Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

4IT421 – Zlepšování procesů budování IS

Semestrální práce ke kurzu 4IT421 - Zlepšování procesů budování IS	
Semestr	**Zimní 2016/2017**
Autoři	**Alina Burková, xbura14** **Jan Teplý, tepj00** **Petr Krejča, xkrep35**
Téma	**WaTerFall requirements in Agile Product Development**
Datum odevzdání	**18. 12. 2016**

2016

Abstrakt

Tato semestrální práce je zaměřena na možnosti propojení vodopádového modelu a agilního vývoje. V práci jsou stručně představeny obě metodiky a jsou uvedeny jejich silné a slabé stránky. Dále jsou představeny Business Requirements Documents a je popsáno jejich použití v agilním vývoji. V závěrečné části je představen pojem hybridní model a možnosti propojení vodopádových a agilních metodik.

Klíčová slova

BRD, hybridní model, agilní metodiky.

Obsah

1 Úvod

Cílem této semestrální práce je seznámit čtenáře s agilními metodikami a vodopádovým modelem a možnostmi jejich kombinace (hybridní model). V úvodní části jsou krátce představeny specifické vlastnosti agilních metodik a vodopádového modelu, jejich rozdíly, výhody a nevýhody.

Další části jsou věnovány možnostem kombinace těchto metodik a představení hybridního modelu. Konkrétně se práce zaměřuje na použití Business Requirements Documents v agilním vývoji a popisu hybridního modelu, jako možné kombinace vodopádového modelu a agilních metodik.

2 Popis agilních metodik

Agilní metodiky jsou moderním přístupem k vývoji nejen software. Oproti tradičním robustním metodikám postaveným na vodopádovém modelu, popisují agilní metodiky spíše principy a praktiky, jak dosáhnout výsledků namísto přesně popsaných procesů. Omezují množství dokumentace projektů a dávají přednost osobní komunikaci. Základem těchto metodik je přímá spolupráce vývoje a zákazníků. Vývojáři se snaží dodat zákazníkovi co nejdříve hotové části systému a očekávají od něj zpětnou vazbu. Umožňují rychle reagovat na tuto zpětnou vazbu a na změny v požadavcích.

2.1 Vlastnosti agilních metodik

Agilní metodiky jsou založeny na Manifestu Agilního vývoje softwaru [Agile Manifesto, 2001], který definuje následující preferenční hodnoty:

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

Kromě těchto preferenčních hodnot se agilní metodiky řídí také těmito dvanácti principy:

1. Naší nejvyšší prioritou je uspokojit zákazníka skrz rychlé a průběžné dodávání kvalitního softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
10. Základem je jednoduchost – umění co nejvíce práce vůbec nedělat.
11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
12. Tým v pravidelných intervalech vyhodnocuje svou práci a upravuje své postupy tak, aby byl co nejefektivnější.

2.2 Příklady agilních metodik

- **DSDM** je původní agilní metodika vytvořená ve Velké Británii v 90. letech. Je postavená přímo na agilních principech.
- **Scrum** je agilní metodikou, která se zaměřuje na management úkolů v týmu a komunikaci jeho členů. Základem jsou krátká setkání týmů, na kterých každý člen představí, na čem pracuje. Vývoj probíhá ve Sprintech a na konci každého sprintu je nová funkcionalita předvedena zákazníkovi a vývoj dále reaguje na jeho zpětnou vazbu. Metodika popisuje tři role. Product Owner má za úkol komunikaci se zákazníkem, Scrum Master zajišťuje fungování týmu a Scrum Team Member je členem vývojového týmu.
- **XP (Extreme Programming)** je metodikou pro malé týmy pracující na projektech s často se měnícím nebo nejasným zadáním. Metodika pracuje také s párovým programováním dvou vývojářů u jednoho počítače.

