Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Why Agile Didn’t Work

SEMESTRÁLNÍ PRÁCE

4IT421 Zlepšování procesů budování IS

Vypracovali: Michal Hric (xhrim06), Jakub Dušek (xdusj22)

Datum odevzdání: 15. 5. 2016

LS 2015/2016

Abstrakt:

Tato semestrální práce se zabývá problematikou implementace agilních metodik vývoje softwaru a klíčovými faktory pro úspěšnost přechodu na agilní přístupy vývoje. V první části práce jsou představeny hlavní charakteristiky a principy agilního vývoje softwaru a uvedeny příklady a specifika třech populárních metodik – scrum, extrémní programování a crystal metodiky. V druhé části práce se zabýváme přímo problémy při neúspěšné implementaci metodik. Identifikovány jsou časté a významné problémy po přechodu týmu na agilní přístupy, stejně tak jako jejich příčiny a následky. Identifikované problémy vychází z článku „Why Agile Didn't Work" od Chen Ping. V závěru práce jsme shrnuli některá podstatná doporučení, kterými by se měli řídit společnosti, pokud zamýšlí implementovat agilní přístupy vývoje softwaru.

Klíčová slova:

Agilní vývoj, problémy implementace agilních metodik, doporučení při implementaci agilních metodik, scrum

Obsah

1. Úvod

V této práci se budeme zabývat převážně problematikou implementace agilních metodik a hodnocením jejich očekávaných a reálných přínosů pro danou společnost či tým lidí. Představa mnoha lidí je v dnešní době totiž taková, že pokud nasadí nějakou agilní metodiku, musí se zlepšit jejich efektivita, produktivita a kvalita výstupu, přičemž to ale není zcela tak jednoduché. Agilní metodiky nemohou ve všech situacích zaručit na 100% zlepšení, v některých situacích ani není vhodné agilní metodiky implementovat s ohledem na složení týmu či požadovaný výstup. Autorka článku „Why Agile Didn't Work" Chen Ping píše, že se jí lidé často ptají na otázky typu: „Implementovali jsme agilní metodiku, ale proč to nefunguje?". A na tuto otázku bychom se v naší práci chtěli zaměřit a odpovědět.

Týmy často implementují agilní metodiky v situacích, kdy se potýkají s problémy s nízkou produktivitou nebo nízkou kvalitou výstupů a doufají, že agilní přístupy jim pomohou zázračně zlepšit situaci. Co si ale tyto týmy neuvědomují je fakt, že agilní přístup je systematický přístup řízení softwarového vývoje. Není to pouze o uspokojování potřeb zákazníka tím, že budeme uspokojovat jeho měnící se požadavky, agilní přístup vyžaduje také „Stand up meeting" každý den, plánování schůzek a retrospektivní hodnocení schůzek a další charakteristiky, které dále v textu rozvedeme. Agilní přístup vyžaduje mnoho manažerské a technické podpory, bez které nepřežije a může se stát, že lidé začnou při špatné implementaci agilní metodiky skrytě tento nový přístup nenávidět a budou se chtít vrátit k předchozím metodikám vývoje. (Ping, 2015)

Cílem této semestrální práce je přiblížit čtenářům problematiku přechodu z rigorózních metod vývoje softwaru na metody agilní. Práce nejprve uvede čtenáře do problematiky agilního vývoje SW, důvodů jeho vzniku a základních charakteristik. Dále budou uvedeny metody agilního vývoje nejčastěji používané v praxi. V další části se tato práce bude zabývat otázkou, proč k nasazení agilních metodik nestačí pouze jejich teoretická znalost. Dle výchozího článku "Why Agile Didn't Work" se totiž problémy uvnitř týmu změnou metodik nevytratí, ba naopak se znásobí. V práci budou uvedeny postupy, jak taková slabá místa v týmu lokalizovat a eliminovat. K zodpovězení této otázky budou spolu s uvedeným článkem využity i další hlubší znalosti nabyté z odborné literatury. V závěru budou shrnuty veškeré poznatky a shrnuta doporučení.

2. Agilní vývoj IS

2.1 Charakteristika agilního vývoje IS

Agilní přístup k vývoji je novější přístup, který začal vznik od poloviny 90. let 20. století a oproti rigoróznímu přístupu se snaží více reagovat na novodobé požadavky, kterými jsou především rychlé a pružné zavedení informačních systémů. Dříve častěji používané rigorózní

metodiky vývoje mohou být, jak jejich název napovídá, příliš zkostnatělé a nedokáží se tak rychle a pružně přizpůsobit měnícím se požadavkům na informační systém. Hlavní myšlenkou agilního vývoje je přístup, kdy se snažíme software nebo jeho část vyvinout co možná nejrychleji, předložit ho zákazníkovi, které zhodnotí nedostatky a poté software upravit. Při agilním přístupu je velmi důležitá komunikace mezi dodavatelem a zákazníkem a přesná specifikace požadavků zákazníkem a následné odsouhlasení požadavků. Při agilním vývoji se snažíme vytvořit přesně a pouze to, co zákazník požadoval bez funkcionalit a prvků navíc. (Buchalcevová, 2009)

Existuje množství agilních metodik, jejich hlavní charakteristiky jsou ale stejné a jsou obsaženy v tzv. „Manifestu agilního vývoje softwaru“, který v roce 2001 sepsali hlavní představitelé agilních přístupů. V manifestu jsou definovány čtyři základní hodnoty agilního vývoje, kterými jsou:

