Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Continuous delivery webových aplikací

Jan Češpivo

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Jan Češpivo

Informatika

Název práce

Continuous delivery webových aplikací

Název anglicky

Continuous delivery of web application

Cíle práce

Cílem práce je implementace procesu continuous delivery pro vybranou firmu v ČR zabývající se tvorbou webových aplikací. Problém bude popsán v první části práce nejprve v univerzální teoretické rovině a v druhé části práce jako konkrétní projekt včetně závěrečné evaluace navržených postupů.

Metodika

1. Přednostní orientace na nástroje open source.

2. Dodržování pravidel systémové integrace a standardů obvyklých v softwarovém inženýrství, především UML a procesně orientovaná notace.

Doporučený rozsah práce

60-80 stran

Klíčová slova

continuous delivery, deployment, nasazování, vývoj software, devops

---

Doporučené zdroje informací

DUVALL, Paul, Steve MATYAS a Andrew GLOVER. Continuous integration. Upper Saddle River, NJ: Addison-Wesley, 2007. ISBN 03-213-3638-0

FITZGERALD, Brian a Klaas-Jan STOL. Continuous software engineering and beyond: trends and challenges. Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering – RCoSE 2014. New York, USA: ACM Press, 2014, s. 1-9. DOI: 10.1145/2593812.2593813.

HUMBLE, Jez a David FARLEY. Continuous delivery. Upper Saddle River, NJ: Addison-Wesley, 2011. ISBN 03-216-0191-2.

HU¨TTERMANN, Michael. DevOps for developers. New York: Apress, 2012. ISBN 14-302-4569-7.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 08. 03. 2016

Čestné prohlášení

Prohlašuji, že svou diplomovou práci Continuous delivery webových aplikací jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2016

Poděkování

Rád bych touto cestou poděkoval panu doc. V. Merunkovi za vedení práce a Zuzce za lingvistické poznámky a velkou podporu. Dále firmě COEX s.r.o., která mi umožnila implementovat teoretický koncept v praxi a mimo jiné i motivovala k výběru tématu.

Continuous delivery webových aplikací

Continuous delivery of web aplications

Souhrn

Práce se zabývá procesem continuous delivery webových aplikací. Teoretická část obsahuje popis a vysvětlení základních pojmů týkajících se continuous delivery. V praktické části jsou navrženy a implementovány nástroje potřebné pro implementaci procesu continuous delivery ve firmě vyvíjející webové aplikace.

Summary

This thesis is dealing with continuous delivery od web applications. The theoretical part contains description and explanation of the basics about continuous delivery. In the practical part is designed and implemented tools for final implementation of continuous delivery in software company.

Klíčová slova:

Continuous delivery, automatické nasazení, vývoj software, webové aplikace, automatizace, automatické testování

Keywords:

Continuous delivery, automatic deployment, software development, web applications, provisionig, automatic testing

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

I když není Henry Ford ten, kdo vynalezl pásovou výrobu, je to jistě první osoba na kterou si každý v souvislosti s tímto výrazným zefektivněním vzpomene. Průmyslová revoluce postupně přešla do revoluce informační, v té teď žijeme a někteří k ní i svojí prací dále přispíváme. I přesto, že se vývoj software považuje za kreativní práci, často se v ní stále setkáváme s opakujícími se až téměř rutinními činnostmi.

Je až s podivem jak běžně mezi profesionálními vývojáři slýchávám, že nestíhají termíny a jak jsou pod tlakem a když v pátek nasazují dlouhou dobu připravovaný projekt, tak už rovnou počítají s tím, že domu dorazí až pozdě večer a na víkend si raději ani nic dalšího neplánují. Přitom nasazování je právě jednou z těchto stále se opakujících se činností. Vývojář nasazuje na svůj stroj pravidelně několikrát měsíčně různé verze aplikace. Nová posila do týmu si také musí projekt na svém počítači rozhýbat. Zákazník si novou funkčnost, kterou si nedávno objednal, také musí vyzkoušet a schválit, než se objeví jako nová položka menu i jeho zákazníkům. Často to bývá na různém testovacím prostředí, které je velmi podobné, nezřídka totožné s tím produkčním.

Přitom je řešení jednoduché. Stačí být jako Henry Ford a nechat své zaměstnance, aby svoji práci zefektivnili a přišli s automatizací. Proces continuous delivery je právě tou automatickou linkou pro výrobu software, jakou je pásová výroba a dnes například systém kanban pro automobilový průmysl. Od první součástky až k zákazníkovi. Od první změny kódu až k hotovému řešení na produkci. To je účelem continuous delivery.

2 Cíl práce a metodika

Cílem teoretické části práce je seznámit se s problematikou continuous delivery a dosadit tyto informace do kontextu vývoje webových aplikací. Teoretická část tak bude základem k praktické části, kdy bude proces continuous delivery implementován do vybrané firmy v podobě demonstrace procesu na vzorové webové aplikaci. Tento proces by měl byl replikovatelný a dále zobecnitelný i na další vyvíjené webové aplikace v dané firmě.

Základem práce je důkladná literární rešerše a vymezení pojmů v teoretické části v kapitole 3 Teoretická východiska, ze které bude posléze vycházet praktická část v kapitole 4 Praktická implementace.

Teoretická část je nutná pro uvedení pojmu continuous delivery, jeho předpokladů a popisu jednotlivých technologií dostupných a využitelných k praktické realizaci procesu.

V teoretické části budou definovány obecné cíle procesu continuous delivery, tak jak je popisují dostupné zdroje. Tyto obecné cíle pak determinují konkrétní cíle praktické implementace. V teoretické části budou také definovány základní metriky pro analýzu stávající situace a evaluaci implementace procesu v závěru praktické části. V úvodu praktické části tedy budou obecné cíle uvedené v teoretické části konkretizovány a rozšířeny o další specifické cíle vycházejí z vnitřního prostředí firmy. Těžiště práce spočívá v návrhu a praktické realizaci procesu continuous delivery a demonstraci implementace procesu na vzorovém projektu.

3 Teoretická východiska

3.1 Vymezení základních pojmů

Verzovací systém je systém pro správu zdrojového kódu. Jde o mechanismus pro udržování více verzí souborů najednou. I po změně souboru jsou stále k dispozici jeho předešlé verze. Součástí je i mechanismus, skrze který mohou různí lidé pracovat na jednom a tom samém souboru (či několika najednou). Centrální úložiště zdrojového kódu se nazývá repozitář. Moderní verzovací systémy umožňují vytvářet i celé větvení verzí a efetivní práci s nimi.[23]

Skript je jednoúčelový program interpretovaný výchozími nástroji operačního systému. Často se vytváří jako automatizace opakovaně se vyskytujících problémů a je zpravidla bez interakce. Jedná se o definici v tomto kontextu používanou zdroji [23, 6, 24, 21].

Nasazení je souhrn aktivit, které vedou k tomu, že je software k dispozici pro používání.[9]

Infrastruktura

V klasickém pojetí zahrnuje pojem infrastruktura operační systémy, servery, síťové přepínače a routery. Podle jiných definicí infrastruktura zahrnuje celé prostředí organizace spolu s podpůrnými službami jako jsou firewally a monitorovací služby.[24] Dále v textu budeme pod pojmem infrastruktura uvažovat rozšířenou verzi tohoto pojmu, tedy všechno, co přímo ovlivňuje běh a použitelnost daného softwaru.

Prostředí

Prostředí umožňuje běh aplikace takovým způsobem, že přímo neovlivňuje jiná prostředí. Těchto prostředí může být více a na každém může běžet jiná verze aplikace. Speciálním případem prostředí je prostředí produkční, je pouze jedno a reprezentuje běžící aplikaci v produkčním režimu pro koncové uživatele.[4]

Continuous integration

Continuous integration je praxe při vývoji software, kdy jednotlivci či týmy pravidelně integrují svoji práci s prací ostatních. Běžně každý vývojář integruje minimálně jednou denně, což vede k několika prováděným integracím během jediného dne. Každá integrace je verifikována automatickým systémem, tak aby došlo k co nejdřívější detekci integračních chyb. Mnoho týmů zjistilo, že tento přístup vede k signifikantnímu zredukování integračních problémů a dovoluje vytvářet ucelený software výrazně rychleji.[17]

Tradiční přístup zařazuje fázi integrace, testování a kontroly kvality až po dokončení fáze vývoje. Tento přístup může vést k nepředvídatelným zpožděním na konci projektu. Chyby se mohou projevit v mnoha rozhraních různých subsystémů a skutečné příčiny chyb se velmi těžko odhalují.[25]

Continuous integration je praxe, kdy se integruje a testuje nový kód oproti stávajícímu kódu při každé změně. [1]

3.2 Definice pojmu continuous delivery

Continuous delivery je přístup, ve kterém vývojový tým produkuje v krátkých cyklech verze software, které jsou okamžitě připravené na produkční nasazení.[10]

Dle [1] nejlépe definuje proces continuous delivery Martin Fowler ve svém článku [18]:

Proces continuous delivery je prováděn právě tehdy když:

- Software je nasaditelný již v průběhu svého životního cyklu,
- nasaditelnost software má vyšší prioritu než práce na nových funkcích,
- každý může snadno a rychle získat informaci o produkční připravenosti systému jakmile je v něm provedena změna.
- lze jednoduše a samostatně nasadit jakoukoliv verzi software na jakékoliv požadované prostředí.

3.2.1 Historie

Pojem continuous delivery se poprvé veřejně objevil v roce 2010 v názvu knihy *Continuous delivery: reliable software releases through build, test, and deployment automation*[23]. Při hledání kořenů samotného procesu se však musíme dostat hlouběji do historie. Dle [25] se již v roce 1988 objevil pojem evolutionary delivery v knize *Principles of Software Engineering Management*[21] jako jeden z prvních inkrementálních přístupů k vývoji software. Z této základní idei pak vychází celé odvětví agilního přístupu k vývoji software. Tuto myšlenku rozvinul hlavně Kent Beck ve své knize *Extreme Programming: Embrace Change, Reading, Mass.*[7] a doplnil ji mnoha příklady z praxe. Martin Fowler spolu s Matthew Foemmel inspirováni touto knihou poprvé v článku [19] definují proces continuous integration. V roce 2006 pak Fowler ve svém článku [17] pojem revidoval a autoři Jez Humble, Chris Read a Dan North na konferenci AGILE 2006 představují článek *The Deployment Production Line.*[22] a následně v roce 2007 vychází kniha *Continuous integration: improving software quality and reducing risk*[13]. Jez Humble se tématu dále věnuje a spolu s David Farley vydávají v roce 2010 knihu zmíněnou na začátku kapitoly [23] a definují tak proces continuous delivery.

3.3 Cíle procesu continuous delivery

V zásadě lze cíle procesu continous delivery definovat již na základě definice od Martina Fowlera z kapitoly 3.2 Definice pojmu continuous delivery Tyto cíle však nejsou samoúčelné a existují pro ně důvody. Abychom mohli definovat v následující kapitole 3.4 Přínosy zavedení procesu continuous delivery přínosy procesu continuous delivery v softwarovém inženýrství, je třeba ukázat nejdříve na některé problémy, které v procesu vývoje software existují a které predeterminují důvody pro snahu o vytvoření procesu continuous delivery [23].

Dle [18] jsou hlavní přínosy zavedení procesu continuous delivery:

- Snížení rizika nasazení: Protože jsou nasazovány menší změny, je zde menší šance, že se něco pokazí a je jednodušší problém opravit.
- Uvěřitelný pokrok: Mnozí sledují postup změn při vývoji softwarového projektu kontrolou, jak jsou jednotlivé práce na projektu „hotovy“. Je mnohem méně uvěřitelné, pokud „hotovo“ znamená, že „vývojáři tvrdí, že je to hotové“ oproti tomu, pokud je změna nasazena na produkčním (nebo produkčně podobném) prostředí.
- Uživatelská odezva: největší riziko pro tvorbu software je, že na konci je vytvořeno něco, co ve skutečnosti není použitelné a neplní tak svůj účel. Čím dříve a častěji je funkční software ukázan skutečným uživatelů, tím rychleji se získá zpětná vazba o tom, co skutečně uživatel potřebuje.

Další důvody pro vytvoření procesu continuous delivery jsou:

- Čím později je chyba odhalena tím je vyšší cena na její opravu [7, 25] Chyby jsou nesnáze odhalitelné a také opravitelné v bodě, kdy jsou zaneseny. Když jsou odhaleny později, je vždy náročnější je opravit. Vývojáři zapomínají, co dělali v čase, když byla chyba do systému zanesena a funkcionalita se mezitím mohla změnit. Defekty jsou nejsnáze (nejlevněji) odstraněny, pokud se ani nedostanou do verzovacího systému jako změna. [23]
- O produkční prostředí se v mnoha organizacích starají úplně jiní lidé než ti kteří vyvíjejí software, který na něm běží. To vede ke konfliktům mezi oběma odděleními a při problémech s nasazením nebo při objevení chyby při běhu na produkčním prostředí ke svalování viny z jedné strany na druhou. [24]

Návrhy řešení uvedených problémů spolu z dalšími konkrétními důvody a cíli jsou uvedeny v následujících podkapitolách.

3.3.1 Automatizace procesu nasazování

> „V komplexním prostředím může být testování a nasazování obtížný a časově náročný proces zahrnující aplikační servery, infrastrukturu pro přenos dat a rozhraní na externí systémy. Viděli jsme nasazování, které zabralo několik dní i přes to, že vývojový tým používal automatické sestavovací systémy k ujištění, že je jejich kód plně otestován.“ [22]

V [23] jsou uvedeny tyto konkrétní příklady problémů:

- Pokud není proces nasazení plně automatizován, objeví se chyby v každém nasazení. Jedinou otázkou je, zda-li to jsou či nejsou chyby závažné. I s excelentním testováním nasazovacího procesu jsou chyby nasazení velmi obtížně odhalitelné.
- Pokud není proces nasazení automatizován, není opakovatelný ani spolehlivý a vede k promarněnému času při řešení chyb nasazení.
- Manuální proces nasazení musí být dokumentován. Udržování dokumentace je tak komplexní a časově náročný úkol vyžadující spolupráci mezi několika lidmi a tedy vždy v nějakém čase vede k obecně nekompletní a zastaralé dokumentaci. Sada automatizačních nasazovacích skriptů slouží jako dokumentace a vždy bude aktuální a kompletní, protože jinak nasazení nebude funkční.
- Automatické nasazování podporuje spolupráci, protože všechno je explicitně uvedeno ve skriptu. Dokumentace předpokládá nějakou danou úroveň znalosti čtenáře a reálně je běžně psaná pouze jako pomůcka pro osobu, která provádí nasazení. To vede k nečitelnosti dokumentace pro ostatní.
- Důsledkem výše uvedeného tvrzení je, že manuální nasazení je závislé na nasazovacím specialistovi. Pokud je tento na dovolené nebo v týmu již nepracuje, nastává problém.
- Provádění manuálních nasazení je nudná a opakující se činnost a ještě vyžaduje jistý stupeň expertních znalostí. Vyžadovat od expertů nudnou a opakující se činnost, která je navíc technicky náročná je ve většině případů jistá cesta k lidské chybě. Automatické nasazování uvolní drahým a vysoce kvalifikovaným pracovníkům ruce k práci na hodnotnějších činnostech.
- Jediný způsob jak testovat manuální proces nasazení je provést ho. Toto je často časově náročné a tudíž drahé. Automatický proces nasazování je levné a jednoduché otestovat.
- V manuálním procesu nasazování neexistuje záruka, že bude správně následována dokumentace. Pouze automatizovaný proces je plně kontrolovatelný.

Proces continuous delivery dle [23] přináší automatizaci procesu nasazení, tak aby byl:

- replikovatelný,
- a spolehlivý.

3.3.2 Sjednocení nasazování na různá prostředí

V tradiční přístupu se nasazení na produkční prostředí provádí až po dokončení vývoje, který dosud kompletně probíhal na vývojovém prostředí. [23] Mnoho společností má také dedikované oddělení pro testování software. Protože nasazování software a jeho údržba často vyžaduje odlišné schopnosti než samotný vývoj software, je vytvořeno speciální produkční oddělení. Rozdělení pracovních oblastí vypadá velmi výhodně i pro management. Navíc specializovaný tým může mít vlastního manažera, který plní individuální požadavky potřebné pro toto specifické oddělení. Naneštěstí i když obě oddělení pracují na stejném projektu, nesdílí stejné cíle. Zatímco produkční oddělení má za cíl stabilitu a vyhýbá se změnám v software pro zajištění neměnných podmínek pro produkční systémy, naopak vývojové oddělení produkuje změny (nové funkce, opravy chyb a práce na základě změnových požadavků) a tyto změny potřebuje aplikovat do produkčního prostředí. [24]

Dále dle [23]:

- Testeři musí testovat systém na vývojovém prostředí, které je ve své podstatě odlišné od produkčního.
- Vydání nové verze je prvním kontaktem lidí, kteří se starají o produkční prostředí, s novou verzí software. V některých případech dokonce existují jiné týmy pro nasazení na předprodukční (staging) prostředí a na produkční. Pak se stává, že lidé zodpovědní za produkční prostředí se seznámí s novou verzí i pouhý den před nasazením.
- Protože vytvoření prostředí, které je identické k produkčnímu je časově náročné a samotné produkční prostředí je velmi citlivé, přístup je striktně kontrolován.
- Vývojový tým vytváří instalátory, konfigurační soubory, databázové migrace a dokumentaci k nasazení pro pracovníky, kteří nakonec skutečně nasazení provádí. Toto všechno není testováno v prostředí, které se chová přesně jako produkční.
- Je zde velmi málo spolupráce mezi vývojovým týmem a lidmi, kteří provádějí nasazení.

Dle [23, 24] proces continuous delivery sjednocuje nasazení na různá prostředí, tak že je proces:

- testovatelný,

- jednotný pro všechna prostředí.

Zdroje [23, 24] uvádí, že pro uspokojivé řešením je nutná úzká spolupráce mezi vývojovým týmem a týmem starajícím se o nasazení a běh aplikace, odpovědnost za nasazení je tak společná oběma týmům a oba týmy mají přístup do nástrojů, které automatizují nasazení na prostředí (vizte předchozí kapitola3.3.1 Automatizace procesu nasazování).

3.3.3 Automatizované vytváření a konfigurace prostředí

Manuální konfigurace produkčního prostředí probíhá prostřednictví týmu operations. Když je třeba provést změnu, je provedna příno na produkčních serverech. V případě zaznamenávání takovéto změny, je proveden záznam v databázi změn. Bez automatizace pak dle [23]:

- trvá příprava prostředí pro nasazení dlouho
- nelze se vrátit o krok zpět k dřívější konfigurace systému, toto zahrnuje operační systém, aplikační server, web server nebo i například nastavení infrastruktury
- Servery v clusteru mohou mít, neúmyslně, odlišné verze operačního systému nebo knihoven třetích stran.
- Konfigurace se provádí modifikací konfiguračních souborů přímo na produkčních systémech a může dojít k chybnému nastavení a tím k jejich nefunkčnosti.

