Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

DIPLOMOVÁ PRÁCE

2012 Tomáš BAUER

Zde bude zadání diplomové práce.

Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

Obor: Aplikovaná informatika

Diplomová práce

AUTOMATIZACE NASAZENÍ PHP APLIKACÍ

Diplomant: Bc. Tomáš Bauer
Vedoucí práce: Ing. Jiří Kosek

Praha 2012

Prohlášení

Prohlašuji, že jsem vypracoval samostatně diplomovou práci na téma *Automatizace nasazení PHP aplikací*. Použitou literaturu a další podkladové materiály uvádím v přiloženém seznamu literatury.

V praze dne 26. června 2012.

. . . . . .
podpis diplomanta

Poděkování

Rád bych tímto poděkoval vedoucímu své bakalářské práce, panu Ing. Jiřímu Koskovi, za odborné vedení, čas věnovaný konzultacím a všechny podnětné tipy a rady, které přispěly ke zvýšení kvality práce.

Abstrakt

Tato práce se zabývá problematikou doručení softwarového produktu koncovým uživatelům a automatizací souvisejících procesů. Po úvodním seznámení s problematikou nasazení tak, jak ji definuje jedna z nejrozšířenějších metodik softwarového inženýrství RUP (*Rational Unified Process*), se práce věnuje přístupům *Continuous Integration*, *Continuous Delivery* a *Countinuous Deployment*, které se na doručení softwarového produktu na bázi automatizovaných procesů přímo zaměřují. Druhá část práce dokumentuje návrh a implementaci řešení pro automatizaci nasazení internetových aplikací v PHP v rámci konkrétní vývojářské firmy. Samotný návrh pak pokrývá analýzu současného stavu ručního procesu nasazení firemních aplikací spolu s možnými způsoby automatizace jeho jednotlivých částí (aktualizace zdrojového kódu aplikace a databázového schématu, konfigurace aplikace, aj.), vymezení základních požadavků na budoucí řešení a analýzu dostupných nástrojů, na jejímž základě jsou vybrány nejvhodnější nástroje tvořící komponenty výsledného řešení. Součástí práce je také popis zajímavých nebo netriviálních částí řešení týkajících se zavedení či implementace jednotlivých komponent, praktický případ použití systému a následné zhodnocení výsledného řešení s jeho odhadovanými přínosy a náměty na budoucí rozšíření funkcionality.

Klíčová slova: Continuous Integration, Continuous Delivery, internetová aplikace, PHP, Phing, automatizace nasazení

Abstract

This diploma thesis deals with the delivery of software products to end users and automation of related processes. After the initial introduction to the issue of deployment, as it is defined by one of the widely used software engineering methodology RUP (*Rational Unified Process*), the thesis devotes to practices as *Continuous Integration*, *Continuous Delivery* and *Continuous Deployment* which are directly oriented to the delivery of a software product based on automated processes. The second part depicts the design and the implementation of the solution for automation of the deployment of web applications in PHP within the specific developer company. The design part itself covers the analysis of the current state of the manual deploying process of business applications along with possible ways of automation its parts (updating the source code and database schema, application configuration, etc.), the definition of essential requirements for the solution and analysis of available tools which is the base for the selection of the most suitable tools for each component forming the resulting solution. The thesis also depicts interesting or nontrivial parts of implementation of each component, the practical case of usage of the system, the subsequent evaluation of the resulting solution along with estimated benefits and suggestions for the future expansion of its functionality.

Keywords: Continuous Integration, Continuous Delivery, web application, PHP, Phing, deployment automation

Obsah

Seznam obrázků

1 Úvod

1.1 Předmluva

Internet je jedním z fenoménů dnešní doby. Výrazné zrychlení internetového připojení, zvýšení kapacity datových úložišť nebo nástup výkonných mobilních zařízení, to vše jsou důsledky bouřlivého rozvoje technologií, které s internetem souvisejí. Díky technologickému vývoji uplynulých let jsou současné internetové aplikace nejen dostupné téměř z jakéhokoliv místa na světě, ale také v mnoha oblastech nabízejí stejné možnosti jako jejich desktopové verze.

Se zvyšující se popularitou a počtem uživatelů internetových aplikací vzrostl také tlak vývojářských firem na rychlost vývoje a způsob doručení nové funkcionality koncovým uživatelům. Pokud firma provádí nasazení nové verze ručně, jedná se vždy o časově náročný proces spočívající v provedení celé řady netriviálních činností. Takový proces je často náchylný k chybám a pro firmu nemá takřka žádnou přidanou hodnotu.

V dnešní době je již běžné, že vývojářské firmy nasazují nové verze svých internetových aplikací i několikrát denně. Extrémním příkladem může být aplikace Flickr[1] určená pro sdílení fotografií na internetu, jejíž vývojový tým v počátcích nasazoval novou verzi každých třicet minut. Jak ale zdroj [12] zároveň uvádí, takový způsob vývoje by nebyl možný bez kvalitního nástroje umožňujícího automatizaci všech činností spojených s nasazením nové verze aplikace.

1.2 Cíle práce

Cílem úvodní části této práce bude seznámení čtenáře s problematikou doručení softwarového produktu koncovým uživatelům a s principy přístupů, které se na tuto část vývoje přímo orientují a jsou založené na plné automatizaci souvisejících procesů.

Dále bude součástí práce rozbor procesu nasazení internetových aplikací v konkrétní vývojářské firmě a definování požadavků na výsledné řešení, jehož cílem bude automatizace veškerých činností souvisejících s nasazením firemních aplikací. Na základě těchto požadavků bude následně provedena analýza a porovnání dostupných nástrojů.

Závěr práce bude věnován návrhu firemního řešení na základě získaných znalostí a popisu nejzajímavějších částí implementace jednotlivých komponent.

[1] Flickr – *http://www.flickr.com/*

2 Problematika nasazení aplikace

Desítky let vývoje softwarových produktů daly vzniknout celé řadě metodik, jejichž cílem je definovat kroky nutné k úspěšnému vývoji a dodání softwarového díla. Nehledě na zvolenou metodiku, poskytnutí výsledného softwaru koncovým uživatelům je vždy nedílnou součástí životního cyklu každého softwarového produktu.

Cílem této kapitoly je krátké nahlédnutí na životní cyklus softwarového produktu tak, jak ho popisuje jedna z tradičních metodik z oblasti softwarového inženýrství - metodika RUP, a blíže čtenáře seznámit s tím, jak je v této metodice proces nasazení aplikace definován. Druhá část kapitoly se pak věnuje základním principům přístupů, které jsou na úspěšné doručení aplikace koncovým uživatelů přímo orientované a v porovnání s metodikou RUP nahlíží na proces nasazení aplikace značně rozdílným způsobem.

2.1 Definice nasazení v metodice RUP

Metodika RUP *(Rational Unified Process)* byla vytvořena a používána společností *Rational Software Corporation*. Tato metodika rozděluje proces vývoje softwaru do čtyř základních fází: [1]

- **Zahájení *(Inception)*** - hlavním cílem této fáze je analýza požadavků na systém, vymezení jeho hranic a způsobu interakce systému s okolím – typicky formou modelů případů užití.
- **Příprava *(Elaboration)*** - popisuje kritickou fázi vývoje, jejímž hlavním výstupem je návrh architektury systému na základě požadavků definovaných v předchozí fázi.
- **Konstrukce *(Construction)*** - hlavním cílem v této fázi vývoje je implementace jednotlivých částí systému, jejich následná integrace a testování. Důraz je zde kladen především na efektivní řízení zdrojů a koordinaci vývoje.
- **Předávání *(Transition)*** - v této fázi dojde k předání finální verze systému koncovým uživatelům. Dále probíhá testovací provoz systému, na jehož základě jsou opravovány drobné chyby.

Jak je patrné z popisu jednotlivých fází vývoje softwaru v metodice RUP, nasazení aplikace je prováděno v rámci poslední fáze – Předávání. Metodika RUP definuje nasazení softwarového produktu takto: [11]

„Nasazení softwarového produktu je disciplína popisující aktivity spojené se zajištěním dostupnosti softwarového produktu pro koncové uživatele.“

2.1.1 Proces nasazení v metodice RUP

Proces nasazení softwarového produktu je v metodice RUP definován posloupností následujících aktivit:

2.1.1.1 Plánování nasazení *(Plan deployment)*

Cílem této fáze je popis způsobu doručení softwarového produktu koncovému uživateli a sestavení časového plánu nasazení. Mimo to by měl být v této části popsán také způsob doručení dalších součástí softwarového produktu, které jsou pro koncového uživatele nezbytné pro jeho bezproblémové používání (např. uživatelské manuály, hardwarové komponenty atp.).

2.1.1.2 Tvorba dokumentace *(Develop Support Material)*

Tato fáze pokrývá tvorbu veškerých informačních materiálů týkajících se daného softwarového produktu, které popisují instalaci, používání a údržbu systému koncovým uživatelem. V některých případech je součástí této fáze také tvorba podkladů pro školení budoucích uživatelů systému.

2.1.1.3 Akceptační testy *(Manage Acceptance Test)*

Před poskytnutím softwarového produktu koncovým uživatelům je kladen velký důraz na ověření jeho funkčnosti. Tato fáze popisuje dva druhy prostředí, ve kterých je systém testován. Prvním z nich je vývojové prostředí, na kterém je systém zpravidla testován průběžně v rámci vývoje. Druhým je pak přímo prostředí testovací, kde je zadavatelem ověřena funkčnost finální verze systému. Testovací prostředí by mělo být instancí prostředí, pro které je systém vyvíjen.

2.1.1.4 Vytvoření instalace *(Produce Deployment Unit)*

Tato fáze spočívá v kompletaci softwarového produktu tak, aby byl připraven na dodání zákazníkovi. V závislosti na stádiu vývoje se typicky jedná o produkt pro účely beta testování či finální verzi.

2.1.1.5 Beta testování *(Beta Test Product)*

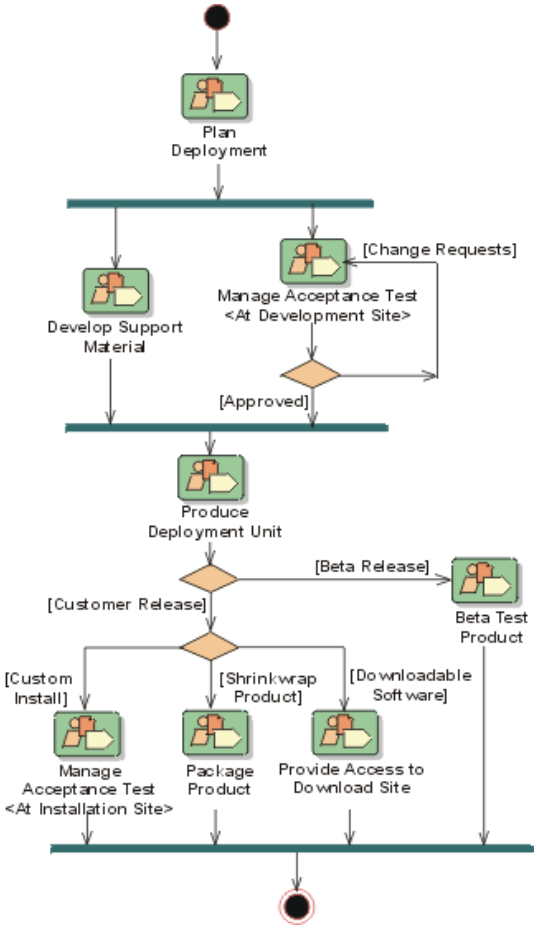
Pro účely beta testování je produkt poskytnut skupině uživatelů, od kterých je následně získána zpětná vazba, na jejímž základě může dojít k některým vylepšením produktu či odstranění nalezených chyb.

2.1.1.6 Výroba softwarového balíku *(Package Product)*

V případě tzv. krabicového softwaru je zde z jednotlivých částí (např. instalační CD, uživatelský manuál atp.) sestaven výsledný produkt a odeslán do výroby. Tato fáze se v oblasti internetových aplikací objevuje jen zřídka (např. poskytnutí off-line verze internetové aplikace) a je zde uvedena spíše pro úplnost.

2.1.1.7 Poskytnutí možnosti stažení produktu *(Provide Access to Download Site)*

V případě, že se jedná o produkt distribuovaný prostřednictvím internetu, je umožněno jeho stažení například na stránkách výrobce. V oblasti internetových aplikací pak tato fáze spočívá ve zpřístupnění produktu koncovým uživatelům prostřednictvím internetu.



Obrázek 2.1: Diagram aktivit pro nasazení produktu v metodice RUP. [11]

Protože je metodika RUP úzce spjata s jazykem UML *(Unified Modeling Language)*, jsou jednotlivé aktivity spojené s procesem nasazení softwarového produktu uvedeny do kontextu pomocí diagramu aktivit. [Obrázek 2.1]

Metodika RUP se fázi nasazení softwarového produktu nevěnuje do takové míry, jako je tomu u ostatních částí životního cyklu. Na doručení softwarového produktu koncovým uživatelů nahlíží spíše jako na jednorázovou činnost, která je provedena až v rámci poslední fáze – po ukončení vývoje a otestování daného produktu.