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

V manifestu je uvedeno, že body nalevo jsou hodnotnější (tučným písmem zvýrazněné), než body napravo. Z hlavních bodů manifestu je patrné pár základních pravidel agilního vývoje. Vývoj je vždy orientován na fungující konečný software, ostatně to, co je pro zákazníka nejdůležitější. Je dán velký důraz na komunikaci a spolupráci se zákazníkem, jelikož pouze tak jsme schopni plně pochopit všechny jeho požadavky a promítnout je to výstupu. Na předchozí bod navazuje i reagování na změny, při časté komunikaci se zákazníkem jsme schopni rychle identifikovat požadavek na změnu a rychle ho implementovat do produktu. (Beck, Agile Manifesto, 2001)

Dále manifest obsahuje 12 principů, které blíže specifikují charakteristiky agilního vývoje, dle Agilního manifestu se agilní vývoj řídí těmito principy:

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

Agilní přístupy musí být rychlé a pružné.

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

Změny v požadavcích jsou vítané, jen pomocí reagování na změny požadavků je možné docílit spokojenosti s výsledným produktem na obou stranách.

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

Snaha o co nejrychlejší dodání softwaru.

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

Denní spolupráce manažerů a vývojářů je nutná, např. pomocí každodenních meetingů.

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

Agilní metodiky kladou důraz na individuální přístup k motivaci lidí a vytváření jim kvalitního pracovního prostředí.

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

Upřednostňování osobní komunikace.

- *„Hlavním měřítkem pokroku je fungující software.“*

Fungující software je pro zákazníka nejdůležitější.

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

Udržení svižného tempa celého týmu a ostatních zainteresovaných osob.

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

Vysoká kvalita výsledného produktu.

- *„Jednoduchost--umění maximalizovat množství nevykonané práce--je klíčová.“*

Jednoduchá řešení, která budou fungovat.

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

Nesmí se opomenout prostor pro kreativitu v týmu.

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

Je důležité, aby se lidé neustále učili a zlepšovali.

(Beck, Principy stojící za Agilním Manifestem, 2001)

Výše uvedené principy obsažené v Agilním manifestu shrnují hlavní a nejdůležitější charakteristiky agilních přístupů k vývoji softwaru. Různé agilní metodiky se poté liší v některých charakteristikách, ale základní charakteristiky jsou dodržovány u všech.

2.2 Agilní metodiky vývoje IS

Mezi agilní metodiky můžeme zařadit více metodik vývoje softwaru, pro účel této práce jsme vybrali některé často používané. V následujících řádcích budou představeny následující agilní metodiky:

- Scrum
- Extrémní programování
- Crystal metodiky

2.2.1 Scrum

Scrum je agilní metodika pro jednodušší i komplexnější projekty. Původně byla metodika vyvinuta pro řízení projektů vývoje softwaru, ale lze použít i na jiné projekty. Název Scrum byl odvozen podle mlýna v ragby, aby se zdůraznila podobnost s ragby v rychlosti a adaptivitě. Zákazník, neboli Product owner vytvoří prioritizovaný seznam požadavků na produkt, který se nazývá Product backlog. Metodika rozděluje vývoj softwaru na několik Sprintů, přičemž každý Sprint je zhruba 30 dnů dlouhý a na jeho konci výsledkem produkt nebo jeho část, která je použitelná. V týmech, které používají tuto metodiku je klasicky každý den ráno krátká porada tzv. „Scrum meeting“, při kterém obvykle každý člen týmu vstane a stručně sdělí ostatním, na čem poslední den pracoval, na čem bude pracovat dnes a v čem vidí potencionální problémy. Tuto každodenní poradu vede a moderuje Scrum master. (Scrum Alliance, 2016)

Scrum definuje čtyři základní fáze:

- **Plánovací fáze**

  V plánovací fázi se specifikují prvotní požadavky, plán dodávek produktu a architektonická a byznys vize

- **Vynášecí fáze**

  V této fázi jsou do seznamu požadavků zapsány požadavky, které nemají vliv na funkčnost systému

- **Fáze vývoje**

  Ve sprintech se vyvíjí produkt, postupuje se podle priorit v backlogu. Na konci sprintu je předán produkt nebo jeho použitelná část.

- **Fáze dodávky**

  Ve fázi dodávky dochází k předání produktu. (Buchalcevová, 2009)

2.2.2 Extrémní programování

Extrémní programování je metodika, která byla poprvé použita 6. 3. 1996 a je jednou z populárních agilních metodik. Extrémní programování podporuje týmovou práci a při vývoji pohlíží na manažery, zákazníky a vývojáře jako na rovnocenné partnery, kteří chtějí dosáhnout stejného společného cíle. Metodika je určena pro malé nebo střední týmy vývojářů, kteří se pracují s málo strukturovaným nebo často se měnícím zadáním. (Wells, 2013)

Mezi základní pravidla extrémního programovaní patří například:

- Rozdělení projektu do iterací.
- Plánování iterace je prováděno na začátku každé iterace.
- Podpora open-office pracovišť.
- Stand up meeting na začátku každého dne (podobně jako u Scrum).
- Jednoduchost.
- Začínat programováním jednotkových testů.
- Párové programování produkčního kódu.
- Všechen kód musí projít jednotkovým testováním.
- Často frekventované malé sestavení. (Wells, 2013)

2.2.3 Crystal metodiky

Rodina metodik Crystal obsahuje jedny z nejstarších agilních metodik, jejich vznik se datuje do roku 1992. Od ostatních metodik se liší převážně tím, že se nejedná o jednu jedinou metodiku, ale o celou rodinu metodik, ze kterých si můžeme vybrat pro náš projektu tu vhodnou. Z rodiny metodik Crystal si vybereme vhodnou metodiku v závislosti na velikost týmu a důležitost produktu. (Cockburn, 2008)

Crystal metodiky se zakládají na třech základních principech:

- **Poháněny člověkem**

Crystal metodiky jsou tzv. „člověko-centrické“, je dáván důraz na lidi v týmy a jejich potřeby jsou zohledňovány.

