Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

TÉMA SEMESTRÁLNÍ PRÁCE:

FOR CONTINUOUS DELIVERY, IT'S ALL ABOUT INTEGRATION

Autoři: Bc. Lukáš Vlček (xvlcl05)

Bc. Kamil Soule (souk03)

Bc. Jiří Čarek (xcarj00)

Kurz: 4IT421 Zlepšování procesů budování IS

Datum odevzdání: 14.5.2016

Semestr: LS 2016

Abstrakt

Cílem této práce je úvod do problematiky průběžného vývoje a DevOps. Hlavním cílem je představení průběžné integrace, průběžného dodání a průběžného nasazení, jakožto důležité části průběžného vývoje. Zvolený přístup k vývoji a provozu software je porovnán oproti klasickým přístupům, jsou posouzeny možnosti zvýšení kapacity pro inovaci a zlepšení uživatelské spokojenosti. V práci jsou srovnány výhody DevOps a posouzeny chybné praktiky při zavádění průběžného vývoje. Přínosem této práce je vhled do funkčnosti konceptu DevOps.

Klíčová slova

Průběžná integrace, průběžné nasazení, průběžné dodávání, DevOps, agilní vývoj

Obsah

1 Úvod

Agilní metodiky se dostávají do popředí a v kombinaci s pokročilou automatizací procesů dostáváme velice efektivní přístup k vývoji a provozu software. Lze rozeznat aktuální trendy continuous integration (průbežná integrace), continuous delivery (průběžné dodávání) a continuous deployment (průbežné nasazení) a DevOps, které jsou založeny na agilních a lean (štíhlých) přístupech. Z agilních přebírají hlavně důraz na spokojenost zákazníka, z lean hlavně odstranění přebytečné práce. Vycházejí z praxe, kde se nalezli úzká místa a našla se snaha tyto problémy adresovat pomocí moderním nástrojů, postupů a změny filosofie.

Vzhledem k tomu, že výše uvedené termíny se v současné době stávají buzzwordy ve světě IT a využívání těchto konceptů má své nesporné výhody, má smysl se tématem zabývat. Cílem této práce je čtenáře blíže seznámit s těmito postupy a koncepty a srovnat je s postupy, které se jsou známé z klasického přístupu k vývoji a provozu software.

Proto je obsahem této práce definice a popis termínů průběžná integrace, průběžné nasazení a DevOps. Pro porovnání s klasickými postupy slouží seznam výhod oproti nim, který je uveden u každého popisovaného pojmu. Aby tomuto srovnání byla dodána větší váha, obsahuje tato práce také srovnání klasického přístupu a DevOps na základě dat získaných z průzkumů. Nakonec pro zájemce o implementaci zmíněných konceptů do praxe, tato práce obsahuje také popis několika špatných praktik při zavádění průběžného vývoje, jako je například špatná práce se systémem pro správu verzí či integrace a systémy pro dodávku a nasazení.

2 Průběžná integrace, nasazení a dodávání

V této kapitole jsou vymezeny základní pojmy, které jsou dále používány v této práci a následně je ukázáno k čemu slouží a jaké jsou vztahy a rozdíly mezi nimi. Závěrem této kapitoly jsou představeny benefity a techniky průběžné integrace, nasazení a dodávání.

2.1 Průběžná integrace (Continuous Integration)

Průběžnou integraci lze považovat za jakýsi postup při vývoji software, při kterém dochází k častému integrování práce jednotlivých členů týmu. Pojem „průběžná integrace“ původně pocházel z procesu vývoje Extrémního programování jako jeden z původních dvanácti postupů. Integrace se stává složitější s nárůstem počtu osob pracujících na daném projektu a externích systémových závislostí. Často totiž každý člen týmu řeší nějaký svůj individuální úkol nebo problém, ale společně pracují na jednom společném celku, který je výstupem. Proto je potřeba práci integrovat průběžně, aby byla zachována konzistence a všechny komponenty projektu spolu správně pracovaly. Integrování, které se provádí až ke konci vývoje softwaru, může být často velmi nákladné a časově náročné, proto se i řada projektů nestihne včas. Každá integrace, která je prováděna v rámci postupu průběžné integrace, bývá ověřována automatizovaným sestavením včetně testů, aby bylo možné co nejrychleji nalézt integrační chyby. Jedním z cílů průběžné integrace je poskytování zpětné vazby. To znamená, že pokud vývojář provede nějaké úpravy v kódu, které nejsou dobré, obdrží okamžitě zpětnou vazbu, která mu umožní identifikovat a opravit problém co nejdříve (Duvall, Matyas a Glover, 2007, s.318),(Fowler, 2006).

Důležitým předpokladem pro úspěšné zavedení průběžné integrace je centrální úložiště se správou verzí. Díky centrálnímu úložišti mají ke změnám zdrojových souborů přístup všichni vývojáři, testeři atd. Dále je třeba zavést server průběžné integrace (dále jen CI server), který vykoná integrační sestavení, kdykoliv někdo z týmu nahraje svoji změnu do centrálního úložiště. Integrační sestavení na CI serveru lze vykonávat i ručně, ale právě automatizování sestavení, testování a dalších procesů patří mezi hlavní rysy průběžné integrace. Právě bez těchto automatizovaných procesů se nejedná o průběžnou integraci (Duvall, Matyas a Glover, 2007, s.318) (Fowler, 2006).