Existují však moderní přístupy jako *Continuous Integration*, *Continuous Delivery* a *Continuous Deployment*, které se této problematice věnují daleko podrobněji a na průběžnou integraci, testování a doručování nové funkcionality aplikace koncovým uživatelům nahlíží jako na činnost, která by měla být prováděna nepřetržitě již v rámci vývoje softwarového produktu. Právě těmto přístupů se budou podrobněji věnovat následující kapitoly.

2.2 Continuous Integration

Continuous integration (česky průběžná integrace) představuje metodiku vývoje softwarového produktu z oblasti extrémního programování. Martin Fowler popisuje tuto metodu vývoje ve svém článku [7] následovně:

„Průběžná integrace je metoda vývoje softwaru, při které členové týmu častěji integrují jejich práci – každý typicky minimálně jednou denně – což vede k několika integracím za den. Každá integrace je pak ověřována automatickým sestavením aplikace (zahrnujícím spuštění testů, analýzy kódu atp.), díky čemuž dojde k odhalení integračních chyb tak brzy, jak je to jen možné. U mnoha týmů tento přístup vede k značnému redukování množství problémů s integrací a umožňuje jim tak výrazně urychlit vývoj softwaru.“

2.2.1 Cyklus průběžné integrace

K snazšímu pochopení celého principu průběžné integrace je třeba blíže popsat, jak celý proces integrace probíhá. Tento proces je označován jako tzv. cyklus průběžné integrace, jehož schéma je znázorněno na obrázku 2.2. [5]

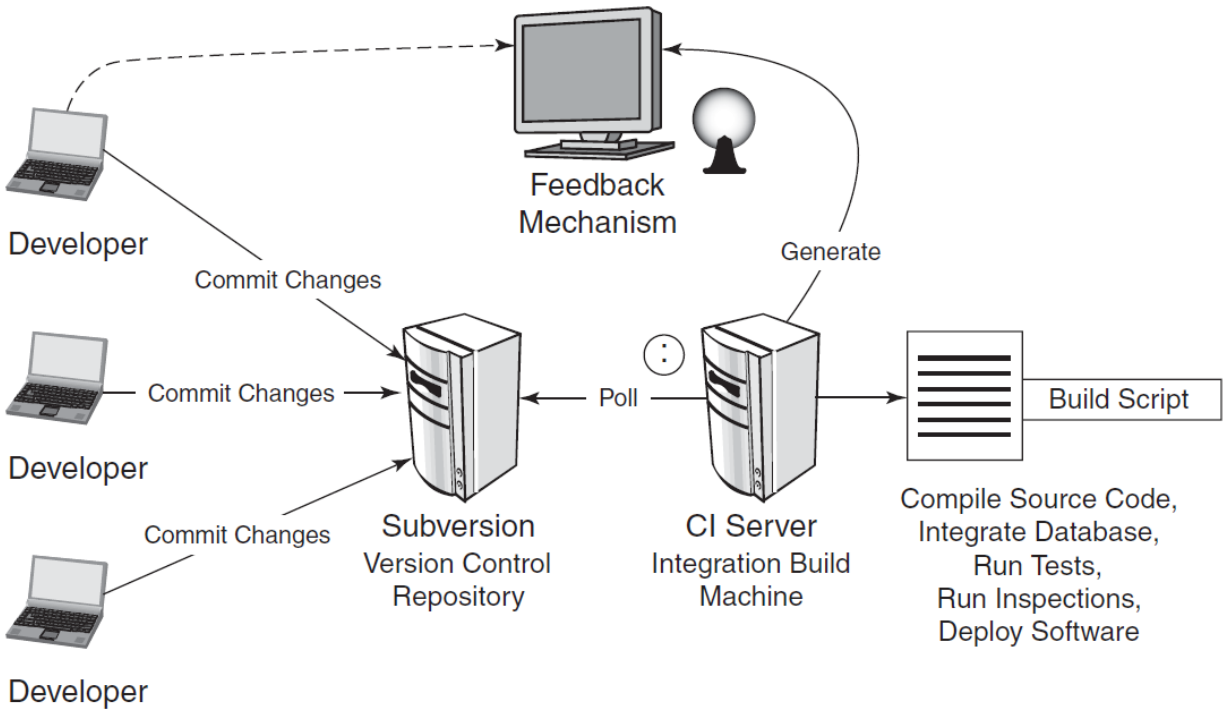
Cyklus průběžné integrace je spouštěn každým nahráním změn do projektového repozitáře a obsahuje následující kroky:

1. Mezitím, co server průběžné integrace (dále CI server) sleduje, zda v projektovém repozitáři došlo k nějakým změnám (typicky každých několik minut), někdo z členů vývojového týmu nahraje změny do projektového repozitáře.

2. Brzy poté CI server detekuje změny v projektovém repozitáři, získá poslední verzi projektu a spustí na ní integrační skript, který provede samotnou integraci softwaru.

3. O výsledcích integrace poté informuje členy projektu obvykle formou e-mailového reportu.

4. CI server se vrátí ke sledování změn v projektovém repozitáři.



Obrázek 2.2: Cyklus průběžné integrace a jeho komponenty. [5]

Z výše popsaného cyklu průběžné integrace je patrné, že samotný proces integrace spočívá ve spuštění integračního skriptu. Sestavení aplikace zde však neznamená pouhou kompilaci zdrojového kódu, ale kromě dalšího je zde zahrnuto také například spouštění testů, analýza kódu nebo automatizované nasazení aplikace typicky do nějakého „production-like“ prostředí. Hlavním cílem tohoto procesu je ověření, zda jsou jednotlivé komponenty vyvíjeného softwaru schopny pracovat jako jeden celek.

2.2.2 Zásady průběžné integrace

Aby mohl vývojový tým z tohoto přístupu vytěžit skutečně maximum, je ve většině případů nutné, aby každý člen týmu změnil své dosavadní návyky při práci na projektu. Nelze totiž spoléhat pouze na technickou stránku věci, ale je také třeba, aby členové týmu přijali za své několik důležitých zásad, které posléze musí být důsledně dodržovány. Tyto zásady shrnuje Paul Duvall [5] v následujících sedmi pravidlech:

- časté nahrávání změn do projektového repozitáře
- nikdy nenahrávat nefunkční kód do projektového repozitáře
- okamžitá oprava neúspěšného sestavení aplikace
- psaní automatizovaných testů

- úspěšný průběh všech testů a analýzy kódu
- lokální spuštění integračního skriptu před nahráním změn do projektového repozitáře
- vyhnutí se získání nefunkčního kódu

2.2.2.1 Časté nahrávání změn do projektového repozitáře

Jedním z nejdůležitějších pravidel je časté nahrávání změn zdrojového kódu do projektového repozitáře. Teprve tehdy členové týmu skutečně pocítí hlavní přínosy průběžné integrace. Intervaly nahrávání změn delší než jeden den způsobují to, že se integrace stane časové náročnou operací a navíc zamezí ostatním členům týmu využít poslední změny v aplikaci při své práci.

Jednou z technik, jak toho lze dosáhnout je ta, kdy vývojář současně upravuje pouze jednotlivé komponenty systému. Rozdělí tedy svůj úkol na více menších částí, například postupuje tak, že napíše test, upraví zdrojový kód aplikace, spustí test a ihned po jeho úspěšném dokončení nahraje své změny do projektového repozitáře.

2.2.2.2 Nikdy nenahrávat nefunkční kód do projektového repozitáře

Další důležitým předpokladem pro úspěšné provádění průběžné integrace je fakt, že každý člen týmu si je vědom toho, že nikdy nesmí nahrát nefunkční kód do projektového repozitáře. Toto pravidlo úzce souvisí s jedním z následujících pravidel o lokálním spuštění integračního skriptu před nahráním změn do projektového repozitáře, neboť právě jeho dodržováním by k nahrání nefunkčního kódu nemělo docházet.

2.2.2.3 Okamžitá oprava neúspěšného sestavení aplikace

Oprava příčiny neúspěšného sestavení aplikace, ať už se jedná o chybu při kompilaci, neúspěšný test či problém s databází, by měla být hlavní prioritou vývojového týmu.

2.2.2.4 Psaní automatizovaných testů

Veškeré činnosti spouštěné v rámci sestavení aplikace by měly být plně automatizované a právě z toho důvodu musí být automatizované také testování aplikace.

2.2.2.5 Úspěšný průběh všech testů a analýzy kódu

U průběžné integrace je klíčové, aby integrační skript aplikace proběhl vždy úspěšně. Vzhledem k tomu, že jak automatizované testy, tak analýza kódu je spouštěna v rámci tohoto skriptu, je nezbytně nutné, aby i ty proběhly se stoprocentním úspěchem. Pokud by byl tolerován kód, který neprošel všemi automatizovanými testy, snížila by se tak kvalita výsledného softwarového produktu. Dobrým zvykem bývá také doplnění integračního skriptu o spouštění nástroje pro analýzu pokrytí aplikace testy.

2.2.2.6 Lokální spuštění integračního skriptu před nahráním změn do projektového repozitáře

Vývojáři by měli simulovat spuštění integračního skriptu již v rámci svého vývojového prostředí včetně všech unit testů a snížit tak riziko nahrání nefunkčního kódu do projektového repozitáře. V této fázi je obzvláště důležité to, aby vývojář integroval svoji novou verzi softwaru s verzemi všech ostatních vývojářů získáním posledních změn z projektového repozitáře a následným sestavením aplikace, čímž dojde k výraznému snížení pravděpodobnosti selhání integračního skriptu.

2.2.2.7 Vyhnutí se získání nefunkčního kódu

V případě, že nastane situace, kdy se v projektovém repozitáři vyskytuje nefunkční verze aplikace (došlo selhání integračního skriptu), neměl by si tuto verzi načítat žádný z vývojářů. Pokud by se tak stalo, byl by nucen vynaložit mnohdy nemalé množství času pouze na to, aby tuto chybu obešel a mohl aplikaci spustit a otestovat tak svojí část kódů. Konec konců, jak již bylo uvedeno výše, oprava nefunkčního sestavení aplikace by měla být pro vývojový tým vždy prioritou číslo jedna.

2.2.3 Přínosy průběžné integrace

Jak je patrné, průběžná integrace je z větší části přístup založený na sadě pravidel a určité disciplíny všech členů projektového týmu. Výše uvedené principy představují pouze nejnutnější základ, bez kterého může vývojový tým jen stěží prohlásit, že provádí průběžnou integraci. Podrobnější popis tohoto přístupu je již nad rámec této práce, nicméně na závěr si ještě shrňme největší výhody, které s sebou tento přístup přináší. [5]

- Snižuje rizika vývoje.
- Snižuje počet opakovaně prováděných manuálních činností.
- Generuje software, který lze kdykoliv nasadit.
- Zlepšuje přehled členů týmu o stavu aplikace.
- Zvyšuje důvěru týmu v daný softwarový produkt.

2.2.3.1 Snížení rizik vývoje

Integrace, která je prováděna několikrát za den, pomáhá snížit rizika spojená s vývojem softwaru tím způsobem, že urychluje a usnadňuje detekci chyb, pravidelně provádí úkony spojené s kontrolou kvality softwaru a ověřuje předpoklady aplikace, například z hlediska konfigurace produkčního prostředí.

- **Rychlejší detekce chyb a jejich oprava** - vzhledem ke spouštění automatizovaných testů a analýzy kódu již v rámci integračního skriptu aplikace jsou chyby objeveny a opraveny dříve, než tomu bývá zvykem u jiných přístupů, kde je většina chyb objevena až v rámci fáze určené k testování aplikace.
- **Měření kvality softwaru** - zahrnutím automatizovaných testů a analýzy kódu v rámci integračního skriptu lze průběžně sledovat ukazatele kvality softwarového produktu.
- **Ověření předpokladů aplikace** - vzhledem k pravidelnému spouštění integračního skriptu v tzv. čistém integračním prostředí jsou ověřovány předpoklady týkající se například nastavení proměnných běhového prostředí či závislostí na knihovnách třetích stran.

2.2.3.2 Snížení počtu opakovaně prováděných manuálních činností

Snížení počtu opakovaně prováděných manuálních činností jejich automatizací ušetří čas, náklady a vynaložené úsilí členů týmu. Toto se týká především činností spojených s kompilací kódu, migrací databáze, testování aplikace, analýzy kódů, nasazení či reportování o výsledcích jednotlivých částí. Automatizace těchto činností v rámci průběžné integrace pak zajistí následující:

- Veškeré tyto činnosti jsou vždy prováděny stejným způsobem.
- Je zajištěno vždy stejné pořadí spouštění jednotlivých částí.
- Všechny tyto činnosti jsou spouštěny při každém nahrání změn do projektového repozitáře.
- Osvobozuje členy týmu od práce, která nemá takřka žádnou přidanou hodnotu, a šetří tak firemní zdroje.

2.2.3.3 Generování softwaru, který lze kdykoliv nasadit

Jeden z nejpodstatnějších přínosů průběžné integrace je ten, že aplikace je neustále udržována ve stavu, ve kterém je možné kdykoliv nasadit její poslední verzi. Pro klienty a budoucí koncové uživatele je právě nasazení nové verze aplikace tím nejdůležitějším momentem, při kterém jim je doručena nejnovější funkčnost aplikace či sada oprav některých chyb. V případě, že při integraci dojde k nějakému problému, členové vývojového týmu jsou okamžitě informováni pomocí automatizovaného reportování a musí neprodleně začít daný problém řešit.

