Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Porovnání DAD a XP – co je v DAD inspirováno XP

SEMESTRÁLNÍ PRÁCE

Předmět:	4IT421 Zlepšování procesů budování IS
Vyučující:	doc. Ing. Alena Buchalcevová, Ph.D.
Termín odevzdaní:	1. ledna 2016
Zpracovatele:	Bc. Kirill Chernov
	Bc. Mykyta Kurdyukov

2016

Obsah

1. Úvod

V oblasti vývoje software jsou v posledních letech na vzestupu tzv. „agilní metodiky“. Jedná se o soubor několika různých metodik, technik a postupů jak vyvíjet programové vybavení počítačů, které nesou společný rys – agilitu, tj. schopnost velmi rychle reagovat na změny požadavků zákazníků, změny prostředí a vývoj technologií, které jsou v oblasti informačních technologií stále častější.

Pro účely zvládnutí některých projektu se modernizují tradiční metodiky jako RUP a přebírají prvky agilních metod. V naši seminární práci pokusíme rozebrat 2 velmi zajímavé metodiky XP a DAD, které pak porovnáme mezi sebou.

2. Cíl páce

Cílem seminární prací je porovnání mezi sebou metod XP a DAD. Pro tento účel budou popsané jednotlivé rysy metod, jejich charakteristiky a specifiky ve kterých tyto metody jsou odlišné. V závěru prací se pokusíme stanovit pro jaké podmínky, které metody jsou nejvhodnější.

3. Přehled framework DAD

Disciplined Agile Delivery je poměrně nová metodika pro postupy a způsoby dodání určitého IT řešení zákazníkovi. Scott W. Ambler na jejím konceptu začal pracovat v roce 2007. Z velké části vychází z již známých agilních metodik (Scrum, Extrémní programování, Agilní modelování a dalších) a staví tedy na základních principech Manifestu agilního programování z roku 2001. Zároveň však tyto principy rozšiřuje, upřesňuje a přidává například popis celého životního cyklu dodávky IT řešení.

„DAD procesní framework adoptuje to, co bychom mohli popsat jako efektivní, riziky řízená a odlehčená verze RUP (Rational Unified Process)“. (AMBLER & LINES, 2012, s. 19)

„Důvodem vzniku DAD byla především absence metodiky, která by umožnila používat agilní přístup zodpovědně i ve větších organizacích a týmech. Následky v případě neúspěchu by tam totiž byly mnohem závažnější než u týmu malého. Dosavadní převahu malých týmů v agilních postupech dokazuje i průzkum “Agile Practices Survey“ z roku 2009, podle kterého je v 68 % agilních týmů pouze 1-10 IT profesionálů.“ (AMBYSOFT, 2009)

„Pokud jde o charakteristiku DAD, dala by se popsat jako hybridní agilní metodika, která klade důraz na jedinečnou úlohu lidí v agilním vývojovém týmu, na význam jejich ochoty a schopnosti se učit, na skutečnou hodnotu vyvíjeného řešení a na životní cyklus dodávky řízený cíli. Dalšími charakteristickými vlastnostmi metodiky je její škálovatelnost a zohledňování okolních podmínek v podniku u každé dodávky IT řešení.“ (Černý, 2014)

3.1. Klíčové principy metodiky DAD

3.1.1. Lidé na prvním místě

DAD považuje lidské zdroje a jejich povahu za hlavní faktor úspěchu (či neúspěchu) každého IT řešení. Věnuje se jednak složení týmu a rolím jeho členů a dále jejich vlastnostem a rozložení jejich schopností.

„Podle DAD by dále všichni členové týmu měli mít komplexní znalosti. Tedy nebýt pouze specialisty ve svém jednom oboru, ale mít i celkový přehled o oborech ostatních, takzvané "cross-skills". Díky tomu mohou totiž pracovat efektivněji a s nižšími náklady - není například nutné dělat podrobnou interní dokumentaci. Zároveň by měli mít maximální možnou snahu nejen o spolupráci a sdílení znalostí s ostatními členy, ale i o získávání a osvojování nových dovedností." (Černý, 2014)

Týmové role, uvádí DAD následující:

- „Stakeholder": Je popsán jako ten, jehož ovlivňuje vyvíjené řešení. Nemusí se tedy jednat vždy jen o koncového zákazníka, může jít například i o partnera, s jehož systémem má ten vyvíjený spolupracovat.
- Vlastník produktu (Product owner): Jedná se o člověka, který zastupuje zákazníka. Jeho hlavním úkolem je vysvětlovat a upřesňovat nejasnosti během vývoje, druhotným úkolem pak představovat práci týmu stakeholderovi.
- Členové týmu (Team members): Lidé, kteří tvoří realizační tým.
- Lídr týmu (Team-leader): Je zodpovědný za efektivní fungování týmu, vytváří mu odpovídající podmínky.
- Architekt (Architecture owner): Dohlíží na architekturu vyvíjeného řešení, používání existujících frameworků, vzorů apod. V malých týmech může být role spojena s Team-leader.