Příklad typického scénáře průběžné integrace:

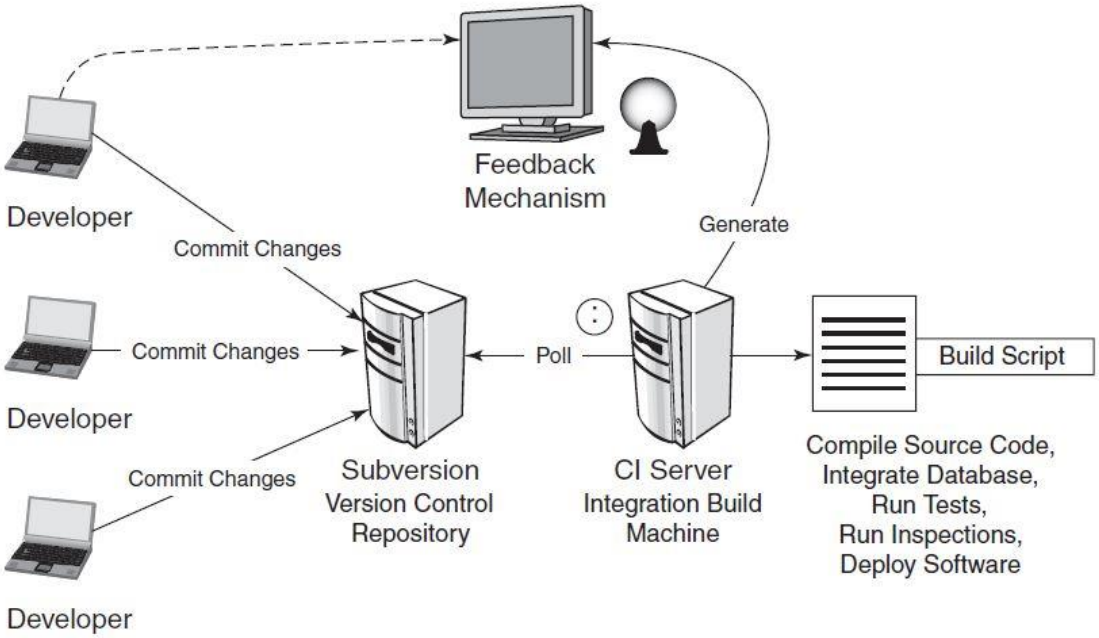
1) Vývojář nahraje svůj nově vytvořený kód na nějaké centrální úložiště se správou verzí (např. SVN či GIT repository).

2) CI server si neustále ověřuje, zda nedošlo v kódu k nějakým změnám prostřednictvím dotazování do centrálního úložiště. Jakmile po nahrání dat vývojáře do centrálního úložiště zjistí CI server, že došlo ke změně, okamžitě si

natáhne zdrojové soubory z centrálního úložiště a provede integraci pomocí sestavovacího skriptu.

3) Jakmile CI server dokončí svoji integrační práci, okamžitě zasílá vývojářům v týmu zpětnou vazbu o výsledku integračního sestavení. Tyto zpětné vazby samozřejmě obdrží nejen vývojáři, ale i celý vývojářský tým.

4) CI server dále dotazuje centrální verzované úložiště, zda nedošlo k dalším změnám ve zdrojových souborech (Duvall, Matyas a Glover, 2007, s.318).



Obrázek 1: Komponenty systému průběžné integrace (Duvall, Matyas a Glover, 2007, s.318).

Automatizované sestavení:

Díky automatizaci integračního sestavení pomocí různého software nebo skriptů je možné eliminovat počet opakujících chyb, zanesených vlivem selhání lidského faktoru, které by vznikly manuálním sestavením. Automatizované sestavení zahrnuje automatizaci integrace, ale také může zahrnovat nasazení projektu do produkce. Jelikož je sestavení a kompilace často velmi komplikovaná, používají se právě nástroje, které tyto procesy automatizují. Výstupem automatizovaného sestavení mohou mimo jiné být i generované dokumentace, různé webové stránky apod. (Fowler, 2006).

Automatizované testování:

Jedná se o velmi efektivní proces, při kterém je možné zjistit mnoho chyb při procesu sestavení. Jedná se o tzv. sebe-testování pomocí sady automatizovaných testů. Výsledek těchto testů pak říká, zda sebe-testování proběhlo úspěšně anebo nějaké testy selhaly. Pokud takto testy selžou, automaticky se nezdaří ani provést sestavení (Duvall, Matyas a Glover, 2007, s. 318). Kombinace průběžného sestavení a automatizovaných testů je někdy

označována také jako „Daily build and smoke test“ (volně přeloženo jako denní sestavení a kouřový test).