- **Ultralehké**

Crystal metodiky snižují administrativní a dokumentační činnosti, i pokud se jedná o velké projekty.

- **Roztažitelné, aby seděly**

Výstupy se tvoří o něco menší, než je potřeba a poté jsou doplněny tak, aby byly požadavky přesně naplněny. (Cockburn, 2008)

3. „Why Agile Didn't Work“ aneb „Proč to u nás nefunguje?“

3.1 Nasazení agilních metodik – Očekávání versus realita

Již v úvodu bylo řečeno, že se koučové agilních metodik často setkávají se stejným vzorcem problémů, které nastanou po nasazení některé z agilních metodik. Od svých klientů, kterým s provozem agilních metodik pomáhají tak dostávají často otázku, proč tolik vychvalované agilní metodiky, které na papíře přeci vypadají tak krásně, u nich po nasazení nefungují vůbec tak, jak si představovali. Lídři a s nimi také celý tým jsou zděšeni, že jejich produktivita a celková morálka dokonce klesla.

V každém týmu existuje zaručeně nějaký nešvar, který je svým způsobem jedinečný. Agilní kouč proto musí disponovat schopnostmi a znalostmi, které mu umožní najít i ty nejhlouběji zakořeněné problémy v organizaci. Takové problémy je třeba identifikovat a řešit hned na počátku. Nelze jen bezostyšně naskočit na módní vlnu agilních metodik a těšit se z nadcházejících zlepšení. Právě takové jednání zaručeně vyvolá potřebu pomoci od experta – kouče agilních metodik. Autorka výchozího článku, koučka Chen Ping dále uvádí, že pohled vedoucího na tabuli v obležení skupinky lidí s fixami může vyvolat falešný pocit, že to zafungovalo. Nikoliv. S agilními metodikami přichází také spousta ne vždy příjemných aktivit, jako jsou například: daily stand up meetings, planning meetings a retrospective meetings. Takový vedoucí si musí uvědomit, že agilní metodiky nebudou fungovat bez podpory vedení a dostatečné technické podpory.

Následuje příklad od autorky článku: V nedávné době pracovala s týmem, který se potýkal s problémy popsanými výše. Tento tým byl větší, rozdělen do sedmi scrum týmů. Každé ráno to bylo stejné, v kanceláři bylo slyšet mnoho hlasů díky několika scrum meetingům pořádaných ve stejný čas. Samozřejmě, že management toto viděl rád, byl to do jisté míry hmatatelný výsledek něčeho, za co zaplatily velké množství peněz.

Chen Ping pak začala pozorovat změnu kurzu, který mířil na pomyslný ledovec. Docházela pravidelně na ranní meetingy. U jednoho týmu si všimla, že byl neustále brzděn tím stejným technickým problémem po několik dní v řadě. Tento tým se pochopitelně jal pracovat na něčem jiném, to ovšem vedlo k množství nedokončeného kódu. Mnoho práce tak bylo vykonáno, leč žádná funkcionalita nemohla být testována, natož předvedena zákazníkům.

V agilních metodikách platí: Scrum master má za úkol překážky odstraňovat. Ovšem v tomto popisovaném týmu to nefungovalo, vůbec. Nelze ovšem hodit vinu jen na Scrum mastera. Ten vysvětlil Ping, že o problému ví, ale není snadné ho vyřešit. Potřebné znalosti k vyřešení nahromaděných problémů totiž má jen pár lidí z celého týmu, bohužel z jiného scrum týmu.

