaží dokumentovat pouze nejdůležitější minimum. Čím větší množství dokumentace tým vytvoří, tím více dokumentace musí následně udržovat. Zabývání se dokumentací stojí čas i peníze, které by mohly být využity ke zlepšení vyvíjeného softwaru. Agilní tým musí sám nalézt optimální rozsah dokumentace, který ho stojí nejmenší úsilí a zároveň mu přináší největší užitek, podobně, jako to dělá při vývoji, kde s postupem času zjišťuje, kolik práce stihne v jednotlivých iteracích dokončit.

Jak vyplývá z předchozího odstavce, tak agilní dokumentace určitě má své místo, jen je nutné udržet ji v mezích právě tak, aby neovlivnila agilnost vývoje.

5 Literatura

agilemanifesto.org. 2001. agilemanifesto.org. *Manifest Agilního vývoje software.* [Online] agilemanifesto.org, 2001. [Citace: 1. duben 2016.] http://agilemanifesto.org/iso/cs/.

Ambler, Scott. 2012a. Agile/Lean Documentation: Strategies for Agile Software Development. *Agile Modeling.* [Online] Ambysoft Inc., 2012a. [Citace: 13. březen 2016.] http://www.agilemodeling.com/essays/agileDocumentation.htm.

—. 2012b. Best Practices for Agile/Lean Documentation. *Agile Modeling.* [Online] Ambysoft Inc., 2012b. [Citace: 13. březen 2016.] http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm.

—. 2002. Examining the Agile Manifesto. *Ambysoft.* [Online] Ambysoft, 2002. [Citace: 16. Duben 2016.] http://www.ambysoft.com/essays/agileManifesto.html.

—. 2005. Just Barely Good Enough Models and Documents: An Agile Best Practice. *Agile Modeling.* [Online] Scott Ambler, 2005. [Citace: 24. Duben 2015.] http://agilemodeling.com/essays/barelyGoodEnough.html.

Cripsin, Lisa a Gregory, Janet. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams 1st Edition.* místo neznámé : Addison-Wesley Professional, 2009. 978-0321534460.

Kasik, Olivia a Silver, Omar. 2015. Agile in the Federal Space: Best Practices for Keeping Documentation Lean. *Agile Alliance.* [Online] prosinec 2015. [Citace: 13. březen 2016.] https://www.agilealliance.org/wp-content/uploads/2015/12/ExperienceReport.2014.KasikSilver.pdf.

Kasik, Olivia. 2015. Successfully Creating Value Added Documentation Using Agile Software Development in Heavily Regulated Environments. *ZapThinkTank.* [Online] Dovel Technologies, 19. říjen 2015. [Citace: 13. březen 2016.] http://zapthink.com/2015/10/19/successfully-creating-value-added-documentation-using-agile-software-development-in-heavily-regulated-environments/.

pmdocuments.com. 2012. Agile vs Waterfall. *Project Management Documents and Templates.* [Online] Project Management Documents and Templates, 21. Srpen 2012. [Citace: 11. Duben 2016.] http://www.pmdocuments.com/2012/08/21/agile-vs-waterfall-are-they-compatible/.

Harrison, S., Tzounis, A., Maglaras, L., Siewe, F., Smith, R. and Janicke, H. (2016). A Security Evaluation Framework for U.K. E-Government Services Agile Software Development. International Journal of Network Security & Its Applications, 8(2), pp.51-69.

Selic, B. (2009). Agile Documentation, Anyone?. IEEE Softw., 26(6), pp.11-12.

ZapThinkTank. (2015). Value Added Documentation in Agile Software Development. [online] Available at: http://zapthink.com/2015/10/19/successfully-creating-value-added-documentation-using-agile-software-development-in-heavily-regulated-environments/ [Accessed 24 Apr. 2016].