Vedlejší výhodou tohoto přístupu je také to, že díky malým blokům změn ve zdrojovém kódu je pro vývojáře podstatně snadnější odhalit a odstranit příčinu daného problému. U projektů, kde není využit přístup průběžné integrace, se tak stává, že většina problémů a chyb aplikace vypluje na povrch až v rámci integrační a testovací fáze softwarového

produktu (typicky před jeho vydáním). To obvykle vyžaduje daleko větší úsilí při jejich řešení a může tak způsobit oddálení či dokonce zabránění vydání daného softwaru.

2.2.3.4 Zlepšení přehledu členů týmu o stavu aplikace

Vzhledem k tomu, že při průběžné integraci jsou v rámci integračního skriptu mimo jiné neustále sledovány kvalitativní metriky aplikace, má tak projektový tým k dispozici důležité informace o směru, kterým se vývoj aplikace ubírá. Na základě těchto informací lze učinit efektivnější rozhodnutí, například při zavádění nových postupů a vylepšení vedoucích ke zvýšení kvality vyvíjeného softwaru.

2.2.3.5 Zvýšení důvěry týmu v daný softwarový produkt

Efektivní aplikace principů průběžné integrace celkově napomáhá ke zvýšení důvěry týmu ve vyvíjený softwarový produkt. Při každém sestavení aplikace se může tým vývojářů spolehnout na to, že proběhly veškeré automatizované testy prověřující jak funkční chování aplikace, tak dodržení určitých standardů z hlediska návrhu a kvality zdrojového kódu.

Bez často prováděné integrace nových změn v aplikaci si vývojáři často nejsou úplně jistí, jaký dopad mají jejich úpravy zdrojového kódu na zbylou část aplikace. Vzhledem k tomu, že systém průběžné integrace na tyto problémy velice brzy upozorní, jsou si vývojáři i ostatní členové týmu při provádění změn podstatně jistější.

2.3 Continuous Delivery

Continuous delivery je dalším z přístupů z této oblasti. Na rozdíl od průběžné integrace zachází ještě o něco dále. Zatímco při průběžné integraci dochází pouze k častému sestavení aplikace, spuštění automatizovaných testů, analýzy kódu atp., zde je ke všemu navíc přidán ještě další krok spočívající v nasazení aplikace na testovacím prostředí. Předpokladem tohoto přístupu je fakt, že testovací prostředí pak co možná nejvíce odpovídá produkčnímu prostředí (tzv. „*production-like*“ prostředí). Nemůže se tedy projektovému týmu stát to, že integrační skript v rámci vývojového prostředí proběhne bezchybně, zatímco v produkčním prostředí se aplikace z nějakého důvodu nebude chovat korektně (např. kvůli problémům s právy atp.).

Jez Humble a David Farley ve své knize [8] uvádí několik běžných anti-vzorů, které ačkoli jsou tak běžné, že by se daly označit za standard, při vývoji kvalitního softwarového produktu je potřeba se jim vyhnout. Jeden z nich s touto prací úzce souvisí a je mu věnována následující kapitola.

2.3.1 Anti-vzor: Manuální nasazování softwarového produktu

Tento anti-vzor popisuje problémy způsobené manuálním nasazením aplikace. Nasazení většiny moderních aplikací bývá netriviálním úkolem, který zahrnuje velké množství rutinních kroků činící tento proces náchylný k lidským chybám.

Nasazení rozsáhlejších aplikací je pak často úkolem několika členů či dokonce týmů, z nichž každý je zodpovědný za nasazení té své části. Každý z nich pak obvykle používá rozdílné postupy a dokonce i v případě, že se nikdo z členů žádné chyby nedopustí, pouhá změna v pořadí prováděných kroků nebo jejich načasování může vést k rozdílným výsledkům, které nejsou žádoucí. Takový způsob nasazování vede k následujícím problémům:

- Potřeba tvorby rozsáhlé dokumentace popisující kritické kroky nasazení aplikace.
- Závislost na manuálním testování správné funkčnosti aplikace.
- Vývojový tým často řeší problémy s nasazením aplikace v den jejího vydání.
- Časté úpravy procesu nasazení v průběhu vydání aplikace.
- Lišící se konfigurace prostředí pracujících v clusteru, například aplikační servery s rozdílnou konfigurací, souborovým systémem atp.
- Proces nasazení aplikace trvá déle než několik minut.
- Nepředvídatelné výsledky nasazení aplikace, které často vedou k problémům vyžadující roll-back.

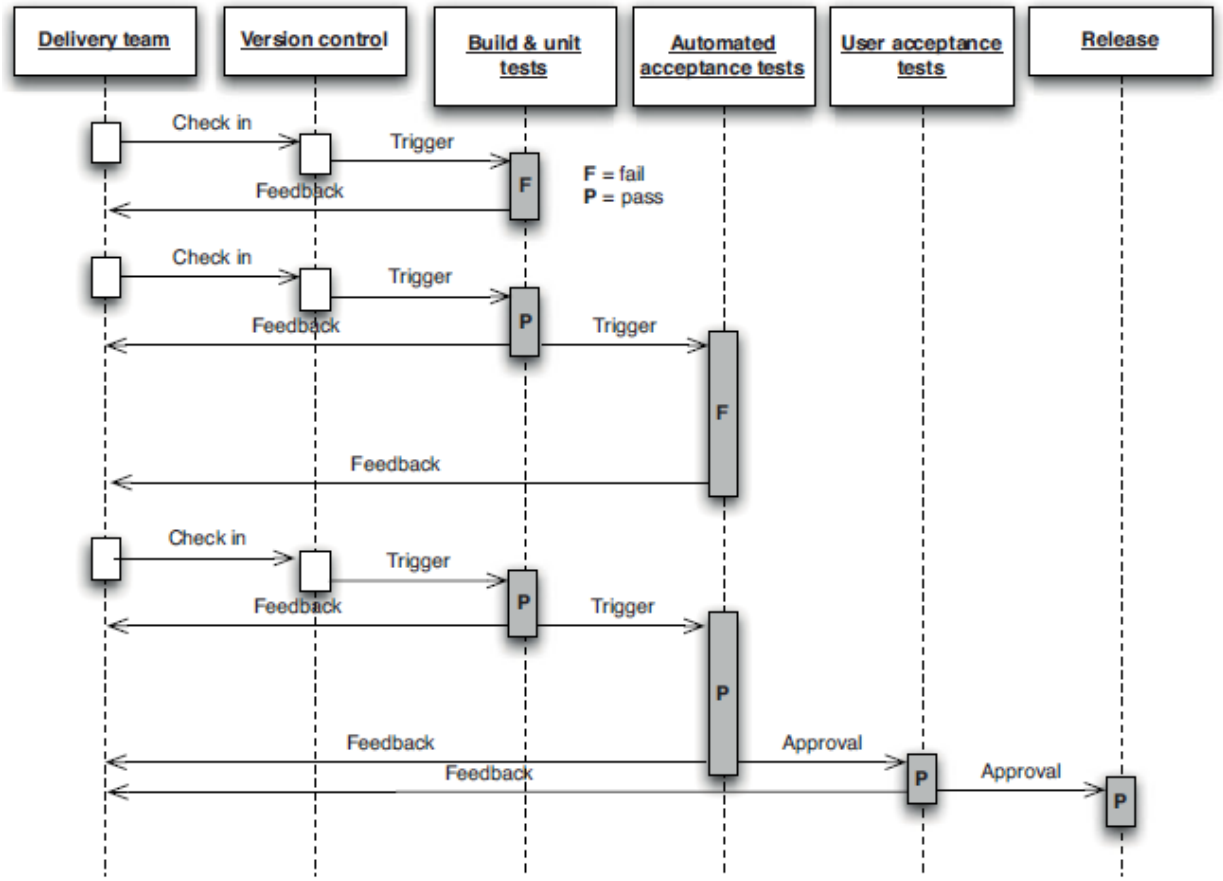
Pokud firma provádí nasazení svých aplikací právě tímto způsobem a často se střetává s výše popsanými problémy, měla by se zamyslet nad úpravou procesu nasazení svých aplikací a to tak, aby v budoucnu dosáhla toho, že celý proces nasazení softwaru bude plně automatizovaný. Proces nasazení aplikace by měl být úkolem vhodným pro jediného člena týmu, v ideálním případě by ho měl být schopen provést například i člen managementu. Při plné automatizaci tohoto procesu je nasazení aplikace věcí tří triviálních kroků:

1. Zvolit verzi aplikace, kterou chceme nasadit.
2. Zvolit prostředí, do kterého chceme danou verzi aplikace nasadit (např. vývojové, testovací, produkční aj.).
3. Stisknout tlačítko „Nasadit“.

Základní principy a pravidla, které je mít na paměti při automatizaci procesu nasazení softwarových produktů, popisuje návrhový vzor *Deployment Pipeline*, jemuž se věnují následující kapitoly této části práce.

2.3.2 Deployment Pipeline

Deployment Pipeline představuje návrhový vzor popisující principy, kterými se řídí proces automatizovaného doručení softwarové aplikace do rukou koncových uživatelů. V každé organizaci je tento proces realizován rozdílným způsobem, nicméně jeho základ se příliš



Obrázek 2.3: Sekvenční diagram vzoru deployment pipeline. [8]

nemění. V podstatě vždy se jedná o automatizaci činností, kterými jsou sestavení, testování a nasazení aplikace. Příklad tohoto procesu je zobrazen na 2.3.

Proces nasazení podle vzoru *Deployment Pipeline* probíhá následovně. Jakákoliv úprava aplikace, ať už se jedná o změnu konfigurace, úpravu zdrojového kódu či nastavení běhového prostředí, vyvolá spuštění procesu nasazení. Prvním krokem procesu je vždy kompilace zdrojového kódu, po které následuje série testů ověřujících správnou funkčnost aplikace (unit testy, akceptační testy atp.). Pakliže všechny tyto testy úspěšně projdou, znamená to, že je možné aplikaci nasadit.

Takováto implementace procesu nasazení s sebou přináší několik důležitých výhod. První z nich spočívá v tom, že každý, koho se proces doručení softwaru týká, má k dispozici informace o všech jednotlivých částech celého procesu, což vede k výrazné podpoře týmové spolupráce. Zároveň jsou díky reportingu z každé části procesu všichni členové týmu informováni o vzniklých problémech tak brzy, jak je to možné. Nespornou výhodou tohoto procesu je také to, že lze do kteréhokoliv prostředí nasadit jakoukoliv verzi softwaru, která tímto procesem úspěšně prošla.

2.3.3 Fáze procesu nasazení dle vzoru Deployment Pipeline

Jak již bylo zmíněno výše, k dosažení ideálního stavu je zapotřebí automatizace jednotlivých fází procesu nasazení aplikace. V naprosté většině případů se jedná o sestavení aplikace, spuštění testů ověřujících, že daná verze aplikace splňuje veškeré funkční požadavky, a její nasazení do testovacího či produkčního prostředí. Tyto činnosti, pakliže jsou prováděny manuálně, jsou velmi často náchylné k chybám a tedy hlavní příčinou vzniku neočekávaných problémů.

V některých případech je tento proces navíc obohacen o některé další typy testů (např. analýza kódu, pokrytí aplikace testy aj.), nicméně její základ je téměř vždy tvořen následujícími fázemi:

- **Fáze nahrání změn do projektového repozitáře *(The commit stage)*** vyvolává spuštění procesu nasazení aplikace. V této fázi dochází ke kompilaci zdrojového kódu, jeho otestování typicky formou unit testů, analýzy kódu či přípravě instalačních souborů.
- **Fáze akceptačního testování *(Automated acceptance test stage)*** má za cíl ověřit správnou funkčnost aplikace z hlediska jejího chování, tedy zda toto chování odpovídá potřebám koncových uživatelů či požadavkům zákazníka.
- **Fáze manuálního testování *(Manual test stages)*** doplňuje automatizované testování. Jejím cílem je odhalení chyb, které nebyly objeveny automatizovaným testováním. Typicky je zde prováděno uživatelské akceptační testování.
- **Fáze nasazení aplikace *(Release stage)*** spočívá v doručení softwaru koncovým uživatelům, buď formou balíčkové aplikace nebo jejím nasazením do produkčního či testovacího prostředí, které je identickou kopií produkčního prostředí.

2.3.4 Pravidla a zásady vzoru Deployment Pipeline

Aby byl proces automatizovaného doručení softwarového produktu co možná nejefektivnější a přinesl vývojovému týmu skutečné výhody, je třeba se držet několik pravidel, které autoři knihy [8] shrnují takto:

- Kompilace zdrojového kódu je provedena pouze jednou.
- Pro nasazení do všech prostředí je používán stejný deployment skript.
- Po každém nasazení aplikace je spuštěn smoke test.
- Testování a integrace probíhá v production-like prostředí.
- Nahrání změn do projektového repozitáře spouští proces nasazení.
- Selže-li některá z fází procesu, odstranění problému je hlavní prioritou týmu.