Plná automatizace vytváření prostředí a jeho konfigurace umožňuje vytvářet nová prostředí snadno a rychle „na požádání“. Změny v konfiguraci jsou prováděny automatizovaně, manuální změny jsou zapovězeny. Konfigurace je uložena ve verzovacím systému, takže se lze vždy vrátit k jakémukoliv bodu a není potřeba jiná další databáze změn konfigurace. [24, 35, 23]

3.3.4 Automatizace testování

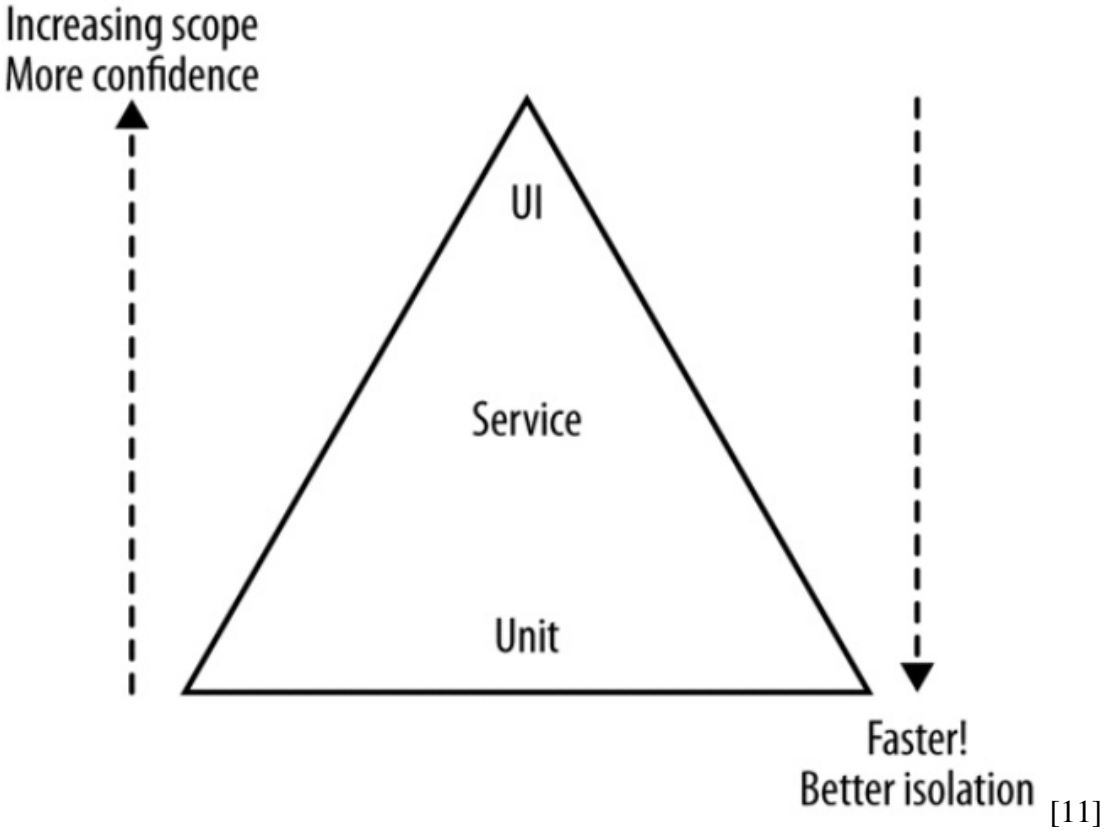
Chyby by měly být detekovány co nejdříve. Automatické testy poskytují rychlou zpětnou vazbu na stav a kvalitu software. [24] Ne však všechny testy mohou být automatizovány, u testů použitelnosti, nebo exploratorních testů je třeba lidská strana. [12]

Jedno z často uváděných rozdělení testů je [11]:

- jednotkové testy (také testy submodulů)
- testy služeb (nazývané také jako testy komponent, nebo testy API)
- testy uživatelského rozhraní

Testy by měly probíhat v takovém pořadí, aby nejdříve proběhly testy, které testují minimální množství programového kódu a mířící na konkrétní body v aplikaci a jsou tak velice rychlé. Úplně nakonec by měly proběhnout testy, které testují celé sady funkčností a mohou trvat výrazně déle. [11] Toto ilustruje obrázek 3.1.

Obrázek 3.1: Pyramida automatických testů



[11]

Některé zdroje nabízejí rozdělení testů dle svého rozsahu a odpovídají testovacím fázím vývoje [25, 23]:

- jednotkové testy
- integrační testy
- systémové testy

Tuto variantu lze také analogicky zobrazit jako pyramidu. Obě varianty jsou si defakto sobě rovné a z velké části se překrývají. Pro účely této práce budeme používat druhého typu rozdělení.

Pyramidální rozdělení v sobě nese ještě jedno další pravidlo. Testů malého rozsahu (jednotkových testů) musí být daleko více než ostatních testů většího rozsahu. Například jeden celosystémový test pokryje velkou oblast kódu a pro kompletní pokrytí jich oproti jednotkovým

stačí pouze několik. Je důležité aby pokrytí rychlými jednotkovými testy bylo co nejvyšší, tak aby se využila jejich největší výhoda, kterou je rychlost jejich provádění. Systémové testy mohou probíhat velmi dlouho a tak se může stát, že je defekt odhalen s velkým zpoždění. [27]

Problematika testování i samotného vytváření testů je velice široká a i když hraje v procesu continuous delivery velmi důležiou roli, přesahuje již rámec této práce. Více informací lze najít v [11, 12, 25]

3.3.5 Automatická správa dat

Pro testování a nasazování různých verzí software musí být pro každé prostředí k dipozici data ve struktuře odpovídající verzi aplikace. Pokud nasazujeme nový kód aplikace, nelze většinou smazat všechna nasbíraná data z předchozí verze. Toto představuje problém, pokud potřebujeme s novou verzí aplikace zároveň modifikovat strukturu nebo i obsah dat. V tomto případě je třeba zavést mechanismus, který dokáže automatizovaným způsobem data zmigrovat z jedné podoby do druhé. [23]

Je třeba implementovat oboustranný migrační mechanismus, který nejen dokáže zmigrovat data ze starší verze do nové, ale i zpětně nová data do staré verze. Toto však není vždy uskutečnitelné, protože existují případy, kdy se dopřednou migrací část informace ztratí a tedy zpětnou migrací nelze dojít ke stejnému stavu, jaký byl před započetím procesu. [5, 23]

Existují tři základní strategie pro alespoň částečné řešení a zmírnění důsledků tohoto problému [23]:

1. Zálohování v době nasazení. V době nasazení nové verze jsou veškerá data zálohována a uložena pro případ nutnosti zpětně zmigrovat. Aby nebyla veškerá nová data ztracena, zmigrují se do starší verze všechny nové záznamy.

2. Verzování datových struktur. Původní data zůstanou zachována a paralelně k původní struktuře se vytvoří nová jako její kopie s odlišným názvem (často obsahující verzi aplikace). Tato kopie je pak standardně zmigrována. Při zpětné migraci se pouze zmigrují nové a aktualizované záznamy struktury a uloží se zpětně do staré struktury. Velkou výhodou tohoto řešení, je schopnost aplikace běžet v několika verzích najednou, migrační skripty pak běží neustále, tak aby v nové i staré verze byly aktuální záznamy.

3. Zpětná kompatibilita verzí. Nová verze pouze rozšiřuje schopnosti starší verze. Pokud je třeba nových nekompatibilních struktur, vytvoří se paralelní struktura tak jako ve scénáři 2 a data jsou ukládána jak do těchto nových, tak do starých v původním formátu. Tento přístup nemusí být vždy plně realizovatelný (např. když nejsou data potřebná pro některou verzi vůbec sbírána z uživatelského rozhraní) Navíc protože tento přístup vyžaduje koexistenci různých mechanismů v několika verzích, které ukládají několikerým způsobem do databáze, zvyšují se nároky na testování a může se i výrazným způsobem

snížit výkon výsledné aplikace. Často se z těchto důvodů implementace omezuje pouze na poslední stabilní verzi, takže zároveň běží pouze dvě verze. V modifikovaném scénáři je pak po dostatečně dlouhé době starší verze z aplikace vyřazena a se zpětnou migrací se tak již nepočítá.

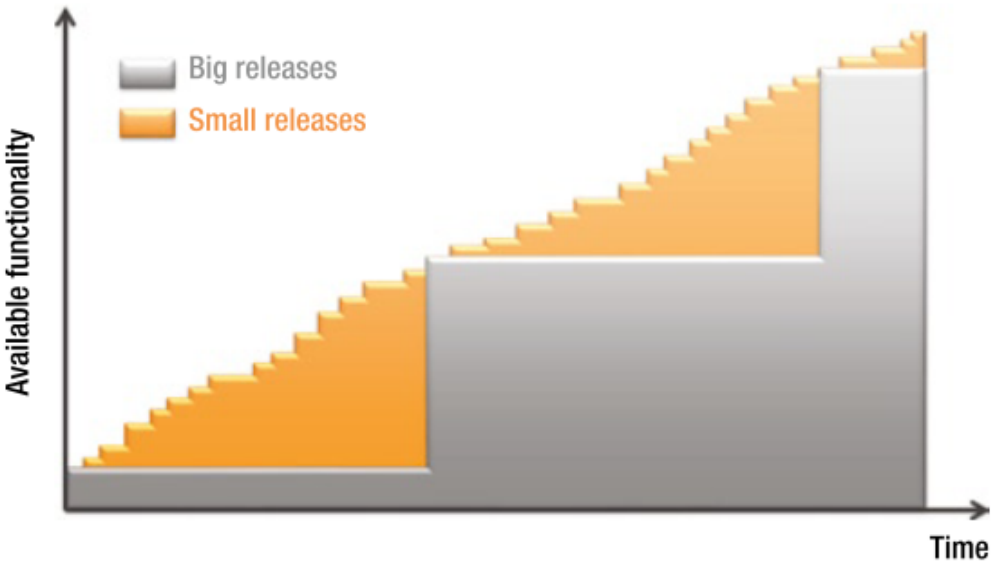
Vyčerpávajícím způsobem se migračními a refaktorizačními technikami datových zdrojů a databází zabývá například [5].

Ne na všech prostředích lze pracovat s kompletními produkčními daty, jejich objem může být příliš velký. Ve většině případů to dokonce nebývá ani žádoucí. Na testovacích prostředích je třeba pracovat pouze s takovými daty, pro které lze efektivně aplikaci testovat. Na vývojových prostředích je zase třeba data z produkčního prostředí v opodstatněných případech dodatečně anonymizovat. [23, 30] Detailní informace o anonymizačním procesu lze najít například v [30].

3.4 Přínosy zavedení procesu continuous delivery

Základní přínos zavedení continuous delivery, tak jak byl popsán v předchozí kapitole, je vytvoření procesu, který je opakovatelný, spolehlivý a predikovatelný a který zredukuje nutný čas na vydání nové verze na minimum a umožní dostat nové funkcionality a opravy chyb v software rychle k uživateli. [23] Jak ilustruje obrázek 3.2, při nasazování velkých sad změn je nová funkcionalita nasazena na produkci později než při nasazování menších sad změn. [24, 6] Kromě toho existují i další přínosy, které bychom původně ani neočekávali. [23]

Obrázek 3.2: Nasazování velkých sad změn oproti nasazování malých sad



[24]

Schopnost jednoduše nasadit jakoukoliv verzi software na jakékoliv prostředí přináší mnoho výhod: [23]

- Testeři mohou porovnávat starší verze aplikace, aby verifikovali, zda-li se chování nové verze skutečně změnilo.
- Podpora může nasadit poslední verzi na své prostředí jen aby reprodukovali nahlášený defekt v aplikaci.
- Nasazení nové verze lze provést pouhým stiskem tlačítka.
- Jednotlivé týmy lépe a více spolupracují, protože nasazování je jejich společná práce.

Dále se objevují další výhody nasazení procesu:

- Frekvence vydávání nových verzí se dramaticky zvýší a tím se výrazně zrychlí i nasazení aplikace do skutečného produkčního využití na trhu. [10]
- Nasazováním malých sad změn dostává vývojový tým velmi brzo zpětnou vazbu od uživatelů a může se soustředit pouze na použitelné funkcionality. [6, 10, 18]
- Vytváření vývojového prostředí je velmi snadné. Dříve trávili vývojáři, testeři i operations velmi mnoho času nastavováním a opravou vlastního vývojového prostředí. Po nasazení continuous delivery se výrazně zvýší efektivita tohoto procesu a tím se zvýší i celková produktivita vývojového týmu. [10]
- Proces nasazení na produkční prostředí je ze své podstaty velmi spolehlivý, riziko při nasazení se minimalizuje. Testování procesu probíhá automaticky při každé změně. Nasazení do produkce je podmíněno otestováním tohoto procesu na dané verzi. [18, 10] Některé implementace procesu continuous delivery umožňují i zpětné nasazení původní verze a tudíž ještě dále snižují riziko chyby. [10]
- Kvalita výsledného produktu se výrazně zvýší. Vzhledem k automatizaci procesu testování a nasazení se významné chyby v produktu téměř neobjevují. [10]

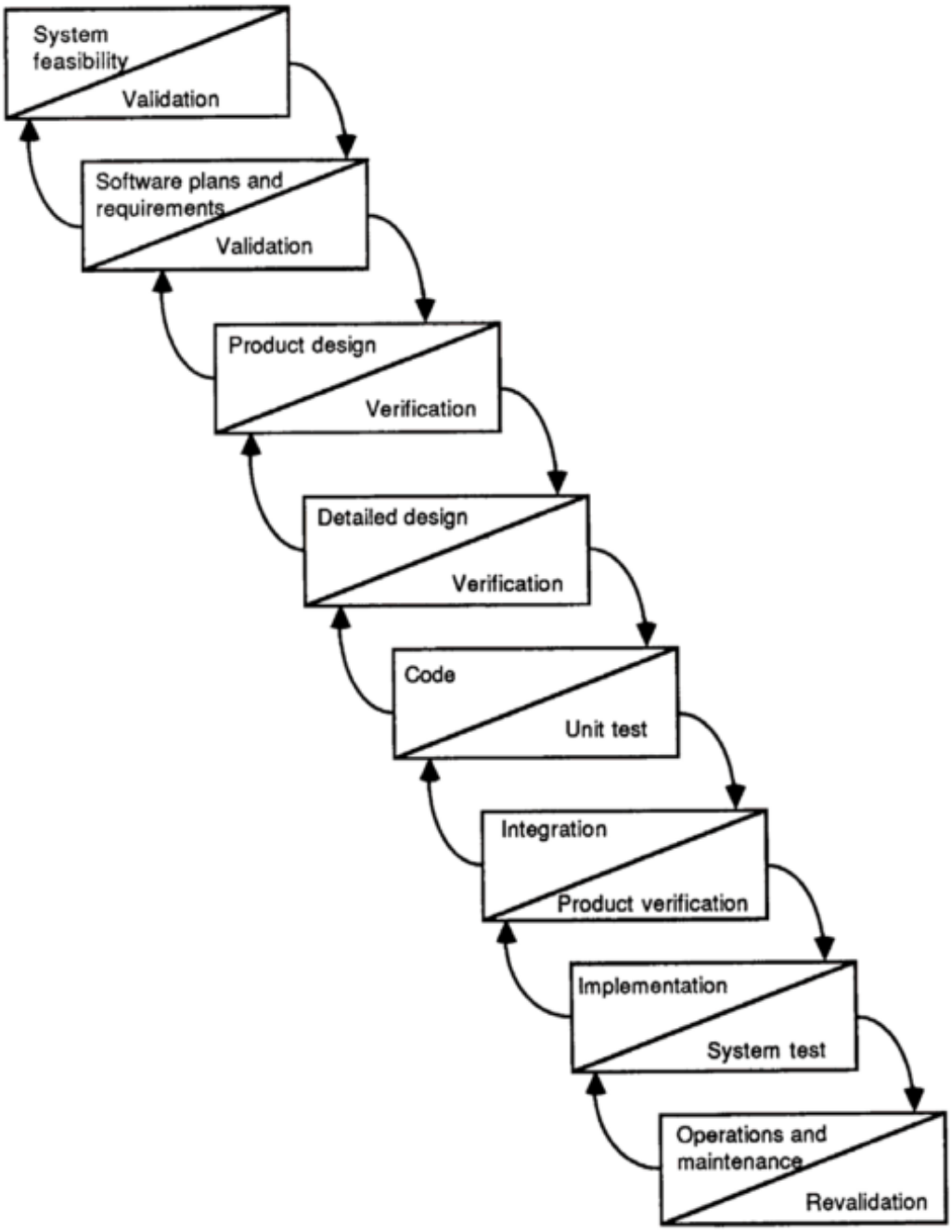
3.5 Implementace procesu continuous delivery

V předchozích kapitolách byl popsán proces continuous delivery jako přístup (či metodika) i se svými cíly a možnými přínosy. Pro uvedení do praxe je ale třeba celý proces implementovat do celého procesu vývoje aplikace.