3.1.2. Důraz na učení

„V této souvislosti DAD rozlišuje tři oblasti, a to doménovou (identifikování zákazníkových potřeb), procesní (zlepšování procesů na individuální i týmové úrovni) a technologickou (efektivní využívání existujících nástrojů). Dále DAD nabádá, aby organizace poskytli DAD týmům podmínky pro sdílení znalostí, a to například zřizováním interních diskuzí, znalostních bází a podobně." (Černý, 2014)

3.1.3. Tým v kontextu podniku

Další klíčovou charakteristikou je fakt, že DAD tým je vnímán v kontextu celého podniku. Podstatou je, že nestačí sledovat pouze tým samotný, naopak je třeba vědět i o jeho okolí.

3.1.4. Orientace na cíle

Touto oblastí se DAD asi nejvíce vyčleňuje z existujících agilních metodik. „Na rozdíl od většiny ostatních se totiž nevěnuje pouze konstrukční fázi projektu" (AMBLER & LINES, 2012, s. 11), ale popisuje i fázi počáteční a fázi přechodovou. Autoři DAD upozorňují na fakt, že „sice existují na jedné

straně “těžké” metodiky, jako například „Rational Unified Process“ (RUP) s 5 detailně popsanými fázemi, a na straně druhé agilní metodiky jako Scrum, kde jsou postupy popsány povrchně“ (AMBLER & LINES, 2012, s. 13). Je evidentní, že chybí jakási střední cesta. Tu metodika Disciplined Agile Delivery představuje.

Definovány jsou tři základní fáze životního cyklu:

- Inception Phase (počáteční fáze)
- Construction Phase (konstrukční fáze)
- Transition Phase (přechodová fáze)

„Orientace na cíle je v tomto případě chápána tak, že pro každou z výše uvedených fází jsou přesně definované cíle, na které by se měl tým zaměřit a které by měl splnit pro úspěšný přechod do fáze následující.“ (Černý, 2014)

3.1.5. Aktivní předcházení rizik

Říká se „zneškodněte rizika dříve, než zneškodní Vás“. Toto heslo vystihuje filosofii přístupu metodiky DAD. DAD adaptuje to, co bychom mohli nazvat „riziky a hodnotou řízeným životním cyklem“ („risk & value-driven lifecycle“). Tím je jakousi nadstavbou hodnotou řízeným metodikám, jakými jsou například Scrum či Extrémní programování. (AMBLER & LINES, 2012, s. 19)

„Z hlediska snížení rizik se metodika DAD snaží inspirovat u jiných agilních metodik. Adaptuje například:

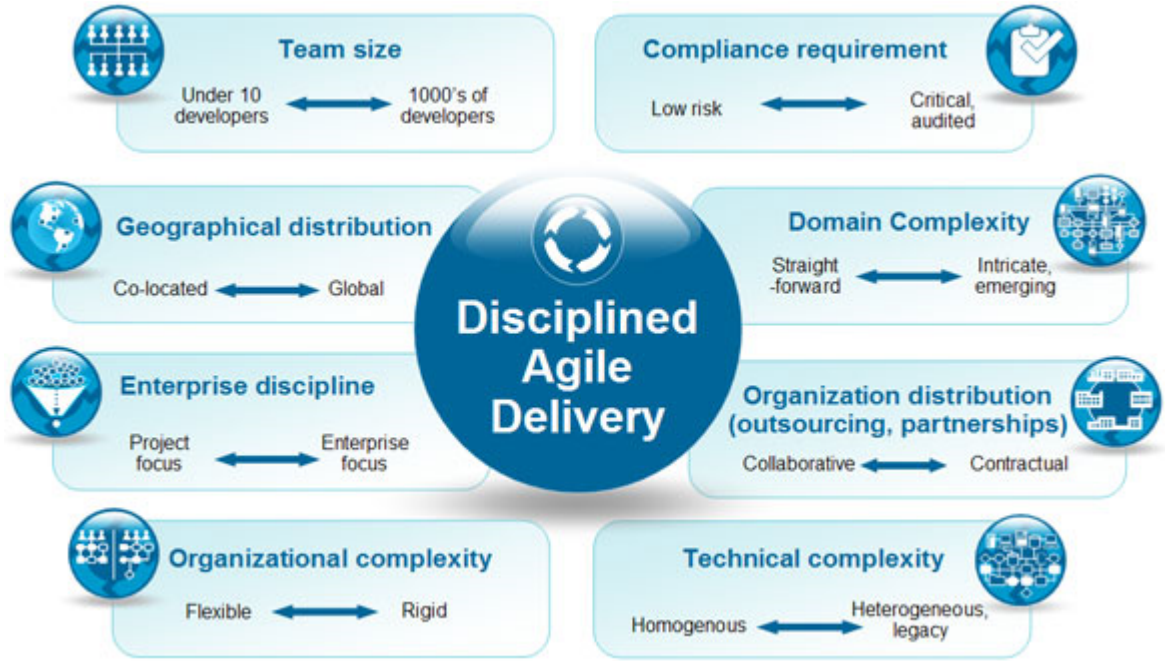
- „Příprava dema“
- Aktivní zapojení Product Ownera a zainteresovaných stran“ (Černý, 2014)