2.3.4.1 Kompilace aplikace je provedena pouze jednou

Jak již bylo popsáno výše, pro sestavení aplikace, spuštění testů, nasazení aplikace atp. by měl být zdrojový kód vždy načítán z projektového repozitáře. Problémy při procesu doručení však může způsobit případ, kdy je tento zdrojový kód opakovaně kompilován pro každou jednotlivou fázi procesu doručení softwaru. Při takovémto postupu totiž existuje riziko toho, že zdrojový kód bude pokaždé zkompilován trochu rozdílným způsobem, což může být zapříčiněno například rozdílnou verzí kompilátorů (nebo jejich konfigurace) pro jednotlivé fáze či použití rozdílných knihoven třetích stran.

Opakovaná kompilace zdrojového kódu pro jednotlivé fáze procesu doručení softwarů narušuje hned dva z principů toho přístupu. Především výrazně snižuje efektivitu celého procesu jeho zpomalením, což vede k pomalejší zpětné vazbě formou reportingu z jednotlivých fází. Dále si je třeba uvědomit, že automatizované testy nemusely být spuštěny na identické verzi aplikace, která je posléze nasazena do produkčního prostředí.

Zkompilované zdrojové kódy aplikace by měly být následně uchovány (mimo projektový repozitář) pro potřeby dalších fází procesu. V tomto případě je ale zapotřebí důsledného oddělení těch částí aplikace, které jsou závislé na běhovém prostředí (např. konfigurační soubory).

2.3.4.2 Pro nasazení do všech prostředí je používán stejný deployment skript

Dalším důležitým pravidlem je jednotný způsob (skript) nasazení aplikace do jakéhokoliv prostředí, ať už se jedná o pracovní stanice vývojářů, testovací či produkční prostředí. Typicky vývojáři nasazují aplikaci nejčastěji, testeři či analytici méně častěji a nasazení do produkčního prostředí je prováděno obvykle jen zřídka. I přesto je ale pro tým nasazení do produkčního prostředí tím nejdůležitějším. Využitím stejného skriptu pro nasazení do jakéhokoliv prostředí zajistí jeho důsledné otestování již v průběhu vývoje aplikace a eliminuje tak téměř všechna rizika s ním spojená.

Jednotlivá běhová prostředí se téměř vždy vzájemně liší (např. umístěním externích knihoven, nastavením databáze, operačním systémem atp.). Tento problém však lze snadno vyřešit pomocí konfiguračních souborů obsahujících nastavení, která jsou specifická pro prostředí, do kterého je aplikace nasazována. Tyto soubory by měly být uchovány v projektovém repozitáři spolu s aplikací a při nasazení aplikace předány deployment skriptu.

I při použití tohoto přístupu se čas od času může stát, že se při nasazení aplikace nějaký problém vyskytne. Pokud ale používáme stejný, mnohokrát otestovaný deployment skript, je velice pravděpodobné, že se chyba bude týkat některé z následujících oblastí:

- Konfigurace aplikace, která je specifická pro konkrétní běhové prostředí.
- Problém s infrastrukturou či některou ze služeb, na kterých je aplikace závislá.
- Konfigurace prostředí, do kterého aplikaci nasazujeme.

2.3.4.3 Po každém nasazení aplikace je spuštěn smoke test

Po nasazení aplikace by měl být automaticky spuštěn skript provádějící tzv. „*smoke test*“. Smoke test představuje velmi jednoduchý test ověřující pouze to, zda je aplikace dostupná/spustitelná. Tento test je typicky realizován spuštěním aplikace či načtením úvodní obrazovky. V některých případech může být testována také dostupnost služeb, na kterých běh aplikace závisí (např. databáze či jiné externí služby).

Smoke test je mnohdy považován za nejdůležitější test v procesu nasazení aplikace dávající vývojovému týmu jistotu, že nasazená aplikace skutečně běží. Pokud dojde k selhání tohoto testu, musí být tento stav okamžitě nahlášen vývojovému týmu nejlépe spolu s informací o příčině tohoto selhání, například nedostupnosti některé s důležitých služeb.

2.3.4.4 Testování a integrace probíhá v production-like prostředí

Jedním z hlavních problémů, se kterým se potýká mnoho vývojových týmů při nasazení aplikace do produkčního prostředí, je ten, že se jejich produkční prostředí výrazně liší od vývojového a testovacího prostředí. Jediným možným způsobem, jak se těmto problémům při vydání softwaru vyhnout, je provádět testování a průběžnou integraci v prostředích, které se co možná nejméně odlišují od prostředí produkčního.

Zajištění toho, aby všechna prostředí byla pokud možno identická, vyžaduje jistou dávku disciplíny a správného využití konfiguračního managementu. V zásadě je vždy nutné ověřit následující:

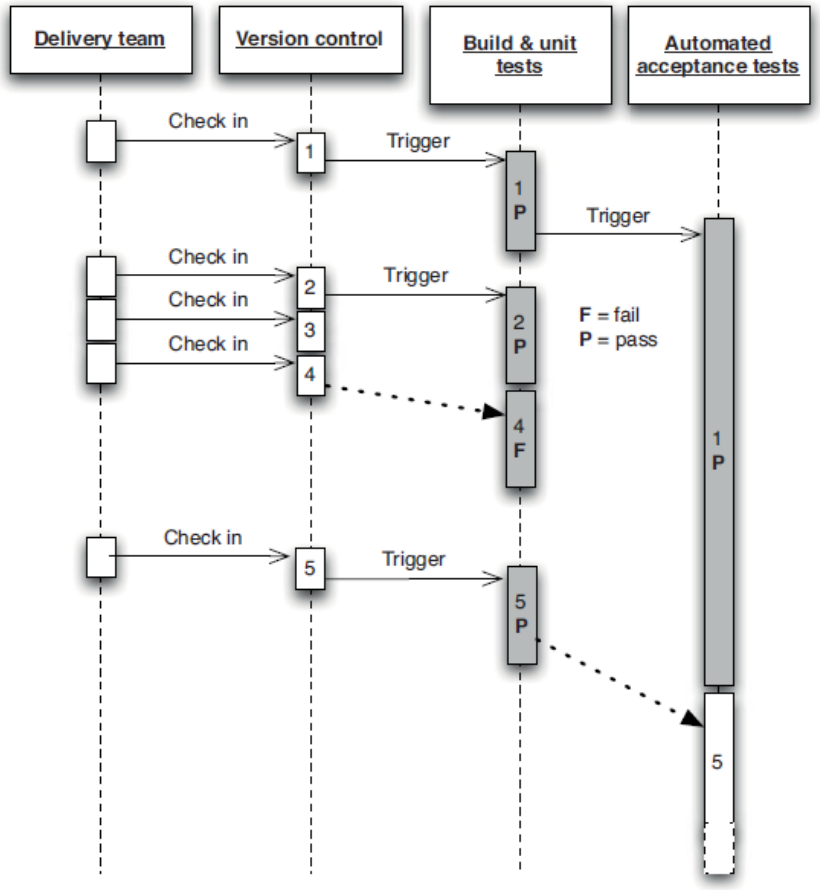
- Použití stejné infrastruktury sítě z hlediska její topologie a konfigurace firewallu.
- Využití stejného operačního systému a jeho konfigurace (včetně patchů).
- Častým zdrojem problémů při vydání nové verze aplikace je migrace dat. Je nutné, aby data využívaná aplikací byla vždy validní a ve stavu, který je očekáván.

2.3.4.5 Nahrání změn do projektového repozitáře spouští proces nasazení

V mnoha firmách, kde není prováděna průběžná integrace softwaru, probíhají její jednotlivé části odděleně v různě stanovených dobách – například sestavení aplikace každou hodinu, akceptační testy každou noc, zátěžové testy o víkendech atp. Proces nasazení podle deployment pipeline však probíhá ihned po nahrání jakékoliv změny do projektového repozitáře.

Problém ale může nastat v případě velkého počtu vývojářů a tedy velmi častému nahrávání změn do projektového repozitáře, kdy jsou do projektového repozitáře nahrány změny ještě v době, kdy není zcela dokončen proces předchozí. Tento problém je znázorněn na sekvenčním diagramu na obrázku 2.4.

Tento příklad popisuje případ, kdy jeden z vývojářů nahraje změny do projektového repozitáře a vytvoří verzi 1, čímž vyvolá spuštění první fáze procesu nasazení (sestavení aplikace, spuštění unit testů atp.). Když je tato fáze úspěšně dokončena, dojde ke spuštění druhé fáze – akceptačnímu testování. Mezitím ale některý z dalších vývojářů opět nahraje



Obrázek 2.4: Plánování spouštění fází procesu nasazení. [8]

změny do projektového repozitáře (verze 2), čímž opět vyvolá spuštění první fáze procesu. Ačkoliv ale tato fáze proběhne úspěšně, nemůže být na této verzi spuštěno akceptační testování, neboť to stále probíhá na verzi první. Navíc v této době dojde k nahrání dalších dvou sad změn do projektového repozitáře (verze 3 a 4).

V takovéto situaci by se integrační server neměl snažit spouštět proces na každé nové verzi, neboť by mohlo velice snadno dojít k neustálému zpožďování v integraci změn aplikace a následnému reportování jejich výsledků. Z toho důvodu zde platí pravidlo, že proces je vždy spouštěn pouze na nejaktuálnější verzi aplikace (v tomto případě na verzi 4). V případě, že tato verze selže hned ve své první fázi, integrační server neumí přesně říci, zda došlo k poškození aplikace nahráním změn třetí nebo čtvrté verze, nicméně pro vývojáře již nebývá velkým problémem zdroj chyb rychle odhalit a odstranit (nahráním verze 5). Když je pak konečně dokončeno akceptační testování verze 1, integrační server spustí další běh akceptačních testu přímo na nejaktuálnější – páté verzi aplikace.

Tento způsob spouštění a plánování automatizovaných fází je zásadní pro každý proces nasazení softwaru dle vzoru deployment pipeline.

2.3.4.6 Selže-li některá z fází procesu, odstranění problému je prioritou týmu

Jak již bylo řečeno, aby vývojový tým dosáhl opravdu spolehlivého, rychlého a opakovatelného procesu nasazení aplikace, musí každá verze aplikace úspěšně projít veškerými automatizovanými testy. Pokud některá z fází procesu nasazení aplikace selže, je odstranění problému, který toto selhání způsobil, hlavní prioritou pro celý vývojový tým.

2.4 Continuous Deployment

Přístup *Continuous Delivery* spočívá v zajištění toho, že vyvíjený software je připraven k nasazení do produkčního prostředí kdykoliv po celou dobu procesu jeho vývoje. Jakákoliv verze aplikace pak může být pouhým stiskem tlačítka doručena koncovým uživatelům během několika málo minut.

Posledním z přístupů zabývajících se automatizovaným nasazením aplikací je technika označovaná anglickým termínem *Continuous Deployment*. Tato technika z velké části vychází z předchozích dvou, avšak odlišuje se tím, že každá úprava aplikace, která úspěšně projde všemi automatizovanými testy, je okamžitě nasazena přímo do produkčního prostředí.

Zde je již opravdu nezbytné, aby měl vývojový tým naprostou důvěru v to, že aplikace funguje přesně tak, jak má. To lze zajistit pouze jejím kompletním pokrytím kvalitními automatizovanými testy. Automatizované testy je zde nutné psát v prvé řadě a to včetně akceptačních testů, které zajistí to, že nová verze aplikace (obsahující například novou funkci) projde akceptačními testy pouze v případě, že je tato funkčnost kompletně dokončena.

Continuous Deployment není pro každého, ne vždy je zcela žádoucí ihned vydat novou verzi aplikace. V ostatních případech ale přináší nespornou výhodou v extrémně rychlé odezvě koncových uživatelů a daleko pružnější reakční době softwarové firmy, než jak je tomu v případě tradičních technik vývoje softwaru.

3 Řešení pro automatizované nasazení aplikací

Cílem této kapitoly je analýza hlavních problémů současného stavu manuálního procesu nasazení internetových aplikací v konkrétní vývojářské firmě včetně identifikace veškerých ručně prováděných činností. Následně se kapitola zabývá možnostmi jejich automatizace a na základě takto získaných informací je postaveno zadání obsahující funkční požadavky na výsledné řešení automatizující nasazení firemních aplikací.

3.1 Současný stav

Ve firmě, pro kterou je toto řešení automatizovaného nasazení webových aplikací připravováno, je doposud nasazení aplikací prováděno čistě manuálním způsobem. Není zde využíván žádný verzovací systém, část úprav zdrojových kódů je prováděna na lokálních stanicích vývojářů či přímo na ostré verzi aplikace.