Standardní vodopádový model vývoje aplikace, tak jak je zobrazen na obrázku 3.3, je základním kamenem modelu vývoje software. Ve své podstatě je obsažen i ve spirálovém

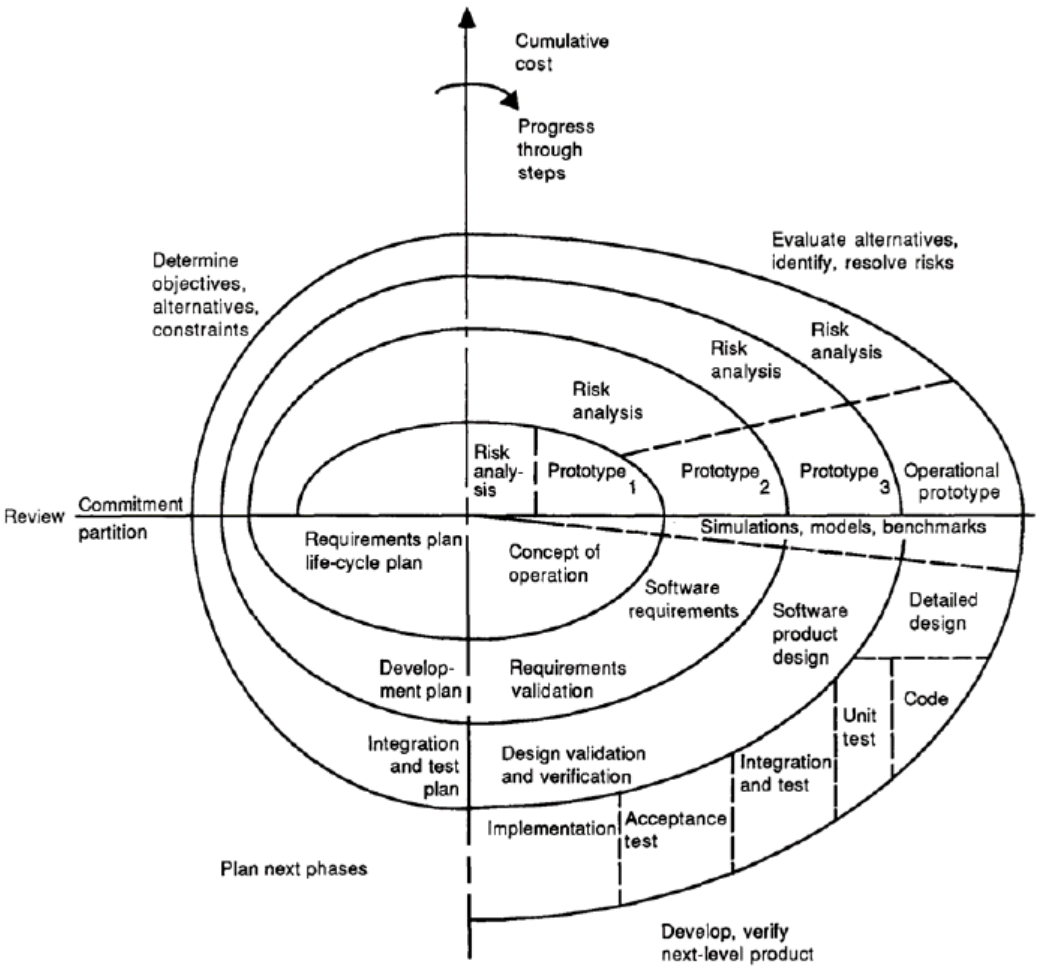
modelu [8] viz obrázek 3.4. Požadavky na jednotlivé kroky se nezměnily ani v agilním přístupu jen se zmenšil objem zpracovaných informací a snížilo množství práce v každém kroku. Kroky se pak iterativně opakují, dokud není aplikace dokončena. [8, 6]

Obrázek 3.3: Vodopádový model vývoje software



[8]

Obrázek 3.4: Spirálový model vývoje software



[8]

Proces continuous delivery zahrnuje automatizaci testování subsystémů a jejich integraci stejným způsobem jako je tomu v procesu continuous integration, z něhož ostatně historicky vychází - jak již bylo zmíněno v kapitole 3.2.1 Historie. Proces continuous integration je nutná součást procesu continuous delivery [23, 18, 1].

V rámci klasického modelu vývoje software proces continuous delivery vyžaduje automatizaci následujících základních podprocesů:

- testování subsystémů a integrační testování - continuous integration,
- zprovoznění infrastruktury a nasazení na testovací prostředí pro automatické či manuální testování,
- zprovoznění infrastruktury a nasazení aplikace na produkční prostředí

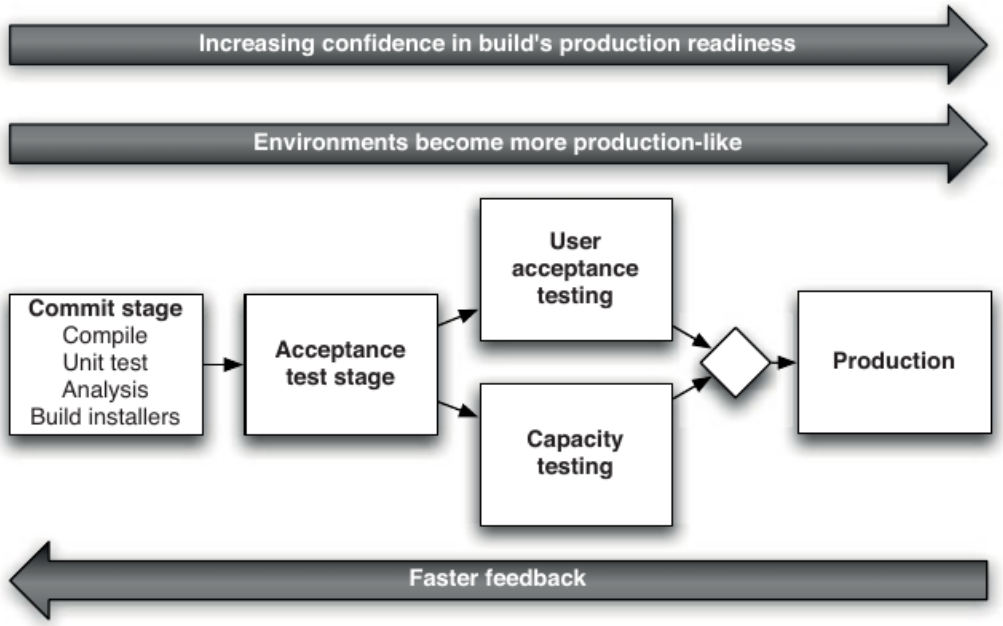
3.5.1 Model implementace procesu

Model implementace procesu continuous delivery se dle [22] nazývá „The deployment production line“ či dle [23, 18] jako „Deployment pipeline“. Pro účely této práce budeme používat druhého uvedeného termínu.

Proces continuous delivery zahrnuje několik fází testování a nasazení na různá prostředí a vyžaduje spolupráci mnoha osob, rolí či týmů. Deployment pipeline tak jak zde bude prezentována je modelem praktické realizace tohoto procesu a je i předlohou jeho konkrétní implementace v podobě automatizačního systému. [23] Proces continuous delivery nepředepisuje konkrétní implementaci a tak ani výsledný automatizační systém nemusí mít nutně podobu tak jak bude v této kapitole prezentován.

Deployment pipeline je realizací procesu změny v aplikaci postupně jednotlivými prostředími od vývojového až po produkční. Změna prochází jednotlivými kroky. Každému kroku odpovídá automaticky či poloautomaticky (na požádání) vytvořené izolované prostředí. Změna v aplikaci může do každé další fáze přejít teprve poté, co úspěšně prošla všemi fázemi předchozími. V některých případech může být proces continuous delivery a tedy i jeho implementace realizována s větveními. [23, 24, 1] Toto je ilustrováno na obrázku 3.5.

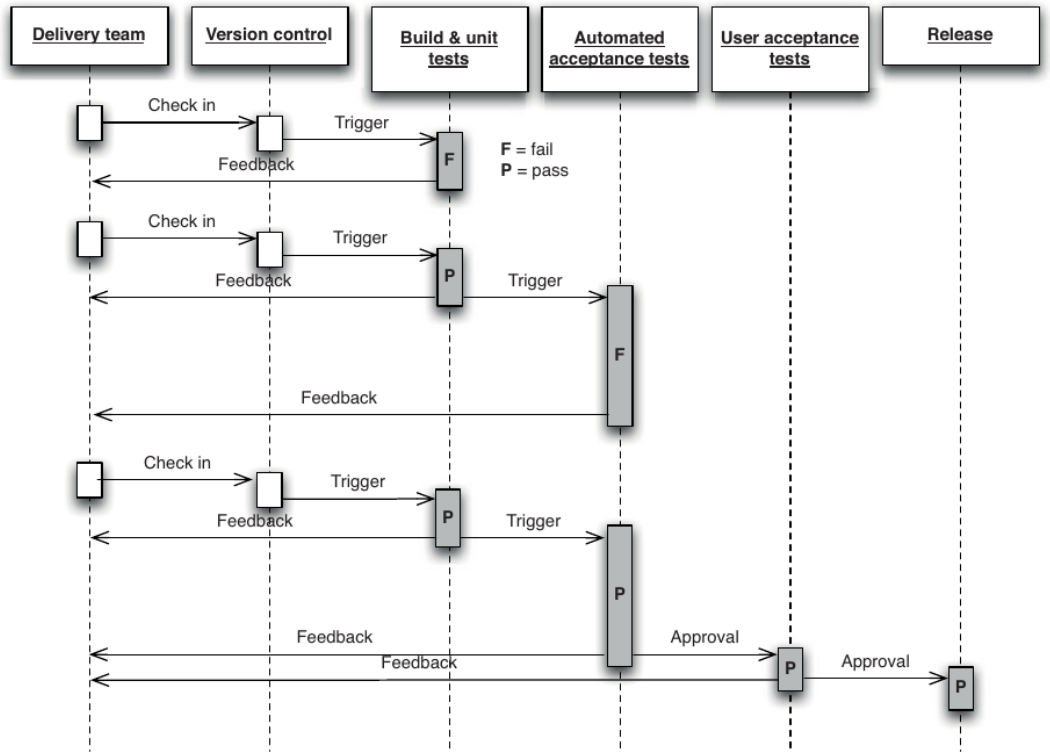
Obrázek 3.5: Deployment pipeline



[23]

Příklad části sekvenčního diagramu v notaci UML, který reprezentuje vnitřní mechanismus modelové deployment pipeline je zobrazen na obrázku 3.6

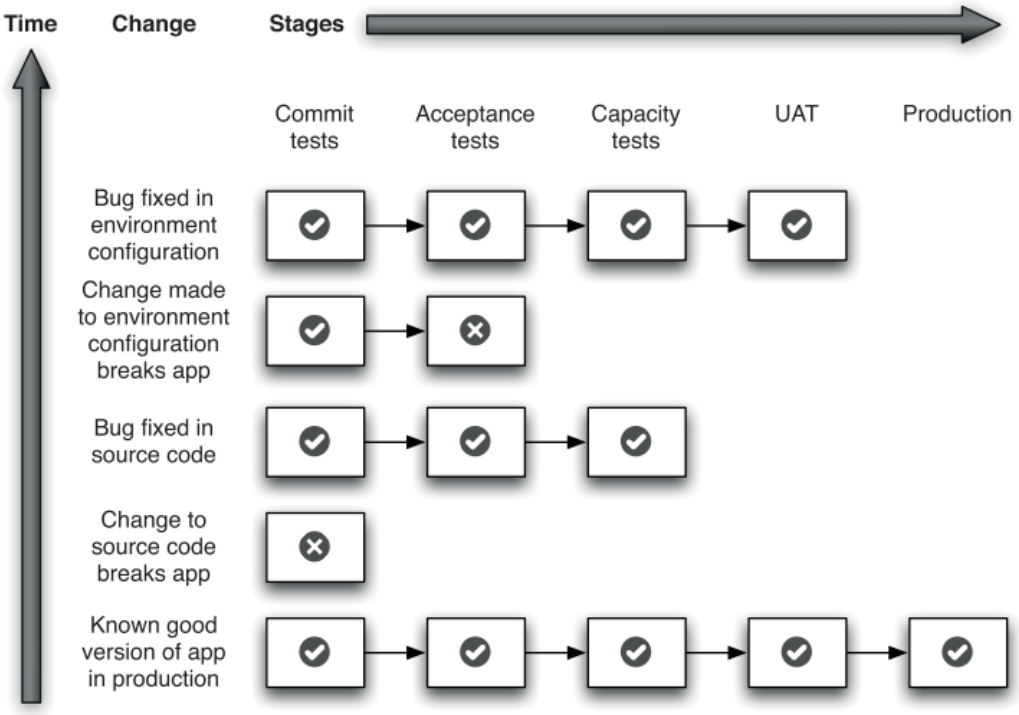
Obrázek 3.6: Sekvenční diagram pro deployment pipeline



[23]

Obrázek 3.7 zobrazuje modelový postup v procesu continuous delivery, tak jak probíhá postupně v čase. Na vertikální ose je čas a na horizontální jednotlivá prostředí. V procesu continuous delivery prochází změna v kódu aplikace jednotlivými prostředími celé deployment pipeline a postupně je v jednotlivých fázích validována a případně propouštěna dále.

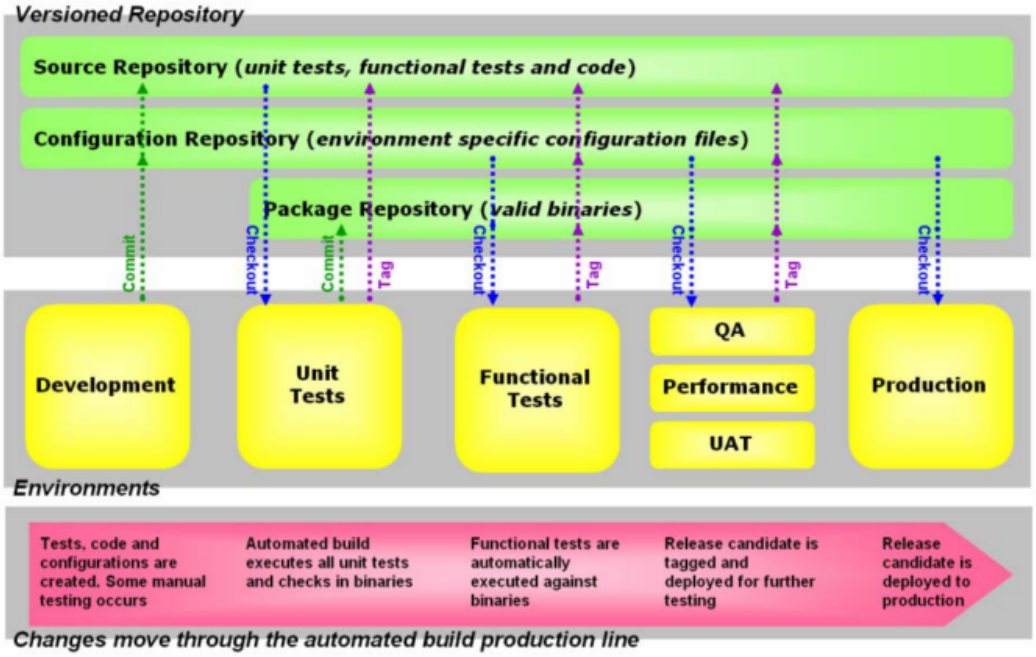
Obrázek 3.7: Změny v čase pro jednotlivá prostředí v modelové deployment pipeline



[23]

I když je vývojové prostředí součástí procesu continuous delivery, ve výše prezentovaném modelu chybí. Vývojářský stroj nelze ve většině případů snadno zahrnout do automatizačního mechanismu a tak je někdy tento krok v modelu vynecháván a v praxi je pak prováděn jako samostatný krok dle aktuální potřeby vývojáře. Pokud však neuvažujeme nad automatizačním procesem pouze v rámci jeho konkrétní implementace jako automatizovaného systému, neměla by táto fáze v modelu chybět - vizte obrázek 3.8.

Obrázek 3.8: Deploment pipeline se zahrnutým vývojovým prostředí / vývojovou fází.



[22]

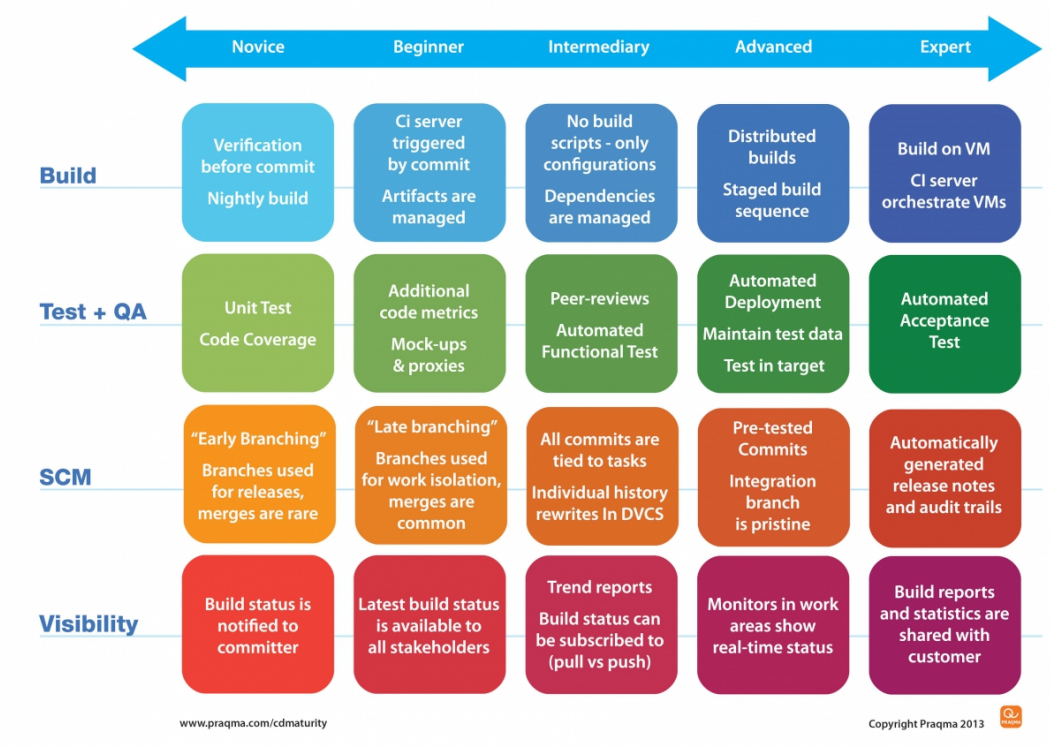
3.5.2 Hodnocení implementace procesu continuous delivery

Dle [28] lze hodnotit stupeň implementace procesu continuous delivery ze 4 hledisek:

- Integrace a sestavení aplikace a konfigurace prostředí
- Testování, kontrola kvality a automatizace nasazování
- Práce s verzovacím nástrojem
- Viditelnost změn

Jak je vidět na obrázku 3.9 hodnocení procesu probíhá pomocí matice, kde na vertikále jsou jednotlivá hlediska a na horizontální ose stupeň zvládnutí procesu od nováčka až po experta.

Obrázek 3.9: Hodnocení implementace procesu continuous delivery.



[28]

Hodnocení tedy probíhá v jednotlivých oblastech a výsledek může být bodově ohodnocen:

- Proces se neprovádí - 0 bodů
- Nováček - 1 bod
- Začátečník - 2 body
- Středně pokročilý - 3 body
- Pokročilý - 4 body
- Expert - 5 bodů

Pokud se při hodnocení v jakékoliv oblasti objeví 0 bodů, nelze vůbec mluvit o procesu continuous delivery.

Hodnocení lze rozšiřovat směrem „dolů“ o další aspekty procesu. Lze diskutovat, zda se uvedený přehled hodí na všechny případy, ale je vhodný jako základ pro další přesnější hodnocení. [28]

3.5.3 Používané technologie a nástroje

Implementace celého procesu by nebyla možná bez odpovídajícího rozvoje technologií nebo by byla velice náročná a drahá. Technologie a nástroje uvedené v následujících kapitolách

také svým způsobem determinují dnešní možnosti continuous delivery. S postupným vývojem těchto technologií se tak dále zvyšuje potenciál celého procesu. [24]

3.5.3.1 Verzování kódu

Zakladním kamenem continous delivery je pokročilá práce s verzovacím systémem včetně vytváření jednotlivých vývojových větví. Proto je nutné mít k dispozici nástroj, který má všechny požadované vlastnosti.