2.3 Práce s požadavky v agilních metodikách

V práci s požadavky jsou dva hlavní rozdíly mezi tradičními a agilními metodikami. Agilní metodiky jsou adaptivní spíše než prediktivní. Jsou připravené reagovat na časté změny a těžit z nich. Oproti tomu v tradičních metodikách jsou požadavky připraveny dlouho dopředu a přesně. Každá případná změna tak může být složitá a časově náročná. Druhým rozdílem je, že se agilní metodiky soustředí hlavně na expertízu, kompetentnost s přímou spolupráci lidí. Tradiční metodiky více spoléhají na procesy dokumentace. [Paetsch, Eberlein, Maurer, 2003]

DSDM používá pro vývoj několik fází. V prvních dvou z nich, Feasibility study a Bussiness study, jsou připraveny základní požadavky systému. Další požadavky jsou vytvářeny během vývoje. Metodika nespecifikuje přesnou metodu tvorby požadavků, proto je možno použít kteroukoliv.

SCRUM spravuje požadavky pomocí Product Backlog pro celý projekt, a Sprint Backlog pro jednotlivé sprinty. Product Backlog obsahuje všechny požadavky na funkce, vylepšení a opravy chyb seřazené podle priority. U požadavků jsou také všechny potřebné informace pro vývoj. Pro Sprint Backlog jsou z Produck Backlogu vybrány požadavky s nejvyšší prioritou, které budou vyvíjeny během následujícího sprintu. Na konci každého sprintu jsou vyhodnoceny splněné požadavky a produkt předveden zákazníkovi. Získané informace jsou použity při plánování dalšího sprintu a přípravě požadavků pro Product Backlog.

XP používá pro správu požadavků tzv. Story Cards. Story Cards jsou vytvářeny během interview a brainstormingů vývojářů se zákazníkem. Vývojáři na základě získaných poznatků odhadují složitost jednotlivých Story. Zákazník stanovuje priority, podle kterých jsou vybírány Story pro další iteraci vývojového cyklu.

2.4 Techniky přípravy požadavků a práce s nimi

V agilním vývoji se často používají následující techniky pro práci s požadavky. [Paetsch, Eberlein, Maurer, 2003]

Začlenění zákazníka je během agilního vývoje velice důležité. Zákazník, nebo jeho zástupce, by měl být vývojovému týmu dostupný k zodpovězení otázek a také se přímo podílí na tvorbě požadavků ve všech částech vývoje. V tradičních metodikách probíhá komunikace se zákazníkem ohledně požadavků hlavně v úvodních krocích vývoje.

Interview - rozhovory se zákazníkem nebo jeho zástupcem jsou hlavním zdrojem informací o požadavcích pro vývojový tým. Přímé rozhovory se zákazníkem zajistí, že se během průchodu přes další lidi nezkreslí a nevedou tak k nedorozuměním a chybám.

Prioritizace se používá ve všech agilních přístupech. Častým použitím je vývoj požadavků s nejvyšší prioritou, tak aby co nejdříve přinesly co největší hodnotu zákazníkovi. Prioritizace probíhá průběžně s přidáváním nových požadavků a získáváním povědomí o projektu.

Brainstorming je technika pro rychlé vytváření a sepisování nápadů. V agilním vývoji ji lze využít při vyváření nových požadavků na funkce, které by zákazník nebo uživatel ocenil.

Validace se používá se zhodnocením splnění požadavků na konci vývojových iterací, a to jak při kontaktu se zákazníkem, tak při schůzkách týmu. Během validace mohou vzniknout nové požadavky na zlepšení nebo opravy.

3 Popis vodopádového modelu

Metodiky postavené na tradičním vodopádovém modelu pracují se sekvenčním vývojem. Původní model rozděluje vývoj na sedm částí:

1. Specifikace požadavků
2. Návrh
3. Implementace
4. Integrace
5. Testování a ladění (validace)
6. Instalace
7. Údržba

Na požadavcích se pracuje v první částí, kdy je vypracováno a schváleno Business Requirements Documents (BRD). Jakékoliv další změny v těchto požadavcích musejí projít složitým procesem schvalování a jakékoliv větší změny požadavků mohou neúměrně prodloužit vývoj. Problematické také je, že software je zákazníkovi předveden až po implementaci a integraci a vývojáři tak nemohou průběžně získávat zpětnou vazbu.

3.1 Výhody a nevýhody

Při vývoji ve vodopádovém modelu se může stát, že se projektu nedostává prostředků a času, řešením toho je vynechat fázi testování. Projekt může být nepřehledný, protože tím, že je fungující software dodán až na konci vývoje, nemusí tým vědět, jak na tom přesně je. V případě, že se objeví problém například v návrhu až na konci vývoje, je často velice namáhavé a drahé ho vyřešit.