Navíc se rizikům snaží předcházet aplikací některých z následujících opatření:

- Konsensus (nejlépe písemný) z hlediska hodnoty a definice celkové vize projektu mezi všemi zainteresovanými stranami (stakeholdery), jenž má za cíl snížení rizika nedorozumění mezi vizí a naimplementovaným řešením.
- Ověřená architektura. Nejlepší způsob, jak předejít nečekaným technickým problémům, je návrh architektury v průběhu počáteční fáze projektu. Dále je vhodné začít s implementací celkové kostry architektury již v rané fázi Konstrukční fáze.
- Ověření životaschopnosti projektu je sice známá praktika doporučovaná metodikou Scrum, nicméně v praxi se jedná o praktiku zřídkakdy vídanou. DAD apeluje, aby jejich týmy minimálně 2x během životního cyklu projektu kontrolovali životaschopnost projektu, jakožto možnost zastavení prohlubování problémů a zvyšujících se nákladů problémových projektů.“ (AMBLER & LINES, 2012, s. 20)

3.1.6. Škálovatelnost

„Pojmem škálování se běžně rozumí individuální úprava jednotlivých praktik agilních metodik a přidávání nových praktik s ohledem na zvětšený objem objektů v projektu.” (ZIKMUND, 2012) Přestože by se mohlo zdát, že hlavním kritériem škálovatelnosti je velikost týmu, DAD představuje celý soubor faktorů, který je třeba brát úvahu.



Obrázek 1: Jednotlivé faktory skulování metodiky DAD

Dá se předpokládat, že tvůrci DAD hodlají udělat z této vlastnosti opravdu stěžejní přednost a dokázat tak, že jde o metodiku vhodnou pro široké spektrum agilních vývojových týmů.

3.2. Životní cyklus projektu

“Jelikož je DAD Framework metodikou zabývající se dodáváním IT řešení zákazníkovi, je logické, že má specifikované postupy a jednotlivé fáze projektu. Projekt se skládá z několika fází.

DAD respektuje tzv. agilní 3C rytmus „coordinate, collaborate and conclude“, tedy koordinace, spolupráce a ukončení, jenž zasahují do všech 3 úrovní tohoto hybridního frameworku, kterými jsou:

a) Z pohledu fáze životního cyklu projektu (Přípravná, Konstrukční a Přechodová) s cílem dokončení projekt, tedy přípravy „Dodávky“ neboli „Release“.
b) Z pohledu jedné iterace od plánování po demo a následnou retrospektivu.
c) Z pohledu jediného dne, počínaje koordinační schůzkou, končící tvorbou fungujícího „buildu“ na konci dne.

DAD Framework rozděluje proces tvorby řešení na tři základní části, a to: zahájení projektu (počáteční fáze), tvorba řešení (konstrukční fáze) a uvolnění řešení (přechodová fáze), přičemž životní cyklus je inkrementálního typu.“ (Černý, 2014)

3.2.1. Počáteční fáze (Zahájení projektu)

DAD je framework, neboli rámec, nikoliv pak normativní metoda, která by ukládala, jak má jít projekt krok za krokem. Výhodu tedy je, že se dá upravovat a aplikovat na celou řadu projektů. Projektový tým si vždy může vybrat cíle, které se k jejich projektu vztahují a které budou sledovat.

Typické pro fázi zahájení projektu jsou cíle, kterými je projekt řízen. Kromě cílů, které se vážou pro tuto fázi, je nutné také respektovat cíle, které provázejí celý projekt.

Hlavní cíle počáteční fáze: formování týmu, identifikace vize projektu, souhlas stakeholders s vizí projektu, konsolidace s podnikovou vizí, identifikace počáteční technické strategie, požadavků a plánu uvolnění, vytvoření pracovního prostředí, stabilizace financování a identifikovat rizika. (AMBLER & LINES, 2012, s. 275)

Zároveň je nutné respektovat a dodržovat cíle, které nás provází celý projekt.

Průběžné cíle jsou: naplňování poslání projektu, zvyšování dovedností a znalostí členů týmu, obohacení stávající infrastruktury, zlepšování procesů a prostředí, využívání stávající infrastruktury a adresování rizik. (AMBLER & LINES, 2012, s. 275)

3.2.2. Konstrukční fáze (Tvorba řešení)

Konstrukční fáze je postavena na principu ostatních agilních metodik, tudíž vývoj je postaven na iteracích. Důležitý je zde kontakt se zainteresovanými stranami, ideálně aby s nimi tým pracoval při každé iteraci a konzultoval definované požadavky. Tím by měla být zajištěna co nejvyšší přidaná hodnota produktu a minimalizováno riziko.