Mezi nejpokročilejší verzovací nástroje patří například Git, Mercurial, Perforce nebo BitKeeper. Narozdíl od starších nástrojů (jako je SVN nebo CVS) jde o nástroje decentralizované. Toto přináší velké množství výhod, hlavně co se týče efektivity práce. Pro založení nové větve nebo uložení nové revize není třeba komunikace se serverem. Synchronizace s prací ostatních se provádí dle domluvy a osobních preferencí každého vývojáře. I přes nesporné výhody decentralizovaných nástrojů, může být někdy centralizovaný verzovací systém dostačující a plně vyhovující. [23]

3.5.3.2 Virtualizace

Virtualizace je technologie, která poskytuje abstraktní vrstvu mezi fyzickým hardware a operačním systémem a aplikacemi, které v něm běží. [33]

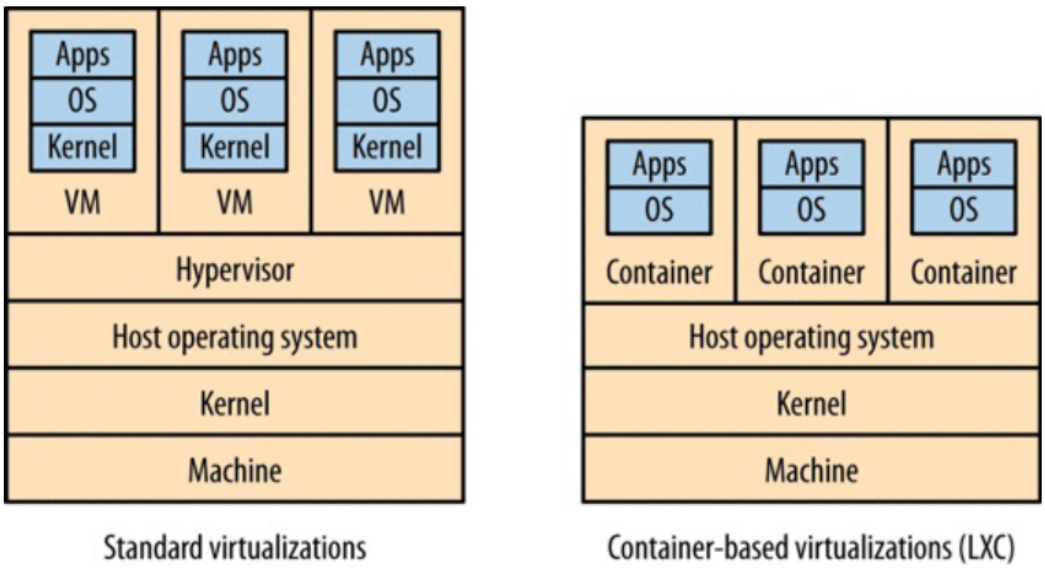
Standardní typ virtualizační technologie virtualizuje celé rozhraní fyzického stroje. [27] Tento typ virtualizace umožňuje spouštět na jednom fyzickém hardware několik izolovaných operačních systémů.

Dle druhu a úlohy operačního systému, který běží přímo na fyzické vrstvě a dále na podmíněné spolupráci s hardware, případně přidané podpoře u hostovaného operačního systému, lze dále virtualizaci klasifikovat na [33, 27]:

- Plnou virtualizaci - „Typ 2“,
- s podporou hardware - „Typ 1“,
- paravirtualizaci - „Typ 0“.

Vedle těchto standardních typů virtualizace existují virtualizace kontejnerové, kdy hostované operační systémy sdílejí jedno jádro hostujícího operačního systému. Rozdíl mezi standardní virtualizací (Typ 2) a kontejnerovou je zobrazen na obrázku 3.10.

Obrázek 3.10: Rozdíl mezi standardní virtualizací a kontejnerovou.



[27]

Velkou výhodou této takzvaně „odlehčené“ virtualizace [27] je téměř neznatelná ztráta výkonu, který se téměř neliší od nevirtualizovaného prostředí [15]. Nevýhodou je pak to, že nelze v kombinovat několik odlišných operačních systémů (například Linux a Windows), protože jednotlivá virtualizovaná prostředí sdílejí jedno jádro operačního systému. [33]

Pomocí virtualizace lze vytvořit celou virtuální infrastrukturu. Pro vytváření dalších prostředí, která jsou nutná pro implementaci procesu continuous delivery tak není třeba dalšího fyzického hardware. Jedinou nutnou podmínkou je schopnost softwaru vytvářet virtuální infrastrukturu automatizovaným způsobem. Nutná je tedy existence dokumentovaného aplikačního rozhraní (API) případně rozhraní příkazové řádky. Běžné grafické rozhraní aplikace je nedostačující. [23, 16]

3.5.3.3 Automatizace konfigurace

V souvislosti s automatizací vytváření a konfiguraci prostředí se objevily nástroje, které umožňují vytvářet a konfigurovat celou potřebnou infrastrukturu automatizovaným způsobem. [24]

Tyto nástroje lze klasifikovat dle způsobu nahrávání konfigurace na prostředí na: [20]

- agentní, které si stahují konfiguraci z centrálního serveru
- bezagentní, které konfigurují prostředí přímo.

Dále lze klasifikovat nástroje podle přístupu ke konfiguraci stroje na:

- deklarativní - popisující, jaký má být konečný výsledek konfigurace,
- imperativní, které definují příkazy, tak aby byla dosažena změna v konfiguraci,

- hybridní, které kombinují jak deklarativní tak imperativní přístup.

Dostupné nástroje s jejich klasifikací shrnuje tabulka 1.

Tabulka 1: Klasifikace konfiguračních nástrojů

Nástroj	Způsob nahrávání konfigurace	Přístup ke konfiguraci
Ansible	Bezagentní	Hybridní
Chef	Agentní	Imperativní
Otter	Bezagentní	Hybridní
Puppet	Agentní	Deklarativní
SaltStack	Bezagentní	Hybridní

3.6 Specifika procesu continuous delivery webových aplikací

Hlavním z důvodů velkého rozšíření webových aplikací je způsob distribuce klientské aplikace koncovým uživatelům. Aplikace je distribuována skrze webový prohlížeč a není tak třeba dalšího dodatečného softwarového vybavení. [3]

Proces continuous delivery webových aplikací tak nemusí pro webové aplikace řešit technologii nasazení na koncová klientská zařízení, stačí aby byla vyřešena serverová část aplikace. Toto zjednodušení ale nevylučuje klientskou část z procesu testování, tak jak je popsáno v kapitole 3.3.4 Automatizace testování. Stejným způsobem, jako je testována serverová součást, musí být testována i klientská. Vzhledem ke komplexnosti webových prohlížečů a odchylek jejich chování mezi různými typy, verzemi a platformami, může být zvláště pak testování uživatelského rozhraní velmi složité. [23, 12]

S výhodou je v implementaci procesu continuous delivery pro webové aplikace využita jejich schopnost distribuovat se a automaticky nasazovat na klientské stanice.

Serverová část webové aplikace může běžet na jakékoliv platformě. V posledním průzkumu společnosti Netcraft (březen 2016) se pro běh webových aplikací nejčastěji používají webové servery:

1. Apache (32,4%),
2. Windows Server (31,7%),
3. nginx (14,3%).

Další webové servery dosahují podílu maximálně 2%. [26]

4 Praktická implementace

Praktická implementace procesu continuous delivery bude provedena ve firmě COEX s.r.o. v oddělení vývoje webových aplikací.

V následující kapitole bude nejdříve analyzován a vyhodnocen stávající stav. Dle tohoto hodnocení bude navrženo základní řešení, definovány konkrétní cíle implementace procesu continuous delivery ve firmě COEX s.r.o. a navrženo schéma realizované deployment pipeline (kapitola 3.5.1 Model implementace procesu) a její funkční prvky.

4.1 Analýza stávajícího stavu

Firma dlouhodobě udržuje cca 30 webových aplikací o různé velikosti a na odlišných infrastrukturách. Každý rok tento počet roste přibližně o 5. Aplikace jsou vyvíjeny na klíč dle požadavků zákazníka. Většina aplikací beží na infrastruktuře o jednom serveru, který zastává jak webovou, aplikační tak i databázovou funkci. Největší aplikace pak běží na 6 samostatných strojích. Ke všem infrastrukturám má firma administrátorský přístup.

Proces vývoje, testování a nasazování aplikací je řešen pro každou aplikaci zvlášť. I přesto lze najít jednotící prvky, které jsou popsány v následujících podkapitolách.

4.1.1 Technologie

Nejvíce je ve firmě pro vývoj aplikací používán jazyk Python, dále následuje PHP. Aplikace psané v Pythonu využívají hlavně Django případně mikroframework Flask. U PHP je ve valné většině použito CMS ModX. Jako webový server převažuje nginx, Apache je také přítomen, ale postupně se od něj upouští. Databázové stroje jsou pro projekty psané v Pythonu výhradně PostgreSQL a pro PHP pak MySQL.

4.1.2 Infrastruktura

V zásadě všechny aplikace běží v prostředí operačního systému GNU/Linux. Jen ve výjimečných případech např. z důvodu nutnosti použít proprietární software třetích stran je použit systém Windows ale i pak se jedná pouze o subsystém, který stojí paralelně k hlavním částem aplikace a je dostupný jako služba pro hlavní aplikace, která běží na operačním systému typu GNU/Linux. Velmi rozličné je však použití různých distribucí operačního systému GNU/Linux. Jedná se následující výčet distribucí:

- Debian 8 (jessie)
- Debian 7 (wheezy)
- Debian 6.0 (squeeze)

- Ubuntu 14.04 (Trusty Tahr)
- Ubuntu 15.04 (Vivid Vervet)
- Ubuntu 15.10 (Wily Werewolf)
- CentOS 5

Je běžné, že jeden projekt ve své infrastruktuře využívá několik různých verzí stejné nebo i úplně jiné distribuce. Je to dáno historickým vývojem aplikací, kdy se postupně dle nutnosti přidávali další stroje do již existující infrastruktury, případně přebíraly projekty od jiných vývojových týmů.

4.1.3 Prostředí a testování

Pro každou aplikaci vždy existuje jedno odpovídající testovací prostředí. Toto prostředí umožňuje držet právě jednu testovací verzi aplikace. Primárně slouží k akceptačnímu testování zákazníka, jen ve výjimečných případech je použito pro interní testování. Pokud je třeba nabídnout více nových funkcí paralelně k akceptaci, je nutno nejdříve integrovat funkce do jedné verze a tuto samostatně nasadit na testovací prostředí. Zákazník tedy akceptuje celý balík funkcí najednou. Samostatně lze funkce nasazovat pouze postupně a pokaždé musí proběhnout samostatná akceptace zákazníkem.

Ve většině případů není testovací prostředí z pohledu infrastruktury úplnou kopií produkčního prostředí. Testovací prostředí má nižší počet strojů, případně jsou výrazně slabšího výkonu.

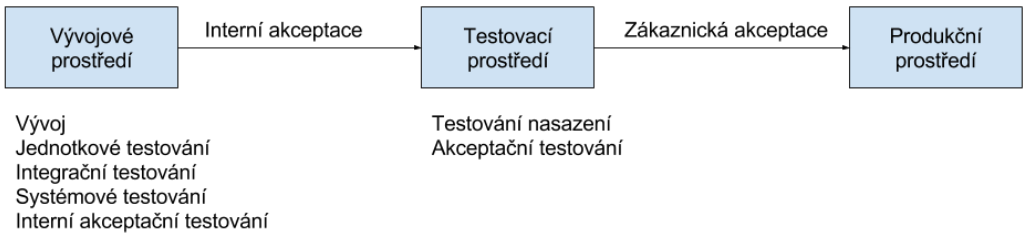
Pro potřeby vývoje si vývojáři vytváří pro každou aplikaci vlastní prostředí na lokálním vývojářském stroji, toto prostředí pak slouží i pro účely interního testování a akceptaci.

Intergrační testování probíhá po integraci lokálně na vývojářské stanici před nasazením na testovací prostředí.

U některých rozsáhlejších aplikací je k dispozici ještě předprodukční prostředí, které které slouží k testování nasazení na produkční prostředí.

Nejběžnější případ je ilustrován na obrázku 4.1

Obrázek 4.1: Stávající stav využití prostředí a testování



[Autor]

Při jednotkovém testování probíhá kontrola rozsahu pokrytí testy a uvedená metrika je zobrazena ve výsledku testování. U tohoto typu testování je vyžadováno >50% pokrytí.

4.1.4 Instalace a konfigurace prostředí

Instalace prostředí se provádí manuálně dle návodu na instalaci uvedeném v dokumentaci každého projektu. Změna konfigurace prostředí probíhá manuálně přihlášením na server pomocí administrátorského přístupu. Změny v konfiguraci jsou ručně zaznamenány v centralizovaném dokumentu a následně se provede i změna v dokumentaci instalace aplikace v projektové dokumentaci. Konfigurace se také provádí manuálně pro každé prostředí zvlášť.

4.1.5 Správa kódu

Veškerý kód aplikace je verzován a zálohován pomocí nástroje Git. Jako centrální úložiště pro Git je pro všechny projekty jednotně používána komerční SAAS služba bitbucket.org. Pro každou novou funkcionality nebo opravu chyby je ve verzovacím nástroji založena nová větev (dále funkční větev) a po otestování zpětně sloučena s hlavní větví. Hlavní větev u každého projektu je zároveň větví produkční. Tedy do produkčního prostředí se dostane pouze kód z hlavní větve. Pro testovací prostředí (a každé další) analogicky existuje odpovídající větev (testovací větev, předprodukční větev atp.).

Každý vývojář při začátku práce na nové funkcionalitě odvodí svoji práci z odpovídající funkční větve, která byla pro funkcionalitu založena a po dokončení práce tuto svoji větev opět do ní zpětně sloučí. Případné konflikty řeší vývojář, který slučuje svoji práci s prací ostatních.

4.1.6 Stav automatizace

Automatizace je k dipozici ve formě připravených testovacích a nasazovacích skriptů. Tyto skripty jsou součástí repozitáře kódu konkrétního projektu a návod na jejich použití je uveden v dokumentaci aplikace. Automatizace instalace a konfigurace prostředí je implementována v omezené míře pouze na některých projektech. Často jde o sdílenou znalost pro několik projektů a je tak použitý stejný vzor, který se pak zautomatizuje do skriptu. Všechny dostupné automatizační skripty jsou psány pro shell bash, případně v jazyce Python pro automatizační nástroj Fabric. Popis instalace a konfigurace je realizován pomocí nástroje Ansible.

Dle typologie testů uvedené v kapitole 3.3.4 Automatizace testování je částečně automatizováno pouze jednotkové testování. Ostatní typy jsou prováděny manuálně.

4.1.7 Hodnocení stávajícího stavu

Dle metodiky z kapitoly 3.9 lze vyhodnotit stav implementace continuous delivery následujícím způsobem:

Integrace a sestavení aplikace a konfigurace prostředí

0 bodů - Automatizace je pouze částečná, vývojáři lokálně testují svoji práci jednoduchými skripty ještě předtím než svoji práci odevzdají. Neexistuje však žádný nástroj, který by na vyžádání provedl automatickou instalaci a otestování všech provedených změn.

Testování, kontrola kvality a automatizace nasazování

1 bod - Probíhá pouze jednotkové testování a je zavedena metrika rozsahu pokrytí kódu testy. Nicméně žádné další metriky nejsou zavedeny a proces není plně automatický.

Práce s verzovacím nástrojem

2 body - Probíhá poměrně sofistikovaná správa kódu pomocí větvení a slučování změn. Částečně existuje vazba k jednotlivých úkolů v systému projektového řízení jako funkční větve, ale tento proces není automatizován.

Viditelnost změn

0 bodů - Neexistuje automatický nástroj pro sestavení prostředí a nikdo není upozorněn na změny v kódu a stav sestavení aplikace, integrace či testování.

4.1.7.1 Celkové hodnocení

Bylo dosáhnuto celkového zisku **3 bodů** z celkových 20 možných.

Ve dvou případech dosáhnulo bodové ohodnocení 0 bodů, **ve firmě COEX s.r.o. není dle této metodiky zaveden proces continuous delivery**.

4.2 Cíle

Z předchozí kapitoly je zjevné, že je třeba se soustředit hlavně na následující dvě oblasti:

- Integrace a sestavení aplikace a konfigurace prostředí
- Viditelnost změn

Nesmíme ale ani opominout hlavní cíle procesu continous delivery, tak jak byly definovány v kapitole 3.3 Cíle procesu continuous delivery.

Analýza procesů uvnitř firmy COEX s.r.o. vznesla i další specifické požadavky na implementaci.

Splnění cílů bude testováno na demonstraci implementace procesu ve vzorovém projektu.

4.2.1 Základní cíle

Základními cíli tedy budou:

1. Automatizované vytváření a konfigurace prostředí
2. Zviditelnění výsledků generovaných automatizačním nástrojem

Již splněním těchto cílů dojde dle metodiky 3.9 k splnění alespoň minimálních požadavků na implementaci continuous delivery.

Pro kompletaci procesu je dle kapitoly 3.3 Cíle procesu continuous delivery třeba přidat další cíle:

3. Automatizace procesu nasazování
4. Sjednocení nasazování na různá prostředí
5. Automatizace testování
6. Automatická správa dat

4.2.2 Specifické cíle

Vzhledem k různým požadavkům napříč různými projekty uvnitř firmy je nutné aby proces byl nastavitelný pro každou aplikaci zvlášť. Proces se může měnit v průběhu vývojového cyklu projektu a tedy by neměl být rigidní.

Všechny prvky procesu by měly být volitelné, tak aby každý vývojový tým mohl využít právě jen jeho výhody a nebyl příliš svazován pravidly. Svoboda by měla být také v rozhodnutí, které změny je třeba nechávat procházet všemi typy prostředí a akceptacemi a u kterých lze vynechat některé kroky. Mělo by také být relativně snadné aplikovat implementovaný proces na již existující projekty. S tímto souvisí, že by měl být proces co nejvíce kompatibilní s již stávajícími nástroji a postupy ve firmě.