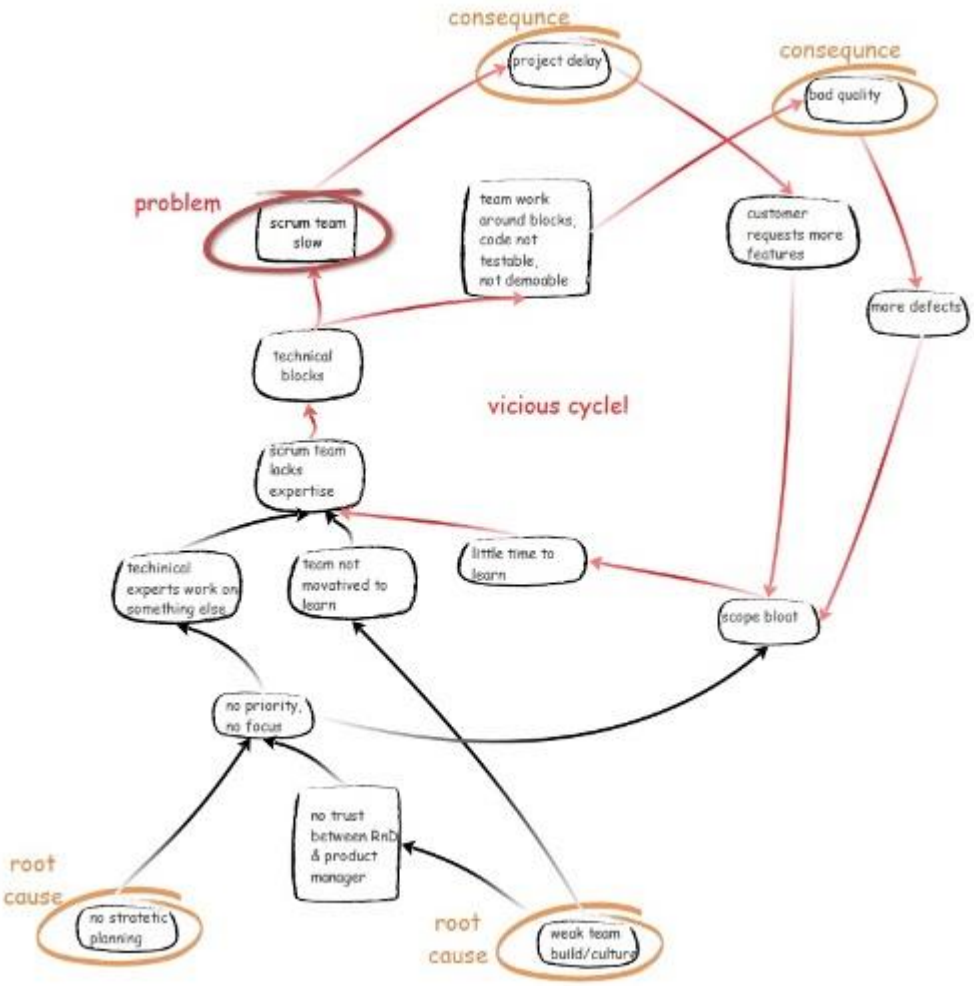
A přesně to je typický problém v agilních vývojářských týmech:

- Metodika praví, že každý si může z backlogu vybrat takový úkol, který chce. To ovšem často vede k situacím, kdy jsou „pěkné“ úkoly vybrané hned a ty zbylé pak vyžadují znalosti okrajových technologií, kterými disponuje jen pár členů týmu. A ti si mezitím pracují na něčem jiném. Ostatní jen těžko namotivujete, aby se začali zabývat technologiemi, které jsou pro ně španělskou vesnicí. Zvláště pak, pokud jde o technologie zastaralé, nyní nevyvíjené.
- Scrum master většinou nemá pravomoci, které by mít měl. Ty většinou zůstávají u manažerů, kteří se jich neradi zbavují. Takové rozdělení může fungovat, pokud je mezi scrum masterem a manažerem oboustranná spolupráce. Přeci jen manažeři jsou často zkušenější v řešení interpersonálních problémů a získáváním podnikových zdrojů. Pokud ovšem tato spolupráce vázne, nastávají problémy.

3.2 Problémy – identifikace, dopady, řešení

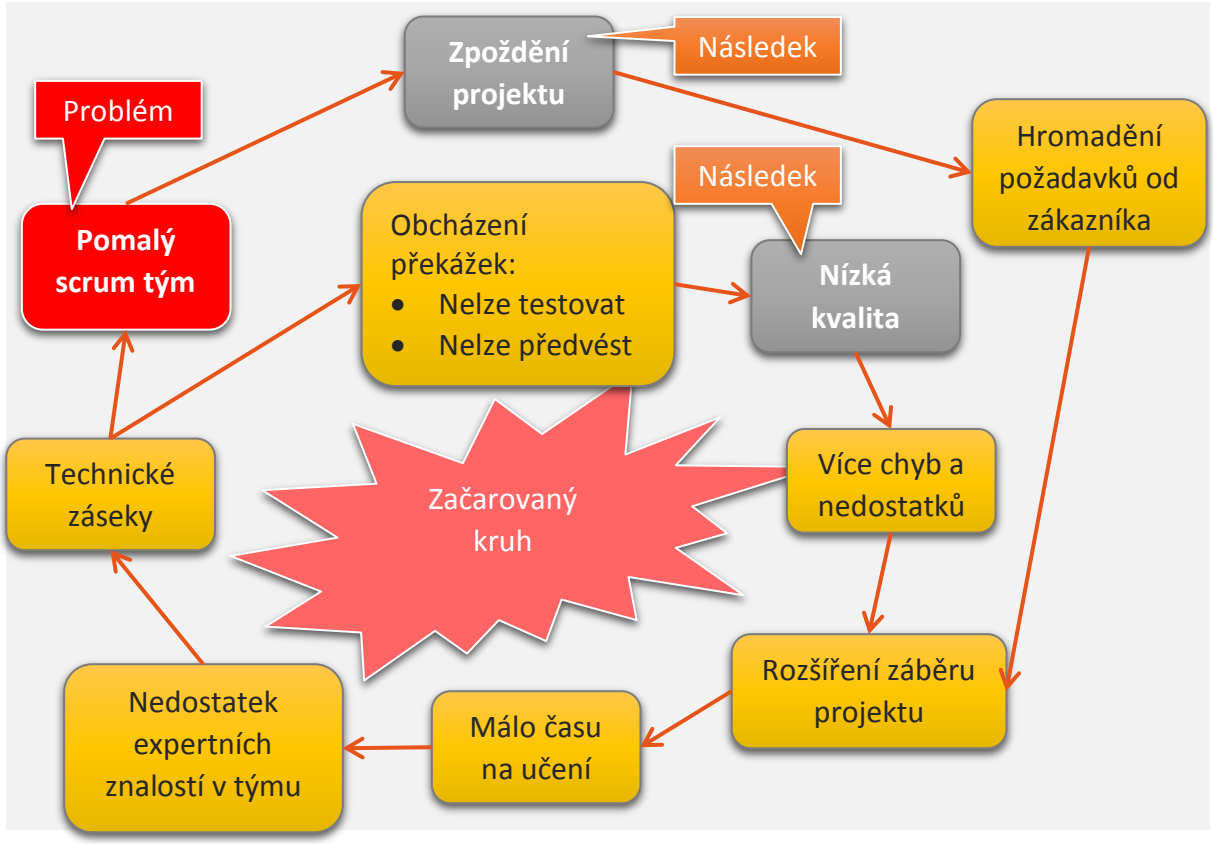
Námi v příkladu popisovaný tým měl větší problémy. Manažer si začal nový projekt bez jakéhokoliv scrum týmu, nebylo tak možné sledovat průběh. Po bližším prozkoumání bylo jasno: Produktový manažer pověřil RnD tým (Research&Developement) jedním úkolem. Tým si i přesto nechal rezervy pro práci na dalších projektech.