Klíčové praktiky průběžné integrace:

- Nahrávat kód velmi často na centrální úložiště.
- Nenahrávat do centrálního úložiště poškozený kód.
- Opravit neúspěšné sestavení co nejrychleji.
- Udržovat vytváření sestavení co nejrychleji.
- Psát si vlastní jednotkové (unit) testy před integračním sestavením (automatizace jednotkových testů)
- Všechny testy musí projít.
- Pro předejití neúspěšného sestavení, by měl vývojář zkusit napodobit integrační sestavení u sebe.
- Vyhnout se stažení poškozeného kódu na svoji pracovní stanici.
- Umožnit členům týmu co nejjednodušší přístup k poslední verzi kódu.
- Každý člen týmu musí být srozuměn o jakýchkoliv událostech.
- Automatizovat nasazení.

2.1.1 Přínosy průběžné integrace:

1) Velmi důležitým přínosem průběžné integrace je redukce rizika a díky automatizaci je možné dobře odhadnout délku integračního procesu.
2) Snadnější vyhledání a opravení chyb.
3) Šetření času při kompilaci, sestavení a jiných procesů v rámci vývoje softwaru.
4) Šetření času testerů díky automatizovaným testům.
5) Automatické kontrolování kódu.
6) Každý člen týmu má přehled o stavu sestavení a změnách.
7) Přehledné verzování souborů a snadný přístup k nim (Fowler, 2006).

2.2 Průběžné dodávání (Continuous Delivery)

Jedná se o další fázi průběžné integrace, ve které je třeba už mít software ve stavu vhodným na produkci. Je to tedy fáze, ve které průběžná integrace, automatizované

sestavení, testování atd. umožňují, aby software mohl být co nejrychleji nasazen na produkci. Na rozdíl od průběžného nasazení (podkapitola 2.3), kde je každá změna automaticky nasazena na produkci. Průběžné dodávání znamená, že je možné nasazovat software na produkci velmi často, ale je možné se rozhodnout, zda se software nasadí nebo ne (Humble a Farley, 2010, s. 497) .

Základní znaky průběžného dodávání:

1) Software musí být schopný nasazení.
2) Nejvyšší prioritou pro tým je udržet software schopný nasazení, i když mezitím pracuje na nových funkcích.
3) Kdokoliv by měl mít možnost dostat automatizovanou zpětnou vazbu o stavu a připravenosti na produkci, kdykoliv někdo udělá změnu.
4) Software by měl být schopný automatizovaně nasadit libovolnou verzi softwaru na libovolné prostředí.

Aby bylo možné průběžně dodávat software, je třeba automatizovat veškeré možné části vývojového procesu (automatizované sestavení, automatizované testování atd.). Samozřejmě je potřebné nepřetržitě integrovat. Je velmi vhodné testovat software před tím, než bude schopný nasazení na produkci, aby byl vyzkoušen na prostředí, které se chová stejně jako produkce. Často toto prostředí firmy nazývají jako pre-produkce (Fowler, 2013).

2.2.1 Přínosy průběžného dodávání

1) Průběžné dodávání umožňuje dané organizaci vyrukovat s novou verzí software znatelně rychleji. Zlepšuje tedy konkurenceschopnost organizace.
2) Lepší produktivita a efektivita vývoje. Díky automatizaci vývojářský tým spoří čas.
3) Lepší kvalita software díky snížení počtu chyb a incidentů s pomocí automatizace testování a identifikace (Chen, 2015).
4) Snížení výskytu problémů při nasazení software. Díky častým integrováním a sestavováním menších částí je snazší objevit problém a opravit ho.
5) Častější vydávání nových verzí umožňuje získávat častěji zpětné vazby od zákazníků, díky tomu je možné dodávat opravdu správný užitečný software, který zákazník požaduje. To pak samozřejmě vede ke zvýšení spokojenosti zákazníků (Fowler, 2013).

2.3 Průběžné nasazení (Continuous Deployment)

Průběžné nasazení funguje také na automatizované bázi. To znamená, že každá aktualizovaná pracovní verze, která vede přes všechny procesy průběžné integrace a dodání je automaticky nasazována na produkci (do reálného provozu). Tyto nasazení mohou probíhat i několikrát denně (např. firmy Facebook, Flickr, Etsy). Jedná se tedy o poslední fázi celé integrace. Díky celému cyklu průběžné integrace (automatizovaných sestavování, testů, průběžných kontrol a integrací databází atd.) je možné nasadit vyvíjený software na jakékoliv prostředí a kdykoliv. Samozřejmě po splnění všech business požadavků na funkcionalitu apod. (Duvall, Matyas a Glover, 2007, s.318). Fáze nasazení se zpravidla týká těchto bodů:

1) Správné označení a přidání popisků k souborům v rámci centrálního úložiště, které k sobě patří. To poté umožňuje sledování historie celé skupiny souborů a ne pouze jednotlivých – ty se mohou totiž snadno rozcházet ve verzích. Díky těmto označením je možné opravovat různé chyby na paralelních větvích v centrálním úložišti.

2) Zajištění, aby se nasazovaný software nerozcházel s verzí operačního systému, databáze, aplikačního serveru atd., které jsou na prostředí, kde bude software nasazován.

