Možnosti využití agilních metodik při provozu a údržbě SW

Seminární práce k předmětu

4IT421 Zlepšování procesů budování IS

xparl10
Lukáš Parma

OBSAH

1. Úvod

Tato práce je zaměřena na agilní provoz a údržbu informačních systémů a IT služeb z pohledu softwarů a aplikací. Zaměřím se nejen na oblast správy infrastruktury IS, ale i na způsob práce na úrovni týmů spojené s potřebou komunikace mezi vývojem a provozem.

O agilních metodikách vývoje najdeme spousty zdrojů. O problematice týkající se konkrétně agilního provozu a údržby se však mnoho nedočteme. Proto budu ve své práci uvádět především principy a praktiky, které sepsal Jaroslav Procházka ve své knize „Provozujte IT jinak“ (1)

2. Definice základních pojmů

Nejprve definuji základní pojmy, o které se v práci budu opírat.

Informační systém podle Laudona lze chápat jako (1):

„Sociálně-technický systém, ve kterém lidé, technologie, podnikové procesy a organizace na sebe navzájem působí ve snaze sbírat, zpracovávat, archivovat a distribuovat informace s cílem podpořit řízení, koordinaci a rozhodování v organizaci.“

IT systém:

„IT systémem rozumíme obecný systém využívající IT infrastrukturu, základní či aplikační software k jistému účelu, kterým nemusí být nutně podnikání. Tento pojem zahrnuje různé typy aplikací a také formy jeho provozu.“

IT služba definovaná podle ITIL® (2):

„Služba je prostředek dodávání hodnoty zákazníkovi tím, že zprostředkovává výstupy, jichž chce zákazník dosáhnout, aniž by vlastnil specifické náklady a rizika.“

Údržbu lze definovat podle IEEE následovně (3):

„Údržba SW je procesem modifikace softwarového systému či komponenty po doručení uživateli za účelem korekce chyby, zlepšení výkonu či jiného atributu nebo adaptace systému změnám prostředí.“

Rozdělit jí můžeme do tří základních kategorií:

- Korektivní údržba – odstraňuje nalezenou chybu .
- Adaptivní údržba – změna funkčnosti SW z důvodů změny prostředí.
- Zlepšující údržba – zlepšování výkonnosti.

Nakonec vysvětlím nejdůležitější pojem: **Agilní přístup** k vývoji, provozu a údržbě SW (4).

Jde o přístup, který má na začátku projektu fixně daný čas a zdroje, tzn. lidské zdroje, vývojová prostředí a jiné. Ovšem funkcionalita je proměnná. Výhodou oproti tradičnímu (rigoróznímu) přístupu je především to, že zákazník dostane produkt včas s proměnnou funkcionalitou. Může se tedy stát, že produkt bude mít jen částečnou funkcionalitu, ovšem díky agilnímu pojetí se snadno upravuje a vylepšuje i za provozu. Pro zákazníka je mnohdy lepší, aby měl částečně funkční produkt v daný čas, nežli kompletní produkt s časovým přesahem a

s tím spojenými zvýšenými náklady na další pokračování vývoje. Zákazník si předem určí priority jednotlivých funkcí, takže jsou případné nedostatky akceptovatelné. Obecně se agilní přístup snaží co nejrychleji navrhnout systém, předložit zákazníkovi a podle zpětných vazeb upravovat.

Z manifestu agilního programování vycházejí pro agilní metodiky základní principy (5):

- Lidé a komunikace jsou důležitější než procesy a nástroje.
- Fungující software je důležitější, lepší než stohy dokumentace.
- Zákazník by měl úzce spolupracovat s vývojovým týmem.
- Odpověď na změnu, její naplnění je podstatnější než (slepé) následování plánu.

Agilní manifest byl vytvořen na popud uznávaných odborníků z IT, kteří chtěli přijít s alternativním přístupem k těm tradičním, které nebyly zcela úspěšné. Pro potřebu této práce definuji kromě základních principů i další přiložená:

- Plnit přání zákazníka.
- Změny požadavků od zákazníků jsou vítány (konkurenční výhoda pro zákazníka).
- Časté dodávky fungujícího SW zákazníkovi.
- Každodenní spolupráce vývojářů s koncovými uživateli pro lepší pochopení jejich požadavků.
- Osobní komunikace zamezující chybné interpretaci.
- Měřítkem projektu je fungující SW, ne dokumentace, modely, či různé specifikace.