Chen Ping na schůzce s týmem pak nakreslila tento diagram příčin a následků, který znázorňuje, kde je zakopaný pes:



Chen Vidíme, že diagram s příčinami a důsledky je užitečný nástroj. Pokud vzniká za spolupráce postiženého týmu, lze díky němu odhalit hlavní příčinu problémů. V tomto případě šlo o začarovaný kruh.

Ten jsem předělal do přehlednější podoby, která vyobrazuje právě onu problematickou smyčku:



Z diagramu je jasné, že se jedná o smyčku opakujících se následků. Všechno začíná rychlostí týmu – je moc pomalý. To má za následek samozřejmě zpoždění projektu. Tím se hromadí požadavky od zákazníka, který je průběžně přidává a mění. Následně se rozšiřuje záběr projektu. Vyvíjený software má nakonec plnit mnohem širší spektrum funkcí. Časová tíseň nedovolí členům týmu získávat nové znalosti, které pak chybějí při vývoji. To působí záseky, které nikdo netuší, jak vyřešit. Takové záseky se pak obcházejí. Není tedy co testovat a předvést zákazníkovi. Klesá kvalita softwaru, objevuje se více chyb a nedostatků. To opět nepříznivě působí na časový fond, což opět vede k nedostatku času k učení.

Technické potíže

Jako další příklad je uveden jiný tým. Ten měl plnou podporu od vedení, i přesto však nešlo vše hladce. Průběžná integrace často končila chybou. Scrum master se rozhodl umístit nad vchod do kanceláře velkou obrazovku, která bude zobrazovat aktuální stav integrace. Zelená barva pro úspěšný výsledek, červená pro neúspěch. V průměru napočítala Chen Ping za den jednu až dvě hodiny zelené obrazovky. To samozřejmě nutilo každého člena týmu neustále sledovat postup projektu. Ustanovilo se pravidlo, že žádný další kód nesmí být přidán k integraci, pokud obrazovka není zelená. Ale nízká úspěšnost integrací vedla k tomu, že vývojáři potají nahrávali nové kusy kódu i přes červenou obrazovku.

Autorka článku varuje před tak obvyklou představou, že vývoj softwaru je velmi odlišný od klasické pásové výroby. Vývoj softwaru je přeci řemeslo spíše intelektuální, na rozdíl od řemeslné práce u výrobní linky. Nicméně jistou podobnost je třeba si připustit. Například onu zmiňovanou výrobní linku lze nalézt ve vývoji softwaru, o to více je to zřetelné právě u

agilních metodik. Software při svém vývoji také prochází jakousi pomyslnou výrobní linkou. Každý vývojář je zpravidla zaměřen na jinou oblast funkcionalit vznikajícího softwaru. Někdy tak může i o odlišné použité technologie. Ale i zde, stejně jako u pásové výroby se celý proces zpomalí, ne-li zastaví, pokud jeden článek výrobního procesu vázne. Nedodělky se pak začnou hromadit. V těchto případech platí pravidlo tzv. úzkého hrdla, tedy že celý tým je pouze tak rychlý, jaká je rychlost nejužšího místa, kterým může být nejpomalejší vývojář.

Pro ilustraci a řešení problému vytvořila Chen Ping tento diagram:

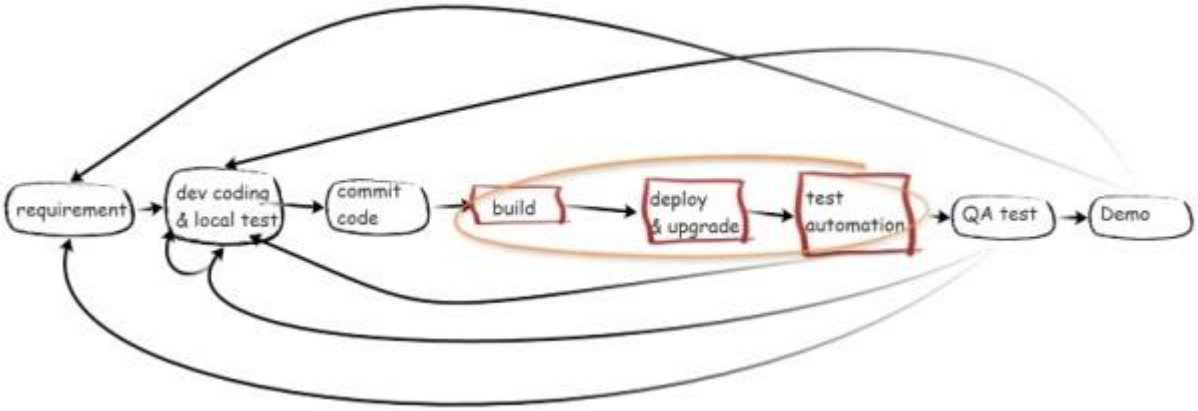
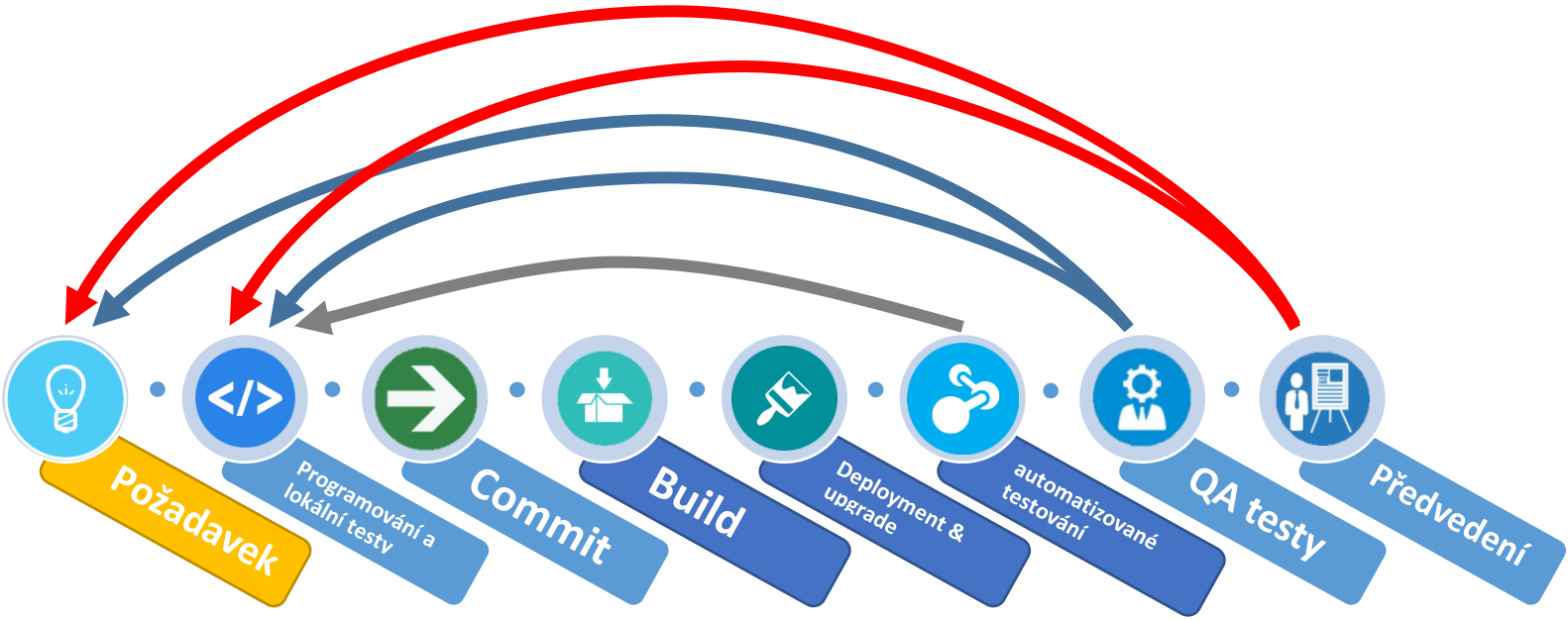


Diagram jsem předělal do české podoby:



Vidíme, že tento proces má mnoho smyček, kdy se práce vrací do dřívějších činností. Zde platí, že čím delší je ona smyčka, tím dražší je náprava chyby, která ji vyvolala. Příkladem může být smyčka „Demo“ – „requirements“ a „Demo“ – „dev coding & local test“. Problém