Hlavním cílem této fáze je vytvořit řešení, které minimálně splňuje, ideálně přesahuje funkční nároky na něj kladené a přidává obchodní hodnotu podniku.

Hlavní cíle konstrukční fáze: produkovat spotřební řešení, reagovat na měnící se požadavky zúčastněných stran, přibližovat se k nasazení schopné „dodávky“, udržovat (ideálně zlepšovat) stávající úroveň kvality a brzy ověřit vhodnost architekturu. (AMBLER & LINES, 2012, s. 275)

3.2.3. Přechodová fáze (Uvolňování řešení)

V této fázi dochází k nasazení řešení zákazníkovi. I tato fáze se může skládat z několika iterací, které by měly být pokud možno co nejkratší. Cíle přechodové fáze: ujištění se o připravenosti řešení k produkci, ujištění se o připravenosti zúčastněných stran k přijmutí řešení, nasazení řešení do produkce. (AMBLER & LINES, 2012, s. 419)

4. Přehled metodiky Extreme programming

Jednou ze základních agilních metodik vývoje software je “eXtrémní programování“ (dále také „XP“). Základní myšlenkou této metodiky je (jak je z názvu patrné) extrémnost. Jednotlivé činnosti, které se osvědčily, či v budoucnu osvědčí jako prospěšné, se provádí „až do extrému“. To znamená, že každá činnost, která se prokáže jako přínosná je stále zlepšována, opakována co nejčastěji a klade se na ni velký důraz.

4.1. Hlavní hodnoty XP

„Metodika eXtrémního programování je založena na následujících 5 hodnotách, jejichž dodržování je klíčové pro úspěch týmu a projektu: [Extremeprogramming.org, 2009]:

a) Komunikace: každý je součástí týmu. Otevřenost jak uvnitř týmu, tak vůči zákazníkovi může pomoci včasně odhalit mnoho problémů a tím zkvalitnit finální výrobek.
b) Jednoduchost: v XP se programuje to, co je požadováno - nic víc, nic míň. Tým se vyhýbá vytváření redundantního kódu. Tato skutečnost vychází ze základní premisy agilního vývoje, že vše se může měnit a co bylo naprogramováno dnes, nemusí být použito zítra.
c) Zpětná vazba: hotová funkcionalita se prezentuje co nejdříve, aby se k ní mohli vyjádřit všichni členové týmu a také zákazník. Včasnou zpětnou vazbou se opět předejde vytváření nadbytečné funkčnosti a zbytečné práci.
d) Odvaha: upřímnost k zákazníkovi (i uvnitř týmu) vyžaduje velkou odvahu. Pomůže však předejít velkým překvapením na konci projektu a zlepší šance na jeho úspěšné zakončení.
e) Respekt: každý si zaslouží jistou dávku respektu, byť se jeho názor může zdát irelevantní. Programátoři respektují zákazníka a jeho uživatelskou zkušenost a naopak zákazník respektuje odborný názor programátorů.“ (Fujdiar, 2014)

4.2. Role

„XP v týmu uvažuje následující role. Některé mohou být obsazené stejným člověkem. Například povinnosti kouče a stopaře většinou plní jeden člověk.

a) Kouč
   - Má na starosti celý tým. Je k dispozici jako vývojový partner, zejména pro nové programátory. Sleduje dlouhodobé cíle refaktorizací. Pomáhá programátorům s jednotlivými technickými dovednostmi jako testování, formátování a refaktorizace. Vysvětluje proces manažerům na vyšších úrovních. Tým by měl řídit spíše jemným nasměrováním než pomocí autoritativních příkazů.
b) Stopař
   - Jeho úkolem je sledovat aktuální metriky a hlídat, zda projekt pokračuje podle plánů a přání. Zároveň zajišťuje, aby tým viděl, jak si na tom právě stojí. Toho dosahuje vyvěšováním různých statistik na tabule v pracovní místnosti.

c) Programátor
    - Je základem týmu, odhaduje pracnost zadání a jednotlivých úkolů během plánovací hry. Psaním unit testů analyzuje požadavky a navrhuje rozhraní vyvíjeného systému. Komunikuje se zákazníkem a ostatními programátory při upřesňování požadavků a jejich řešení. Během párového programování dva programátoři píší testy, zdrojové kódy a integrují změny do systému.

d) Zákazník
    - Rozhoduje, co se má dělat. Během plánovací hry píše karty zadání, následně upřesňuje nejasnosti, a tak se učí psát karty přesněji. V případě komplikací při vývoji rozhoduje, která část funkčnosti má menší prioritu a může být přesunuta do následující iterace a dodána až v další verzi. Zákazník s pomocí testera vytváří testy funkcionality, které slouží jako akceptační testy pro dodaný systém.