Po pravidelných dodávkách funkčností se následně opravují nalezené chyby. Nevytváří se tak nesystematické opravy generující další problémy. Zásadním znakem agilního přístupu je, že vše dělá jeden tým. Z toho tedy vyplívá, že je v podstatě agilní provoz a údržba součástí životního cyklu agilních projektů.

Agilní komunita se již delší dobu zabývá aplikací agilních principů do oblasti provozu a údržby SW. Tyto praktiky můžeme rozdělit na dvě oblasti:

- Využití agilních praktik v již existujících přístupech – např. Scrum, Extrémní programování, Lean Software Development či Open Source agilní procesy.
- Využití agilních praktik v oblasti údržby a provozu bez ohledu na metodu.

3. Základní problémy provozu a údržby IT systémů

V současné době je stále zvyšující se trend používání informačních systémů ve formě IT služeb, přístupu k provozu ve formě Application Service Providing (ASP), či ve formě software jako služba (Software as a Service – SaaS). Podniky se tak zbavují starostí s technickou a provozní stránkou informačních systémů (dále jen IS), či IT služeb.

Provoz IS trvá mnohokrát déle než samotný vývoj IS. Některé systémy mohou fungovat i několik desítek let. Stává se tedy, že systémy jsou zastaralé a musí se vydávat velké náklady na provoz a údržbu. Veškeré rozšiřování těchto systémů se stává složitějším, jelikož se musí počítat s propojením všech vazeb a s testováním každé změny vůči celému systému. I proto v současné době stále rostou náklady na provoz a údržbu IT systémů a snižují se náklady na inovace.

Z jednoduchého upgradu hardwaru se může stát také velký problém. V izolovaném prostředí je tato změna triviální, ovšem pokud by šlo o složitější propojenou infrastrukturu, bude tato změna vyžadovat další opatření např. změnu middleware, úpravu databází, či upgrade aplikačních serverů. Tomuto jevu se říká tzv. dominový efekt.

Dalším důležitým problémem při provozu a údržbě IT systémů bývají bohužel lidé a jejich komunikace. Neochota jedinců k týmové práci, nedostatečná komunikace s vývojáři, nesdílení znalostí a řešení může vést k mnoha zbytečným problémům, např. nezakomponování funkčních priorit zákazníka, či přetížení jednotlivých členů týmu, což může vést k neefektivitě práce. Toto jsou tzv. měkké aspekty provozu a údržby, o kterých se více dozvíte v práci mého kolegy Vlastimila Kouřimského.

Tyto problémy vznikají mezi třemi základními skupinami lidí v organizaci:

- Koncoví uživatelé – využívají systémů ke svému užitku, komunikují s vývojáři a předávají jim svá stanoviska.
- Správci IT systémů – starají se o správný chod IT infrastruktury, mohou být v organizaci nebo součástí externí strany (tzv. outsourcing), komunikují s vývojáři, aby dosáhli dostatečné znalosti produktu, výkonu, možnosti rozšiřitelnosti a udržovatelnosti
- Vývojáři IT systémů – navrhují a implementují požadavky koncových uživatelů, mohou být v organizaci, součástí externí strany či může být vyvinutý produkt zakoupen.

Každý z nich mluví svým jazykem, a proto se často stává, že dojde ke špatnému pochopení na všech stranách. Agilní princip chce tyto nedostatky odstranit především dostatečnou komunikací mezi těmito skupinami a zapojením všech rolí do projektu. Charakteristické jsou časté prezentace spustitelných verzí, které budou prezentovány různým rolím v jejich jazyce.

4. Principy agilního a štíhlého provozu a údržby

V tradičních metodikách a standardech najdeme spoustu principů jak provozovat a udržovat IS. Disponují značnými dokumentacemi, jak se má přesně postupovat a vše je podrobně popsané. Předpokládají, že výstup bude stejný nezávisle na zaměstnanci, jelikož má každý stejnou dokumentaci, podle které se musí řídit. Opomíjejí tak na lidský aspekt a na jeho tvůrčí prosazení. V této kapitole se zaměřím právě na tuto stránku věci a popíši základní principy agilního provozu a údržby bez zbytečné byrokracie.

4.1. Základní principy agilního provozu a údržby

Čtyři základní principy, které bez velkých investic pomohou posunout provoz a údržbu IT služeb/IS na vyšší kvalitní stupeň (1):

1. Více disciplíny, méně byrokracie.
2. Spolupráce všech zainteresovaných stran a rolí (uvnitř týmů, mezi týmy, viditelnost) a systémový pohled.
3. Proaktivita (analýza historických dat, predikce trendů, technologické simulace scénářů budoucího vývoje organizace) a učení.
4. Riziky řízený přístup (rizikové, neznámé věci provádíme nejdříve, tzv. koncept „make your hands dirty“).