3) Vytváření unikátních identifikátorů a popisků pro jednotlivé sestavení. Jedná se o popisky a identifikátory binárních výstupů sestavení (.jar, .zip ...), na rozdíl od bodu 1), kde se jedná o jednotlivé soubory kódu třeba i nezkompilované.

4) Veškeré automatizované a další testy musí před nasazením úspěšně projít.

5) Automatizované vytváření reportů a generování zpětných vazeb, aby zainteresované osoby měly představy o tom, zda byly splněny veškeré požadavky nebo ne. Dále zainteresované osoby získají přehled o chybách, které byly řešeny, opraveny atd.

6) Je třeba zajistit, aby vždy byla možnost návratu na předchozí verzi. To znamená, že pokud má stávající verze nějaké problémy, je možné ji dočasně nahradit předchozí verzí, která fungovala lépe (Duvall, Matyas a Glover, 2007, s.318).

3 Propojení vývoje a provozu software – DevOps

Dalším buzzwordem posledních let spojeným s IT světem, kterým se tato práce zabývá je DevOps. Tento název vznikl spojením anglických výrazů development (vývoj) a operations (provoz). DevOps jako takový je pouze koncept, filozofie a přístup k vývoji software. Neskrývá se za ním rigorózní metodika či dokumentace, která by zájemce dokázala krok po kroku dovést do stavu vývoje, který by se dal označit pojmem DevOps. To je také důvodem, proč neexistuje žádná přesná definice tohoto pojmu. Různé definice však vždy zachovávají stejnou klíčovou myšlenku. Pro objasnění tohoto termínu využijeme definice uvedené v knize (Sharma, 2014, s. 1):

„DevOps je přístup k vývoji software, který je založen na agilních a lean principech, jež se snaží propojit business vlastníky, vývojáře, provozní správce IT a oddělení pro zajištění kvality, s cílem dodávat software průběžně, což umožňuje businessu rychle se chopit obchodních příležitostí a snížit čas pro získání zpětné vazby od zákazníka“.

DevOps vychází z myšlenek z agilních principů vývoje software. Na rozdíl od obecného agilního přístupu k budování softwaru, se DevOps snaží zaměřit zejména na plynulost a efektivitu procesu souvisejícího s dodávkou software a získání zpětné vazby. Principy DevOps lze aplikovat na libovolný projekt, bez ohledu na jeho business doménu, či využívanou agilní metodiku pro vývoj. Některé z DevOps praktik lze využít i v případě vývoje pomocí klasických metodik (např. využití průběžné integrace a testování). Pro využití kompletního spektra DevOps praktik se však předpokládá agilní přístup k vývoji.

3.1 Hlavní principy a myšlenky DevOps

Cíle DevOps, čili zefektivnění dodávky software a zlepšení zpětné vazby, jsou dosaženy zejména pomocí aplikace několika hlavních myšlenek a principů, mezi které patří:

- Automatizace procesů souvisejících s nasazením na různá aplikační prostředí
- Zlepšení komunikace mezi účastníky na vývoji a provozu software
- Společná odpovědnost a vzájemná důvěra účastníků na vývoji a provozu
- Zlepšení získání zpětné vazby

3.2 Výhody DevOps

Pokud bychom se na DevOps podívali velmi detailně, dalo by se o jeho výhodách psát velmi dlouho. Detailní pohled na DevOps však není cílem této práce, a proto zde uvádíme pouze tři hlavní benefity , kterými jsou:

- Zlepšení uživatelské spokojenosti
- Zvýšení kapacity pro inovaci
- Zrychlení času pro vytvoření hodnoty (tvz. „time to value")

3.2.1 Automatizace procesů

Automatizace procesů vede k znatelnému snížení nákladů a to díky nižším nákladům na manuální práci zaměstnanců, jejichž hodinová sazba bývá často vysoká. Automatizací nedochází k potřebě přítomnosti pracovníka a celý proces proběhne mnohem rychleji. Díky mechanizaci procesů nedochází k nepředvídatelným chybám způsobeným lidským faktorem, a tedy nejsou ani náklady potřebné na odstraňování těchto chyb. Může však nastat problém kompatibility různých automatizačních aplikací. Tento problém se může řešit koupí komplexního aplikačního balíku, avšak musíme vzít v úvahu také provozní náklady na licence produktu. Automatizace se využívá také pro nasazování aplikací a tím se snižuje potřebný čas pro jejich uvedení do provozu.

3.2.2 Zlepšení uživatelské spokojenosti

Toho je docíleno díky zlepšení získávání zpětné vazby – pokud víme, co zákazník opravdu potřebuje, můžeme mu to dodat. Navíc lze dodávat v kratších intervalech, což zákazník ocení zejména v případě požadavků na změny stávající funkcionality či opravě chyb, které se dostaly do produkce. Zdržení nasazení takových změn by totiž mělo za následek zhoršení efektivity práce se systémem. Další zlepšení vyplývají z ostatních benefitů, které se také více či méně promítají do spokojenosti zákazníka.