e) Tester
    - Většinu testování v XP zajišťují programátoři, role testera pomáhá zákazníkovi vybírat a psát testy funkcionality. Zajišťuje správný chod všech testovacích nástrojů. Má na starosti pravidelné spouštění testů funkcionality a seznámení týmu s výsledky těchto testů. „ (Kotrla, 2005)

f) Konzultant
    - Pomáhá týmu ve specifických problémech, na které tým občas narazí. Problémy však nikdy neřeší sám někde v ústraní, ale vždy s alespoň jedním programátorem, který se tímto od něj učí.

g) Velký šéf
    - Zodpovídá za vývojový proces jako celek. Zajišťuje nové lidi, když jsou potřeba. Neměl by se týmu plést do cesty, pokud nemá podezření, že se tým ubírá špatným směrem. V takovém případě je na týmu, aby svoje jednání obhájilo. Když to tým nedokáže, směřoval vývoj nejspíše špatným směrem a velký šéf splnil svou povinnost, když nenechal tým dál pokračovat, ale naopak ho přiměl k zastavení a zamyšlení.

4.3. Postupy

„XP je založeno na dvanácti postupech, které jsou stručně popsány v této kapitole, více informací lze najít v knize Extrémní programování. Tyto postupy použité samostatně mohou způsobit projektu vážné problémy, ale dohromady tvoří silný celek, který urychluje vývoj software.“ (Kotrla, 2005)

a) Plánovací hra
    - Plánovací hra slouží v XP k řízení projektu. Zákazník jejím prostřednictvím rozhoduje o šíři celkového zadání, stanovuje priority požadavků, určuje, jaké funkčnosti spolu souvisejí a mají se tak objevit společně v další verzi uvolněného software, podle svých záměrů (například účast na veletrhu) upravuje datum, kdy se mají jednotlivé verze uvést do provozu. Programátoři během plánovací hry odhadují potřebný čas na vývoj jednotlivých požadavků, radí zákazníkovi při technických rozhodnutích (například volba databáze) a vytvářejí podrobný harmonogram právě začínající iterace.

b) Malé verze
    - Pro zákazníka je důležité, aby novou funkčnost mohl začít používat co nejdřív, proto XP dodává malé verze a v krátkých časových intervalech v řádu měsíce nebo dvou, v případě rozsáhlejšího systému maximálně po půl roce.

c) Metafora
    - Metafora zajišťuje komunikaci mezi vývojovým týmem a zákazníkem. Je společným „příběhem“, kterým popisuje zákazník provádění stávajících procesů ve své společnosti, a programátoři mluví o chování vyvíjeného software. V XP metafora zastupuje význam architektury.

d) Jednoduchý návrh
    - Vyvíjený software má neustále nejjednodušší možnou podobu, která pokrývá aktuální požadavky zákazníka. Nic se nevyvíjí dříve, než je to potřeba, protože se může stát, že to nebude potřeba nikdy.

e) Testování
    - Programátoři pomocí testů jednotek ověřují svůj zdrojový kód a udržují si tak víru ve své schopnosti a důvěru k dříve vyvinuté části systému. Zákazník z výsledků testů funkcionality pozná, jaká část systému je již úspěšně vyvinutá a kolik funkčnosti ještě chybí.

f) Refaktorizace
    - Programátoři pomocí refaktorizace udržují zdrojové kódy, tak aby neobsahovaly duplicity, aby šlo lehce přidat novou funkčnost, aby zvýšili čitelnost zdrojového kódu, aby zlepšili návrh aplikace atd. Při této činnosti jim pomáhají hotové testy, které jim dávají jistotu, že se chování systému nezměnilo. Důležité je, aby programátoři rozlišovali, kdy refaktorují a kdy vyvíjí novou funkčnost, a nikdy tyto dvě činnosti neprováděli současně.

g) Párové programování
    - Při XP se zdrojový kód píše v páru, jeden programátor píše nový test, kód nebo refaktoruje, druhý mu přitom radí a hlídá, jestli výsledek činnosti prvního odpovídá celkové strategii. První programátor při své práci uvažuje jen o části systému, kterou mění či doplňuje. Důsledky pro systém jako celek musí mít na mysli druhý, v případě potřeby musí prvního opravit. Páry se mění většinou dvakrát za den.

h) Společné vlastnictví
    - Všichni členové týmu mají stejnou odpovědnost za celý vyvíjený systém, proto pokud objeví někde chybu nebo příležitost ke zlepšení, mají pravomoc (dokonce povinnost) chybu opravit nebo zlepšení realizovat sami, namísto, aby upozornily odpovědnou osobu a cekali, až tato osoba bude mít čas.