4.1.1. Více disciplíny, méně byrokracie

Většina organizací má pevně daná pravidla, podle kterých se zaměstnanci musí striktně řídit - jak mají řešit incidenty, zpracovávat změny, jak testovat a integrovat do produkčního prostředí. Ne každý zaměstnanec však tyto pravidla dodržuje z různých důvodů – je líný nebo se nepodílel na vzniku pravidel a má lepší nápad jak to řešit, ale nikdo ho neposlouchá – forma revolty. Výsledkem jsou další a další pravidla, která jsou vydána na popud nedodržení těch původních. Zaměstnanci jsou tak zahlceni zbytečnou byrokracií a nemají čas na svou vlastní práci, jelikož musí vyplňovat množství nesmyslných dokumentů.

Dalším cílem tohoto principu je mít motivované zaměstnance, či týmy. Fungující týmy jsou složené z vhodně

zvolených jednotlivců, kteří mají možnost prosadit se a kreativně pracovat. Pokud přenecháme zaměstnancům více rozhodovacích pravomocí, budou se cítit více oddáni přidělenému úkolu. Navíc se budou chtít zasloužit o pochvalu, a proto na sebe budou brát více odpovědnosti, aby dokázali, že byl projekt pod jejich dohledem.

Členové týmu mohou ostatním přispívat svými poznatky o různých omezeních a možnostech zlepšení. Ideální nastavení organizace je proto takové, kde každý zaměstnanec (i na posledním hierarchickém postu) může navrhovat změny a vylepšení.

Provádění retrospektiv je silným nástrojem v pracovním prostředí. Ohlédnutím zpět můžeme zjistit, co funguje a co ne. Samozřejmě nejde jen o ohlédnutí zpět, ale i následné řešení a navrhnutí změn. Změny se však musí provést.

Častým problémem bývá nejasná zodpovědnost za určitý proces a činnosti. Stává se to především předáváním zodpovědnosti manažerů týmu na další manažery, kteří to posílají dále. Pokud nastane problém, je jen těžko dohledatelný zodpovědný pracovník. Řešením je tedy plochá hierarchická struktura, která přenáší operativní i taktická rozhodnutí na tým. Rozhodování a pravomoc je tedy zcela na nejlepším svědomí celého týmu a nerozhoduje o tom žádné vedení. V dnešní době, kdy se rychle mění podmínky, je zapotřebí flexibilnějšího a efektivnějšího přístupu, a proto se ustupuje od hierarchické organizační struktury a centrálního řízení.

4.1.2. Spolupráce všech stran a systémový pohled

Jak již bylo řečeno, komunikace mezi koncovými uživateli, vývojáři a údržby je složitá díky různým používaným jazykům. Tuto propast je nutné odstranit. Článkem pro překonání této propasti je sdílení stejných společenských cílů, plánů organizace a mapování cílů na IT služby či projekty.

Jednotlivým týmům je důležité sdělovat cíle a plány celého projektu. Týmy pak mohou rozhodovat ve shodě s celopodnikovými cíli a ví, že jdou správným směrem. Pokud vhodně definujeme cíle na úrovni týmu/ projektu/ služby, každý uvidí svůj přínos, kterým může přispět do projektu a tím splnit společný podnikový cíl.

IT je v podniku často chápáno jako náklad, jelikož není zcela jasně vidět přínos a nejde ho spočítat. Propojením cílů podnikání s cíli IT však můžeme dosáhnout měřitelnosti těchto přínosů. Podnikové cíle si rozdělíme na operativní cíle, po jejichž naplnění dojde i ke splnění cílů podniku. Existují sady praktik, které naplnění operativních cílů umožní. Operativní cíle rozdělujeme do čtyř základních skupin:

- Zvýšená produktivita, efektivita (effort) – více užitné IT služby, eliminace částí, které nepřináší hodnotu.
- Zlepšení kvality (quality) – snížení defektů, zajištění správnosti, prevence problémů.
- Velikost, komplexnost (size, complexity) – zvýšená inovace.
- Termín, plán, rychlost dodávky (Time-to-Market, Time-to-Value) – zkrátit dobu od počátku k přínosům.

Tyto operativní cíle se dají naplnit různými praktikami a technikami v závislosti na typu organizací či trhů.

Postup jak definovat, mapovat, implementovat a měřit podnikové cíle ve spojení s IT:

1. Definice očekávaných podnikových cílů.
2. Určení cesty jak se k cílům dostat.
3. Provádění kroků cesty.
4. Měření a přizpůsobení.