Agilní přístup řeší tyto problémy tím, že všemi fázemi vývoje projde během jednoho cyklu. [Rasmusson]

4 Vodopádové požadavky v agilním vývoji

V současné době skoro všichni ví, co znamená pojem Business Requirements Documents (dokumenty o byznys požadavcích). Obecně BRD jsou psané byznys analytiky, kteří reprezentují organizační vrstvu, která se usadí mezi koncovými zákazníky a vývojáři softwaru. BRD popisuje, co přesně zákazník od systému čeká a chce, a to včetně funkčních a nefunkčních požadavků. BRD je sada komplexních, obsáhlých a detailních dokumentů, sestavených na základě dalekosáhlých plánů, které pokrývají několik fází a vydání vývoje softwaru.

Historicky byly BRD používané v sekvenčním/fázovém vývoji produktu, kde podrobná a otevřená analýza a dokumentace byla rozšířená o důkladný návrh a architekturu systému, o programování po dílčích částech, o QA (Quality Assurance) a konečně o UAT (User Acceptance Testing). Ve vodopádovém procesu v okamžiku, kdy je BRD vytvořen, je předán dál dolů po řetězci, od jedné organizační vrstvy k další pro dokončení a schválení. BRD představuje formální smlouvu mezi různými organizačními jednotkami v rámci stejné organizace a dále mezi zákazníky a vývojáři. Tyto smlouvy obvykle obsahují předpoklady o tom ‚Co' musí být dodáno, ‚Kdy' to má být dodáno a ‚Kolik' toto dodání bude stát. [Gendel 2016]

4.1 Obecné nedostatky BRD

4.1.1 BRD vylučují snadné změny

Vytvoření komplexní specifikace pro vývoj předpokládá, že všechna nejlepší řešení jsou známá předem. Dalším požadavkem je, že v průběhu cyklu vývoje by zákazníci neměli měnit své požadavky a neměli by přicházet s novými a lepšími nápady. Veškeré změny totiž aktivují proces 'požadavek na změnu', který je standardním protokolem pro vypořádaní s rozsahem dotvarování. Během tohoto procesu vytvářejí byznys analytici objemnou dokumentaci a poté dokončují požadavky bez přímé vstupní informace od vývojářů.

Nejlepší nápady a řešení přicházejí často mnohem později, kdy je vývoj v plném proudu. Také není neobvyklé, že zákazníci změní názor na původně stanovené požadavky až v průběhu vývoje. V takovýchto případech, aby se dalo odůvodnit rozsah, musí se projít zdlouhavým a byrokratickým procesem změny, který vyžaduje množství času a úsilí. Návrhově jsou BRD určeny odolávat změnám, cokoliv co by vyžadovalo aktualizaci potom, co jsou BRD dokončeny a schváleny, bude vést k vytváření problémů. Když jsou BRD vypracované bez účasti vývojářů, tak obvykle vytváří mnoho nerealistických přání a očekávaní zákazníků, kteří často hledají složitá a nákladná řešení.

4.1.2 BRD jsou produkovány prostředníky

Typicky jsou BRD produkované byznys analytiky (BA), skupinou lidí, která představuje odlišnou funkční vrstvu, řízenou vlastním liniovým vedením. Vzhledem k tomu, že BA nejsou zákazníci, tak nemají úplnou představu o skutečných potřebách byznysu. Ani BA, ani vývojáři softwaru nemohou znát všechny nuance a specifika technických možností, závislostí nebo omezení. Fakticky jsou byznys

analytici prostředníci, umístění mezi byznysem a vývojáři. Tato dodatečná vrstva komplikuje celý proces a často vede k nedorozumění, prodloužení komunikace a tím celkového času práce.

4.1.3 Smluvní chování způsobené BRD

BRD představuje smlouvu mezi byznysem a technologiemi. Je to vlastně neoficiální interní smlouva mezi různými částmi té samé organizace, rozděluje ji (organizaci) na "my" a "oni". Každá organizační vrstva je zaměřena na své osobní cíle a ztrácí pohled na společné strategické cíle firmy. Každá organizační vrstva je motivovaná peněžními a dalšími výhodami, které po splnění tohoto cíle dostane, a proto je především soustředěna na dodání své části co nejlepším způsobem a pak předání této části na další organizační vrstvy s tím, že se stanou zodpovědné za ní.