3.2.3 Zvýšení kapacity pro inovaci

Pokud dokážeme rutinní práci zautomatizovat, ušetříme tak cennou část personální (zrychlením procesů leckdy i technické) kapacity, kterou lze využít k práci, která vytváří skutečnou business hodnotu pro zákazníka. Také díky zlepšení zpětné vazby (ať již pomocí dřívějšího testováním, či zpětné vazbě od zákazníka), dochází k úbytku zbytečné práce a tím navýšení kapacit.

3.2.4 Zrychlení času pro vytvoření hodnoty (tvz. „time to value")

Samotný požadavek na změnu či přidání funkcionality systému ještě nevytváří pro zákazníka žádnou hodnotu. Teprve až po nasazení začíná ze změny skutečně těžit. Zrychlení celého cyklu nasazení systému (průběžné integrace, dodávání a nasazení) pomáhá zrychlit čas k vytvoření skutečné hodnoty.

4 Průběžný vývoj a DevOps oproti klasickým přístupům

V předchozím kapitolách jsme si představili součásti tvz. průběžného vývoje (continuous development) - průběžnou integraci, průběžné dodávání a průběžné nasazení. Dále také koncept zvaný DevOps. Tato kapitola je zaměřena na srovnání uvedených praktik s tvz. tradičním přístupem k vývoji a provozu software. Abychom se vyhnuli prostému vyjmenování výhod, které se objevili v předešlých kapitolách, budou zde uvedeny data z průzkumu, který srovnával DevOps přístup a tradiční přístup. Průzkum, z kterého bylo čerpáno (Badrinarayanan, Kabanov, James a White, 2013), se dotazoval 620 pracovníků IT z různých firem na jejich zkušenosti ohledně vývoje a údržby software. Bližší informace o metodice průzkumu lze nalézt v uvedeném zdroji, který je dostupný online.

Je nutné si uvědomit, že s aplikací DevOps souvisí zavedení průběžné integrace, dodávky a nasazení, a proto lze na toto srovnání pohlížet také jako na srovnání využívání těchto praktik, oproti jejich absenci v tradičních IT týmech.

Konkrétně lze v této kapitole nalézt srovnání doby trávené na běžných provozních aktivitách a délky doby nutné pro zajištění zotavení po chybě.

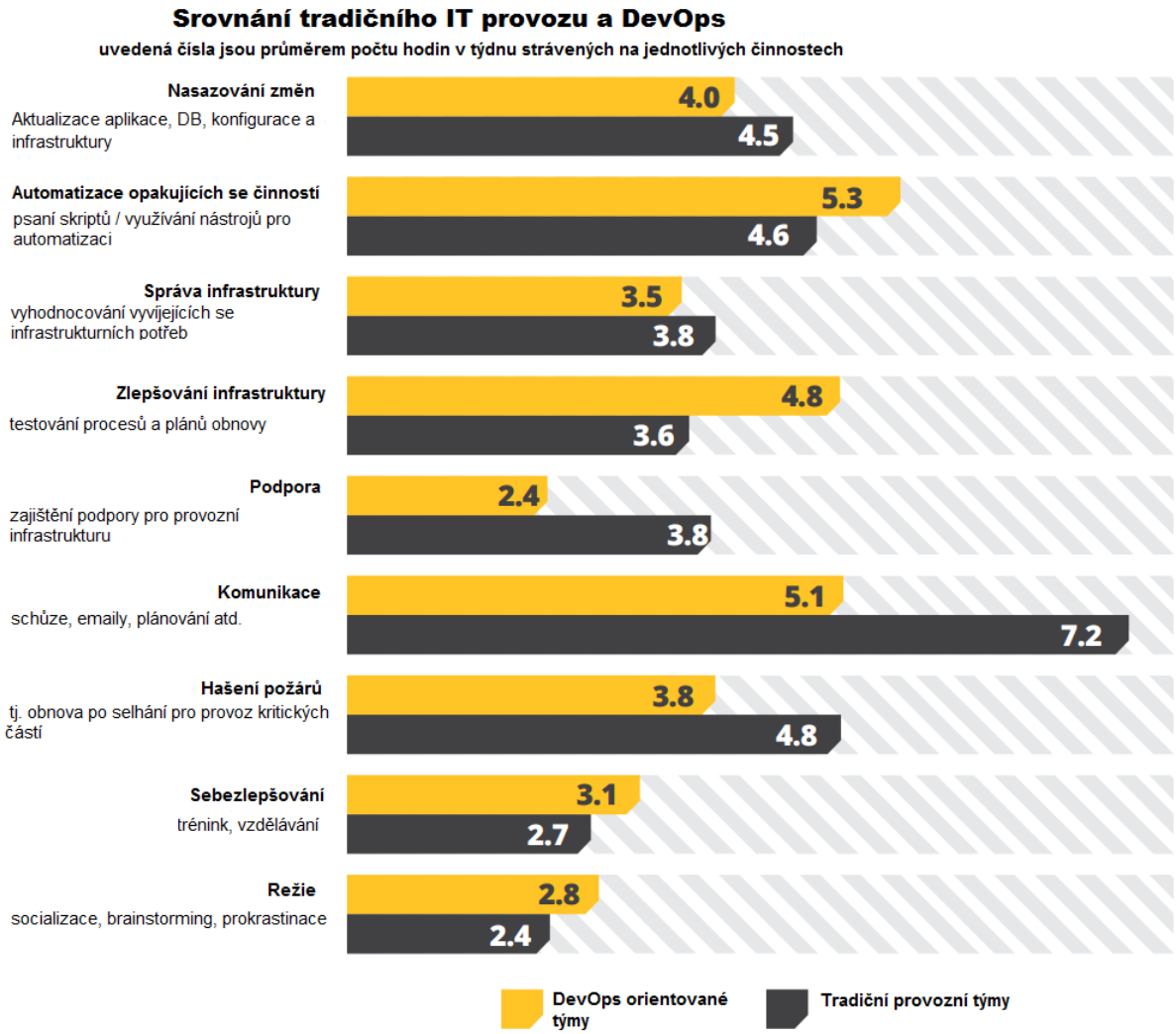
4.1 Srovnání doby trávené na běžných provozních aktivitách