Pro lepší provázanost se doporučuje týmy vývoje a údržby přidělit do týmu zákazníka. Pravidelná komunikace přispívá k lepšímu pochopení potřeb zákazníka a vytváří se tak vhodnější a použitelnější IT služby. Jak už bylo zmíněno, pro úspěšnější projekt jsou důležité pravidelné demonstrace aplikací, při kterých zákazník zkouší experimentovat, zkoušet opravené či implementované scénáře a komentovat je.

Při komunikaci v týmu bychom měli pamatovat na respektování druhého, jeho odlišností a cílů. V diskusi dejme stejný prostor pro vysvětlení svého pohledu i dalším, snažme se nalézt kompromis a společný cíl. Při řešení konfliktů pomáhá mít s sebou mediátora (třetí stranu), který drží směr diskuse a má nadhled. Otevřená a upřímná komunikace předchází problémům nepochopení druhé strany.

Další praktikou, která zkvalitňuje IT služby je rotování lidí v týmu. Jde o pravidelnou změnu pozice s jednotlivými kolegy od vývoje až po údržbu. Tato praktika přispívá k celkově lepšímu pochopení problémové domény a pracovních postupů. Speciální formou je právě neoddělování týmů vývoje od provozu a údržby.

4.1.3. Proaktivita a učení

Jako první praktikou či vzorem je důležité porozumět problémové doméně, specifikům organizace a podnikovým cílům. Pro vývojáře nebo řešitele incidentů jsou podrobnosti o podnikových procesech velice důležité. Potřebuje přesně vědět, na co se bude IT služba používat a jak jí bude uživatel používat, protože např. testováním životně důležitých kritických systémů (nemocniční SW) se bude vývojář zabývat déle a podrobněji než testováním webové prezentace pro objednávání rezervací v hotelu. Nestačí tedy znát pouze aktivitu uživatele, ale i celý kontext a celý systém či proces, ve kterém danou aktivitu bude provádět.

Dalším vzorem proaktivity je identifikace trendů vývoje IT služeb, jejich rostoucí zatížení a využití. Napomůžou nám k tomu data o incidentech a problémech, kdy a proč se incidenty objevily a jaké byly jejich příčiny. Pomocí těchto dat můžeme provést simulace, měření a navrhnout řešení do budoucna.

Proaktivita je realizována především:

- analýzou historických dat
- predikcí trendů
- technologickými simulacemi podnikových scénářů
- řešením problémů a předcházením incidentů
- implementací prototypů a návrhem řešení na základě měření, simulací a predikcí

Proaktivitu je těžké do týmu prosadit, protože je postavena nad rámec standardních povinností zaměstnanců. Pomocí procesu prototypování se však proaktivita dá trochu zavést do způsobu práce. Tzn., pokud chceme přijít s vylepšením stávajícího produktu, je dobré vytvářet množství prototypů na papíře, ve formě fyzických modelů či verzí služeb nebo produktů a pravidelně je podrobovat diskusím a kritice od týmových odborníků (vývojáři, provozní pracovníci, obchodníci, specialisté na design, marketéři, produktoví specialisti).

Rychlá zpětná vazba stimuluje a urychluje naše učení. Napomáhá pochopit přímé důsledky našeho jednání a umožňuje neopakovat stejnou chybu dvakrát. Na úrovni týmu se zpětná vazba definuje pravidelnou demonstrací změn a nových funkčností zákazníkovi, novými integracemi verzí, vývojářskými testy a neustálou komunikací mezi analytiky, vývojáři a testery. Na úrovni řízení je to denní sledování trendu incidentů, změn, problémů a ústní zpětná vazba.

Organizace by měla podporovat sebevzdělávání a vyhradit na něj pracovníkovi čas.

Poslední vzor tohoto principu je přenášení znalostí na úrovni týmů i mezi nimi. Projekt, který dobře funguje v jednom týmu, nemusí tak dobře fungovat i v druhém z důvodů rozdílného kontextu, technologií, znalostí a zkušeností jednotlivých týmů. Proto je důležité, aby sdílení mezi projekty a službami probíhalo na obecnějších úrovních praktik a vzorů. Povolává se zkušený pracovník (mentor, kouč), který pomáhá s výběrem a implementací konkrétních vhodných praktik a vzorů. Cíl mentora a kouče je co nejdřív se stát nepotřebným pro tým - aby tým pracoval samostatně. Používá se rotování lidí v týmu i mezi týmy vývoje, provozu, údržby, podpory nebo prodeje.