4.1.4 Lokální optimalizace

Vzhledem k tomu, že každá organizační vrstva je zaměřená na to, co má dodat/předat, tak optimalizuje interní procesy pouze pro své potřeby, nikoliv v rámci celé organizace. Tomu se říká místní optimalizace.

Organizační vrstva, která je zodpovědná za vytváření dokumentace (většinou byznys analytici) pokládá dokončení BRD za svůj místní úspěch. Aby se to uskutečnilo, organizační vrstva, která směřuje ke svému, místnímu cíli, začíná expandovat a nabírat víc lidí, aby mohla splnit cíl včas. Poté, co je dokumentace podepsaná a předána dál jiné organizační vrstvě (architekti, designeři, vývojáři a testeři), vrstva zodpovědná za dokumentaci bude nevyužitá. Organizační vrstva nabrala množství nových lidí a potřebuje odůvodnit, proč tam jsou. Proto začnou pracovat na nepodstatných věcech, které mají nízkou byznys prioritu a vrstva se proto místně optimalizuje, aby odůvodnila svoji velikost a aktivity. Pokud to neudělá, tak se bude muset zmenšit cestou propouštění svých lidí. [Gendel 2016]

4.2 Výjimky

Mohou nastat některé výjimečné situace, kde předem zpracovaná dokumentace většího rozsahu bude opodstatněna. Takové výjimky jsou spíše vzácné, ale jsou tu některé příklady takových situací:

- Organizace podepsala externí smlouvu s třetí stranou (dodavatel), ve které je jasně definován výpis práce (SOW) a legální smlouvy definující vztahy. V tomto případě, kdy práce bude probíhat v sekvenci (vodopádu), může být nutné mít detailně vypracovanou dokumentaci, aby nedošlo k nedorozumění na obou stranách. Nicméně stále není doporučeno stanovit vše předem: rozsah, rozpočet a časový harmonogram. S největší pravděpodobností budou rozsah a rozpočet fixní (oproti agilnímu vývoji, kde je rozpočet obvykle flexibilní), ale pokud tomu tak je, tak potom by časový harmonogram měl zůstat flexibilní. [Gendel 2016]

- Organizace/skupina/oddělení pracuje na limitovanému rozsahu, kde se předpokládá dokončení ve velmi krátkém časovém období. V takových případech jsou rizika odbočení z cesty a setkání se s klasickými výzvami vodopádového životního cyklu podstatně nižší.

- Další situace je taková, že organizace chce z provozu vyřadit starší systém a nahradit jej jiným systémem, který má novější technologie, a s tím, že chce zachovat většinu stálé funkčnosti a

nechce vybudovat úplně nové řešení. Tady je dokumentace použita k zachycení nálezu během reverzního inženýrství stávajícího systému. [Gendel 2016]

4.3 Problémy s BRD v agilním vývoji

Jsou známé případy, kdy týmy a jejich Product Owner zkusily využít BRD v agilním vývoji. V následujících odstavcích jsou uvedeny dva případy, jak to může dopadnout.

Tým se snaží vyvíjet po iteracích a sprintech na základě svého porozumění celému BRD. Vzhledem k tomu, že BRD je všezahrnující dokument, který obvykle pokrývá práci na mnoho týdnů a měsíců dopředu, rozložení na menší nezávislé požadavky je velice náročné.

V důsledku toho, že požadavky nejsou nezávislé, týmu se špatně určuje, jak má rozdělit systém na menší části. Product Owner dá přednost částem BRD, které se dají rozdělit. Pracuje se tak na náhodně vybírané části BRD, která se nedá klasifikovat jako MMF (minimal marketable features) nebo PSPI (potentially shippable product increments), kterou lze samostatně dodat. V důsledku toho nebudou zákazníci a Product Owner schopní vidět funkční produkt na konci každého sprintu. Tým se zabývá vývojem několika určitých části systému v úvodních sprintech (databáze, API, UI), dále pak propojením těchto částí a řešením defektů ve sprintech dalších. Jednání mezi týmem a vlastníkem produktu se zastaví kvůli tomu, že BRD jsou "všechno nebo nic" kontrakt, a tým není schopen dodat kompletní část softwaru.