Specifické požadavky lze tedy shrnout na:

1. Konfigurace procesu zvlášť pro každý projekt
2. Konfigurovatelnost procesu v čase
3. Volitelnost jednotlivých kroků procesu
4. Nenáročnost zavedení procesu na již existující projekty
5. Kompatibilita s již používanými nástroji a postupy

4.3 Návrh řešení

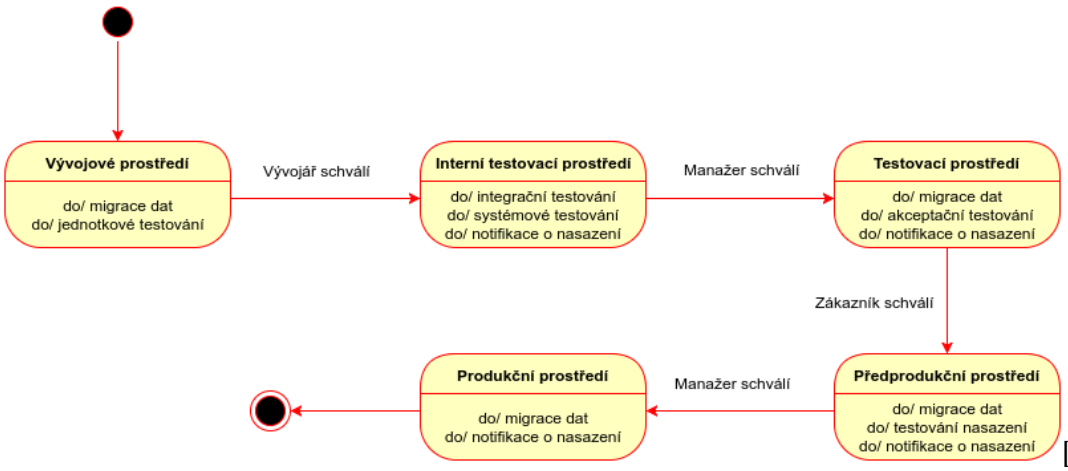
Postup návrhu řešení bude následující

1. Návrh modelu implementovaného procesu dle kapitoly 3.5.1 Model implementace procesu)
   a) Specifický návrh
   b) Obecný návrh
2. Dekompozice na samostatné celky a jejich návrh

4.3.1 Model implementace procesu

Výchozím bodem návrhu je stavový diagram, který ukazuje jedno z možných řešení modelu implementace continuous delivery (deployment pipeline viz kapitola 3.5.1 Model implementace procesu). Diagram 4.2 reprezentuje aktuální **úspěšné umístění změny** na prostředí, tedy postup změny přes různá prostředí až na produkční. Na každém prostředí budou automaticky provedeny specifické úkony, jako je například automatické testování nebo migrace dat. V tomto bodě je třeba si uvědomit, že pro každou změnu je nejdříve vytvořeno specifické prostředí daného typu (toto neplatí pro produkční prostředí, které je společné pro všechny změny).

Obrázek 4.2: Stavový diagram úspěšného umístění změny na prostředí



[Autor]

4.3.1.1 Zobecnění procesu

Pokud sjednotíme vytváření a konfiguraci prostředí a automatizujeme nasazení změny na prostředí:

1. splníme tím dva základní cíle: 3 a 4,

2. můžeme proces zobecnit.

Zobecněný stavový diagramu úspěšného umístění změny na prostředí je pak uveden na obrázku 4.3.

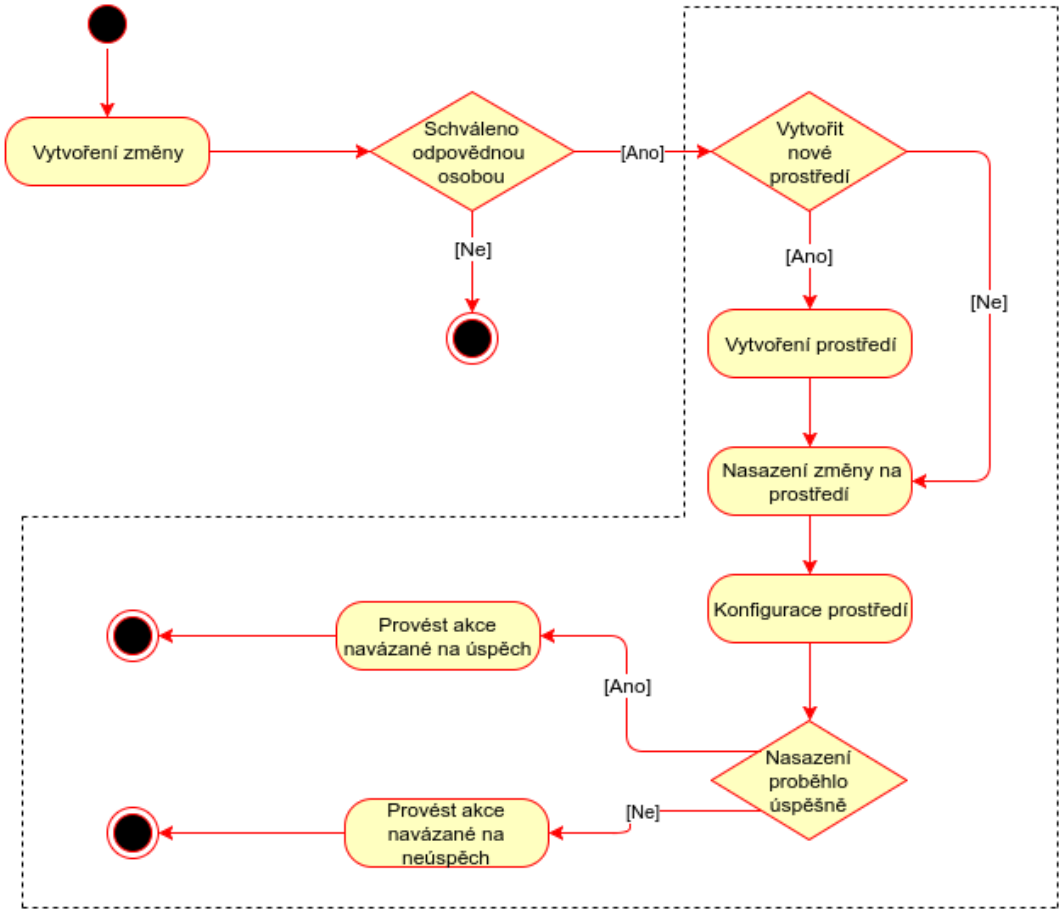
Obrázek 4.3: Zobecněný stavový diagram úspěšného umístění změny na prostředí

[Autor]

4.3.1.2 Dekompozice procesu

Proces je pomocí diagramu aktivit podrobně rozepsán na obrázku 4.4. Již je využito zobecnění z předchozího kroku a zohledňuje i neúspěšné nasazení na prostředí a možné akce, které se v tomto případě mohou provést.

Obrázek 4.4: Dekompozice procesu



[Autor]

V černém rámečku je část procesu která je předmětem procesu nasazení změny na prostředí. Vně rámečku se pak nachází proces vytvoření změny a schvalovací proces, který pro účely této práce spojíme do jednoho procesu vytvoření změny a jejího schválení.

4.3.1.3 Proces vytvoření změny a její schválení

V rámci této práce nás zajímají pouze dvě obecné varianty procesu vytvoření a schválení změny:

1. Vytvoření změny na lokální stanici vývojáře a její neformální schválení pro nasazení na vývojové prostředí.

2. Vytvoření změny ve verzovacím systému a její formální schválení pro nasazení na vybrané prostředí.

První varianta nevyžaduje žádnou zvláštní automatizaci, vývojář pouze spustí proces nasazení změny na prostředí (konkrétně nasadí změnu v lokálním souborovém systému na své vývojové prostředí).

Druhá varianta již vyžaduje nahrání změny do verzovacího systému. Pokud je v rámci konkrétního projektu vyžadováno proces automatizovat, lze pomocí dalších podmínek definovat, zda-li jde o schválení změny a automaticky reagovat nasazením změny na konkrétní prostředí.

Podmínky mohou být například:

- nahrání změny do specificky pojmenované větve verzovacího systému
- přidání specifického příznaku k objektu změny ve verzovacím systému

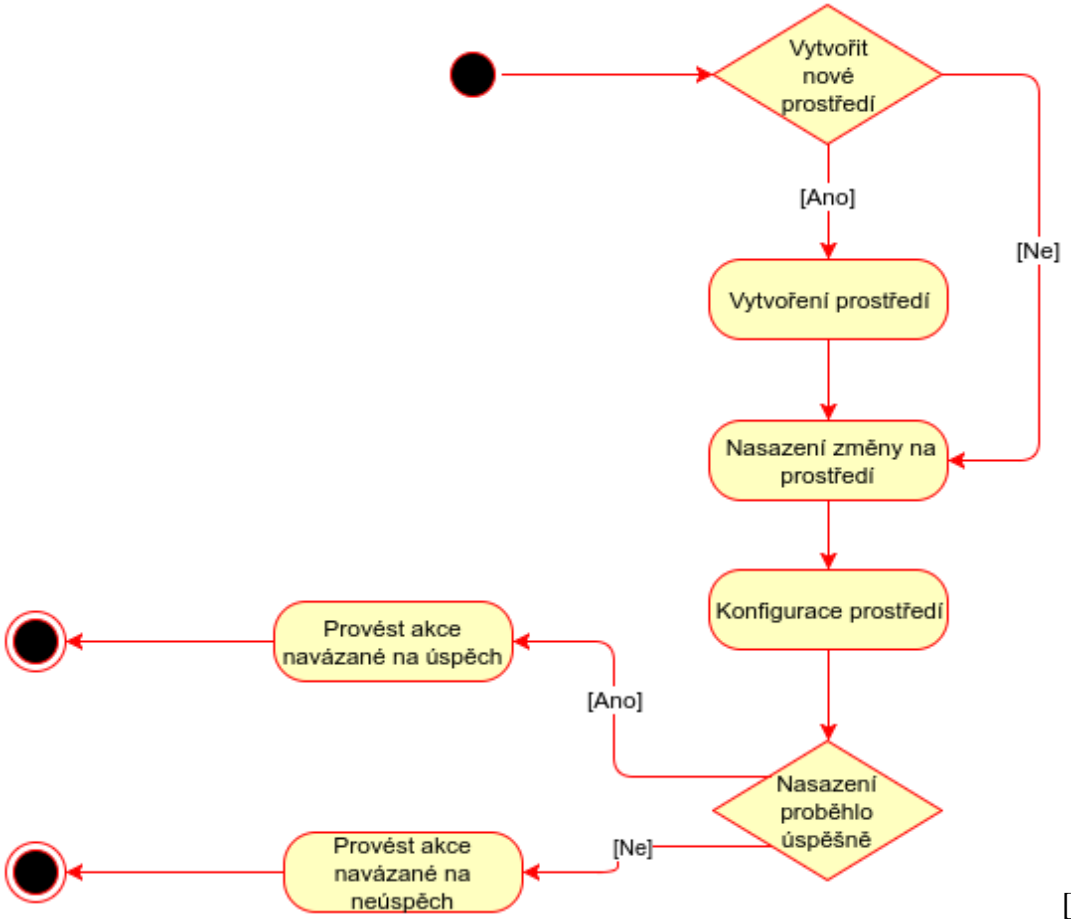
Pro zajištění plné automatizace bude jedním z předmětů implementace automatická reakce po nahrání změny do verzovacího systému.

Formální schválení může proběhnout také v jiném systému (například v informačním systému projektového řízení), jenž následně spustí proces nasazení změny na prostředí.

4.3.1.4 Proces nasazení změny na prostředí

Na obrázku 4.5 je samostatně zobrazen proces nasazení změny na prostředí.

Obrázek 4.5: Dekompozice procesu



[Autor]

Zatímco podprocesy vytvoření prostředí, nasazení změny na prostředí nebo konfigurace prostředí jsou poměrně jednoznačné a společné pro všechny varianty, podprocesy navázané

na úspěch a neúspěch nasazení musí být volitelné a konfigurovatelné v rámci konkrétního projektu. Pro splnění vytyčených cílů by akce spuštěné při úspěšném nasazení měli být například tyto:

- Migrace dat
- Testování
- Zobrazení informace o úspěchu

Při neúspěšném nasazení pak

- Zobrazení informace o neúspěchu
- Testování

Akce testování nebo migrace dat jsou komplexními procesy, které nejsou jednoduše zobecnitelné napříč projekty a měl by je tedy již samostatně řešit konkrétní projekt. Tyto body souvisí s naplněním základních cílů 5 a 6, nesmí být tedy vynechány v demonstraci implementace procesu ve vzorovém projektu.

Naopak zobrazení informace o úspěchu či neúspěchu nasazení je relativně transparentní proces, který lze implementovat i na základě neúplných znalostí o projektu.

Dále tedy bude předmětem implementace nástroj, které umožňuje realizaci následujících procesů.

- Vytváření prostředí
- Nasazení změny na prostředí
- Konfigurace prostředí
- Spouštění volitelných akcí po úspěchu nebo neúspěchu předchozích procesů

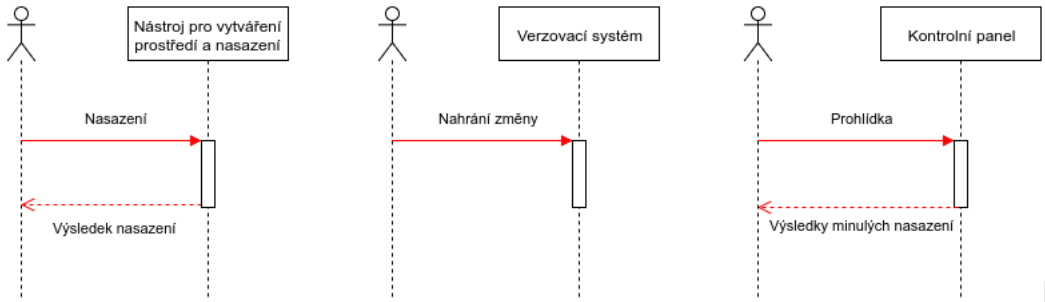
Tento nástroj budeme dále nazývat „Nástroj pro vytváření prostředí a nasazení“. Díky tomu, že tento nástroj zahrnuje automatizaci vytváření a konfiguraci prostředí, jeho realizací naplníme základní cíl 1.

Samostatně pak bude implementován nástroj, který zobrazí informaci o úspěchu či neúspěchu a který bude napojitelný na „Nástroj pro vytváření prostředí a nasazení“, tento nástroj budeme dále nazývat „Kontrolní panel“ a který naplňuje základní cíl 2.

4.3.1.5 Návrh celkového propojení nástrojů

Základní použití nástrojů nebude zahrnovat žádné propojení mezi nástroji jak ilustruje sekvenční diagram 4.6.

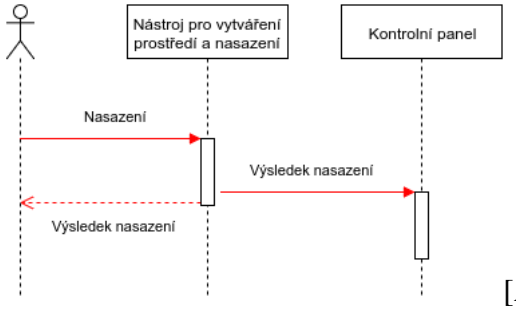
Obrázek 4.6: Základní použití nástrojů



[Autor]

Přímé nasazení na prostředí Vývojář použije „Nástroj pro vytváření prostředí a nasazení“ přímo a tento nástroj po provedení operace nahlásí výsledek nasazení do nástroje „Kontrolní panel“. Tuto situaci zobrazuje diagram 4.7.

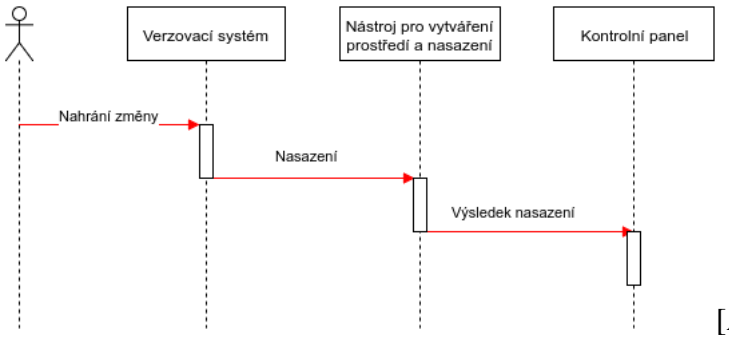
Obrázek 4.7: Přímé nasazení na prostředí



[Autor]

Nepřímé nasazení na prostředí Vývojář nahraje změnu do verzovacího nástroje. Pokud verzovací nástroj vyhodnotí, že má změnu nasadit, zavolá „Nástroj pro vytváření prostředí a nasazení“. Po provedení operace nahlásí „Nástroj pro vytváření prostředí a nasazení“ výsledek nasazení do nástroje „Kontrolní panel“. Tuto situaci zobrazuje diagram 4.8.

Obrázek 4.8: Nepřímé nasazení na prostředí



[Autor]

4.3.2 Nástroj pro vytváření prostředí a nasazení

Základním nástrojem pro úspěšnou implementaci continuous delivery je nástroj pro vytváření prostředí a nasazení. Tento nástroj musí umět vytvářet požadovanou infrastrukturu a na ní nasadit a nakonfigurovat všechny aplikace.

4.3.2.1 Požadavky

Požadovaný nástroj musí být:

- decentralizovaný - tak aby uměl vytvářet prostředí nejen na serveru ale i na lokálních stanicích vývojáře
- rozšiřitelný - pokud by nějaká část nevyhovovala, aby mohla být doplněna.
- zpětné kompatibilní - musí umět instalovat celé systémy i staršího data
- konfigurovatelný - musí umět instalovat různé infrastruktury na různá prostředí

Volitelně by pak měl být:

- open-source - požadavek vyplývající ze zadání práce

4.3.2.2 Návrh

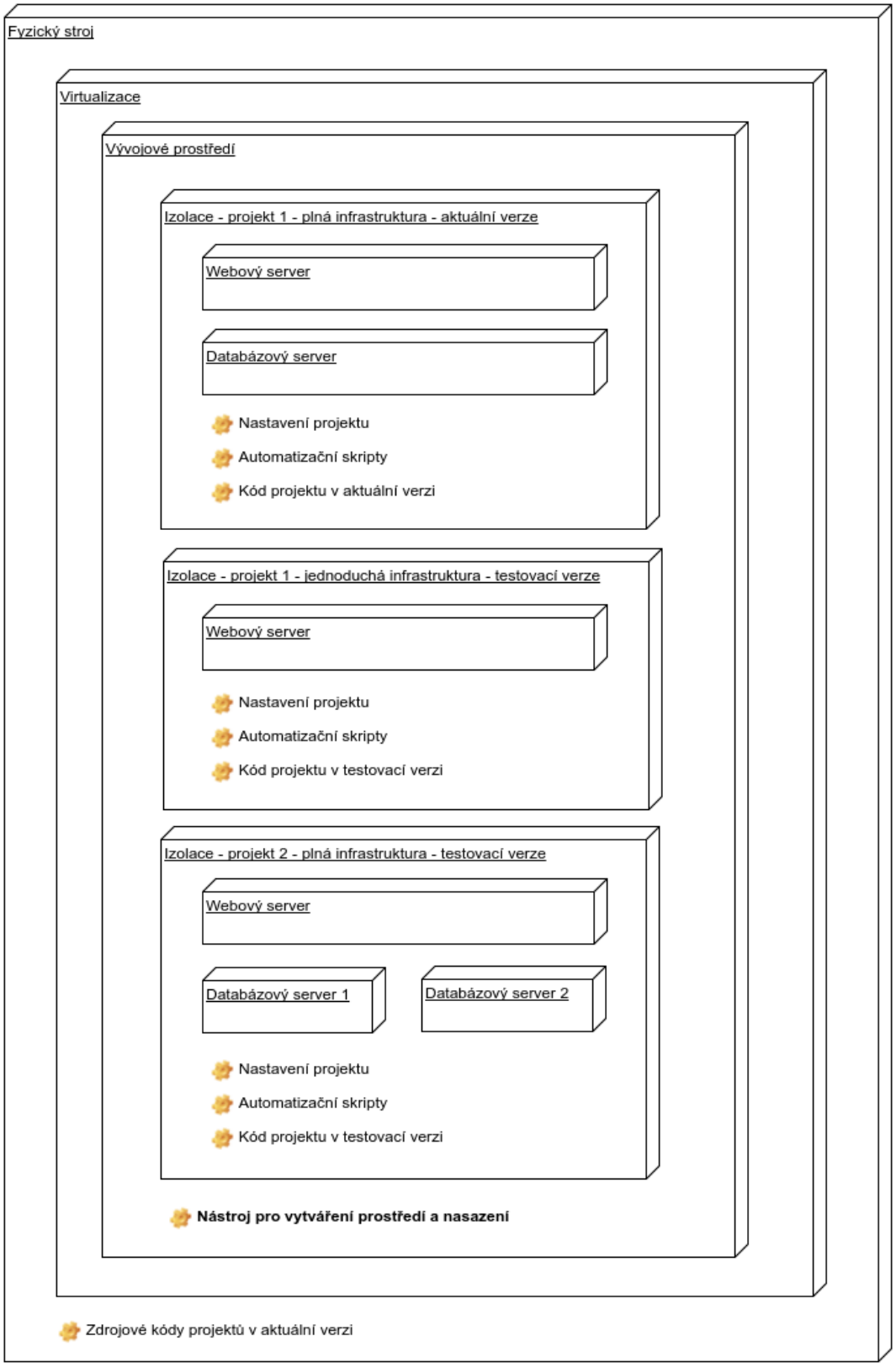
Na základě požadavků nebylo nalezeno žádné vyhovující existující řešení. Existující nástroje umí řešit pouze dílčí problémy.