4.1.4. Riziky řízený přístup

Zdá se, že už se při provozu a údržbě nemusíme zabývat riziky řízeného přístupu, jelikož všechny rizika by měly být vyřešeny ve vývoji a implementaci. Ovšem stávají se často technologická či integrační překvapení na konci projektu nebo selhání z důsledku neidentifikovaných, neřízených nebo nově se objevujících rizik.

Zdroje rizik mohou být:

- s rychle se měnícím trhem jsou vyžadovány rychlé dodávky a změny IT služeb
- opravy chyb nebo nové funkčnosti IT služby
- nové příležitosti k podnikání
- implementace zákonů a vyhlášek z ministerstev
- implementace do existujících technologií IT služeb/IS
- členové týmu – znalosti, odchod, motivace

Definice **rizika** podle PMI (Project Management Institute):
„Riziko je nejistá událost či podmínka, která pokud nastane, má negativní nebo pozitivní dopad na jeden nebo více cílů projektu."

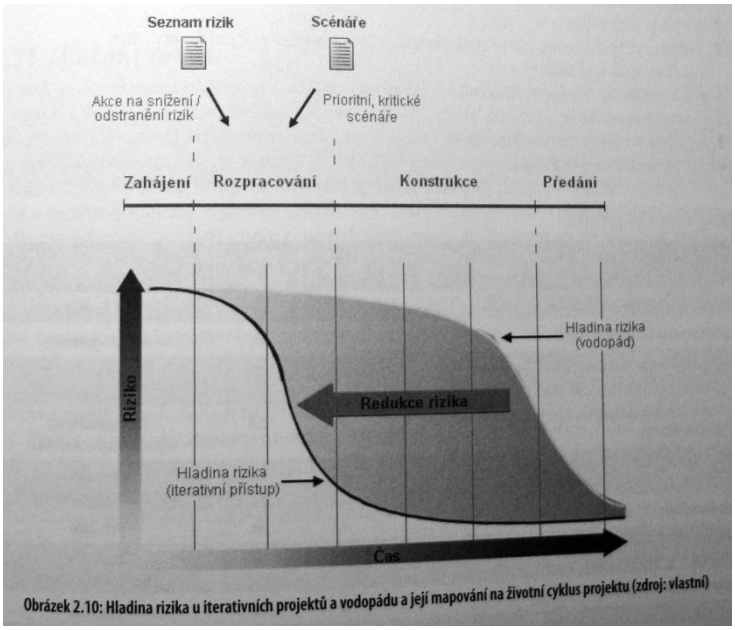
Pokud bychom se nezabývali riziky, mohlo by vzniknout několik důsledků:

- údržba a provoz by byl dražší, jelikož bychom museli „hasit požáry"
- pozdní odhalení problému
- překročené limity
- přetížení zaměstnanci

Riziky řízený přístup k provozu a údržbě se řídí proaktivně, tudíž se snaží rizika identifikovat co nejdříve, předcházet je a samozřejmě je odstranit. Po identifikaci rizika se řeší jak s rizikem naložit, aby to pro firmu bylo nejekonomičtější. Rozhodování můžeme zařadit do tří základních oblastí:

- předcházení riziku (risk avoidance) – reorganizace projektu (např. změna členů týmu či technologie, nábor konzultanta apod.)
- přesun rizika (risk transfer) – reorganizace projektu, tak aby riziko nesl jiný subjekt (zákazník, subdodavatel, banka)
- akceptace rizika (risk acceptance) – budeme s rizikem pokračovat v projektu, ale provedeme minimálně tyto kroky:
  - vytvoříme alespoň nějaké proaktivní kroky a monitorujeme symptomy rizika
  - vytvoříme záložní plán, který použijeme, pokud riziko nastane

S řešením rizik se můžeme setkat v životním cyklu iterativně inkrementálního projektu. Iterativní projekt je charakterizován řetězcem iterací, kdy výstup z jedné iterace je vstupem pro iteraci následující. Fáze iterativně inkrementálního projektu jsou následující: Zahájení (Inception), Rozpracování (Elaboration), Konstrukce (Construction) a Předání (Transition). Pro řešení rizik je určena fáze rozpracování (Elaboration). Na obrázku níže je vidět jak a kdy se zaměřujeme na rizika. Abychom snížili rizika na povolenou hranici, musíme nastavit jako prioritní implementaci kritických scénářů. Akce seznamu rizik se tak stanou cílem fáze rozpracování. Výhodou je, že můžeme už na začátku projektu vyzkoušet různé druhy architektury, která na projekt nejlépe sedne, zda implementace složitých algoritmů bude funkční apod. Pokud zjistíme, že neexistují varianty a nelze pokračovat, nestojí nás důsledky rizika tolik jako u tradičních metodik.