Druhý případ je, když tým nepracuje s BRD, ale Product Owner dál používá BRD v komunikaci se zákazníkem. Zde je situace taková, že tým pracuje v agilním režimu, ale vlastník produktu komunikuje se zákazníkem ve vodopádovém režimu, pak přepracovává požadavky z BRD do formy pro backlog, což je časově hodně náročné, a zavádí do požadavků často omyly a vede to k nedorozuměním. Tím mohou vzniknout mylné předpoklady o protistranách: tým předpokládá, že zákazník je naladěný na agilní vývoj, vyjednávání, změny v požadavcích a jiné činy typické pro tento typ vývoje. Naopak zákazník předpokládá jasné splnění požadavků, a to v termínech stanovených v BRD. Tím, že obě strany čekají od sebe něco jiného, se ztrácí důvěra.

5 Hybridní model

Předchozí část se věnovala tvorbě a použití business requirements documents (BRDs) v agilním vývoji, která představuje jednu ze snah spojit vodopádový model s agilním vývojem a umožnit tak snazší přechod firem a zákazníků na plnohodnotný agilní vývoj. Jak se ukázalo, vytváření detailních, komplexních dokumentací nepředstavuje vhodný způsob zkombinování těchto dvou přístupů. Tato kapitola se proto zabývá trochu odlišným způsobem spojení vodopádového a agilního přístupu, tzv. hybridním modelem.

5.1 Popis hybridního modelu

Definování hybridního modelu není úplně jednoduché. Nejjednodušeji ho lze popsat jako přístup k vytváření softwaru, který kombinuje praktiky z vodopádového modelu a zároveň z agilního vývoje. Tato definice je velice volná a způsobuje, že jako hybridní model lze označit v podstatě každý přístup, který není zcela agilní a používá některé prvky z vodopádového modelu. Lze se tedy setkat s termíny jako Hybrid Agile, Hybrid Waterfall, Agile-Waterfall Hybrid a další, které všechny označují hybridní model. Používání těchto termínů nemá pevně definovaná pravidla a záleží spíše na preferenci autorů. [Hering, 2016] V této práci je používán termín hybridní model, případně hybridní přístup.

Hybridní přístup je řadou odborníků, převážně zastánců agilního vývoje, považován za chybný a nefungující. Argumentují, že hybridní přístup nikdy nebude tak úspěšný jako čistě agilní, protože při použití hybridního modelu vznikají problémy mezi oběma přístupy a je nutné přistoupit na některé kompromisy, aby hybridní model mohl fungovat. [Hering, 2016] [Johnson, 2013]

Nesprávné použití hybridního přístupu může mít pro firmu neblahé následky a o jeho nasazení by se mělo rozhodovat na základě charakteru projektu a také firemní kultury a zkušeností lidí ve firmě. Firmy se také často chlubí, že vyvíjejí agilně, ale ve skutečnosti si vypomáhají vodopádovými prvky. [Oluwole, 2015]

Byla by ale chyba hybridní model kompletně odepsat. Podobně jako agilní vývoj vznikl jako reakce na nedostatky vodopádového modelu, hybridní model byl vytvořen kvůli snaze odstranit slabé stránky agilního vývoje. Snaží se kombinovat to nejlepší z obou přístupů a tím odstranit jejich nedostatky a omezení. [Harshpal, 2016]

5.2 Důvody existence hybridního modelu

Hlavním důvodem pro zachování částí vodopádového modelu jsou obavy ze změn. Firmy nechtějí měnit zažité postupy a strach z nepředvídatelnosti a radikálnosti agilního vývoje, který nemá přesný plán, je přiměje setrvat u ověřených metod. Tento problém často vzniká u vládních organizací nebo velkých korporací, které se kvůli byrokracii a skepticismu mění pouze pomalu a stěží. [Oluwole, 2015]

Dalším důvodem je fakt, že firmy si začínají uvědomovat, že agilní vývoj nepředstavuje vždy nejlepší řešení a pro některé projekty je dokonce nevhodný. Jedná se především o vývoj velkých, kritických systémů, kde je nutné přesně splnit požadovaná kritéria a zajistit dostatečnou bezpečnost systému. S