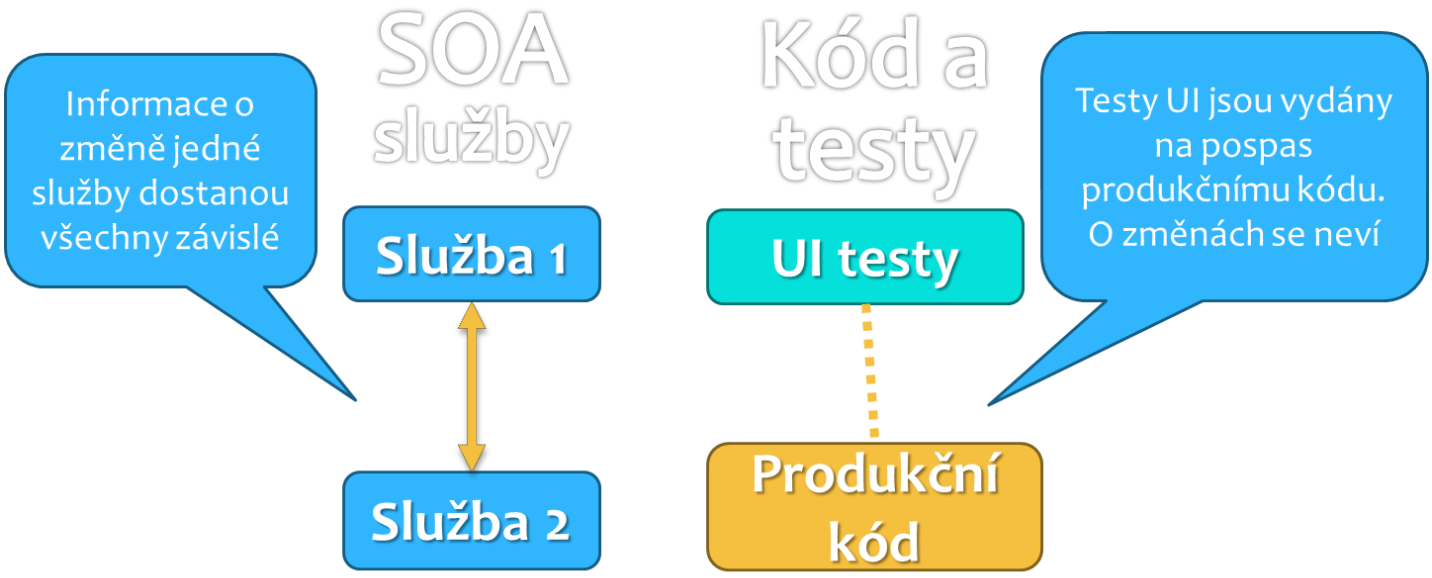
Chen Ping nalezla v činnostech Build-Deploy-Automation. Problém zhoršila i výše popsaná skutečnost, že tým nedodržoval pravidlo červené obrazovky. To mělo za následek více problémů, které ovšem ještě nebyly na řadě.
Tým byl seznámen se zjištěním a poté následovaly návrhy řešení, které by vyladily stávající proces. Mezi ně patřila například důslednější kontrola kódu, nebo přísné dodržování pravidla červené obrazovky. Co se týče automatizovaných testů, řešení už nebylo jednohlasné. Jedna skupina tvrdila, že v současnosti používané testy byly napsány externími pracovníky, kteří jsou již dávno pryč a že takové testování není použitelné, neboť se více času strávilo nad chybami testů, než nad chybami testy zachycenými. Na druhé straně stáli ti, kteří tvrdili, že testy sice špatné jsou, ale jsou lepší než žádné testy a napsání testů nových by způsobilo nepřiměřené časové ztráty nejen pro QA.

Pro vyřešení této neshody bylo opět využito diagramů: Chen Ping pověřila tým, aby vytvořil rychlé propočty významu jednotlivých automatických testů.

Pro představu funkčnosti zavedených procesů pro automatické testování, vytvořila autorka tuto dvojici diagramů.

- Na prvním je znázorněn princip SOA, kde - při optimálně napsaném kódu - se při změně ve funkci jedné služby dozví o této změně i všechny služby závislé. Díky tomu je možné do nich příslušné potřebné úpravy promítnout ihned. Není tak nutné čekat na chyby, které by se objevily bez aplikace změn.

- Na druhé části je znázorněn vztah produkčního kódu a automatizovaných testů grafického prostředí. Pokud se produkční kód změní - například ID nebo jméno prvku – testy o této změně nejsou automaticky informovány. Často se tak případná potřeba změny kódu testů ukáže až po chybě, která nutně nastane.



Ukázalo se, že některé testy byly přijatelné, obzvláště pak některé jednotkové testy a testy komponent. Ostatní testy už tak dobře nedopadly. Závěr byl tedy jasný: Nemělo význam vkládat tolik úsilí do něčeho, co mělo tak nízký přínos. Chen Ping poznamenala, že tvorba a údržba automatizovaných testů nebyla záležitostí jen QA týmu. Testy grafického rozhraní jsou plně závislé na vnitřním kódu. V Service Oriented Architecture (SOA) jsou služby základním stavebním prvkem. Je tedy pravidlem, že takové služby mají perfektně vypracované API. V případě, že je potřeba v jedné službě API změnit, dojde k notifikaci všech služeb na tuto napojených, aby se případná změna promítla i do jejich kódu. Testy grafického prostředí jsou ovšem plně podrobeny vnitřnímu kódu. Pokud se pak například změní ID nebo jméno grafického prvku, testy UI bez potřebných změn nutně skončí chybou.
Chen Ping proto doporučuje vytvořit Framework pro automatické testy, který umožní tvorbu testů silných a přesto flexibilních. Takový framework by fungoval podobně jako zmíněna SOA architektura, ovšem pro testy a produkční kód. Samozřejmě nelze na tuto práci najmou externí pracovníky, stejně tak jako pověřit QA tým, aby vytvořil tuto architekturu pro testování. V námi popisovaném případě totiž QA tým neměl dostatečné znalosti, aby mohl vytvořit tento Framework. Padlo tak rozhodnutí, že se tohoto úkolu chopí vývojářský tým. Spolu s QA týmem pak měli za úkol udržovat UI testy.
Chen Ping uvádí, že tato procesní změna, popsaná na několika řádcích trvala celé tři měsíce a zahrnovala mnoho vyjednávání.