Obrázek 2.10: Hladina rizika u iterativních projektů a vodopádu a její mapování na životní cyklus projektu (zdroj: vlastní)

Vývoj více variant (Set Based Development) se používá v nejasném prostředí, kdy je riziky řízený přístup nevhodný buď z důvodu nízkého snížení rizik nebo z časové náročnosti. Je založen na vyvíjení více paralelních variant, ze kterých se postupem času vybere ta nejlepší varianta vzhledem k informacím, které jsme na začátku neměli. Tento přístup je finančně náročnější, ovšem pokud nejsme schopni vytvořit tradiční plán, ani identifikovat rizika, je tento přístup velmi přínosný.

5. AGILNÍ PRAKTIKY

V minulé kapitole jsem představil základní principy agilního a štíhlého provozu a údržby, které definují základní chování, nezbytné předpoklady a jádro budoucího procesu. Podle potřeby na toto jádro můžeme nabalovat jednotlivé praktiky. Praktiky obsahují způsob práce a zavádějí se inkrementálně podle přínosu. Tato kapitola obsahuje sadu základních praktik a pomůcek pro naplnění principů (1).

Pomocí konceptu MCIF® (Measured Capability Improvement Framework) s podporou nástroje Rational Method Composer definuji výhody praktik, které můžeme různě skládat a využívat k plnění podnikových cílů:

- Praktiky představují užitečnou jednotku znalosti.
- Praktiky se mohou skládat a nasazovat nezávisle na sobě.
- Praktiky mohou být spojeny s různými druhy školení či radami jak je používat.
- Praktiky jsou základním stavebním kamenem procesů-lze vybrat sadu praktik zasahujících do celého životního cyklu projektu.
- Podle cílů můžeme vybrat jednotlivé praktiky, které napomohou k dosažení cíle.
- Mohou se zavést nezbytné základní aspekty nebo kompletní praktiky včetně její automatizace – různý stupeň implementace.
- Zavedení praktik je měřitelné. Můžeme tedy zjistit, zda je daná praktika účinná a zda přispívá podnikovým cílům.

5.1. Iterativní přístup

Tato praktika se týká organizace týmu a způsobu jeho fungování. Je charakterizována společným týmem pro vývoj i údržbu. Vývojáři s údržbou pravidelně rotují mezi sebou podle předem stanovené délky iterace. V každé iteraci opakujeme stále stejné aktivity. Tato praktika se využívá v principech (4.1.2.) a (4.1.4.).

Výhody praktiky:

- Stejný tým pro vývoj a údržbu sdílí znalost architektury systému.
- Lepší kvalita výstupu, jelikož stejný tým vyvíjející nové funkčnosti po sobě musí opravovat chyby.
- Na konci každé iterace se může demonstrovat funkčnost zákazníkovi.
- Lepší odhad předpokládaného dokončení projektu díky iteracím.

5.2. Podnikové scénáře

Podnikové scénáře definují směry vývoje podniku, produktů a IT služeb zákazníka. Tyto scénáře se dělají kvůli dodavatelům IT, aby přesně věděli, jaký dopad může mít vývoj na IT infrastrukturu. Nemělo by tedy díky podnikovým scénářům docházet k překvapením, která by zásadně změnila provoz IT infrastruktury, IT služby či týmu.

Ve scénáři by mělo být:

- IT služby a jejich parametry
- IT infrastruktura
- Lidé, týmy, experti
- Prostory a energie pro IT infrastrukturu a zaměstnance
- Software a licence

5.3. Rotace

O této praktice jsem se už v práci zabýval několikrát. Jde o propojení a větší spolupráci zainteresovaných stran především vývoje s provozem a údržbou. V pravidelných intervalech rotují zaměstnanci na různé posty mezi týmy. Praktika popisuje vzor spolupráce, který je možný měnit podle potřeby a dané domény.

Praktika rotace má příznivý vliv na:

- kvalitu práce
- motivaci zaměstnanců
- úroveň znalostí

5.4. Boj s mnohoznačností

Největším problémem ve vývoji a údržbě je nejednoznačnost lidského jazyka. Je těžké komplexně popsat složitý abstraktní systém, což je problém především při specifikaci požadavků na SW. Jak bylo již řečeno v principech (4.1.1.) a (4.1.2.) odstranit tyto nedostatky lze častými demonstracemi implementovaných požadavků zákazníkovy a dobrou znalostí problémové domény.