Subjektem prvního porovnání je srovnání doby trávené na běžných aktivitách. Srovnává se zde čas odhadovaný týmy IT provozu, které pracují tradičním způsobem, a týmy, které využívají principů DevOps. Data jsou zanesena v grafu (obr. 2).

Ze srovnání vyplývá zejména (Badrinarayanan, Kabanov, James a White, 2013, s. 9):

- DevOps týmy tráví více času automatizací procesů a zlepšováním infrastruktury
- DevOps týmům stačí méně času pro komunikaci
- DevOps týmy nemusí tolik „hasit požáry“
- DevOps týmy tráví méně času na administrativní podpoře
- DevOps týmům stačí méně času k provádění běžných činností

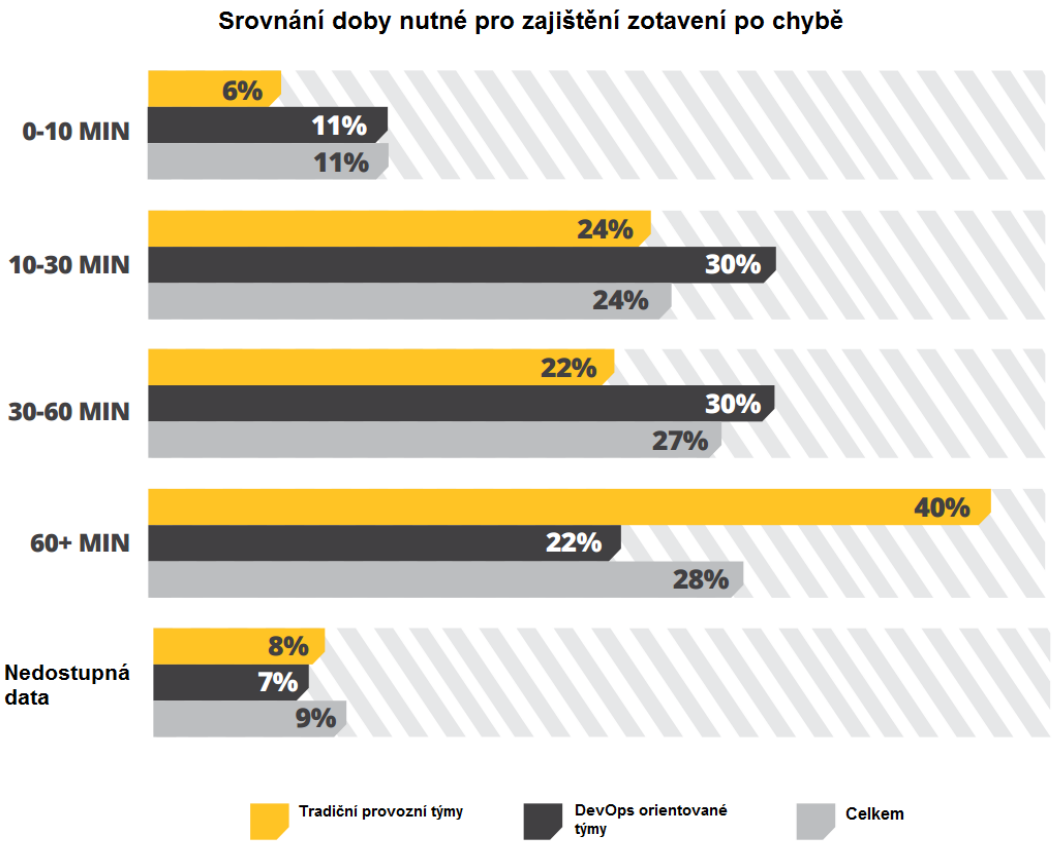
Tyto data také potvrzují tvrzení o výhodách DevOps postupů a principů, zmíněných v předchozích kapitolách.