Vytvoříme nový nástroj propojením již existujících nástrojů, tak aby splňoval výše uvedená kritéria.

Vytváření prostředí Na obrázku 4.9 je ilustrována situace, kdy je na vývojářské stanici najednou nainstalováno několik odlišných verzí různých projektů. Aby mohlo vývojové prostředí fungovat nezávisle na platformě fyzického stroje a jeho operačního systému, je celé prostředí virtualizováno.

Je nutno také provádět izolaci jednotlivých projektů a jejich infrastruktur v jednotlivých verzích. Izolaci lze realizovat také pomocí virtualizace. Dostupnými nástroji je nutné vyřešit kontrolované propojení vnějšího prostředí do vnitřního izolovaného včetně autentizačních mechanismů.

Obrázek 4.9: Návrh zanoření virtualizovaných prostředí na vývojářské stanici



[Autor]

Konfigurace aplikace Konfigurace bude uložena v čitelném a snadno editovatelném formátu u každého projektu zvlášť, tak aby byla verzována spolu se zdrojovým kódem projektu. Datový model konfigurace je zobrazen na diagramu tříd 4.10.

Základními třídami jsou:

- Projekt,
- Prostředí,
- Infrastruktura,
- Instalace.

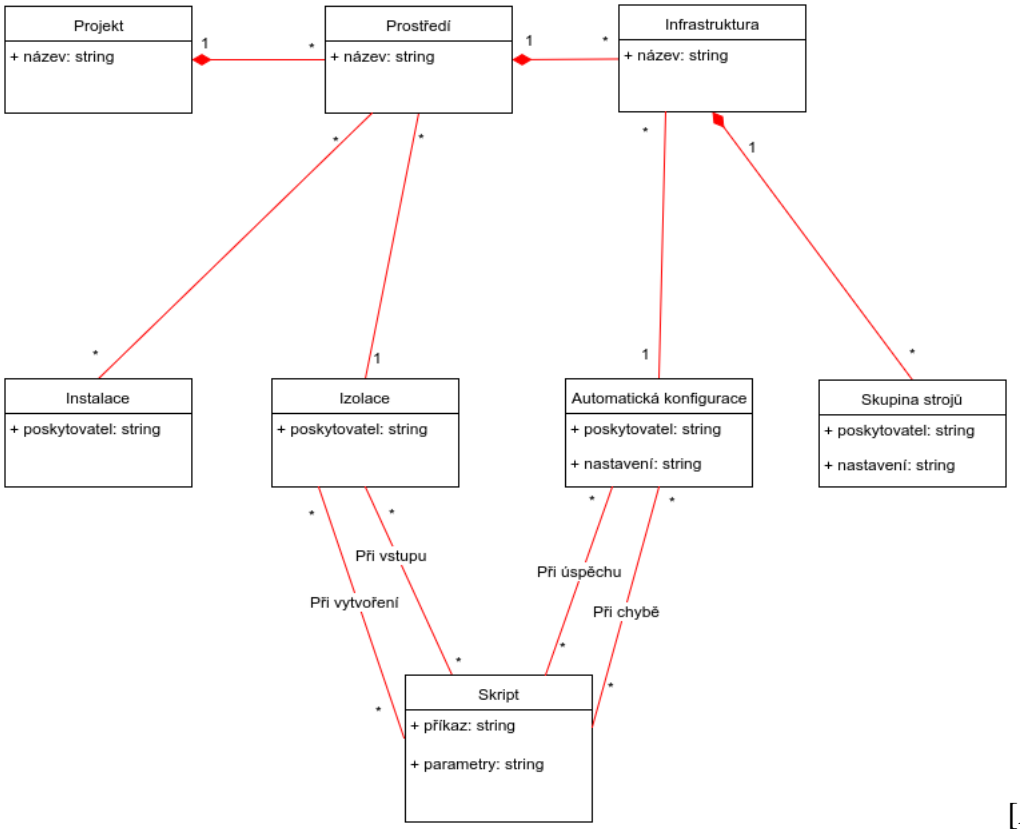
V konkrétní konfiguraci (již jako objekty) definují, který **projekt** se nainstaluje na jaké **prostředí**, jaká **infrastruktura** mu bude poskytnuta a odkud se načte **zdrojový kód (instalace)**.

Dalšími třídami jsou:

- Izolace - definuje jakým způsobem bude instalace na prostředí izolována.
- Automatická konfigurace - určuje jakým způsobem bude projekt na prostředí nainstalován.
- Skripty - spouštěný skript navázaný na určitou událost izolace nebo automatické konfigurace.
- Skupina strojů - popis strojů seskupených dle typu.

Třídy které mají atribut „poskytovatel“ jsou instanciovány dynamicky a poskytují tak možnosti rozšíření pomocí modulů konkrétních „poskytovatelů“.

Obrázek 4.10: Diagram tříd konfigurace nástroje pro vytváření prostředí a nasazení

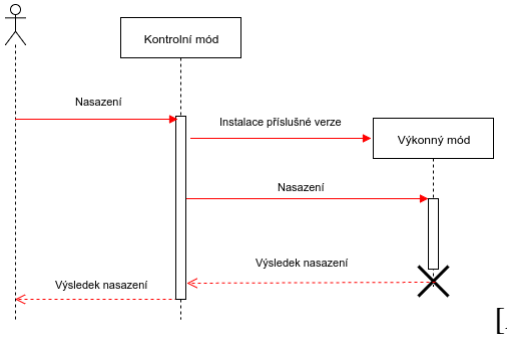


[Autor]

Zpětná kompatibilita Pro zajištění zpětné kompatibility bude implementován mechanismus, kdy bude použitá verze nástroje pro vytváření prostředí a nasazení uložena do konfiguračního souboru k projektu. Tato verze pak bude při každém vytvoření prostředí nainstalována do izolace a teprve tam spuštěna.

Nástroj spuštěný z vnějšího prostředí bude pouze kontrolovat výstup nástroje spuštěného v izolaci a nebude nijak do jeho chodu zasahovat. Vnější spuštění budeme nazývat „Kontrolní mód“, spuštění v izolaci pak „Výkonný mód“. Proces je ilustrován na sekvenčním diagramu 4.11.

Obrázek 4.11: Kontrolní a výkonný mód nástroje pro vytváření prostředí a nasazení



[Autor]

Konfigurace prostředí Automatická konfigurace prostředí bude prováděna automaticky po vytvoření požadovaného prostředí a infrastruktury. Pro konfiguraci bude použit jeden z decentralizovaných konfiguračních nástrojů z kapitoly 3.5.3.3.

Nasazení změny na prostředí Nasazení změny na prostředí bude realizováno dvěma různými způsoby dle nastaveného způsobu instalace:

- Pomocí verzovacího nástroje,
- nástrojem pro vzdálenou synchronizaci.

Spouštění volitelných akcí Pro podporu spouštění dalších akcí a pro notifikaci a propojení s dalšími nástroji bude implementována podpora spouštění standardních skriptů. Při úspěchu či neúspěchu výsledku procesu bude spuštěna odlišná sada skriptů dle konfigurace.

4.3.3 Kontrolní panel

Nástroj „Kontrolní panel“ slouží k zobrazení informace o provedených nasazení. Mělo by jít o přehlednou a jednoduchou aplikaci, ve které je na první pohled vidět, která nasazení proběhla úspěšně, která neúspěšně a jak postupují úspěšná nasazení přes všechna testovací prostředí až do produkčního.

4.3.3.1 Požadavky

Aplikace by měla být:

- Jednoduchá,
- schopná zobrazovat obsluhovat více projektů,
- škálovatelná,

volitelně pak:

- open-source.

4.3.3.2 Návrh

Na základě požadavků nebylo nalezeno žádné vyhovující existující řešení. Vytvoříme nový nástroj propojením již existujících nástrojů, tak aby splňoval výše uvedená kritéria.

Půjde o webovou aplikaci, na kterou se lze připojit pomocí API. Data jednotlivých nasazení budou ukládána do databáze. Klientská část bude sloužit pouze k zobrazení informací.

Síťový model Na obrázku 4.12 je navržen síťový model aplikace.

Obrázek 4.12: Síťový model aplikace „Kontrolní panel“

Kontrolní panel

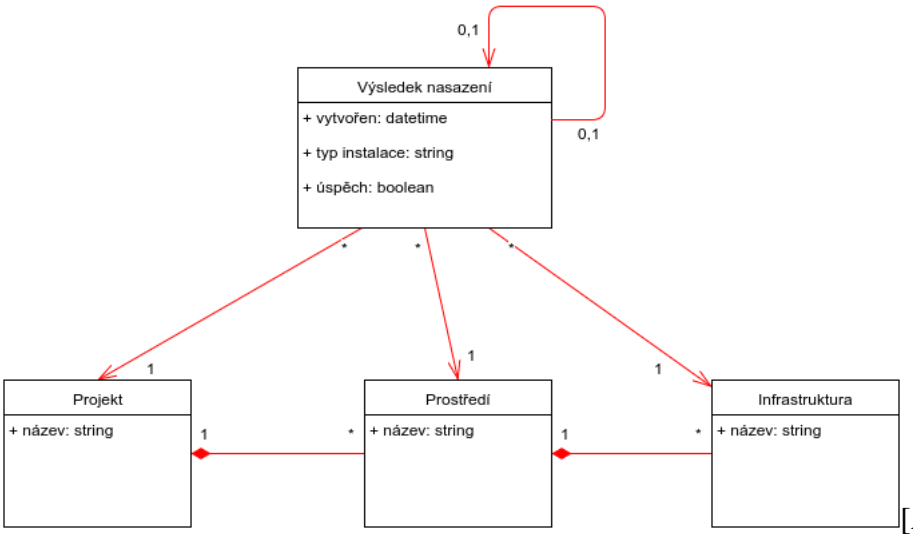
Název projektu

Instalace	Infrastruktura	Prostředí 1	Prostředí 2	Poslední změna
git:master	full	OK	OK	21.3.2016
git:testing	full	OK	ERROR	20.3.2016
actual:demo	full	ERROR		19.3.2016
git:feature_123	full	OK		17.3.2016

[Autor]

Datový model Datový model aplikace kopíruje základní konfiguraci nástroje pro vytváření prostředí a nasazení jak ukazuje obrázek 4.13. Dále přidává třídu „Výsledek nasazení“, která představuje jednotlivá nasazení. Jde o jednosměrný spojový seznam reprezentující postup změny přes jednotlivá prostředí.

Obrázek 4.13: Diagram tříd nástroje „Kontrolní panel“



[Autor]

4.4 Implementace

Zdrojové kódy všech implementovaných nástrojů jsou nedílnou součástí této práce a nachází se na přiloženém CD/DVD.

4.4.1 Nástroj pro vytváření prostředí a nasazení

Jedná se o nástroj bez grafického rozhraní a pro konzolové výstupy programu byl použit anglický jazyk. Pro strojové zpracování výstupu programu je angličtina z důvodu existence množství kódování vhodnější a navíc je v tomto oboru defakto průmyslovým standardem. Software je šířen jako open-source [1].

4.4.1.1 Rozšiřitelnost

V návrhu tohoto nástroje je zahrnuta modulárnost řešená pomocí „poskytovatelů“. Poskytovatel zastřešuje konkrétní použitou technologii a zpřístupňuje ji skrze jednotné API. Toto řešení umožňuje, že případná nevhodná volba technologie nebude bránit dalšímu rozvoji aplikace. Lze paralelně vytvořit další modul, který používá jinou alternativní technologii a pouze v konfiguraci projektu nastavit jiného poskytovatele.

4.4.1.2 Volba technologií

Výsledný nástroj má být hlavně propojením mnoha dalších již existujících nástrojů. Pro toto použití je vhodné použít nějaký systémově-integrační jazyk.[29] Takovýto jazyk, který poskytuje vysokou míru abstrakce a flexibility a zároveň je dobře zavedený v technologickém zázemí firmy COEX s.r.o., je například Python. Pro zajištění podmínky open-source pak zvolíme jako základní platformu operační systém GNU/Linux.

Autentizace a autorizace Pro autentizaci a autorizaci bude použit standardní nástroj Secure Shell (SSH) umožňující šifrováním zabezpečené připojení na vzdálené systémy. S výhodou mohou být použity přidružené nástroje jako je `ssh-agent`, který poskytuje tranzitivní připojení na další systémy a nebo také `scp` pro bezpečný přenos souborů.[2]

Virtualizace Protože půjde o zanořené řešení, použijeme pro izolaci a vytváření infrastruktury odlehčenou verzi virtualizace v podobě kontejnerů. Standardní jádro Linux podporuje kontejnerizaci pomocí jmenných prostorů. Tuto technologii používají ve stabilní verzi tři nástroje:

- Docker
- systemd-nspawn
- LXC

Docker je aplikační kontejner a tudíž ho z důvodu požadavku na kompatibilitu se staršími projekty musíme vyřadit.

[1] Zdrojové kódy nástroje jsou uveřejněny pod licencí Apache 2.0 na adrese https://github.com/baseclue/codev.

Systemd-nspawn je původně vývojový nástroj pro init systém systemd. Jeho omezením je rozšířenost pouze na distribucích používající systemd včetně na serverech používané distribuce Debian.

Použijeme tedy LXC, které má podporu ve všech nejpoužívanějších distribucích.

Automatická konfigurace Ve firmě COEX s.r.o. se již pro některé projekty používá automatizace konfigurace pomocí nástroje Ansible, který je bezagentní 3.5.3.3 a tudíž splňuje podmínku uvedenou v návrhu aplikace.

4.4.1.3 Uživatelská dokumentace

Nejdůležitější součástí nástroje pro vytváření prostředí a nasazení je flexibilita umožněná individuální konfigurací každého projektu pomocí konfiguračního souboru.

Konfigurační soubor Název souboru je `.codev` a měl by být umístěn v hlavním adresáři projektu.

Formát konfiguračního souboru je lidmi čitelný a zároveň strojově zpracovatelný `yaml` [14]. Všechna konfigurace je explicitní, nejsou implementované žádné výchozí hodnoty.

Struktura se drží návrhu dle obrázku 4.10.

```
project: demo_project              # název projektu

environments:                      # definice prostředí
  development:                     # název prostředí
    installations:                 # povolené typy instalace
      - actual                     # aktuální soubory
      - git                        # repozitář git
    performer:                     # způsob spouštění
      provider: local              # poskytovatel spouštění
                                   #   - pouze 'local'
    infrastructures:               # definice infrastruktur
      full:                        # název infrastruktury
        isolation:                 # definice izolace
          provider: lxc            # poskytovatel izolace
                                   #   - pouze 'lxc'
          connectivity:            # definice přemostění portů
                                   #   zevnitř izolace ven
                                   #   na ip adresu izolace
            webserver_1:           # název skupiny strojů
```

```
                        #   a za znakem '_'
                        #   pořadí tohoto stroje
      80: 80            # port v izolaci: vnější
  scripts:
    oncreate:           # seznam skriptů spuštěných
                        #   po úspěšném vytvoření
                        #   izolace
      - install_vpn.sh
    onenter:            # seznam skriptů spuštěných
                        #   po vstupu do izolace
      - connect_vpn.sh
machines:               # definice skupin strojů
  database:             # název skupiny strojů
    provider: lxc       # poskytovatel strojů
                        #   - 'real' nebo 'lxc'
    specific:           # specifické nastavení
                        #   poskytovatele
      distribution: ubuntu
      release: trusty
provision:              # definice automatické
                        #   konfigurace
  provider: ansible     # poskytovatel automatické
                        #   konfigurace
  specific:             # specifické nastavení
                        #   poskytovatele
    version: 1.9.4
    playbook: provisioning/ansible/full.yml
 scripts:
    onsuccess:          # seznam skriptů spuštěných
                        #   po úspěšné konfiguraci
      - run_tests.sh
      - notify_success.sh
    onerror:            # seznam skriptů spuštěných
                        #   při chybě nasazení
      - notify_error.sh
```

Příkazy Aplikace je reprezentována jediným spustitelným souborem `codev`, který dále obsluhuje další podpříkazy. Všechny příkazy a nápovědu k jejich použití lze vyvolat příkazem `codev` bez parametrů.

Výstupem pak je:

```
Usage: codev [OPTIONS] COMMAND [ARGS]...

Options:
  --version  Show version number and exit.
  --help     Show this message and exit.

Commands:
  deploy    Deploy project.
  destroy   Destroy isolation.
  execute   Execute command in isolation.
  info      Show info about the deployment.
  install   Install project.
  run       Run command in project context in isolation.
  shell     Invoke an isolation shell.
```

Příkaz pro instalaci je:

```
codev install -e [prostředí] -i [infrastruktura] -s [
    ↪ instalace]
```

Příkaz pro získání informací o izolaci a infrastruktuře:

```
codev info -e [prostředí] -i [infrastruktura] -s [instalace]
```

Příkaz pro zrušení izolace:

```
codev destroy -e [prostředí] -i [infrastruktura] -s [
    ↪ instalace]
```

Příkaz pro spuštění automatické konfigurace je:

```
codev deploy -e [prostředí] -i [infrastruktura] -s [instalace
    ↪ ]
```

Tento příkaz se ale běžně nevolá přímo ale zprostředkovaně ho zavolá v izolaci příkaz `codev install` viz kontrolní a výkonný mód zobrazený na obrázku 4.11.