Techniky pro formalizaci požadavků:

- Paměťová heuristika – ptaním se vývojářů a zákazníků na reálné požadavky zjistíme, na které si nemohou vzpomenout a které pravděpodobně i zcela nechápou. Na ty se musí více zaměřit a přeformulovat, aby byly

zapamatovatelné.

- Technika klíčových slov – identifikují se klíčová slova popisující požadavek, vypíší se všechny jejich možné definice a z rozdílných interpretací se vybere ta nejvhodnější pro daný kontext.
- Technika zdůrazňování – definice se čtou nahlas a zdůrazňují se jednotlivá slova, abychom dosáhli maxima rozdílných interpretací. Pokud je správná jedna, přeformulujeme požadavek. Pokud jich je více, je potřeba vymyslet další odpovídající požadavky na budoucí systém.

5.5. Defenzivní programování

V defenzivním programování se popisuje způsob myšlení a následování týmových praktik při tvorbě zdrojových kódů. Příkladem technik jsou:

- Inicializace všech proměnných před jejich využitím.
- Konzistentní zápis návrhových hodnot.
- Používání jen jednoho bodu pro opuštění procedury/metody.
- Používání příkazu assert jen pokud má smysl.
- Psaní čitelného kódu, ze kterého jsem po půl roce schopný vyčíst zamýšlenou podstatu.

5.6. Testy řízený přístup (TDD)

Všechny změny v kódu, které ve vývoji či údržbě uděláme, musí projít testy. Tyto testy jsou pro programátory zpětnou vazbou, pomocí které se mohou zdokonalovat a učit. Čím rychlejší zpětná vazba, tím je ponaučení z chyb větší. Pro testy řízené přístupem (TDD – Test Driven Development) platí, že se nejdříve vytvoří testy a až poté se začne s implementací funkčností. Tento přístup pochází z Extrémního programování (XP). Výhoda tohoto přístupu je v tom, že při vytváření testů si automaticky rozmýšlíme vlastní kód a většinou nás napadne několik lepších variant řešení. TDD zabere samozřejmě více času, trvá přibližně o 10-20% déle. Ovšem vrací se nám aktiva z rychlého, efektivního učení programátorů a především z oblasti provozu a údržby, kde se díky mnoha vytvořeným testům nevyskytují chyby.

5.7. Refraktoring

Refraktoring kódů je jakýkoliv zásah do zdrojového kódu programu, který pomůže zlepšit přehlednost, čitelnost či strukturu bez změny chování programu. Nezbytnou součástí jsou unit testy, které kontrolují, zda nebyly refraktoringem zaneseny chyby do kódu. Cílem není jen zpřehlednit kód, ale i odstranit nevyužívané části, které mohou vést ke stabilnější a kvalitnější aplikaci. Místa v kódu můžeme rozpoznat pomocí symptomů, které jsou např. duplicitní kód, dlouhé metody a seznamy parametrů, datové shluky a podobně. Pomocníkem při refraktorování bývají IDE (Integrated Development Environment) nástroje, které umožňují rychlé změny v kódu, např. změny názvu metody se automaticky přepíší v celém kódu.

5.8. Párová práce

Nejznámější párovou prací je párové programování, které prosazuje Kent Beck v Extrémním programování. Dva programátoři pracují na stejném počítači, střídají se, doplňují se nebo jeden píše testy a druhý přemýšlí na implementaci dané třídy. Nemění se jen tito dva programátoři mezi sebou, ale rotují spolu v celém týmu v pravidelných intervalech. Manažeři této praktice často nevěří, protože tak využívají dvě kapacity na jeden problém. Ovšem při výzkumech bylo dokázáno, že výstup z párového programování je o 40% rychlejší a navíc s kvalitnějším algoritmem.

5.9. Konvence

Jednou z nejvíce ceněných věcí při údržbě jsou konvence pro zápis kódu. Tyto konvence přispívají k nižší chybovosti díky čitelnosti, přehlednosti kódu a snadno se udržuje. Konvence se mohou definovat pro jednotlivé projekty nebo se mohou požívat obecné konvence od výrobců technologií či komunit, které jsou sdruženy kolem této technologie. Konvence by měli pokrývat všechny vrstvy a technologie zahrnuté v dané aplikaci či IT službě:

- pojmenování balíků a tříd
- pojmenování proměnných a konstant
- zápis návrhu tříd implementující návrhové vzory
- zápis rozhraní
- komentáře a pravidla použití pro další práci
- webová část a seznam používaných standardů a zdrojů
- databáze a pojmenovávání tabulek a sloupců
- ukázka zápisu zdrojového kódu třídy, metody či webové stránky

5.10. Retrospektiva

O retrospektivě jsem už v práci mluvil, proto tuto praktiku jen shrnu. V tradičních metodách se používala retrospektiva (pohled zpět) na konci každého projektu. Přínos pro daný projekt byl nulový. Proto se podle agilních principů používá retrospektiva mnohem častěji už v průběhu projektu či provozu IT služby, aby se mohli zaměstnanci neustále učit a zlepšovat jejich chování. Retrospektiva probíhá na konci každé iterace nebo při výskytu významného incidentu. Základní otázky pro retrospektivu jsou: Co fungovalo? Co nefungovalo? Co bychom měli udělat jinak?