3.3 Obecná doporučení při nasazování agilních metodik

Na základě výše uvedených problémů při implementaci agilních metodik vývoje softwaru a jejich reálných řešení bychom chtěli shrnout některá obecná a užitečná doporučení pro úspěšné nasazení agilní metodiky.

- Rozdělení si úkolů v týmu tak, aby velmi specializované úkoly měli na starosti členové týmu, kteří mají potřebné znalosti a zkušenosti k zpracování úkolu. Pokud necháme vždy členům týmu možnost vybrat si, jakýkoliv úkol chtějí řešit, může nastat problém se zbylými specializovanými úkoly.

- Reagovat na změny požadavků co nejdříve, jelikož čím později budeme na změny požadavků reagovat, tím nákladnější bude změnu do systému zapracovat. S tímto doporučením se váže pravidelná a aktivní komunikace mezi vývojáři, managementem dodavatelské společnosti a zákazníkem.

- Řídit se dle pravidla úzkého hrdla a při plánování vyhodnotit potencionálně úzká místa a odstranit je, případně je ošetřit tak, aby nebrzdila ostatní činnosti projektu.

- Pohlížet na výsledný software jako na celek, který je funkční a je schopný zákazníkovi přinést požadovaný užitek. Pokud bude mít tým naprogramováno 95% úkolů, ale nebude schopný vyřešit zbylých 5%, tak takový produkt zákazník nemůže akceptovat.

- Řídit se dle Paretova principu 80:20. Podle Paretova principu 80% důsledků pramení z 20% příčin. Tým by se měl převážně zaměřit na činnosti, které jsou pro výsledek nejdůležitější.

- Použít diagram příčin a následků, při jejich tvorbě bude mít každý prostor pro jeho nápady. Na základě diagramu následně identifikovat klíčové příčiny, které v týmu způsobují problému a odstranit nebo napravit tyto příčiny.

- Nesnažit se implementovat agilní metodiky za každou cenu. Toto doporučení platí převážně pro týmy, které jsou například zvyklé na rigorózní metodiky, v takovém případě by stálo za uvážení například vnést do týmu pouze některé agilní prvky, pomocích kterých bychom mohli zvýšit produktivitu týmu.

4. Závěr

V této semestrální práci jsme přinesli přehled dostupných agilních metodik spolu s jejich popisem a charakteristikou a zabývali jsme se převážně problémy při přechodu z rigorózních metodik na agilní. Vyvrátili jsme domnění mnoha lidí, že nasazení agilních metodik je jen prostá změna procesů, kterou lze dokončit během několika dní s pomocí dokumentace, a že taková změna nutně vede ke zvýšení efektivity, produktivity a kvality výstupu. Agilní metodiky se totiž nehodí do každého prostředí. V některých situacích je nutné zvážit, zda je společnost, či tým ochotný potřebnou transformaci podstoupit a zda má pro to dostatečné předpoklady. V semestrální práce je uvedeno, že současný módní trend přechodu na agilní metodiky je v mnoha případech kontraproduktivní a ne výjimečně končí potřebou pomoci

od profesionálního kouče agilních metodik. Právě takovým koučem je Chen Ping, která je autorkou článku Why Agile Didn't Work. Tento článek je primárním zdrojem naší semestrální práce. Autorka v něm kombinuje obecnou teorii agilních metodik s praktickými zkušenostmi, kterých nabyla právě jakožto koučka agilních metodik. Jejím úkolem totiž bylo pomáhat týmům, které přechod na agilní metodiky nezvládly. V semestrální práci díky tomu nechybí konkrétní příklady problémů a také postup a řešení, kterých Chen Ping využila.

Z článku jsme také extrahovaly doporučení, která autorka dává k přechodu na agilní metodiky. Ty jsou v závěrečné části přehledně vypsány. K popisu a pochopení dané problematiky jsme použili vlastních verzí náčrtků a diagramů, které jsou ke zmiňovanému článku přiloženy. Doplnily jsme také důkladné vysvětlení, které pomůže k pochopení příčin problémů.

Při vypracování této práce jsme se seznámili se zásadami, které jsou profesionály doporučeny k dodržení při přechodu z klasických rigorózních metodik na metodiky agilní. Díky příkladům z praxe jsme se také seznámili s obvyklými problémy a způsoby jejich řešení i díky vlastnoručním vizualizacím příčin a následků. Tyto poznatky jsme přinesli v pro čtenáře přívětivé podobě.

Citovaná literatura

Beck, B. B. (2001). *Agile Manifesto.* Načteno z Manifest Agilního vývoje software: http://agilemanifesto.org/iso/cs/

Beck, B. B. (2001). *Principy stojící za Agilním Manifestem.* Načteno z Agile Manifesto: http://agilemanifesto.org/iso/cs/principles.html

Buchalcevová, A. (2009). *Metodiky budování informačních systémů.* Praha: Oeconomica.

Cockburn, A. (2008). *Crystal methodologies.* Načteno z Allistair Cockburn: http://alistair.cockburn.us/Crystal+methodologies

Ping, C. (15. 9 2015). *Why Agile Didn't Work.* Načteno z InfoQ: http://www.infoq.com/articles/agile-didnt-work

Scrum Alliance. (2016). *Why Scrum.* Načteno z Scrum Alliance: https://www.scrumalliance.org/why-scrum

Wells, D. (2013). *Extreme Programming: A gentle introduction.* Načteno z Extreme Programming: http://www.extremeprogramming.org/