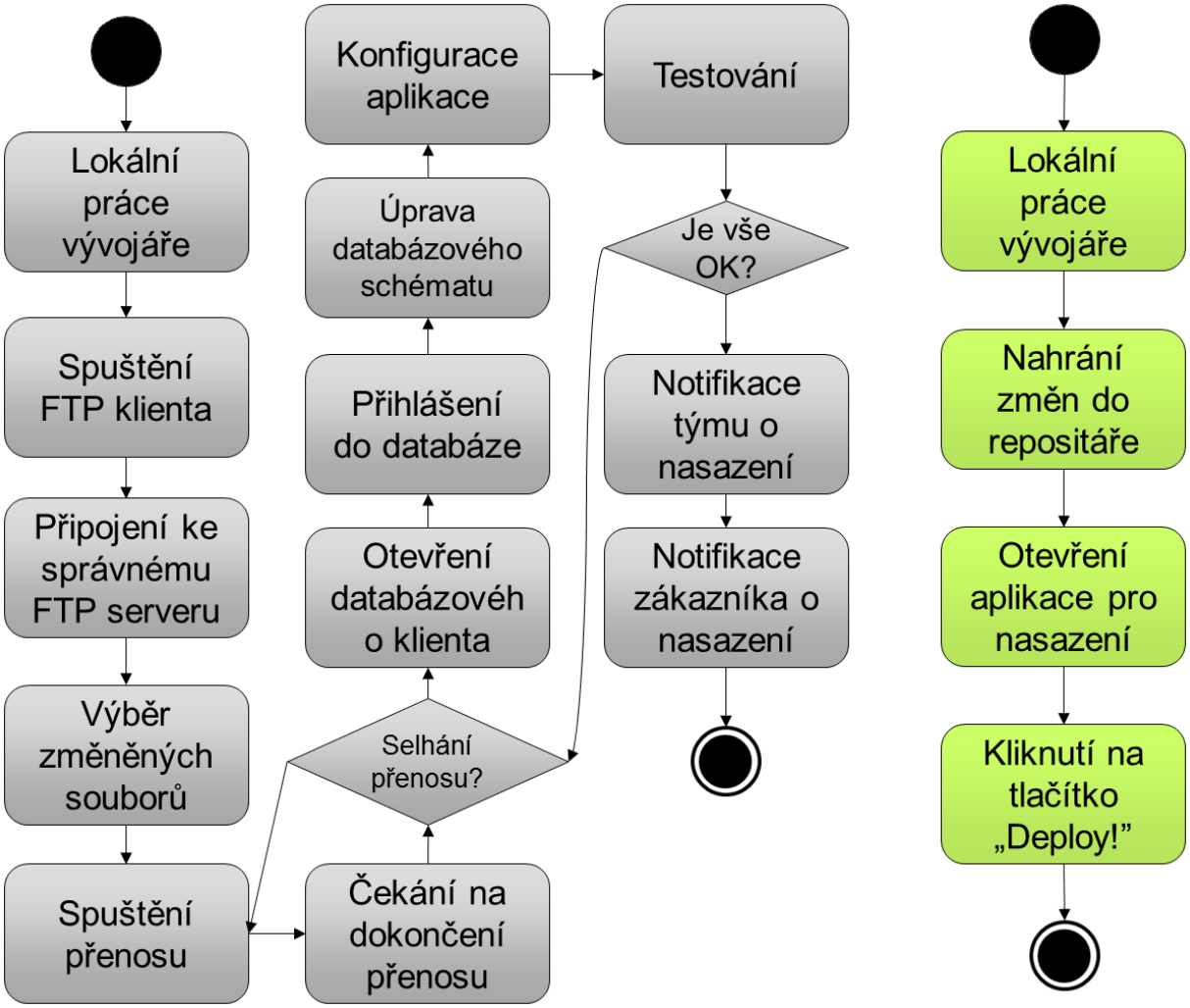
Pakliže se jedná o rozsáhlejší úpravu aplikace, je obvykle prováděna na lokální stanici vývojáře. Takový způsob práce je na rozdíl od úpravy zdrojového kódů ostré verze aplikace nepochybně správným přístupem. Nese s sebou ovšem jednu nepříjemnou povinnost, kvůli které se často vývojář uchýlí právě k úpravám na ostré verzi, a tou je nutnost nasazení změn na ostrou verzi aplikace. To je dáno především tím, že nasazení webové aplikace znamená provedení celé řady rutinních činností. Ne vždy se ale jedná o činnosti triviální a je zde tedy značný prostor k chybám. Celý proces nasazení lze shrnout do následujících čtyř částí a spolu s budoucí podobou procesu je znázorněn na obrázku 3.1:

1. **Přenos upravených zdrojových souborů** - je obvykle prováděn pomocí FTP (*File Transfer Protocol*) klienta. U rozsáhlejších úprav obvykle není možné vybrat a přenést pouze ty soubory, které byly skutečně upraveny, a je proto kopírována kompletní aplikace. V takovém případě je přenášeno velké množství souborů o malé velikosti (knihovny frameworku, soubory WYSIWYG editoru atp.), díky čemuž přenos souborů typicky trvá až v řádu desítek minut. Existuje zde také nemalé riziko vzniku nekonzistence souborů v případě, že přenos některých souborů selže nebo se připojí jiný uživatel při právě probíhajícím přenosu.

2. **Úprava databázového schématu** - obvykle znamená provedení většího množství nevratných operací, které při troše nepozornosti mohou snadno vést ke ztrátě dat.

3. **Úprava konfigurace aplikace** - zahrnuje úpravu databázového připojení, nastavení cest ke knihovnám frameworku, smazání cache atd. Opomenutí jakékoliv z těchto operací vede k nefunkčnosti aplikace.

4. **Notifikace členů týmu či klienta o provedených změnách** - znamená sepsání e-mailové zprávy o provedených změnách a jejich nasazení do produkčního prostředí.



Obrázek 3.1: Porovnání současného a budoucího procesu nasazení.

Dalším problémem při manuálním nasazování webových aplikací je atomicita výše uvedených operací. Vzhledem k tomu, že provedení všech činností spojených s nasazením nové verze aplikace vývojáři trvá typicky desítky minut, webová aplikace se po celou tuto dobu stává pro koncové uživatele nedostupnou a je proto na místě těmto uživatelům zobrazit hlášení o probíhající údržbě. V opačném případě riskujeme její neočekávané chování, které může vést například k poškození dat či jiným problémům.

V případě, že nastane situace, kdy je potřeba, aby na konkrétním projektu pracovalo více vývojářů, je při současné situaci řešena tak, že členové týmu provádějí úpravy na sdíleném úložišti. Takovéto využití sdíleného úložiště, v tomto případě FTP serveru, není o nic lepším způsobem práce více členů týmu, než je tomu u posílání změněných souborů e-mailem. Jedinou výhodou je zde to, že všichni, kdo se na projektu podílejí, pracují se stejnou verzí souborů. Sdílené úložiště ovšem neřeší problém paralelního upravování stejných částí aplikace, tedy ty případy, kdy mezitím, co jeden z vývojářů upravuje soubor na sdíleném úložišti, jiný z vývojářů na server nakopíruje svoji upravenou verzi. Aby bylo možné takovým problémům předejít, je navíc nezbytně nutná domluva na tom, kdo do kterých zdrojových souborů přistupuje.

V neposlední řadě spatřuji značný nedostatek v chybějícím verzování jakýchkoliv úprav vyvíjených aplikací, díky které nelze provést jakoukoliv formu roll-backu v případě pozdějšího neočekávaného výskytu chyby. Tento fakt je obvykle kompenzován rozsáhlejším a také časově náročným manuálním testováním funkčnosti aplikace. Nástroje, které uvedené problémy pomáhají řešit, jsou verzovací systémy, kterým se budu věnovat v následujících kapitolách této práce.

3.2 Proces nasazení

Dříve než se budu podrobněji věnovat jednotlivým částem procesu nasazení, je potřeba jasně definovat, ve kterých běhových prostředích bude aplikace během svého životního cyklu provozována a jaké jsou jejich základní charakteristiky.

3.2.1 Běhová prostředí webové aplikace

U webových aplikací je typické, že jsou nasazovány do různých běhových prostředí, z nichž každé je nastaveno a optimalizováno pro jiné účely. Peter Murray ve svém článku [13] popisuje čtyři nejčastěji využívaná běhová prostředí - vývojové, testovací/integrační, staging a produkční.

3.2.1.1 Vývojové prostředí

Vývojové prostředí je určeno a konfigurováno pro samotný vývoj aplikace. Obvykle jej představují lokální stanice jednotlivých vývojářů, díky čemuž pak rozsáhlejší zásahy do zdrojového kódu aplikace neovlivňují ostatní členy týmu. I přesto, že je to obvykle věcí samotného vývojáře, i vývojové prostředí by mělo svou konfigurací co nejvíce odpovídat produkčnímu prostředí. Výjimkou je samozřejmě to nastavení, které vývoj aplikace přímo usnadňuje, jako například způsob výpisu chyb, logování, omezení cachování atp.

3.2.1.2 Testovací prostředí

Testovací prostředí, pro které se často používá také označení integrační, je prostředí, kam vývojáři nahrávají své úpravy zdrojového kódu aplikace. Zároveň by zde již neměla

být vyvíjena žádná nová funkcionalita, neboť cílem tohoto prostředí je integrace úprav jednotlivých vývojářů a kompletní automatizované testování aplikace jako celku před tím, než je nasazena do staging prostředí. Je zde také navíc kladen důraz na to, aby konfigurace tohoto prostředí co možná nejvíce odpovídala produkčnímu prostředí. Takový přístup pomáhá maximálně redukovat riziko, že se aplikace i při jejím úspěšném otestování nebude v produkčním prostředí chovat tak, jak má.

3.2.1.3 Staging prostředí

Toto prostředí by již mělo být z hlediska jeho konfigurace prakticky identické s produkčním prostředím. Jeho účelem je především otestování procesu nasazení aplikace (tedy snížení rizika výskytu chyby při ostrém nasazení do produkčního prostředí) a také její zpřístupnění klientům před nasazením do produkčního prostředí. Klient pak v tomto prostředí může provést tzv. akceptační testování, pomocí kterého je ověřeno, zda aplikace nabízí takovou funkcionalitu, která byla definována v zadání. V rámci tohoto prostředí již vývojáři nemají přístup ke zdrojovým kódům aplikace, tedy nejsou zde prováděny žádné úpravy jejího zdrojového kódu, neboť by je již nebylo možné řádně otestovat.

3.2.1.4 Produkční prostředí

Produkční prostředí je to, kde aplikace obvykle setrvává nejdéle a které je určeno pro její zpřístupnění koncovým uživatelům. I přesto, že všechna výše uvedená prostředí lze bez problémů provozovat na jediném fyzickém serveru, produkční prostředí by mělo být vždy provozováno odděleně od ostatních. Při jeho konfiguraci je kladen důraz především na vysokou úroveň zabezpečení a optimalizaci pro vysoký výkon. V tomto prostředí vývojáři opět nemají přístup ke zdrojovým kódům aplikace a veškeré úpravy jsou prováděny výhradně nasazením nové verze aplikace ze staging prostředí.

Připravovaný systém pro podporu nasazení by měl umožňovat snadný přenos mezi jednotlivými prostředími tak, aby členové vývojového týmu necítili potřebu provádět úpravy zdrojového kódu mimo vývojové prostředí a verzovací systém. Nezbytná je pak také správná konfigurace běhových prostředí s ohledem na postupně zvyšující se důraz (od vývojového prostředí k produkčnímu) na to, aby konfigurace napodobovala konfiguraci produkčního prostředí.

Prostředí, do kterého bude umožněn přístup klienta za účelem provedení akceptačních testů, by mělo navíc obsahovat možnost pravidelné aktualizace dat z produkční databáze, která klientům pomůže získat lepší představu o podobě aplikace za použití reálných dat.

Vzhledem k potřebám menší firmy, pro kterou je toto řešení připravováno a kde se na vývoji aplikace zřídkakdy podílí větší počet vývojářů, se sloučením testovacího a staging prostředí omezím pouze na tři různá běhová prostředí - vývojové (lokální stanice vývojářů), testovací (plnící navíc funkci staging prostředí s umožněním přístupu k aplikaci klientům) a produkční prostředí.

3.2.2 Možnosti přenosu zdrojových souborů aplikace

Přenos zdrojových souborů aplikace mezi jednotlivými prostředími je jedna ze stěžejních částí celého procesu nasazení. V následujících několika odstavcích se zaměřím na nejpoužívanější nástroje z této oblasti, z nichž některé byly pro tyto účely přímo navrženy, jiné naopak vyžadují dodatečné úpravy pro využití v rámci procesu nasazení.

3.2.2.1 Nástroj rsync

Rsync je unixový nástroj, který oproti klasickému kopírování souborů (například pomocí FTP) provádí jejich synchronizaci. Takový způsob přenosu souborů má celou řadu výhod. Z hlediska nasazování aplikací je zcela jistě nejvýznamnější výhodou minimalizace velikosti přenášených dat, díky které lze podstatně zkrátit dobu nasazování aplikace. K tomu nástroj rsync využívá speciální algoritmus (tzv. „*delta encoding*“), pomocí kterého lze při aktualizaci dvou různých verzí souborů přenášet pouze rozdílové informace. Skutečný obsah dvou kopií souboru je následně porovnán na základě kontrolního součtu.

Mezi další výhody tohoto nástroje patří například možnost zabezpečeného přenosu, kopírování symbolických odkazů, zachování oprávnění přenášených souborů či komprese přenášených dat.