Obrázek 2: Srovnání doby trávené na běžných aktivitách – tradiční IT provoz vs DevOps (Badrinarayanan, Kabanov, James a White, 2013, s. 6)

4.2 Srovnání doby nutné pro zajištění zotavení po chybě

Zajištění co nejrychlejšího zotavení po selhání systému v produkčním prostředí je velmi důležité. Každá minuta, kdy systém není schopen provozu, znamená finanční ztrátu pro zákazníka. Na grafu (obr. 3) je zobrazeno srovnání doby nutné pro zajištění zotavení po chybě v případě tradičních týmů oproti DevOps orientovaných týmů. Z grafu je patrné, že DevOps týmy jsou schopny zajistit rychlejší zotavení po chybě. Znovu se tedy potvrzuje, že filosofie DevOps pomáhá zefektivnit práci IT týmů



Obrázek 3: Srovnání doby nutné pro zajištění zotavení po chybě (Badrinarayanan, Kabanov, James a White, 2013, s. 14)

4.3 Závěr srovnání

Ze srovnání pomocí dat z uvedeného průzkumu vychází jako vítěz DevOps orientované týmy. Tento výsledek není překvapující. DevOps, společně s postupy průběžné integrace, dodávání a nasazení, totiž vycházejí přímo z praxe a jejich snahou je eliminovat úzká místa a nadbytečnou práci spojenou s tradičními postupy.

5 Řízení životního cyklu

Životní cyklus softwarových aplikací je založen na jednotlivých vývojových fázích, které jsou úzce propojené. Metodika byla vyvíjena v období vodopádového vývoje aplikací a snažila se tento systém zrychlit a zvýšit efektivitu. Naproti tomu agilní metodiky pohlížejí na vývoj, jako na ucelený proces, a z nimi byl později inspirován koncept DevOps. Vzhledem k tomu, že DevOps již počítá s vývojem, jako celkem, částečně tak nahrazuje Životní cyklus vývoje softwarových aplikací, neboť je to již jeho součástí. DevOps přejímá z řízení životního cyklu automatizaci procesů a dává na ni ještě větší důraz, avšak jsou vynechány počáteční a koncová fáze životního cyklu a tím se z vývoje stává neustálá činnost, kterou je třeba kontrolovat a regulovat. Přístup DevOps je plynulejší, ovšem stále se jedná o tentýž záměr, tedy řízení životního cyklu.

6 Špatné praktiky při zavádění průběžného vývoje

V přechozích kapitolách jsme se seznámili s postupy průběžného vývoje (Continuous Development) – průběžnou integrací, dodáváním a nasazením. Ačkoliv tyto postupy a principy jsou pokrokové, z jejich výhod lze těžit, jen pokud jsou správně zavedeny. Bohužel k těmto konceptům neexistuje jednotná vše pokrývající „kuchařka“, která by zájemce krok po kroku navedla k jejich úspěšnému zavedení ve firmě. Proto je účelem této kapitoly poukázat na některé špatné praktiky, se kterými je možné se setkat při snaze uvést tyto principy do praxe.

6.1 Špatná práce se systémem pro správu verzí a systémem integrace

Zavedení průběžné integrace nespočívá jen ve výběru a nasazení některého z dostupných nástrojů pro integraci a jeho napojení na systém pro správu verzí (zkráceně verzovací systém). Důležitým faktorem, který zajistí kýžené zlepšení je správné využívání těchto systémů.

Jednou z častých chyb týkajících se nesprávného využívání verzovacího systému je **špatný výběr souborů (artefaktů)**, které mají být verzovány. Všechny potřebné soubory pro chod systému by měli být dostupné ve verzovacím systému (včetně testovacích dat, pomocných skriptů apod.). Naopak je správné vynechat soubory, které lze generovat sestavením systému (např. binární soubory). Dále se jedná o **nevhodnou práci s větvemi**[1] v repozitáři. Jde například o porušení pravidla častého slučování větví (APPLETON, Berczuk Cabrera a Orenstein, 1998 s. 12), které vede k tvz. merge hell – pozdější slučování větví vede k mnohým konfliktům ve změnách, které je třeba dlouho řešit (na úkor produktivní práce). Další špatnou praktikou je opomenutí nastavit politiku pro vytváření větví, což způsobuje nadbytečné větvení.

[1] Větvení je vlastnost moderních verzovacích systémů (např. Git, TFS i staršího SVN). Umožňuje paralelní vývoj. Je důležitý např. pro oddělení vývoje nové funkcionality od údržby a opravy chyb systému na produkci.

Mezi chyby s prací s integračním systémem patří např. (Duvall, 2011):

- Dlouhá perioda mezi integračním sestavením (např. pouze noční sestavení)
- Změny nejsou vývojáři odesílány na denní bázi, ale nechávají si práci na svých strojích i po dobu několika dnů
- Testování není součástí integračního sestavení, nebo prováděno pouze na strojích vývojářů a nikoliv na jednom centrálním místě
- Nejsou odesílány notifikace o úspěšnosti sestavení, či jsou tyto upozornění ignorovány, nebo nejsou rozesílány všem členům týmu
- Sestavení probíhá na počítačích vývojářů, nikoliv na centrálním místě