5.11. Denní schůzky

Denní schůzky jsou pravidelné, ve stejný čas, s celým týmem a neměly by přesáhnout 15 minut. Odpovídá se v rychlosti na tři základní otázky: Co jsem udělal včera, co budu dělat dnes a co mě zdržuje v práci. Pro potřebu vývoje a údržby mírně upravíme tyto otázky na:

- Co zásadního se vyskytlo při poslední směně?
- Proč, jaká byla příčina?
- Co je třeba dokončit?
- Kdo je vlastníkem tohoto úkolu?
- Kdy má být úkol dokončen?

5.12. Vizualizace

Vizualizace je technikou, kterou se dají velmi dobře najít problémy, které pracovník či tým nemohl najít. Tato metodika je i u tradičních metod, ovšem agilní přístupy využívají silnějších nástrojů jako je Kanban nástěnka nebo Burn down chart.

Koncepce Kanban je charakterizována neustálým průtokem požadavků ze vstupu na výstup. Není omezen žádnými bloky či dávkami jako je tomu u tradičních metodik, které se řídí principem „jakmile mám hotovo, pošlu to dál“. Kanban je založený na tahu, ne tlaku jako tradiční metodiky. Práci si z předchozí aktivity bere daný tým sám, až když má volnou kapacitu. Kanban nástěnka má za cíl vizualizovat a zviditelnit proces, omezení rozpracovaných úkolů, pomoci identifikovat problémy a umožnit řešení.

Burn down chart je graf zpracovaných požadavků. Nejprve se rozdělí práce do denních úkolů. Po vypracování úkolu se graf pokaždé aktualizuje. Pokud se na grafu objeví nějaká anomálie, přijmou se nezbytné kroky pro nápravu.

Dále je užitečnou technikou kapacitní varování v podobě signálního kolečka. Využívá se především v provozu a údržbě, kdy se často mění zátěž jednotlivých pracovníků. Na kolečku se mohou měnit tři barvy a vždy je vidět pouze jedna. Pokud má pracovník problémy s nějakým úkolem a nestíhá, posune kolečko na oranžovou nebo červenou barvu. Jiný pracovník, který má své úkoly splněné a má na kolečku zelenou může kolegovi pomoci.

5. Závěr

Skoro 70% z IT rozpočtu je vydáváno na provoz a údržbu IT služby či IS. Proto je velmi důležité zabývat se touto problematikou. V současnosti existují především řešení orientované na tradiční rigorózní procesní standardy a metodiky, které nejsou tak účinné. V práci jsem se proto zabýval přístupem založeným na agilních a štíhlých principech a praktikách v provozu a údržbě, který oproti rigorózním metodám zapojuje více lidskou přirozenost, chování a osobní potřeby každého z nás. Popsané principy a přístupy jsem čerpal převážně z knihy „Provozujte IT jinak“ od Jaroslava Procházky. Definoval jsem stručně problémy, které se vyskytují při provozu a údržbě IT služby/IS a navázal jsem na ně řešením v podobě čtyř principů, které využívají konkrétní praktiky. Těchto praktik je celá řada, proto jsem v práci pospal ty nejpoužívanější. Pouhým následováním těchto principů a praktik však nedosáhneme očekávaného výsledku. Klíčem k úspěchu je motivace zúčastněných na projektu a jejich propojení s projektovými cíli.

6. ZDROJE

1. **Procházka, Jaroslav.** *Provozujte IT jinak.* místo neznámé : Grada, 2011. str. 288. 978-80-247-4137-6.

2. **Ltd, APM Group.** ITIL. [Online] http://www.itil-officialsite.com/home/home.aspx.

3. [Online] http://www.ieee.org/index.html.

4. **Hajdin, Tomáš.** [Online] 2005. http://is.muni.cz/th/39440/fi_m/dp.pdf.

5. Manifest agilního vývoje software. [Online] 2001. http://agilemanifesto.org/iso/cs/.