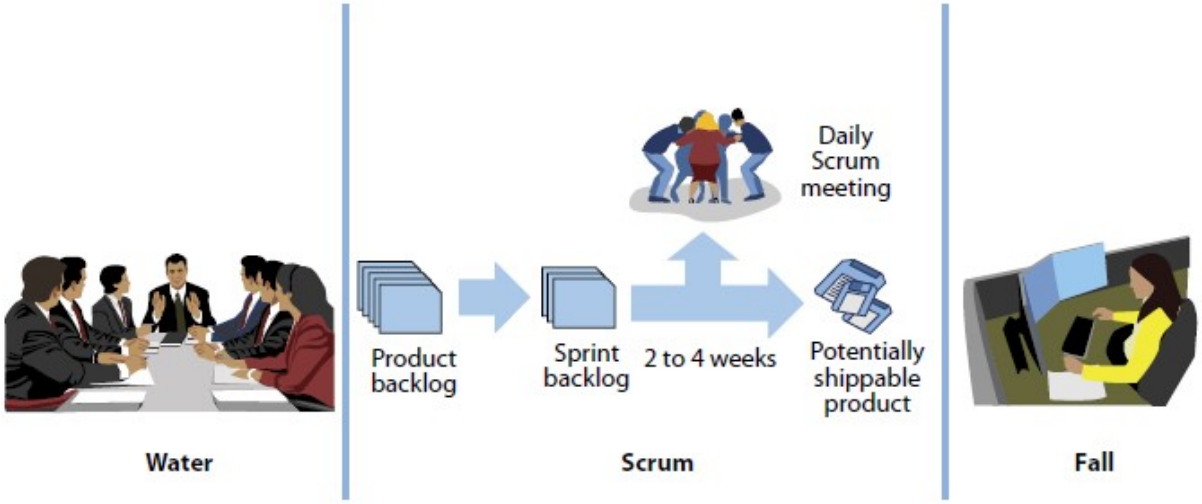
tím souvisí riziko agilního vývoje spočívající ve špatném navržení architektury systému, které lze částečně eliminovat použitím hybridního modelu, kdy se architektura systému navrhne dopředu jako ve vodopádovém modelu. Samotný vývoj pak probíhá v menších iteracích agilně. Rozdíl v přístupu je také na základě úrovně řízení. Zatímco malé týmy, které programují, mohou vyvíjet zcela agilně, na vyšší úrovni řízení se postupuje podle vodopádového modelu. [Finnegan, 2014]

Jako další důvod existence hybridních modelů je zaměření agilního vývoje na praktické věci, které vede k tomu, že se týmy soustředí na to, co mohou samy ovlivnit. Především tedy na samotný tým a jeho fungování. Už ale neovlivní věci mimo sféru jejich vlivu, a proto se byznys analýza a uvolňování softwaru stále řídí pomocí tradičního, vodopádového modelu. Agilní vývoj tedy funguje pouze na úrovni týmu, ale ne na úrovni celé firmy. [West, 2011] [Oluwole, 2015]

Jak bylo zmíněno výše, hybridní model je definován volně a může se jednat o libovolnou kombinaci vodopádového a agilního přístupu k vývoji softwaru. V následující podkapitole je popsán jeden z hybridních modelů, který lze realizovat ve firmě. [West, 2011]

5.3 Water-Scrum-Fall

Označení Water-Scrum-Fall vzniklo podle procesu vývoje softwaru ve firmách, kdy fáze plánování je totožná s vodopádovým modelem, software je vyvíjen agilně a jeho nasazení je opět realizováno pomocí vodopádového modelu (viz obrázek 1). [West, 2011]



Obrázek 1: Water-Scrum-Fall (Zdroj: West, 2011)

5.3.1 Fáze Water

Water je první fáze hybridního přístupu, zabývá se analýzou a specifikováním požadavků na software. Řada firem vyžaduje definovat požadavky a plány dopředu. Potřebují vědět, co se bude dělat, kolik to bude stát a jak dlouho to bude trvat. Tento požadavek je ale jen obtížně slučitelný s agilním vývojem, který nemá přesně daný plán a je založen na změnách. V hybridním modelu se lze setkat s následujícími problémy [West, 2011]:

- Definování příliš mnoho požadavků dopředu – je důležité nespecifikovat dopředu všechny požadavky dopodrobna, aby byl prostor k úpravě požadavků a nedocházelo k neustálým změnám a zvyšováním objemu práce na projektu.