Nasazování webových aplikací je však typické tím, že pouhá aktualizace zdrojových souborů obvykle nestačí a je potřeba provést celou řadu dalších operací, jako například aktualizaci databázového schématu, spuštění testů, smazání cache atd. Zde lze při snaze o plnou automatizaci tohoto procesu za použití nástroje rsync narazit na problém s tím, že na dokončení synchronizace souborů nelze navázat další operace. Tento nástroj se tedy příliš nehodí pro použití v rámci komplexnějších deployment skriptů.

Další nevýhody nástroje rsync spatřuji v absenci jakékoliv možnosti práce s verzemi a hlavně nemožnosti snadněji provést návrat do stavu před zahájením synchronizace souborů. Jediný způsob, kterým pak lze řešit případný roll-back nasazené aplikace, spočívá v návratu do původního stavu v rámci zdroje přenosu souborů a v opětovném spuštění synchronizace s cílovým umístěním.

3.2.2.2 PEAR repozitář

PEAR[1] (*PHP Extension and Application Repository*) je distribuční systém knihoven a rozšíření pro aplikace psané v jazyce PHP. Jedná se tedy o volně dostupný repozitář kódu, který obsahuje širokou škálu knihoven sdružených do tzv. balíčků. Ty obvykle pomáhají vývojářům řešit problémy, se kterými se při vývoji PHP aplikací střetávají nejčastěji.

Po nainstalování tohoto nástroje lze velice snadno stahovat a instalovat dostupná řešení, stejně tak jako aktualizovat již používané balíčky. Mezi knihovnami lze také snadno definovat závislosti, na jejichž základě jsou pak při instalaci konkrétního balíčku automaticky doinstalovány všechny související knihovny.

[1]PEAR – *http://pear.php.net/*

Prvním problémem, na který lze narazit při použití systému PEAR jako nástroje pro nasazování aplikací je fakt, že PEAR jako takový je moderován. To znamená, že před zařazením nové knihovny do tohoto repozitáře je důsledně ověřeno, zda balíček splňuje veškerá pravidla a konvence definované přímo v dokumentaci PEAR repozitáře. Přichází tedy v úvahu pouze varianta, kdy si firma musí na vlastním serveru provozovat svůj privátní PEAR repozitář a řešit s tím spojená úskalí.

Dle Shahara Evrona [6] není však použití repozitáře PEAR pro potřeby automatizovaného nasazování aplikací příliš vhodné, neboť s sebou nese nutnost dodatečné implementace velké části funkcionality pro potřeby tohoto procesu. Naopak vzhledem k tomu, že byl tento nástroj původně navržen pro jazyk PHP, spatřuje jeho výhodu ve snadnější integraci do prostředí PHP aplikací a v jeho multiplatformnosti.

3.2.2.3 Verzovací systém

Použití verzovacích systémů v rámci vývoje aplikace je již v dnešní době standardem. Využití takového nástroje také pro přenos souborů má výhodu především v tom, že vývojářský tým již obvykle takový nástroj zná a denně používá při úpravách zdrojových kódů aplikace (verzování, větvení, tagování atd.). Použití verzovacího systému v rámci nasazování aplikací tedy skvěle zapadá do procesu jejich vývoje.

Verzovací systém je do procesu nasazení aplikací obvykle začleněn pomocí tzv. *„hooků“*, kterými lze na příkazy verzovacího systému navázat některé další operace spojené s nasazením aplikace do konkrétního prostředí.

Nevýhody verzovacích systémů při jejich použití v procesu nasazení Shahar Evron spatřuje [6] především v tom, že tyto systémy nebyly původně pro tento typ použití navrženy. Navíc při využití verzovacího systému pro nasazení aplikace do produkčního prostředí s sebou nese bezpečnostní riziko spojené s nutností umožnit přístup z produkčního prostředí do projektového repozitáře.

3.2.2.4 Balíčkovací systém

Balíčkovací systémy poskytují sadu nástrojů pro správu instalovaného softwaru. Tyto systémy jsou známé především z oblasti unixových operačních systémů, kde lze jejich pomocí snadno konkrétní software stáhnout, nainstalovat a později také odinstalovat.

Mezi nejznámější formáty těchto balíčků patří formát RPM, na kterém jsou postaveny balíčkovací systémy distribucí operačního systému Linux, jako jsou Mandriva, SUSE, RedHat či Fedora. Dalším populárním formátem je pak také formát DEP, který je prozměnu využíván balíčkovacím systémem APT v distribuci Debian GNU.

Hlavní předností balíčkovacích systému je to, že každý balíček kromě daného softwaru obsahuje také sadu metainformací, které kromě identifikace balíčku (autor, veze, popis atp.) obsahují také informace o případných závislostech na jiných balíčcích. V případě využití balíčkovacího systému v rámci procesu nasazení pak lze velice snadno v rámci baličku uvést také informace o závislostech dané aplikace na konfiguraci běhového prostředí (např. na určitých PHP rozšířeních). Při instalaci balíčku pak systém tyto závislosti sám zkontroluje

a případně se automaticky postará o jejich doinstalování či aktualizaci na požadovanou verzi.

Hlavní nevýhodou těchto systémů je závislost na konkrétním operačním systému (někdy na konkrétní distribuci), který daný balíčkovací systém využívá, a také poměrně obtížná integrace s ostatními nástroji, které jsou součástí procesu nasazení.

3.2.2.5 Specializovaný nástroj pro automatizaci nasazení aplikací

Nástrojů pro podporu automatizovaného nasazení aplikací existuje celá řada. Jejich společným cílem je umožnit vývojářům automatizované spouštění jednotlivých úkolů v rámci procesu nasazení aplikace. I přesto, že se obvykle jedná o nástroje vytvořené pro nasazení aplikace v konkrétním jazyce, obvykle je lze s minimálním úsilím použít i v prostředí jiného jazyka.

Jedním z nejpoužívanějších open-source nástrojů z této oblasti je *Capistrano*[2], který byl původně vyvinut pro nasazování aplikací v jazyce Ruby. I přesto je ale poměrně často používán také pro nasazování PHP aplikací. Nevýhodou tohoto nástroje je fakt, že pro jeho provoz je zapotřebí mít kromě PHP nainstalované také běhové prostředí pro jazyk Ruby. Krom toho je třeba počítat s tím, že při potřebě rozsáhlejších úprav procesu nasazení se nelze vyhnout úpravě skriptů v jazyce Ruby, což může být problém především v prostředí firmy, která své aplikace vyvíjí výhradně v jiných jazycích.

Zajímavé pak může být široké spektrum rozšíření tohoto nástroje, kterými jsou například *Capistrano Ash* vyvinuté pro nasazení aplikací Wordpress, Drupal a aplikací využívajících framework Zend nebo rozšíření *Capifony* specializované pro nasazení aplikací vyvíjených za pomoci frameworku Symfony.

Dalším často využívaným nástrojem je nástroj *Apache Ant*[3], který opět, ačkoliv byl vyvinut v jazyce Java a je tedy určen především pro prostředí tohoto jazyka, lze použít také pro nasazení aplikací v jazyce PHP. Jeho princip je takřka shodný s unixovým nástrojem *GNU Make*[4]. Stejně jako *Capistrano* dokáže na základě skriptu v jazyce XML automatizovat spouštění veškerých činností spojených s procesem nasazení. Pro prostředí PHP je však vhodnější jeho varianta přímo určená pro jazyk PHP - *Phing*[5].

Hlavní výhodou nástroje *Phing* je to, že byl vyvinut v jazyce PHP, díky čemuž lze veškeré úpravy procesu nasazení provádět výhradně v tomto jazyce. Proces nasazení aplikace zde sestává z jednotlivých úkolů, které jsou definované pomocí PHP tříd. Vykonávání těchto úkolů je pak orchestrováno podobně jako u nástroje *Apache Ant* pomocí XML souboru.

Protože firma, pro kterou je toto řešení připravováno, vyvíjí především aplikace založené na frameworku Symfony, posledním nástrojem z této oblasti, o kterém bych se rád zmínil, je nástroj *Pake*[6], na kterém se vedle komunity vývojářů kolem frameworku Symfony podílel

[2]Capistrano – *https://github.com/capistrano/capistrano/wiki/*
[3]Apache Ant – *http://ant.apache.org/*
[4]GNU Make – *http://www.gnu.org/software/make/*
[5]Phing – *http://www.phing.info/*
[6]Pake – *https://github.com/indeyets/pake/wiki*

sám jeho tvůrce Fabien Potencier. Zajímavostí tohoto nástroje je to, že pro orchestraci procesu nasazení nástroj nevyužívá XML soubory, které jsou mnohdy hůře čitelné než samotný PHP kód, a lze tedy při řízení jednotlivých úkolů využít možností jazyka PHP, jako je například větvení, cykly atd. Nevýhodu tohoto nástroje pak spatřuji ve skromnější dokumentaci v porovnání s výše zmíněnými nástroji.

3.2.3 Aktualizace databázového schématu

Aktualizace databázového schématu (též databázová migrace) je činnost, které se při implementaci nových funkcí aplikace nelze vyhnout, a je třeba dobře zvážit, jakým způsobem provést úpravu databáze při změně verze aplikace na jednotlivých běhových prostředích.

Aktualizace databázového schématu při nasazení nové verze aplikace je netriviálním úkolem, který bohužel nelze řešit pouhým zkopírováním souborů, jako je tomu u aktualizace zdrojového kódu. Aktualizace databázového schématu je vždy prováděna spuštěním sady příkazů v jazyce SQL. Způsoby, kterými lze tuto činnost provádět, shrnul Ivo Jansch v jednom ze svých článků [9] a na jejich základě definoval pět úrovní zralosti úpravy databáze při aktualizaci aplikace.

1. Aktualizace databázového schématu je prováděna ručně.
2. Během vývoje se změny databázového schématu ukládají do souboru v podobě SQL příkazů. Ten se při aktualizaci ručně spustí nad databází.
3. Při aktualizaci databázového schématu se zároveň vytváří seznam tzv. *„undo“* SQL příkazů, které umožní návrat k původní verzi databázového schématu před jeho aktualizací.
4. Aktualizace databázového schématu je prováděna automaticky na základě připravených SQL příkazů vytvořených dle druhé úrovně.
5. Seznam SQL příkazů pro úpravu databáze je generován automaticky při vývoji nové verze aplikace. Při jejím nasazení je aktualizace provedena automaticky.

Ve firmě, pro kterou je toto řešení připravováno, je aktualizace databázového schématu prováděna výhradně ručně a dle modelu tedy odpovídá první úrovni. Budoucí řešení by pak mělo tuto činnost maximálně automatizovat a odpovídat tak alespoň čtvrté úrovni modelu zralosti.

Jedním z nástrojů často používaným pro tyto účely je nástroj *DbDeploy*[7]. Tento nástroj byl původně vyvinut pro aplikace v jazyce Java, nicméně lze jej využít i v prostředí jazyka PHP. Myšlenka tohoto nástroje je poměrně jednoduchá. Využívá tzv. delta souborů, které obsahují seznam SQL příkazů pro aktualizaci databázového schématu při přechodu z jedné verze aplikace na druhou. Nepovinnou součástí tohoto souboru jsou také výše zmíněné *„undo“* SQL příkazy pro případ potřeby návratu k předchozí verzi databázového schématu.

[7] DbDeploy – *http://dbdeploy.com/*

Tyto soubory jsou ve svém názvu opatřeny číslem verze a uchovávány v projektovém repozitáři spolu s aplikací.

Zároveň s tím je součástí databáze tabulka, která obsahuje informaci o aktuální verzi databázového schématu. Pomocí této tabulky pak lze snadno vybrat sadu delta souborů, které je potřeba aplikovat při přechodu z jedné verze na druhou. *DbDeploy* obsahuje také nástroj pro příkazovou řádku, díky čemuž jej lze snadno spouštět v rámci procesu nasazení aplikace. Nevýhodou tohoto nástroje je pak poměrně chudší dokumentace a fakt, že se v současné době jedná již o mrtvý projekt.

Dalším silným nástrojem z této oblasti je nástroj *Doctrine Migrations*[8], který je distribuován jako rozšíření vrstvy pro databázovou abstrakci ORM frameworku *Doctrine* a nabízí podobně jako *DbDeploy* funkcionalitu pro automatizovanou aktualizaci databázového schématu.

Velkou výhodou tohoto nástroje je možnost automatizovaného generování výše zmíněných rozdílových delta souborů, kterou provádí na základě porovnání objektově relačního mapování, které je součástí aplikace, se skutečnou podobou databázové struktury. Další výhodou je mimo jiné možnost použití jazyka PHP pro popis změn databázové struktury místo čistých SQL příkazů, čímž je zajištěna nezávislost na konkrétním databázovém stroji.

ORM framework *Doctrine* je navíc distribuován jako součást frameworku Symfony2, který vývojáři firmy, pro kterou je toto řešení připravováno, dobře znají a používají. Využití rozšíření *Doctrine Migrations* pro aktualizace databázového schématu v procesu nasazení se tedy v tomto případě přímo nabízí.