i) Nepřetržitá integrace
    - Procesy testování a integrace se opakují v XP během velmi krátkého času, vždy alespoň jednou za den. Pro integraci je vyhrazen přímo jeden počítač, ke kterému si dvojice sednou, když dokončí implementaci svého úkolu. Na tomto vyhrazeném počítači pak provedou integraci s již vyvinutým systémem, to znamená, vyřeší všechny případné kolize, provedou překlad zdrojových kódů a následně spustí testy. Pokud testy neprojdou na sto procent, musí chyby odstranit nebo svůj kód zahodit a začít znovu implementovat.

j) 40hodinový týden
    - Přesčasy jsou známkou problémů při vývoji software. Unavení programátoři nemůžou vyvíjet rychle a kvalitně, proto XP zakazuje přesčasy v dvou následujících týdnech. Pokud je potřeba, aby byl tým v práci dlouhodobě přesčas, je nezbytné rychle identifikovat a odstranit problém, který to způsobil. To může nakonec samozřejmě způsobit i přeplánování zbytku iterace.

k) Zákazník na pracovišti
    - Všechny agilní metodiky mají společný požadavek, aby byl zákazník kdykoliv k dispozici pro upřesnění nejasností v zadání a poskytnutí konzultací z oboru podnikání zákazníka. Nejvhodnější je, pokud je zákazník v jedné místnosti s týmem a je tak možná rychlá a přímá komunikace. Zákazník sice musí být k dispozici kdykoliv, těžko ho však bude tým neustále zahlcovat dotazy, a tak se může zákazník po většinu času věnovat vlastní práci.

l) Standardy pro psaní zdrojového textu
    - Standardy jsou nutné kvůli společnému vlastnictví kódu, všichni programátoři musí znát a dodržovat standardy, které si celý tým vzal za své, aby se snadno a rychle vyznali v kódu, který psal někdo jiný.

4.4. Životní cyklus

„Metodika XP se přizpůsobuje podmínkám projektu a potřebám týmu, životní cyklus se proto bude projekt od projektu lišit. V této části práce je popsáno, jak probíhá ideální životní cyklus.

a) Průzkum (Exploration)
    - Každý projekt začíná průzkumem. Během této fáze si programátoři zkouší navrhnout a částečně realizovat různé architektury plánovaného systému. Ze svých pokusů pak vyberou nejvhodnější architekturu. Zároveň se učí odhadovat, kolik casu jim zaberou jednotlivé úkoly, porovnáním odhadovaného a skutečně potřebného času.
    - Pokud úspěšné zvládnutí začínajícího projektu vyžaduje po programátorech znalosti nové technologie, seznamují se s touto technologií právě v této fázi. Zároveň tato fáze slouží k ověření výkonnostních limitů požadovaného systému. Ověření je prováděno pomocí vhodných prototypů.
    - Ve stejné době se zákazník učí psát své požadavky v podobě karet zadání, aby byly srozumitelné pro programátory. V této etapě by měl sepsat požadavky na první verzi systému. Délka etapy je závislá na znalostech a zkušenostech programátorů s danými nástroji, technologiemi a oborem podnikání zákazníka, pohybuje s v rozmezí několika týdnů až měsíců, přičemž by neměla přesáhnout půl roku.

b) Plánování
    - Etapa plánování navazuje na etapu průzkumu. Pokud se během průzkumu zákazník i tým dostatečně připraví, pak samotná etapa plánování trvá jeden či dva dny, během nichž se vyberou karty zadání pro první verzi systému, a dohodne se termín jejího uvolnění. Tento termín bývá v období následujících dvou až šesti měsíců. Pokud je tým schopen první verzi dodat dřív, je to jen dobře, ale většinou se to nestává. Naopak doba delší než šest měsíců znamená zvýšení rizika.