- Komunikace se zákazníkem – zákazníci často dopředu nevědí, co chtějí a jaké požadavky budou mít na software. K softwaru se dostanou až ve fázi vývoje, a proto požadavky na změnu softwaru vznikají až v pozdní fázi.

- Lidé se soustředí na vlastní výstup – zaměstnanci se často zaměřují pouze na splnění svého úkolu, nespolupracují s ostatními odděleními ve firmě a úkol pro ně končí předáním výstupu dál. Nevytváří tak hodnotu pro zákazníka, ale pouze si plní svůj úkol.

Z výše uvedeného je patrné, že není vhodné věnovat příliš času popisu požadavků na systém, protože bude docházet k jejich změnám. Rovněž je nutné se zaměřit na spolupráci mezi odděleními a komunikaci se zákazníkem.

5.3.2 Fáze Scrum

Fáze Scrum zahrnuje implementaci softwaru a jeho testování. V hybridním modelu je software vyvíjen agilně, často pomocí metodiky Scrum, která se těší velké popularitě. Při používání Scrumu v hybridním modelu je ale třeba dodržovat určité zásady [West, 2011]:

- Různorodý tým - Scrum tým musí obsahovat lidi z různých profesí a oblastí. Nelze ho uplatnit pouze na vývojáře softwaru. Jeho součástí by měli být i testeři a byznys analytici.

- Zahrnutí testování do sprintu – hybridní model svádí k odložení testování softwaru na poslední fázi Fall a tím pádem ke zrychlení sprintů. To by ale byla chyba. Odložení testování má za následek pozdní zpětnou vazbu a nalezení chyb, což může prodloužit a prodražit vývoj.

- Zapojení byznysu – v rámci hybridního modelu je zapotřebí udržovat kontakt s byznysem i po fázi Water, tedy po dokončení specifikování požadavků na software. Byznys požadavky nemají být detailním a neměnným dokumentem. V případě nejasností či změn musí vývojáři komunikovat s byznys analytiky, aby byli schopni vyřešit případné problémy.

- Přijmutí změn – vodopádová fáze Water má definovat celkový směr projektu, ale nemá specifikovat detailní požadavky a architekturu. Vývojáři softwaru musí být připraveni na změny, které nevyhnutelně přijdou a musí jim být umožněno tyto změny provést.

5.3.3 Fáze Fall

Poslední fáze se nazývá Fall a zabývá se vydáváním a nasazením softwaru. Myšlenka agilního vývoje je spojená s častým vydáváním softwaru, aby vývojáři dostávali rychlou zpětnou vazbu. Ne každý podnik má ale dostatečnou infrastrukturu, která umožňuje časté a rychlé vydávání nových verzí. Proto se raději soustředí na méně frekventované vydávání verzí, které obsahují více změn. Nasazení agilního vývoje těžko změní firemní přístup k vydávání softwaru, nicméně je vhodné přijmout určitá opatření pro realizaci hybridního modelu [West, 2011]:

- Propojení vývoje a nasazení – firemní modely jako IT Infrastructure Library (ITIL) a Capability Maturity Model Integration (CMMI) podporují spíše oddělení odpovědnosti a vlastnictví a spoléhání se na dokumentaci. Nepodporují spolupráci, budování týmu a společné cíle, které jsou potřeba pro správné fungování hybridního modelu.

- Vylepšení procesu nasazení softwaru – nalezení a odstranění skutečností, které brání častějšímu vydávání a nasazení softwaru.

- Zahrnutí aktivit, které souvisí s nasazením softwaru, do sprintu – postupně přidávat další aktivity do sprintu. Může se jednat o předprodukční testování, migraci dat, bezpečnostní nebo výkonnostní testování. Realizace těchto aktivit do sprintu zajistí rychlejší zpětnou vazbu a také dřívější nalezení chyb. Důraz je také třeba klást na automatizaci těchto procesů, aby zbytečně nepřidávaly práci.