3.2.4 Konfigurace aplikace

Vzhledem k předpokladu, že aplikace bude nasazována do různých běhových prostředí, z nichž každé vyžaduje různou konfiguraci (jako jsou například přihlašovací údaje do databáze, způsob výpisu chyb nebo cáchování), je důležité, aby byly tyto informace odděleny od samotného kódu aplikace. Typicky se pro tyto účely využívá speciálních konfiguračních souborů, které obsahují nastavení svázaná s konkrétním běhovým prostředím.

Příkladem vhodného řešení muže být způsob, kterým tuto problematiku řeší framework Symfony[9]. Aplikaci ve frameworku Symfony lze již po instalaci spustit ve třech různých prostředích (lze nadefinovat libovolný počet), přičemž v každém z nich je aplikace spuštěna s různou konfigurací. Pro každé definované prostředí je pak načten konkrétní konfigurační soubor:

1. pro vývojové prostředí `app/config/config_dev.yml`
2. pro testovací prostředí `app/config/config_test.yml`
3. pro produkční prostředí pak soubor `app/config/config_prod.yml`

[8]Doctrine Migrations – *http://www.doctrine-project.org/projects/migrations*
[9]Symfony – *http://symfony.cz/*

Je zřejmé, že konfigurace aplikace se pro různá prostředí liší jen z části. Z toho důvodu aplikace obsahuje také společný konfigurační soubor `app/config/config.yml`, který obsahuje výchozí konfiguraci aplikace a který je následně importován do konfiguračních souborů, které jsou specifické pro různá běhová prostředí aplikace. Tyto soubory pak pouze obsahují (nebo přepisují) to nastavení, které se pro dané prostředí liší od výchozího nastavení.

Spuštění aplikace v konkrétním prostředí vyvoláme volbou konkrétního PHP skriptu (tzv. „*front-controlleru*“) pro dané prostředí. Výjimkou je pak testovací prostředí, ve kterém dochází ke spouštění automatizovaných testů a nelze k němu přistoupit z prohlížeče. Pro zbylá dvě prostředí to jsou:

1. pro vývojové prostředí `app_dev.php`
2. pro produkční prostředí pak soubor `app.php`

3.2.4.1 Uložení citlivých konfiguračních informací

Dalším problémem, se kterým je třeba se při nasazování aplikace vypořádat, je způsob předání citlivých informací aplikaci. Typicky se jedná například o přístupové údaje k produkční databázi. Nebylo by totiž vhodné, kdyby byly tyto informace součástí konfigurace aplikace (nebo dokonce součástí jejího zdrojového kódu) a společně s ní uloženy v projektovém repozitáři.

Příkladem správného řešení tohoto problému může být opět framework Symfony, který pro tyto účely využívá speciální konfigurační soubor `parameters.ini`, ve kterém jsou uloženy pouze citlivé informace. Tyto hodnoty jsou pak při načtení konfigurace vloženy do konfiguračních souborů aplikace, kde se místo nich doposud vyskytovaly pouze unikátní názvy proměnných označené speciálními symboly ve tvaru například `%database_password%`.

Díky tomuto způsobu vyčlenění citlivých informací do samostatného konfiguračního souboru pak stačí pouze zajistit existenci konkrétní verze tohoto souboru v každém z běhových prostředí, do kterého je aplikace nasazována.

3.2.5 Notifikace členů týmu o nasazení aplikace

Informování členů vývojového týmu o nasazení nové verze aplikace může být, stejně jako ostatní části procesu, snadno automatizováno. Obvykle tuto činnost provádí sám nástroj pro automatizované nasazení a to tak, že každý člen je informován o výsledcích procesu nasazení aplikace formou e-mailového reportu. Tím lze snadno zaručit, že v případě, že se během procesu vyskytnou nějaké problémy, vývojový tým se o nich dozví tak brzy, jak je to možné.

V případě, že proces nasazení proběhl úspěšně, lze tuto funkcionalitu obohatit také o možnost informování klienta, ať už o kompletním nasazení aplikace či jen doručení nové funkcionality.

3.3 Požadavky na řešení

Cílem této kapitoly je jasné vymezení a specifikace jednotlivých požadavků kladených na řešení pro podporu automatizovaného nasazení internetových aplikací. Požadavky jsou pro větší přehlednost rozděleny na obecné požadavky týkající se celého řešení, funkční a kvalitativní požadavky na část řešení odpovědné za nasazování aplikací a požadavky na aplikaci, která bude sloužit především jako uživatelské rozhraní systému.

3.3.1 Obecné požadavky

3.3.1.1 Analýza a využití dostupných nástrojů

Součástí náplně práce je mimo jiné analýza dostupných nástrojů, na jejímž základě budou vybrány ty nástroje, které umožní maximální využití již hotových komponent a minimalizují tak nutnost vlastního vývoje. Správný výběr nástrojů dále pomůže maximálně zkrátit dobu implementace, zajistí vyšší kvalitu řešení a snadnou údržbu systému.

3.3.1.2 Modularita

Hlavním rysem obou částí připravovaného řešení bude jejich modularita, která zajistí oddělení jednotlivých součástí systému a zajistí tak jeho vysokou flexibilitu. Kvalitní návrh umožní zapouzdřit jednotlivé funkční celky do modulů, které následně mohou nebo nemusí být použity při konkrétní instalaci systému. Modulární uspořádání funkčních celků také mimo jiné zvýší přehlednost zdrojového kódu a možnost systém dále snadno a efektivně rozšiřovat.

3.3.1.3 Oddělení uživatelského rozhraní

Striktní oddělení platí také pro aplikaci poskytující uživatelské rozhraní, kterou tak bude možné provozovat zcela odděleně a nezávisle na systému odpovědném za automatizované nasazení aplikací, se kterým bude aplikace komunikovat vzdáleně pomocí zabezpečeného komunikačního protokolu. Oddělení těchto částí zajistí nezávislost na konkrétní firemní architektuře serverů a lze pak v případě potřeby na obě části řešení aplikovat různou úroveň zabezpečení.

3.3.2 Systém pro automatizované nasazení aplikací

Tato část aplikace představuje jádro celého řešení a jejím hlavním cílem je automatizace veškerých činností v rámci procesu nasazení internetových aplikací. Kromě procesu nasazení bude systém navíc automatizovat některé z dalších rutinních činností, které jsou ve firmě v průběhu vývoje aplikací často prováděny.

3.3.2.1 Běhová prostředí

Systému bude umožňovat nadefinování libovolného počtu běhových prostředí, do kterých lze aplikace nasazovat. V rámci procesu nasazení bude systém umožňovat načtení takové konfigurace, která je specifická pro konkrétní běhové prostředí a která bude definována ve formě konfiguračního souboru.

3.3.2.2 Aktualizace zdrojových kódů

Systém bude schopný komunikovat s projektovým repozitářem a bude umožňovat aktualizaci zdrojových kódů na libovolnou verzi aplikace v kterémkoliv z nadefinovaných prostředí. Při aktualizaci zdrojových kódů aplikace nesmí dojít k poškození či ztrátě uživatelských dat, kterými jsou data uložená v session, soubory nahrané uživateli, databáze aj.

Aktualizace zdrojového kódu aplikace musí být provedena vždy prostřednictvím zabezpečeného komunikačního protokolu. Veškeré soubory aplikace budou navíc přenášeny v komprimované podobě tak, aby byl zajištěn minimální objem přenášených dat.

3.3.2.3 Aktualizace databázového schématu

Součástí systému bude také možnost automatické aktualizace databázového schématu. Způsob aktualizace musí odpovídat minimálně čtvrtému stupni zralostního modelu uvedeném v kapitole 3.2.3 a bude součástí procesu nasazení aplikace i návratu aplikace do stavu před nasazením nové verze. Jakýkoliv zásah do databázové struktury aplikace musí být navíc doprovázen kompletní zálohou databáze.

3.3.2.4 Možnost návratu do stavu před nasazením

Součástí systému bude také mechanismus umožňující bezproblémový návrat aplikace do stavu, ve kterém byla před provedením nasazení nové verze. Spolu s aplikací musí být do původního stavu uvedeno také databázové schéma.

Dále bude systém obsahovat formu historie, kdy bude možné definovat určitý počet historických verzí, které budou pro každé běhové prostředí uchovány. Systém bude schopný provést efektivní přechod mezi libovolnými dvěma historickými verzemi aplikace.

3.3.2.5 Logování

Systém bude obstarávat důkladné logování veškeré činnosti související s procesem nasazení aplikací. O výsledku procesu nasazení bude systém informovat členy projektového týmu prostřednictvím e-mailového reportu, ke kterému bude přiložena také HTML verze tohoto logu.

3.3.2.6 Nasazení projektového repozitáře

Systém bude umožnovat nasazení aplikace ve formě projektového repozitáře, které bude prováděno výhradně na lokálních stanicích členů vývojového týmu. Hlavní požadavkem na tuto funkcionalitu je nezávislost na konkrétní platformě, neboť na lokálních stanicích vývojářů nelze zaručit jednotné běhové prostředí.

3.3.2.7 Nedostupnost aplikace

Proces nasazení internetových aplikací bude orientován na minimalizaci času, po který je nasazovaná aplikace nedostupná. Maximální doba nasazení projektu se musí pohybovat maximálně v řádu několika minut s tím, že po tuto dobu bude koncovým uživatelům zobrazena stránka informující o probíhající údržbě systému.

3.3.2.8 Specifika různých projektů

V rámci procesu nasazení bude systém umožňovat spouštění procedur specifických pro nasazení konkrétního projektu. Typickým příkladem jsou procedury svázané s typem či verzí frameworku, na kterém je daná aplikace postavena a které se mohou pro jednotlivé projekty lišit. Systém pak musí umožnit nadefinování sady těchto procedur pro každý projekt, které budou následně začleněny do vhodných částí procesu nasazení.

3.3.2.9 Založení nového projektu

Systém bude také umožnovat automatizaci činností spojených se založením nového projektu, které obvykle zahrnují následující:

- Vytvoření databáze v každém běhovém prostředí, do kterého bude aplikace nasazována.
- Pro každou vytvořenou databázi vytvořit unikátního databázového uživatele s právy na tuto konkrétní databázi.
- Vytvoření nového projektového repozitáře a jeho konfigurace (např. nastavení práv).
- V případě, že nový projekt vychází z některého z firemních referenčních projektů, je referenční projekt naklonován do nově vytvořeného projektového repozitáře.

3.3.3 Uživatelské rozhraní

Uživatelské rozhraní systému odpovědného za automatizované nasazování aplikací bude realizováno ve formě webové aplikace. Hlavním úkolem této aplikace bude nabídnout uživatelům prostředí pro správu firemních projektů a snadnou obsluhu všech funkcí systému. Mimo to bude aplikace obsahovat také informace o všech běhových prostředích firmy spolu s konfigurací specifickou pro jednotlivé projekty.

3.3.3.1 Nástěnka

Po úspěšném přihlášení se uživateli zobrazí titulní strana ve formě nástěnky. Tato stránka bude sloužit jako vstupní bod celé aplikace a kromě funkce rozcestníku bude uživateli v přehledné formě zobrazovat nejdůležitější informace tykající se správy firemních projektů.

3.3.3.2 Správa projektů

Aplikace bude obsahovat prostředí pro správu firemních projektů a jejich konfigurace. V této části aplikace bude ke každému z projektů možné získat informace o

- doméně projektu,
- názvu repozitáře,
- revizi, která je nasazena na jednotlivých prostředích,
- akcích, které byly na projektu spuštěny.

V rámci správy projektů bude dále možné obsluhovat většinu funkcí systému odpovědného za automatizované nasazení aplikace.

3.3.3.3 Logování

Aplikace bude zajišťovat logování veškerých akcí spojených s činností systémů, kdy ke každému projektu budou logovány následující informace:

- Uživatel, který akci vykonal.
- Běhové prostředí, na kterém byla akce spuštěna.
- Typ akce (např. vytvoření projektu, nasazení, roll-back, aj.).
- V případě nasazení a roll-backu identifikátor revize projektu.
- Odkaz na kompletní report, který si lze na vyžádání nechat zaslat e-mailem.
- Datum a čas akce.

3.3.3.4 Správa uživatelů

Součástí aplikace bude možnost správy uživatelů a uživatelských skupin. Uživatelé budou moci do aplikace přistupovat pouze prostřednictvím svého uživatelského účtu, který bude náležet do právě jedné uživatelské skupiny. Každé uživatelské skupině pak bude možné přiřadit určitou podmnožinu práv, na jejichž základě bude řízen přístup ke konkrétním funkcím systému.

4 Analýza dostupných nástrojů

Cílem této kapitoly je analýza dostupných nástrojů, které by bylo následně možné využít jako základní komponenty celého řešení. Mezi hlavní komponenty řešení patří verzovací systém, nástroj pro automatizaci jednotlivých činností, které je potřeba vykonat v rámci procesu nasazení, a také nástroj pro bezproblémovou aktualizaci databázového schématu. Správný výběr nástrojů mimo jiné pomůže minimalizovat nutnost vlastního vývoje a zároveň zvýší kvalitu výsledného řešení.