Další příkazy `shell`, `execute` a `run` nejsou při standardním použití potřeba, slouží spíše k ladění konfigurace a nasazení.

Další informace o příkazech a jejich parametrech lze získat pomocí syntaxe:

```
codev [příkaz] --help
```

4.4.2 Kontrolní panel

Kontrolní panel lze implementovat pomocí běžných technologií na vytváření webových aplikací. Protože jde o velmi malou aplikaci, je vhodné využít některý z mikroframeworků.

4.4.2.1 Technologie

Vzhledem k tomu, že tato aplikace se bude dále vyvíjet, tak jak budou postupně růst potřeby firmy, je pro implementaci zvolen jediný mikroframework, který se ve firmě používá - Flask [32].

Flask je velmi modulární framework, který pomocí jednoduchého napojení poskytuje i ORM vrstvu pro databázi [31].

Jako webový server je vybrán nginx, který je ve firmě téměř výhradně používán. Neočekává se příliš mnoho simultánních přístupů ani výrazně častých zápisů a proto bylo pro databázi zvoleno nejjednodušší řešení pomocí sqllite. V případě výskytu větší zátěže lze databázi přesunout na robustnější a škálovatelnější databázový stroj.

4.4.2.2 Konfigurace

Konfigurace nastavení databáze je uložena v souboru `/etc/codev-dashboard/config.ini`, který je případně přetížen souborem `~/.config/codev-dashboard/config.ini`.

4.4.2.3 API

Pro napojení na aplikaci jsou určeny dva přípojné body:

- `/add_success/` - přidává úspěšný výsledek nasazení
- `/add_failure/` - přidává neúspěšný výsledek nasazení

Oba body přijímají identifikátory nasazení `project`, `environment`, `infrastructure` a `installation` pomocí HTTP metody POST.

4.4.2.4 Klientská část

Klientská část je implementována pomocí HTML 5 a CSS frameworku bootstrap v3.3.6 dle síťového modelu z obrázku 4.12.

4.4.3 Automatická reakce na změnu ve verzovacím systému

Pro implementaci automatické reakce na změnu nahranou do verzovacího systému je nejprve třeba zvolit vhodný verzovací systém, který umožňuje snadnou implementaci tohoto chování. Ve firmě COEX s.r.o. je již používán verzovací nástroj Git, který umožňuje spustit jakýkoliv skript na základě vyvolané události v repozitáři zdrojových kódů [34].

Implementace proběhla v programovacím jazyce Bash. Jde o jednoduchý skript spuštěný po každé události `post-receive`, který pouze zkontroluje zda-li je provedena změna ve větvi, u které chceme spustit automatické vytvoření prostředí a nasazení. Následně tento proces spustí pomocí již implementovaného nástroje `codev`.

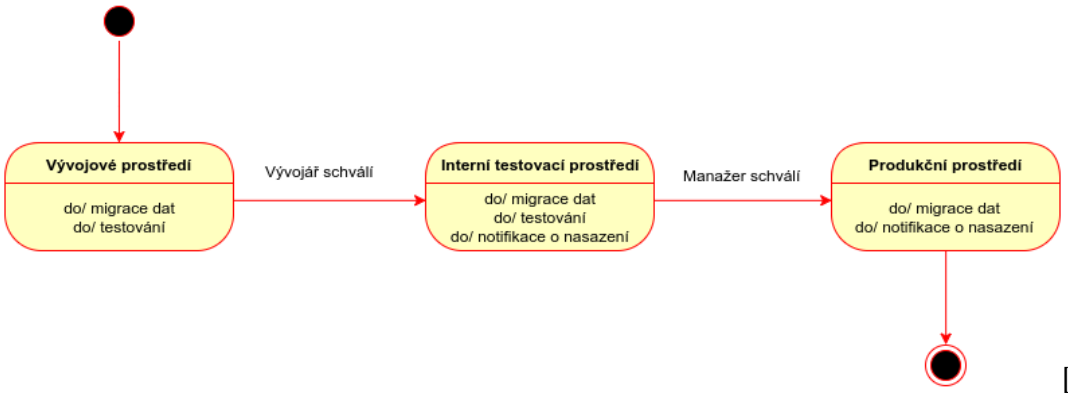
4.5 Demonstrace implementace procesu na vzorovém projektu

Aplikace demonstrující proces a její součásti včetně zdrojových kódů jsou nedílnou součástí této práce a nachází se na přiloženém CD/DVD v adresáři `demo`.

Vzorový projekt je velmi jednoduchý, účelem demonstrace je ukázka procesu continuous delivery nikoli popis tohoto projektu. Pro naše účely stačí, že jde o webovou aplikaci, která se připojuje k databázi[2]. Vzorový projekt je umístěn v podadresáři `demo_project`.

Na obrázku 4.14 je zobrazen implementovaný proces na vzorovém projektu. V rámci použitých nástrojů lze pomocí jejich konfigurace realizovat i jednodušší proces, ale zde jde o minimální verzi procesu, který by vůbec měl být implementován na reálném projektu. Ve většině případů bude implementován složitější proces jako například již použitý návrh možného řešení na 4.2.

Obrázek 4.14: Stavový diagram implementovaného procesu na vzorovém projektu



[Autor]

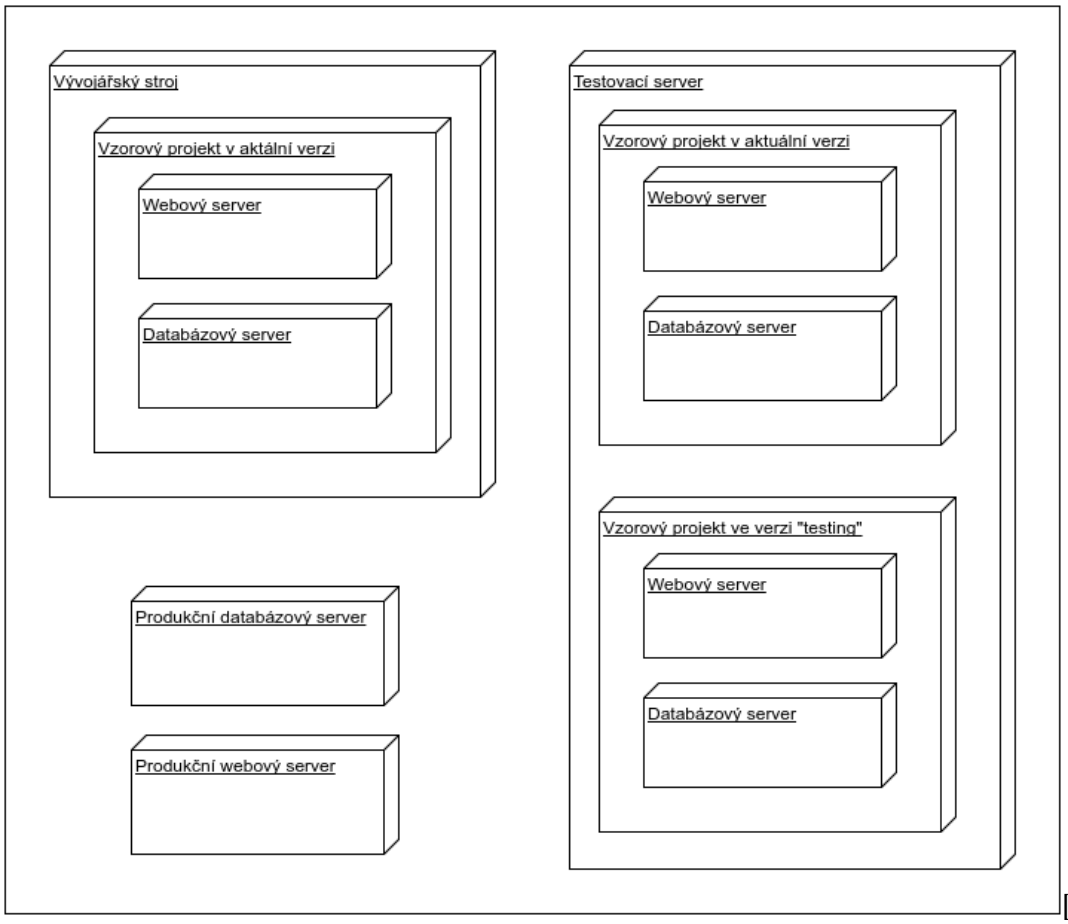
Následující instalační manuál je určen pro 64bitovou verzi operačního systému Ubuntu Server 15.10, ale s drobnými úpravami je vhodný i pro další varianty distribuce Ubuntu.

[2]Vzorový projekt je odvozen z open-source aplikace „Django-sample-app“ autora Daniel Rus Morales. Tato aplikace je šířena pod licencí GPLv3 a původní zdrojové kódy najdete na adrese https://github.com/danirus/django-sample-app.

4.5.1 Instalace infrastruktury

Abychom nemuseli instalovat 4 různé počítače (ať fyzické či virtualizované) můžeme s výhodou použít již implementovaný nástroj pro vytváření prostředí a nasazení `codev`. V rámci následujících kroků bude nainstalována infrastruktura uvedená na obrázku 4.15.

Obrázek 4.15: Diagram nasazení implementovaného procesu na vzorovém projektu



[Autor]

Aplikace je umístěna v adresáři `demo`, do nějž přejdeme pomocí příkazu:

```
local$ cd demo
```

Prerekvizity pro jeho použití i samotná instalace proběhnou automaticky po spuštění příkazu (jsou potřeba administrátorská privilegia a proto je nejprve uživatel požádán o heslo):

```
local$ ./install.sh
```

Vytvořit privátní a veřejný klíč pro automatické připojování na servery:

```
local$ ssh-keygen -t rsa -b 4096 -f codev_id_rsa -N ""
```

Zapnout ssh agenta a přidat do něj vygenerovaný pár klíčů, tak aby byla možno provést autentizaci bez nutnosti znalosti hesla.

```
local$ eval `ssh-agent`
```

```
local$ ssh-add codev_id_rsa
```

Jako poskytovatel instalace je zvolen program `ansible`, jehož konfigurační soubory se nachází ve stejnojmenném adresáři. V souboru `.codev` je nadefinována celá infrastruktura a instalační procedura.

Instalaci pak lze spusti pomocí příkazu:

```
local$ codev install -e demo -i demo -s actual:demo --force
```

Příkaz by měl skončit úspěšně tak, aby na posledním řádku vypsal:

```
[CONTROL] [INFO] Installation has been successfully completed
    ↪ .
```

V případě výskytu chyby je tato vypsána. Často jde o chybu způsobenou nestabilitou připojení. Někdy stačí pouze příkaz zopakovat, ale v některých případech je třeba nejdříve vyčistit instalaci pomocí příkazu `codev destroy` a pak opět zkusit `codev install` z předchozího kroku.

Příkaz pro vyčistění instalace:

```
local$ codev destroy -e demo -i demo -s actual:demo --force
```

Následujícím příkazem pak lze nechat vypsat celou nainstalovanou infrastrukturu:

```
local$ codev info -e demo -i demo -s actual:demo
```

V našem případě bude výstup vypadat následovně:

```
Isolation:
    ident: 54a6f1832ebfbb9eb8728c9b28bee338
    ip: 10.0.3.198
Infrastructure:
    Machines group: develop
        10.0.4.10 develop_1
                8000 -> 8000
                22 -> 221
    Machines group: testing
        10.0.4.100 testing_1
    Machines group: production_web
        10.0.4.200 production_web_1
                80 -> 9000
```

```
    Machines group: production_database
        10.0.4.210 production_database_1
```

IP adresa 10.0.3.198 uvedená v sekci „`Isolation`“ je jediná adresa dostupná přímo, jedná se o adresu izolovaného prostředí. Konkrétní IP adresa může lišit podle toho jaká byla přiřazena lokálním DHCP serverem. Ostatní uvedené IP adresy jsou navázané na jednotlivé virtuální stroje (v tomto případě kontejnery) a jsou dostupné pouze zevnitř izolace.

Ve výpisu je vidět i nastavené přesměrování portů z jednotlivých virtuálních strojů na několik vnějších portů izolace. Proto se lze například z vnějšího prostředí připojit na stroj `develop_1` přímo pomocí protokolu ssh.

Například následující příkaz nás připojí na stroj `develop_1`.

```
local$ ssh -A codev@10.0.3.198 -p 221
```

Není třeba zadávat heslo, veřejný klíč vygenerovaný z počátku procesu byl automaticky rozdistribuovaný mezi všechny virtuální stroje.

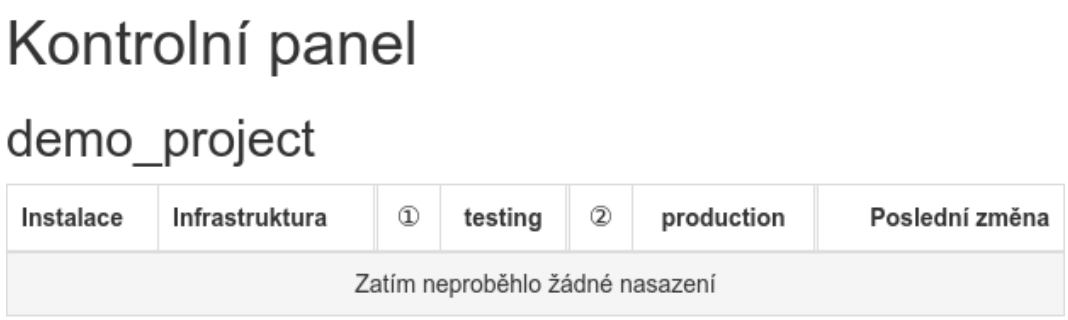
Parametrem -A říkáme, že ssh agent protunelujeme i do nového připojení a můžeme tedy dále pracovat v autorizovaném režimu.

4.5.1.1 Kontrolní panel

V předchozím kroku došlo ke kompletní instalaci infrastruktury včetně aplikace Kontrolní panel.

Aplikace je nyní dostupná přes webový prohlížeč na adrese izolace http://10.0.3.198/ viz obrázek 4.16.

Obrázek 4.16: Nainstalovaná aplikace „Kontrolní panel“



[Autor]

4.5.2 Instalace vzorového projektu

Po připojení na stroj `develop_1` stáhneme prázdný Git repozitář ze serveru `testing`.

```
develop_1$ git clone codev@10.0.4.100:demo_project.git
```

Do nově vytvořeného adresáře nahrajeme například pomocí programu rsync z lokální stanice celý podadresář `demo_project`:

```
local$ rsync -re 'ssh -p 221' demo_project/ codev@10.0.3.198:
  ↪ demo_project
```

Přesuneme se do adresáře projektu

```
develop_1$ cd demo_project
```

Následujícím příkazem lze zjistit, jaké změny byly v repozitáři provedeny.

```
develop_1$ git status
```

Na výpisu vidět, že byly přidány soubory a adresáře projektu `demo_project`.

```
On branch master

Initial commit

Untracked files:
        (use "git add <file>..." to include in what will be
          ↪ committed)
                .codev
                AUTHORS
                LICENSE
                ansible/
                demo/

nothing added to commit but untracked files present (use "git
  ↪ add" to track)
```

Nyní je projekt ve stavu, kdy vývojář provedl změnu v kódu a ještě nedošlo k její nahrání do verzovacího systému.

4.5.3 Nasazení vzorového projektu na vývojovém prostředí

Kompletní definice nasazení na všechna prostředí včetně vývojového je v souboru `.codev`. Vývojové prostředí je identifikováno jako `development` a jeho jedinou infrastrukturou je

`full`.

4.5.3.1 Instalace na lokální prostředí

Následující postup je pro instalaci projektu na prostředí lokálního stroje, tak abychom demonstrovali schopnosti instalace i na lokální prostředí a viditelnost nainstalované aplikace pro vývojáře.

Z hlavního adresáře `demo` přejdeme do podadresáře obsahující vzorový projekt:

```
local$ cd demo_project
```

Pro instalaci na lokální prostředí použijeme nástroj `codev`:

```
local$ codev install -e development -i full -s actual:demo --
    ↪ force
```

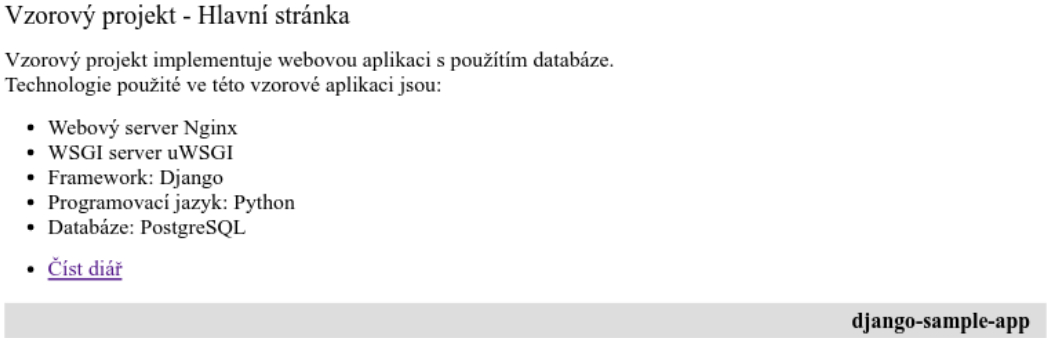
Příkazem codev info zjistíme informace o nainstalované infrastruktuře:

```
local$ codev info -e development -i full -s actual:demo
```

```
Isolation:
    ident: 292fed6331022a02557fe76d09d2d5e3
    ip: 10.0.3.4
Infrastructure:
    Machines group: database
        10.0.4.3 database_1
    Machines group: webserver
        10.0.4.2 webserver_1
                80 -> 80
```

Ve webovém prohlížeči na adrese izolace: http://10.0.3.4/ je nyní k dispozici aktuální verze vzorové aplikace viz obrázek.

Obrázek 4.17: Výstup nainstalované vzorové aplikace



[Autor]

Dle konfigurace došlo k automatickému testování v rámci nasazení na vývojové prostředí. Testování lze také snadno pustit pomocí příkazu `codev run`:

```
local$ codev run -e development -i full -s actual:demo --
  ↪ provisioning/scripts/run_tests.sh {machine_webserver_1}
```

4.5.3.2 Instalace na simulované vývojové prostředí

Pro instalaci na simulovaném vývojové prostředí provedeme analogické akce z předchozí ukázky:

```
develop_1$ codev install -e development -i full -s actual:
  ↪ demo --force
```

Příkazem codev info zjistíme informace o nainstalované infrastruktuře:

```
develop_1$ codev info -e development -i full -s actual:demo
```

```
Isolation:
    ident: 292fed6331022a02557fe76d09d2d5e3
    ip: 10.0.3.69
Infrastructure:
    Machines group: database
        10.0.4.3 database_1
    Machines group: webserver
        10.0.4.2 webserver_1
                80 -> 80
```

Nyní by měl vývojář zkontrolovat, zda-li je vše v pořádku. Ve webovém prohlížeči na adrese izolace: http://10.0.3.69/ uvidí své změny v aplikaci.