6 Závěr

Z výše uvedených skutečností vyplývá, že hybridní model nabízí některé zajímavé výhody oproti čistě vodopádovým a agilním metodikám. Díky vodopádovým prvkům umožňuje specifikaci požadavků a termínů dopředu, což ocení především vedení firem. Začlenění agilního přístupu zase přináší lepší komunikaci se zákazníkem, rychlejší vývoj a zpětnou vazbu. Také podporuje spolupráci mezi lidmi ve firmě a urychluje vydávání a nasazení softwaru. Používání hybridního přístupu ale přináší také řadu problémů a rizik. Je nutné přijmou opatření a zásady, například vyhnout se specifikování detailních požadavků dopředu, nebo zajistit dostatečnou komunikaci mezi jednotlivými odděleními.

Použití hybridního modelu není vždy tím nejlepším řešením a o jeho nasazení by se mělo rozhodovat na základě na charakteru projektu, firemní kultuře a zkušeností lidí, kteří na něm budou pracovat. Dále je nezbytné připomenout, že nesprávně realizovaný hybridní model může mít pro firmu neblahé následky v podobě prodloužení a prodražení vývoje softwaru.

Samostatné začlenění BRD do agilního vývoje se nedoporučuje, protože přináší mnoho problémů a nedorozumění. Účinným a bezpečným způsobem jak přivést zákazníky a technologie blíž k sobě a přitom udržet Product Owner stále zapojeného do procesu, je sdílené vzdělání. Bylo by nerozumné očekávat, že školení pouze jeho a týmu o agilních přístupech, normách a rámcích vyřeší problém nedorozumění a nedůvěry mezi byznysem a technologiemi. Zbytek organizace nemůže zůstat mimo, také by měl byt poučen o základních hodnotách agilního vývoje produktu. [Gendel 2016]

Seznam literatury

AGILEMANIFESTO, 2001. *Manifesto for Agile Software Development* [online]. Allaboutagile.com [cit. 2016-10-10]. Dostupné z: http://agilemanifesto.org/

FINNEGAN, Matthew, 2014. *Agile and waterfall – is a hybrid approach best for enterprise app development?* [online]. Computerworlduk.com. [cit. 2016-10-14]. Dostupné z: http://www.computerworlduk.com/it-management/agile-waterfall-is-hybrid-approach-best-for-enterprise-app-development-3572875/

GENDEL, Gene, 2016. *Waterfall Requirements in Agile Product Development* [online]. Infoq.com. [cit. 2016-10-14]. Dostupné z: https://www.infoq.com/articles/waterfall-requirements-agile

HARSHPAL, Singh, 2016. *Case Study: How to Eliminate Flaws of Waterfall and Agile Development Processes Using a Hybrid Model* [online]. Softwaretestinghelp.com [cit. 2016-10-14]. Dostupné z: http://www.softwaretestinghelp.com/agile-waterfall-hybrid-model/

HERING, Mirco, 2016. *Is There Really Such a Thing as a "Hybrid Agile" Method?* [online]. Infoq.com. [cit. 2016-10-12]. Dostupné z: https://www.infoq.com/articles/does-hybrid-agile-exist

JOHNSON, Eva, 2013. *Agile-Waterfall Hybrid: Smart Approach or Terrible Solution?* [online]. Intland.com. [cit. 2016-10-14]. Dostupné z: https://intland.com/blog/agile/agile-waterfall-hybrid-smart-approach-or-terrible-solution/

OLUWOLE, Dele, 2015. *Hybrid Agile Versus Agile?* [online]. Scrumalliance.org [cit. 2016-10-14]. Dostupné z: https://www.scrumalliance.org/community/articles/2015/may/hybrid-agile-versus-agile

PAETSCH, Frauke. EBERLEIN, Armin. MAURER, Frank, 2003. *Requirements Engineering and Agile Software Development* [online]. Allaboutagile.com [cit. 2016-10-10]. Dostupné z: http://ase.cpsc.ucalgary.ca/uploads/Publications/PaetschEberleinMaurer.pdf

RASMUSSON, Jonathan, *Agile vs. Waterfall* [online]. [cit 2016-12-18]. Dostupné z: http://www.agilenutshell.com/agile_vs_waterfall

WEST, Dave, 2011. *Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today* [online]. Forrester Research. [cit. 2016-12-01]. Dostupné z: http://www.storycology.com/uploads/1/1/4/9/11495720/water-scrum-fall.pdf