c) Iterace do uvolnění první verze
    - Iterace, které probíhají do uvolnění první verze, trvají jeden až čtyři týdny. Pro první iteraci se vybírají taková zadání, aby pokryla celý systém, tedy aby výsledkem první iterace byla kostra systému. Která zadání jdou do dalších iterací rozhoduje zákazník na základě svých priorit. Na konci každé iterace by měla být malá oslava dodání plánované funkcionality. Výsledkem poslední iterace je pak první verze software určená k nasazení do ostrého provozu.
d) Zprovozňování
    - Tato etapa slouží k nasazení první verze systému k zákazníkovi. Nejprve je nutné systém důkladné otestovat, případně pomocí paralelního testování nového a starého systému. Zároveň se v této době tým pokouší o vyladění výkonu, protože má už dostatek znalostí o systému a zároveň má k dispozici provozní hardware.
    - Během této etapy většinou dochází ke zkrácení iterace ze tří týdnů na jeden. Zároveň dochází i ke zpomalení vývoje, protože veškeré změny musejí být lépe promyšleny, neboť systém je už nasazen do ostrého provozu. Nové myšlenky tykající se implementace se sepíší a odloží do následujících iterací. Nasazení systému je vhodné uzavřít menší oslavou, jde totiž o významný okamžik celého projektu.
e) Údržba
    - Po nasazení systému do ostrého provozu dochází ke zpomalení vývoje. Jedním z důvodů je, že programátoři musejí vedle implementace nových úkolů zároveň zajišťovat servis běžící aplikace. Proto je vhodné v tento okamžik přibrat nové programátory do týmu. Jejich rychlé zaškolení je realizováno pomocí párového programování se zkušenějším kolegou.
    - Vývoj dalších verzí začíná vždy průzkumem, v rámci něhož se můžeme pokusit o odvážnou změnu architektury nebo muže zákazník přijít s novým přelomovým zadáním. Tým však musí být více opatrný, protože pracuje s ostrými daty a běžícím systémem. Zároveň se musí snažit o co nejčastější slučování nového zdrojového kódu s provozním, aby se později vyhnul velkým integračním problémům.
    - Tým muže předem odhadovat, jak jeho efektivitu sníží souběžný servis běžícího systému. Důležité je tyto odhady během údržby potvrdit nebo vyvrátit pomocí měření. Stejně tak je vhodné programátory poskytující servis postupem času prostřídat. Během servisu se mohou naučit mnoho důležitých věcí, na druhou stranu jim jejich práce nepřijde tak zábavná, jako při implementaci nových úkolů.
f) Smrt
    - Smrt znamená konec vývoje systému. Ten muže nastat ze dvou důvodů. Méně častým důvodem je, že zákazník nemá už žádné nové zadání, které by chtěl realizovat a systém je dokonale vyladěný, tudíž vývojový tým nemá, co by víc udělal. Častěji je však ukončení vývoje způsobeno neschopností vývojového týmu přidat efektivně novou funkcionalitu.
    - Během této etapy se sepisuje několikastránkový dokument o vyvinutém systému, který by měl týmu pomoci, kdyby se k systému po čase vrátil. Taktéž je vhodné prozkoumat příčiny, které vedly k ukončení vývoje, a vzít si z nich poučení do dalších projektů.“ (Kotrla, 2005)

5. Porovnání DAD a XP

5.1. Životní cyklus

DAD adoptuje tak zvaný životní cyklus řízený rizikem a hodnotou, od běžných metod jako Scrum nebo XP (které používají životní cyklus řízený pouze hodnotou). Pomoci životního cyklu řízeného hodnotou v každé iterace se vyvíjí pružný hotový software, který případně můžeme poslat zákazníkovi. Dodávaná funkčnost představuje to, co bylo požadováno v backlogech, jako nejvyšší hodnota z pohledu zákazníka. V případě, že používáme rozšíření životního cyklu řízeného hodnotou o riziko, hned musíme uvazovat rizikovost jako nejvyšší prioritu. V tomto případě, musíme hned informovat projektového manažera o těchto rizicích, aby on byl přesvěcen včas. Životní cyklus řízeny hodnotou může upozornit především o třech důležitějších rizicích, a sice:

I. riziko nedoručení projektu vůbec,
II. riziko doručení nesprávné funkcionality,
III. politické riziko plynoucí z nedostatku viditelnosti produktu z vnějška tymu.

Včas informovány projekt o těchto rizicích je výborným začátkem, avšak není úplným snížením rizikovosti projektu. (Ambler & Lines, 2012, str. 19)

5.2. Stakeholder vs. Customer

Co se tedy skrývá pod pojmem „Stakeholder“ (česky: „Zainteresovaný subjekt“) na rozdíl od pojmu „Customer“ (česky: „Zákazník“)? Ačkoli termín „Zákazník“ je výborným terminem a velice používaným u příznivců XP, bylo zjištěno, že mnohé agilní tymy neúmyslné omezuji pojem „Zákazník“ na jenom konečného uživatele, přičemž v této interpretaci chybí partnery a ostatní zainteresované strany uvnitř podniku. Ze zkušenosti vyplývá to, že i když pojem „Stakeholder“ je trošku formálnější než „Zákazník“, při jeho použiti vedoucí agilních tymů vybírají vice vyspělé strategie prací.

Jeden úkol spolu se Stakeholderem, který je ve skutečnosti zákazníkem, posiluje se „jejich vs. naše“ způsob myšleni převládající v mnohých IT odděleních. Když se používá pojmy „Stakeholders“, nebo „Zákaznici“, nebo „business“, oni nesou s sebou smysl, se kterým cítíme sebe samostatnou skupinkou, jinými slovy necítíme, že „my“ a náš zákazník, jsme jedním celkem. Ale dnes, skutečnost je taková, že není „byznys“, ale je „náš byznys“. Musíme působit společně. Takže je třeba velice pečlivě volit terminy a pojmy, které se budou používat uvnitř tymu. (Ambler & Lines, 2012, str. 67)

5.3. Práce ve dvojicích

XP disponuje praktikou "programovaní ve dvojicích", která navrhuje, že by developeři měli pracovat spolu a sdílet jednu klávesnici během kódovaní. Tohle je typ extrémní revizi designu či kódu, když v reálném čase, jeden člověk kouká na kód druhého. Klíčový přinos je v lepší kvalitě, protože defekty jsou indikované před tím, než jsou naprogramované. Další přínosy zahrnuji, lepši dokumentovaný kód a design, které jsou lepe pochopitelné.