4.5.4 Nasazení vzorového projektu na testovací prostředí

Pro instalaci na testovací prostředí máme nyní dvě možnosti. Buď použijeme přímé nasazení nebo nasadíme nepřímo přes verzovací systém.

4.5.4.1 Přímé nasazení

Pro přímé nasazení použijeme opět nástroj `codev` a přepínačem `-e` nastavíme prostředí na `testing`.

```
develop_1$ codev install -e testing -i full -s actual:demo
```

Instalace je typu `actual` a změna tudíž nemusí být nahraná do verzovacího systému. Pro některé typy prostředí nemusí být toto chování žádoucí a lze ho jednoduše nepovolit neuvedením typu instalace `actual` do povolených instalací. I v našem případě nechceme do prostředí `testing` nasazovat změny, které nejsou uložené ve verzovacím systému. Výstupem tohoto příkazu bude:

```
Error: Installation 'actual' is not allowed installation for
    ↪ environment 'testing'
```

Při povolení instalace `actual` by došlo k nasazení na testovací prostředí. Protože je však aplikace nainstalována do izolace, není možné se dostat k výstupům aplikace jinak než ze zevnitř testovacího prostředí. Proto je tento způsob instalace vhodný spíše pro automatizaci testů, které probíhají dlouho a nejsou tak vhodné pro běh na stroji konkrétního vývojáře.

4.5.4.2 Nepřímé nasazení

Pro nepřímé nasazení nahrajeme změny do verzovacího systému do větve `testing`. Verzovací systém sám spustí nasazení na testovací prostředí pomocí nástroje `codev` a zpřístupní výstup aplikace z vnějšího prostředí.

Vytvoříme lokální větev `testing`.

```
develop_1$ git checkout -B testing
```

Přídáme jako změnu všechny soubory z adresáře:

```
develop_1$ git add .
```

Nahrajeme změnu do repozitáře:

```
develop_1$ git push --set-upstream origin testing
```

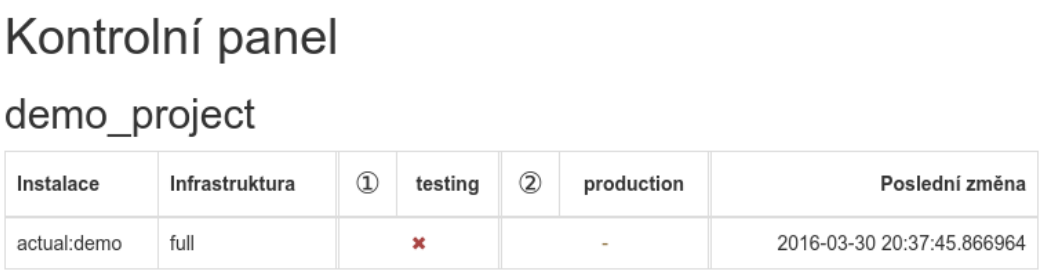
O výsledku nasazení se vývojář dozví z nástroje „Kontrolní panel“

4.5.4.3 Zobrazení výsledku nasazení na kontrolním panelu

V obou typech nasazení nakonec dojde k odeslání informace o úspěchu či neúspěchu nasazení do aplikace „Kontrolní panel“

Příklad neúspěšného nasazení je vidět na obrázku 4.18.

Obrázek 4.18: Neúspěšné nasazení na testovací prostředí v „Kontrolním panelu“



[Autor]

4.5.5 Nasazení vzorového projektu na produkční prostředí

Pro instalaci na produkční prostředí máme nyní opět dvě možnosti. Buď použijeme přímé nasazení nebo nasadíme nepřímo přes verzovací systém.

4.5.5.1 Přímé nasazení

Pro přímé nasazení použijeme opět nástroj codev a přepínačem `-e` nastavíme prostředí na `production`.

```
develop_1$ codev install -e production -i full -s git:master
```

Tímto způsobem došlo k nasazení na produkční prostředí.

4.5.5.2 Nepřímé nasazení

Pro nepřímé nasazení nahrajeme změny do verzovacího systému do větve `master`. Verzovací systém sám spustí nasazení na produkční prostředí pomocí nástroje `codev`.

Přepneme pracovní větev na `master`.

```
develop_1$ git checkout master
```

Sloučíme změny z větve `testing`:

```
develop_1$ git merge testing
```

Nahrajeme změnu do vzdáleného repozitáře.

```
develop_1$ git push
```

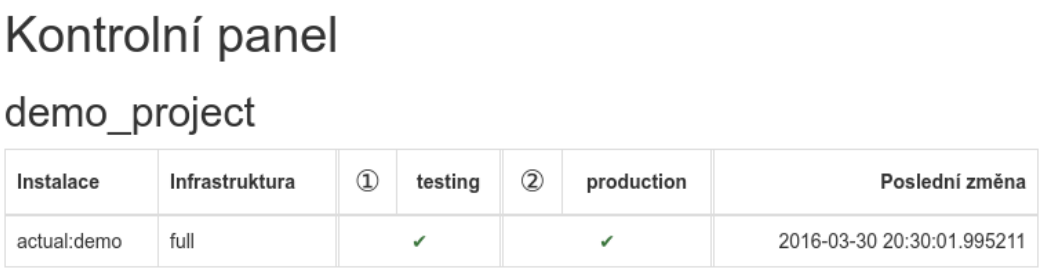
V tomto případě jsou si instalace rovnocenné. Výhodou nepřímého nasazení může být možnost vypnout vývojářský stroj. Nasazení je prováděno pomocí nástroje `codev` na serveru verzovacího systému (či kdekoliv jinde, dle konfigurace).

4.5.5.3 Zobrazení výsledku nasazení na kontrolním panelu

V obou typech nasazení nakonec opět dojde k odeslání informace o úspěchu či neúspěchu nasazení do aplikace „Kontrolní panel“

Příklad tentokrát úspěšného nasazení je vidět na obrázku 4.19.

Obrázek 4.19: Úspěšné nasazení na produkční prostředí v „Kontrolním panelu“



[Autor]

5 Vyhodnocení a diskuze

V kapitole 4.2 jsme definovali konkrétní cíle, které by měla demonstrace implementace procesu continuous delivery ve vzorovém projektu splnit. Stávající stav implementace procesu byl v kapitole 4.1.7 ohodnocen 3 body a poznámkou, že proces continuous delivery není ve firmě zaveden. Toto hodnocení provedeme znovu.

5.1 Vyhodnocení cílů

Pro každý specifikovaný cíl provedeme kontrolu jeho splnění.

5.1.1 Základní cíle

1. Automatizované vytváření a konfigurace prostředí

Byl vytvořen nástroj pro automatizované vytváření a konfigurace prostředí.

2. Zviditelnění výsledků generovaných automatizačním nástrojem

Byl vytvořen nástroj „Kontrolní panel“ pro zviditelnění výsledků generovaných automatizačním nástrojem

3. Automatizace procesu nasazování

Bylo implementována automatická reakce na změnu ve verzovacím systému, která spustí nástroj pro automatizované nasazení

4. Sjednocení nasazování na různá prostředí

Nástroj pro vytváření prostředí a nasazení využívá jednotnou konfiguraci pro všechna prostředí.

5. Automatizace testování

V rámci implementovaného nástroje pro vytváření prostředí a nasazení bylo automatizováno spouštění testovacích skriptů.

6. Automatická správa dat

V rámci implementovaného nástroje pro vytváření prostředí a nasazení bylo automatizováno spouštění instalačních a migračních skriptů.

5.1.2 Specifické cíle

1. Konfigurace procesu zvlášť pro každý projekt

Každý projekt má nadefinovanou vlastní konfiguraci procesu pomocí souboru `.codev`.

2. Konfigurovatelnost procesu v čase

Proces je konfigurovatelný v čase, pomocí změny v konfiguračním souboru `.codev` a případně návazných skriptů. Obojí je verzováno společně se zdrojovým kódem aplikace.

3. Volitelnost jednotlivých kroků procesu

Kroky procesu jsou volitelné a i samostatně konfigurovatelné pomocí konfiguračního souboru `.codev`.

4. Nenáročnost zavedení procesu na již existující projekty

Implementovaný nástroj pro vytváření prostředí a nasazení podporuje i starší operační systémy a je navržen tak, aby byl schopen se napojení na již existující automatizační skripty.

5. Kompatibilita s již používanými nástroji a postupy

V rámci implementace byly zvoleny firmě blízké technologie a nástroje a v maximální míře byly zachovány již existující postupy.

5.1.3 Celkové vyhodnocení cílů

Všechny cíle byly splněny.

5.2 Hodnocení stavu po implementaci procesu

Dle metodiky z kapitoly 3.9 lze vyhodnotit stav implementace continuous delivery následujícím způsobem:

Integrace a sestavení aplikace a konfigurace prostředí

4 body - Automatizace je úplná, závislosti jsou spravovány pomocí automatických instalačních skriptů. Nasazení lze delegovat na server spravující zdrojový kód a dochází k postupnému přechodu změn přes jednotlivá prostředí až po produkční.

Testování, kontrola kvality a automatizace nasazování

1 bod - Probíhá pouze jednotkové testování a je zavedena metrika rozsahu pokrytí kódu testy. Nicméně žádné další metriky nejsou zavedeny.

Práce s verzovacím nástrojem

2 body - Probíhá poměrně sofistikovaná správa kódu pomocí větvení a slučování změn. Částečně existuje vazba k jednotlivých úkolů v systému projektového řízení jako funkční větve, ale tento proces není automatizován.

Viditelnost změn

2 body - Na změny kódu a stav nasazení aplikace je viditelný v nástroji „Kontrolní panel“. Všechny zúčastněné osoby se mohou podívat jak nasazení proběhlo.

5.2.1 Celkové hodnocení stavu

Bylo dosáhnuto celkového zisku **9 bodů** z celkových 20 možných.

Ani jednou nedosáhnulo bodové ohodnocení 0 bodů, **ve firmě COEX s.r.o. je dle této metodiky zaveden proces continuous delivery**.

5.3 Celkové vyhodnocení

Všechny cíle práce byly splněny a do firmy COEX s.r.o. byl úspěšně zaveden proces continuous delivery. V hodnocení stavu implementace procesu bylo získáno 9 bodů z 20 možných, je zde tedy ještě prostor pro další zlepšování.

5.4 Další přínosy a očekávání

Již po krátké době používání se projevily další přínosy zavedení procesu continuous delivery:

- Zrychlení odezvy od uživatelů systému,

- zrychlení zapojení pracovníků do projektu z důvodu zkrácení doby rozfungování aplikace na vývojářeké stanici.

V douhodobějším horizontu se také očekávají další přínosy uvedené v kapitole 3.4. Jsou to zejména:

- Zvýšení celkové kvality,
- snížení nákladů na opravy chyb.

Na poli open-source nástrojů pro automatizaci neexistuje mnoho takto implementovaných komplexních řešení. Největším přínosem této práce je rozhodně nástroj pro vytváření prostředí a nasazení `codev`. Tento nástroj je také uvolněn jako open-source. Oproti ostatním „konkurenčním“ řešení zohledňuje i vývoj projektů v čase. Je díky implementovanému mechanismu kontrolního a výkonného módu zpětně kompatibilní aniž by výrazně omezoval své novější implementace. Zároveň je koncipován jako decentralizovaný nástroj, a tak umožňuje sjednocení nasazení nejen na sdílená prostředí ale i na prostředí vývojová. Toto ve větší míře umožňují pouze nástroje založené na technologii Docker, které však nejsou tak snadno implementovatelné do starších již dlouho běžících projektů. Modulární architektura řízená konfigurací pro každý projekt zvlášť také umožňuje nevázat se na jednu konkrétní technologii a dostat se tak do slepé uličky vývoje.

6 Závěr

Základní cíl, implementace procesu continuous delivery webových aplikací ve zvolené firmě COEX s.r.o., byl úspěšně splněn. Navíc byl při praktické implementaci vyvinut nástroj `co-dev`, který přesahuje rámec této práce. Je uveřejněn jako open-source projekt a jeho další vývoj tak již není pouze v rukou jediného autora. V architektuře tohoto nástroje byla zúročena mnohaletá praktická zkušenost i relativně čerstvě nabité teoretické znalosti, ale vhled cizích očí a práce dobrovolníků umožní nástroji dále růst.

Další očekávání přicházejí směrem od implementace procesu. Bude velmi zajímavé sledovat, jestli a jaké přínosy proces do budoucna ještě přinese. Již nyní lze subjektivně připustit, že se po implementaci automatizace zvýšil klid při nasazení a stává se z tohoto dříve velmi bolestivého procesu téměř rutinní záležitost.

Implementace nejmodernějších technologií má ale i své stinné stránky. Tak jak jsou systémy stále složitější, jsou i náchylnější na chyby zanesené dalšími i nepřímými závislostmi. Při vývoji došlo několikrát k aktualizacím jak kontejnerové virtualizace, tak i k vydáním nového jádra operačního systému, které spolu v kombinaci s dalšími softwarovými aktualizacemi zapříčinilo nečekané pády aplikace a dočasné znefunkčnění nástrojů. Někdy vyvstanula i nutnost zasáhnout do již stabilizované verze nástrojů. Očekávám ale, že se postupem času situace více uklidní. Kontejnerová virtualizace LXC, která je výchozím modulem pro izolaci a vytváření prostředí implementovaného nástroje vstoupila mezi výchozí nainstalované balíčky distribuce Ubuntu 16.04 LTS s dlouhou podporou.

Autora nepotěšila i naprostá absence jakýkoliv českých zdrojů o problematice a to i přesto, že se v České republice o continuous delivery ve vývojářských kuloárech a na konferencích poměrně dost hovoří.

Řešení uvedené v této práci přináší do světa automatizačních řešení pro nasazení a vývoj softwarových projektů novou krev. Všechny vlastnosti implementovaného systému byly navrženy pro maximalizaci svobody volby nejen použitých technologií ale i samotných uživatelů. Software by podle názoru autora neměl člověka omezovat, i když to mohlo původně být myšleno dobře, ale měl by rozvíjet jeho schopnosti a pouze ho dobrovolně zbavit opakujících se a nekreativních činností.

Reference

[1] *Continuous Delivery : What It Is and How to Get Started*. Puppet labs, 2013. Dostupné z: <https://puppetlabs.com/2014-devops-report>.

[2] *OpenSSH* [online]. Dostupné z: <http://www.openssh.com/>.

[3] *PCMag: Encyclopedia* [online]. Dostupné z: <http://www.pcmag.com/encyclopedia/term/54272/web-application>.

[4] Ambler, S. W. – Lines, M. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. 2012. ISBN 9780137041138.

[5] Ambler, S. – Sadalage, P. *Refactoring databases: Evolutionary database design*. 4. Pearson Education, 2006. ISBN 978-0321774514.

[6] Beck, K. – Andres, C. *Extreme Programming Explained: 2nd edition*. Addison-Wesley Professional, 2004.

[7] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000. ISBN 0-201-61641-6.

[8] Boehm, B. W. SPIRAL MODEL OF SOFTWARE DEVELOPMENT AND ENHANCEMENT. *Computer*. 1988, 21, s. 61–72. ISSN 00189162. doi: 10.1109/2.59.

[9] Carzaniga, A. et al. A Characterization Framework for Software Deployment Technologies. *Technical Report*. 1998, s. 1–6.

[10] Chen, L. Continuous Delivery: Huge Benefits, But Challenges Too. *IEEE Software*. 2015, 32, 2, s. 50 – 54. ISSN 0740-7459. doi: 10.1109/MS.2015.27.

[11] Cohn, M. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley, 2009. ISBN 9780321660565.

[12] Crispin, L. – Gregory, J. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley, 2008. ISBN 9780321616937.

[13] Duvall, P. – Matyas, S. – Glover, A. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007. ISBN 978-0321336385.

[14] Evans, C. C. *The Official YAML Web Site* [online]. 2011. Dostupné z: <http://yaml.org/>.

[15] Felter, W. et al. An Updated Performance Comparison of Virtual Machines and Linux Containers. *Technology*. 2014, 25482, s. 171–172. doi: 10.1109/ISPASS.2015.7095802.

[16] Ford, N. – Richards, M. *O'Reilly - Software Architecture Fundamentals* [online]. O'Reilly Media, 2014.

[17] Fowler, M. *Continuous Integration* [online]. 2006. Dostupné z: <http://martinfowler.com/articles/continuousIntegration.html>.

[18] Fowler, M. *ContinuousDelivery* [online]. 2013. Dostupné z: <http://martinfowler.com/bliki/ContinuousDelivery.html>.

[19] Fowler, M. – Foemmel, M. *Continuous Integration* [online]. 2000. Dostupné z: <http://www.martinfowler.com/articles/originalContinuousIntegratic

[20] Gajda, W. *Pro Vagrant*. Apress, 2015. ISBN 9781484200742.

[21] Gilb, T. *Principles of Software Engineering Management*. Addison-Wesley, 1988. ISBN 0201192462.

[22] Humble, J. – Read, C. – North, D. The Deployment Production Line. In *AGILE 2006 (AGILE'06)*, s. 113–118. IEEE. doi: 10.1109/AGILE.2006.53. Dostupné z: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=166 ISBN 0-7695-2562-8.

[23] Humble, J. – Farley, D. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010. ISBN 860-1401501176.

[24] Hüttermann, M. *DevOps for developers*. Apress, 2012. ISBN 978-1-430-24569-8.

[25] McConnell, S. *Code complete*. Microsoft Press, 2004. ISBN 0735619670.

[26] Netcraft. *March 2016 Web Server Survey* [online]. 2016. Dostupné z: <http://news.netcraft.com/archives/2016/03/18/march-2016-web-serve

[27] Newman, S. *Building Microservices*. O'Reilly Media, Inc., 2015. ISBN 9781491950357.

[28] Praqma. *Continuous Delivery Maturity Model* [online]. 2013. Dostupné z: <http://tech-notes.fransimo.info/wp-content/uploads/2013/08/cdmatu

[29] Prechelt, L. Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Advances in Computers*. 2003, 57, s. 205–270.

[30] Raghunathan, B. *The Complete Book of Data Anonymization: From Planning to Implementation*. CRC Press, 2013. ISBN 978-1439877302.

[31] Ronacher, A. *Flask-SQLAlchemy* [online]. Dostupné z: <http://flask-sqlalchemy.pocoo.org/2.1/>.

[32] Ronacher, A. *Flask* [online]. 2013. Dostupné z: <http://flask.pocoo.org/docs/0.10/>.

[33] Sahoo, J. – Mohapatra, S. – Lath, R. Virtualization: A survey on concepts, taxonomy and associated security issues. *2nd International Conference on Computer and Network Technology, ICCNT 2010*. 2010, s. 222–226. doi: 10.1109/ICCNT.2010.49.

[34] Scott Chacon, B. S. *Pro Git*. Apress, 2014. ISBN 9781484200766.

[35] Walls, M. *Building a DevOps Culture*. O'Reilly Media, Inc., 2013. ISBN 978-14-4936-417-5.