6.2 Špatná práce se systémy pro dodávku a nasazení

Další částí řetězce průběžného vývoje jsou systémy pro dodávku a nasazení. Mezi chyby, se kterými je možné se setkat při práci s těmito systémy patří (Duvall, 2011):

- Odsun automatických testů, které lze provádět dříve až na předprodukční prostředí
- Různé skripty pro nasazení pro různá prostředí, oproti jednomu centrálnímu
- Nemožnost automatického návratu (rollbacku) k předchozí verzi v případě chyby
- Vynechání automatizace nastavení prostředí (např. nastavení sítě)
- Rozvrhování nasazení na produkci bez ohledu na to, kolik uživatelů bude nasazením ovlivněno
- Nevyužívání tvz. blue green deploymentu – podle tohoto principu by se měl systém nedříve nasadit a plně zprovoznit a poté pouze „přepnout“ stávající produkční verzi na novou, nikoliv odstavit systém po celou dobu nasazení
- Nedodržení integrity binárních souborů – systém by se měl sestavit jen jednou, a poté jen nasazovat, nikoliv sestavovat na každém prostředí
- Opomenutí automatizace práce s databází (např. provádění manuálních změn schématu)

7 Závěr

Na základě vymezení a představení prvků průběžného vývoje (integrace, dodání a nasazení) bylo možné dojít k závěrům, že DevOps a agilní metodiky pro automatizaci procesů znatelně snižují potřebu manuální práce, čímž se velmi snižují náklady na vývoj a snižuje čas potřebný pro uvedení software na trh. Díky automatizovaným procesům se zvyšuje konkurenceschopnost firmy a tedy i její potenciál na trhu. Přínosem této práce je určitě také identifikování přínosů a klíčových praktik pro správné zavedení průběžného vývoje. Tento obzor pak rozšiřují také identifikované špatné praktiky, činnosti a rizika, které mohou bránit ve správném zavedení průběžného vývoje.

Vzhledem k omezení délky práce nebylo možné se jednotlivými kapitolami zabývat více do hloubky, jako například zabývat se dalšími měřeními z průzkumu ohledně DevOps. Avšak zvolené množství uvedených informací plně postačuje k pochopení uvedených principů a pro jejich srovnání s klasickým přístupem k vývoji a provozu software. Všechny stanovené cíle práce byly tedy naplněny.

Seznam zdrojů a literatury

DUVALL, Paul M, Steve MATYAS a Andrew GLOVER. *Continuous integration: improving software quality and reducing risk*. Upper Saddle River, NJ: Addison-Wesley, c2007. ISBN 0321336380.

HUMBLE, Jez a David FARLEY. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Upper Saddle River, NJ: Addison-Wesley, 2010. ISBN 9780321601919.

CHEN, Lianping. Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*. 2015, **32**(2), 50-54. DOI: 10.1109/MS.2015.27. ISSN 0740-7459. Dostupné také z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7006384

FOWLER, Martin. Continuous Integration. *Martin Fowler* [online]. 2006 [cit. 2016-04-18]. Dostupné z: http://www.martinfowler.com/articles/continuousIntegration.html

FOWLER, Martin. Continuous Delivery. *Martin Fowler* [online]. 2013 [cit. 2016-04-18]. Dostupné z: http://martinfowler.com/bliki/ContinuousDelivery.html#footnote-when

SHARMA Sanjeev. DevOps For Dummies®, IBM Limited Edition. Hoboken, New Jersey: John Wiley & Sons, Inc., 2014. ISBN 978-1-118-73378-3.

FITZGERALD, Brian a Klaas-Jan STO. Continuous software engineering: A roadmap and agenda. Journal of Systems and Software. Elsevier, 2015.

BADRINARAYANAN, Kabanov, James a White. IT Ops & DevOps productivity report 2013. *Rebellabs* [online]. 2013 [cit. 2016-05-1]. Dostupné z: http://pages.zeroturnaround.com/rs/zeroturnaround/images/it-ops-devops-productivity-report-2013%20copy.pdf?mkt_tok=3RkMMJWWfF9wsRouva7MZKXonjHpfsX66e0kWq6g38431UFwdcjKPmjr1YAATcB0aPyQAgobGp5I5FEATrbYTK13t60OXw%3D%3D

APPLETON, Berczuk Cabrera a Orenstein. Streamed Lines: Branching Patterns for Parallel Software Development. Hillside [online]. 1998. [cit. 2016-05-4]. Dostupné z: http://www.hillside.net/plop/plop98/final_submissions/P37.pdf

DUVALL, Paul M. Continuous Delivery Patterns and Antipatterns in the Software Lifecycle. DZone [online]. 2011. [cit. 2016-05-4]. Dostupné z: http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/lecturas/10_Continuous_Delivery_Dzone_Refcardz.pdf

Řízení životního cyklu (Development – Operation - ALM [online]. 2016 [cit. 2016-05-07]. Dostupné z: https://www.bestpractice.cz/)

RILEY, CHRIS. *Is ALM Dead in the World of DevOps?* [online]. 2015 [cit. 2016-05-07]. Dostupné z: http://devops.com/2015/04/15/is-alm-dead-in-the-world-of-devops/

Seznam obrázků