I když mnoho skeptiků říkali, že použiti dvou programátoru pro psaní jednoho kódu je luxusem, který firmy nemohou dovolit, studii ukázaly, že párové programovaní přináší vetší produktivitu, než individuální programovaní, když je vidět celkový vliv.

V DAD myšlenka párového programovaní je rozšířena na všechny tymy, nejenom na programátory. Je běžné, že analytici pracuji spolu a modeluji na tabuli, nebo testeři vytvářejí dvojici s byznys experty, aby ověřili řešeni. Dvě hlavy je lepe než jedna. (Ambler & Lines, 2012, str. 343)

5.4. Používaní metodik

Na stránkách Scotta Ambera, který je autorem metodiky DAD, je uvedené porovnání různých kategorií projektů a jsou k nim přiřazené jednotlivé metodiky. XP a DAD jsou uvedené jako vhodné pro projekty kategorii Integraci/výměny a vývoje nových aplikaci. Podle tohoto faktu můžeme říčí, že obě metodiky jsou vhodné pro stejné účely, protože jak již bylo uvedené DAD převzala hodně z Extrémního programováni. Jedině co by mohlo vypadat, jako zvláštní je to, že pan Amber uvedl jenom metodiku XP jako vhodnou pro projekty, které jsou kritické z pohledu bezpečí. K takovým projektům patří například systém pro řízení letového provozu.

Project Category	**Description**	**Suggested Methods**
Integrace / výměna	Integrace dvou či vice exitující systémů, nebo získaní přístupu do jednoho či vice systémů, anebo běžná výměna jednoho systému za jiný.	DAD ISO/IEC 12207 XP
Vývoj nových aplikaci; Component/object technologies (e.g. Java)	Vývoj nové aplikaci či systému.	DAD XP
Kritický pro bezpečí (Safety Critical)	Systém působící na zdravotní stav nebo bezpečnost lidí. Například jako, systém řízení letového provozu, analýza dat u lékařských výzkumů a systémy monitorování srdce.	ISO/IEC 12207 XP

Tabulka 1: Porovnání použití metod DAD a XP
Zdroj: http://www.agiledata.org/essays/differentStrategies.html

6. Závěr

Agilní metodiky zaujímají větší a větší podíl na celkovém objemu uskutečněných projektů. Cení se pružnost a přizpůsobivost těchto metodik ke konkrétní situaci nebo zákazníkovi. V naší práci jsme udělali pohled na 2 velmi známé agilní metodiky jako DAD a extrémní Programování (XP).

XP zahrnuje v sobě hodně zajímavých praktik, jako například práce ve dvojicích. DAD přebírá tyto praktiky a dává jim větší zralost a komplexnost z pohledu projektu. Tohle zahrnuje lepší dokumentaci k projektu a přenesení nejlepších praktik z jedné činnosti nebo odděleni na celý projekt, aby praktiky převzaté do DAD z jiných agilních metodik byly využité co nejefektivněji. Takový přístup k tvorbě a vývoji metodiky DAD dává velké perspektivy pro její dlouhý a úspěšný život, protože autoři DAD využili princip evoluci nejlepšího, co dosud existuje. Extrémní programováni je stále také používáno pro projekty a její postup je již docela známý pro firmy. Ale v této metodice chybí komplexní přístup k tvorbě projektu, který je důležitou a někdy nutnou podmínkou pro velký projekt.

Navíc jsme, spolu s kolegy, kteří porovnávali DAD se SCRUMem a RUPem, připravili velmi jednoduchou tabulku, ve které jsou tyto všichni čtyři agilní metodiky. Tato tabulka je přílohou této práce.

7. Seznam zdrojů

Ambler, S., & Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile.* USA: IBM.

Beck, K. (1999). *Extreme Programming Explained.* Addison Wisley.

Beck, K. (2000). *Planning Extreme Programming.* Addison Wisley.

Klepetko, B. R. (2011). *Disciplined Agile Delivery (DAD) framework.* Praha: Vysoká škola ekonomická.

Nagler, R. (2012). *Extreme Programming in Perl.*

Šedivec, T. (2013). *Disciplined Agile Delivery (DAD) Framework: Koncept a současný stav.* Praha: Vysoká škola ekonomická v Praze.

Wake, W. C. (2000). *Extreme Programming Explored.*