4.1 Verzovací systémy

Využití verzovacího systému je již v dnešní době nedílnou součástí životního cyklu větších softwarových projektů a rovněž hraje významnou roli v procesu automatizovaného nasazení aplikací. Verzovací systémy slouží jako úložiště zdrojových kódů a všech dalších důležitých souborů týkajících se daného projektu, u nichž zaznamenává jejich historii formou revizí. Verzovací systém značně ulehčuje týmovou spolupráci, neboť lze paralelně pracovat na více verzích aplikace, jednoduše zjistit, kdo ve zdrojových kódech aplikace provedl jakou změnu a v případě, že se tato změna v budoucnu ukáže jako chybná, lze se snadno vrátit k verzi původní.

- **Zachování historie vývoje** - Nejdůležitější vlastností verzovacích systémů je uchování všech verzí souborů repozitáře. Umožňuje jednoduchý návrat k přechozím verzím zdrojových souborů v případě, že provedené změny neodpovídají zamýšleným představám. Součástí každé verze je také sada informací, díky kterým lze snadno vysledovat, kdo danou změnu provedl a čeho se tato změna týkala.
- **Podpora práce v týmu** - Verzovací systém se stává nepostradatelným nástrojem ve chvíli, kdy na stejné části aplikace současně pracuje více členů vývojového týmu. V takových případech odpadají problémy způsobené vzájemným přepisování zdrojových souborů umístěných na sdíleném úložišti.
- **Vývoj více verzí aplikace** - Verzovací systém umožňuje spravovat více verzí stejné aplikace. V takovém případě je vývoj aplikace rozdělen do více větví, a zatímco v jedné z nich vývojový tým pracuje na funkcích pro novou verzi aplikace, v jiné větvi lze například provádět a následně nasazovat opravy pro předchozí verzi aplikace, která je již nasazena v produkčním prostředí.

4.1.1 Centralizované verzovací systémy

Verzovací systémy lze rozdělit do dvou základních kategorií. Tou první jsou verzovací systémy pracující na základě architektury klient-server a označují se jako centralizované. Centralizované verzovací systémy se vyznačují tím, že projektový repozitář obsahující kompletní historii změn je uložen pouze na straně serveru, zatímco na straně klientů se nachází pouze pracovní kopie souborů projektu.

Hlavní nevýhodou centralizovaných verzovacích systémů je potřeba interakce se serverem kdykoliv, kdy potřebujeme například procházet historii projektu, vytvořit novou větev či jen nahrát změny do projektového repozitáře. Tato vlastnost pak může mít nepříjemný dopad na rychlost práce s repozitářem například v situacích, kdy vývojář nemá k dispozici rychlé a spolehlivé připojení nebo dokonce dojde k výpadku serveru, na kterém je verzovací systém spuštěn. Vzhledem ke shromažďování veškerých dat o historii projektu na jednom místě je také nutné věnovat zvýšenou pozornost způsobu zálohování serveru.

Výhodou centralizovaných verzovacích systémů je pak numerické číslování jednotlivých verzí projektu a také možnost uzamknutí určité části repozitáře, což v případě distribuovaných verzovacích systémů samozřejmě nelze.

4.1.1.1 Výhody centralizovaných verzovacích systémů:

- jednodušší architektura systému
- numerické číslování verzí
- možnost uzamknutí části repozitáře

4.1.1.2 Nevýhody centralizovaných verzovacích systémů:

- potřeba časté interakce se serverem
- potřeba připojení k serveru při práci s repozitářem
- lokálně pouze pracovní kopie souborů repozitáře
- potřeba spolehlivého zálohování dat repozitáře

4.1.2 Distribuované verzovací systémy

Opakem jsou pak distribuované verzovací systémy založené na architektuře peer-to-peer, u které jsou si všechny uzly rovnocenné. To znamená, že každý s vývojářů má na své lokální stanici plnohodnotný projektový repozitář s jeho kompletní historií změn, ke kterým se může libovolně vracet lokálně, tedy bez nutnosti komunikace se serverem. Některý z takových repozitářů obvykle slouží pro synchronizaci a sdílení práce mezi jednotlivými vývojáři a z toho důvodu bývá označen jako „centrální“. Technicky se ale takový repozitář nijak neliší od ostatních. Jedná se pouze o jednu z mnoha metodik, jak distribuovaný

verzovací systém používat. Stále platí fakt, že každý z projektových repozitářů může vůči jinému tzv. klonu vystupovat v roli serveru, stejně jako v roli klienta.

Hlavní výhodou distribuovaných verzovacích systému je možnost klonování projektového repozitáře. Tato vlastnost umožňuje pracovat s repozitářem lokálně, díky čemuž je provedení jednotlivých příkazů značně rychlejší, a také v případech, kdy vývojář momentálně nemá připojení k internetu. Další výhoda distribuovaných verzovacích systémů spočívá v bezpracném zálohování, neboť zde neexistuje pouze jedno jediné místo shromažďující veškerá data projektu. Vzhledem k tomu, že každý s vývojářů má na svém lokálním disku plnohodnotnou verzi projektového repozitáře, nemůže dojít k tomu, že by tým přišel o veškerou práci v případě, že by na straně serveru došlo k poruše vedoucí ke kompletní ztrátě dat. Stejně tak ani v případě výpadku serveru hlavního repozitáře nedojde k narušení práce vývojářů. U distribuovaných systému navíc existují daleko širší možnosti ve volbě metodiky práce s projektovým repozitářem. Zatímco centralizované systémy nabízejí vždy jen jeden scénář práce, u distribuovaných systémů může projektový tým použít takovou metodiku práce, která nejvíce odpovídá jeho dosavadním zvyklostem. Tyto zásady se pak obvykle promítají do způsobu, jakým dochází k větvení a spojování projektu a dynamickým změnám rolí (klient-server) jednotlivých projektových repozitářů, typicky při spolupráci několika vývojových týmů.

Nevýhodou distribuovaných verzovacích systémů je pak složitější hierarchie rolí, v kterých repozitáře vystupují a způsob synchronizace změn. Ta je ale dána konkrétní metodikou práce vývojového týmu s projektovým repozitářem, neboť i v případě distribuovaného verzovacího systému lze používat přístup, který odpovídá architektuře klient-server používané u centralizovaných verzovacích systémů. Za nevýhody lze také považovat nelineální způsob verzování formou hashování revizí či nemožnost uzamčení části projektového repozitáře, které se uplatňuje především u binárních souborů, u kterých nelze realizovat spojení.

4.1.2.1 Výhody distribuovaných verzovacích systémů:

- možnost klonování projektového repozitáře
- off-line práce s projektovým repozitářem
- bezpracné zálohování
- široké možnosti volby metodiky práce s projektovým repozitářem

4.1.2.2 Nevýhody distribuovaných verzovacích systémů:

- složitější hierarchie rolí projektových repozitářů
- nelineární číslování verzí formou hashování revizí
- nemožnost uzamčení částí repozitáře

4.1.3 Volba verzovacího systému

Centralizované verzovací systémy již za sebou mají delší historii a i přesto, že jsou stále hojně používané, jsou v současné době spíše na ústupu. To je dáno především atraktivními možnostmi distribuovaného řešení, kterými jsou například možnost jeho použití off-line nebo jejich snadnější zálohování. Na základě výše uvedeného srovnání obou architektur jsem se rozhodl pro výběr verzovacího systému z rodiny distribuovaných verzovacích systémů. Mezi nepopulárnější distribuované verzovací systémy současnosti patří Git [1] a Mercurial [2].

4.1.3.1 Mercurial

Mercurial je open-source verzovací systém, který je dílem Matta Mackalla a společnosti Selenic. Jedná se o multiplatformní verzovací systém naprogramovaný v jazyce Python, který byl navržen pro rozsáhlejší projekty a vydán pod licencí GNU GPLv2.

Při vývoji tohoto verzovacího systému byl dbán velký důraz na jeho výkonost, díky čemuž se dnes řadí mezi nejrychlejší verzovací systémy. Oblibu mnoha vývojářů získal také díky kvalitně zpracované dokumentaci a užšímu sortimentu příkazů, díky čemuž je poměrně snadné naučit se s ním pracovat.

Mercurial je navíc dodáván s příjemným webovým rozhraním, pokročilejší uživatele také jistě potěší široká škála rozšíření, pomocí kterých lze integrovat nové funkce přímo do jádra verzovacího systému.

4.1.3.2 Git

Git je v posledních letech rychle se rozšiřujícím open-source verzovacím systémem, který vyvinul sám autor Linuxu – Linus Torvalds – pro potřeby vývoje Linuxového jádra. Git je naprogramován v jazyce C, Bashi a Perlu a je, stejně jako Mercurial, licencovaný pod GNU GPLv2.

Specialitou verzovacího systému Git je jeho vlastní stejnojmenný protokol určený pro přenos dat, pomocí kterého lze komunikovat zabezpečeně pomocí SSH (*Secure Shell*). I přesto je možné použít také běžné komunikační protokoly, kterými jsou HTTP či FTP. V porovnání s ostatními systémy Git nabízí širší škálu příkazů, nicméně základní práce s projektovým repozitářem není o moc složitější, než je tomu u jiných systémů.

Velkou výhodou verzovacího systému Git je také jeho hlavní poskytovatel hostingu, server GitHub [3], který se velkou měrou podílel na jeho popularizaci tím, že uživatelům Gitu nabídl efektivní grafické rozhraní pro správu příspěvků k jejich repozitářům, díky čemuž kolem Gitu vznikla velmi rozsáhlá komunita vývojářů.

[1] Git – *http://git-scm.com*
[2] Mercurial – *http://mercurial.selenic.com*
[3] GitHub - *https://github.com/*

4.1.3.3 Porovnání systémů Git a Mercurial

Na stránkách Google Code [15], které patří mezi významné zdroje informací o vývojářských nástrojích a produktech, byla uveřejněna podrobná analýza verzovacích systémů Mercurial a Git. Průzkum nebyl orientován na výčet veškerých vlastností obou systémů, ve kterých si jsou oba systémy velmi podobné, ale zaměřil se pouze na ty oblasti, v nichž se tyto verzovací systémy liší nejvýznamněji. Jako výhody verzovacího systému Mercurial uvádí:

- **Učící křivka** - Významnou výhodou verzovacího systému Mercurial je jeho učící křivka. Jeho základní verze neobsahuje takové množství příkazů a funkcí, jako je tomu u systému Git. Tato vlastnost spolu s kvalitní a rozsáhlou dokumentací umožňuje novým uživatelům snadnější osvojení si způsobu práce se systémem. Přispět může také značná podobnost jeho příkazů rozšířeným verzovacím systémům Subversion a CVS.
- **Podpora systému Windows** - Na rozdíl od systému Git, který má nejblíže k systému Linux, je Mercirual možné bezproblémově provozovat na všech běžných platformách včetně Windows. Přestože Git na systému Windows provozovat lze, citelně zde zaostává například v podpoře a optimalizaci některých jeho funkcí.
- **Údržba** - Vzhledem ke způsobu ukládání repozitářů Mercurial v porovnání s Gitem nezabírá tolik místa na disku. Git vyžaduje pravidelnou údržbu repozitářů, nicméně nabízí sofistikovanější systém pro správu místa na disku klienta (viz správa klientského úložiště Gitu).
- **Stálost historie** - Zatímco Git nabízí velké množství možností také v oblasti změn v historii repozitáře, Mercurial ji zachovává v nezměněné podobě. I přesto, že může být zásah do dat historie repozitáře nebezpečný, v některých případech je taková možnost naopak výhodou. Navíc v případě vlastního Git serveru lze pomocí konfigurace ztrátu historických dat zakázat.

Výhody verzovacího systému Git jsou shrnuty takto:

- **Správa klientského úložiště** - Git i Mercurial umožňují selektivně nahrávat větve z jiných repozitářů, čímž uživateli umožňují snížení velikosti lokálně uložených dat. Git navíc umožňuje odstranění nahraných větví a tzv. prořezávání starších dat historie větví lokálního repozitáře, zatímco v případě Mercurialu je potřeba mít uložena data o kompletní historii všech přítomných větví.
- **Počet rodičů** - Git není při slučování omezen počtem rodičů. U Mercurialu je v případě slučování většího počtu rodičů nutné provádět slučování po dvou.
- **Rebasování** - příkaz „rebase“ umožňuje přesunutí větve do některé z novějších revizí a začlenění provedených změn. V době uvedení této analýzy tuto funkcionalitu nabízel pouze Git, nicméně nyní ji již doplnil i Mercurial.

Myslím, že výběrem jakéhokoliv z těchto dvou verzovacích systémů se nedopustím závažnějšího omylu. V obou případech se jedná o kvalitní řešení a funkcionalita, kterou tyto systémy nabízí, se liší jen nepatrně. Nicméně vzhledem ke značné popularitě především v oblasti vývoje webových aplikací, které se v současné době těší verzovací systém Git, padla moje volba právě na něj.

Verzovací systémy jako Mercurial nebo Git mohou výrazně usnadnit celý proces nasazení webové aplikace především proto, že je v nich snadné určit změny, které mají být nasazeny, a změny, na kterých se stále pracuje. Konkrétní metodika práce s projektovým repozitářem pak zaleží na způsobu, kterým je konkrétní aplikace vyvíjena a zvyklostech projektového týmu.

4.2 Automatizace procesu nasazení

Další důležitou komponentou celého systému je orchestrační nástroj, jehož cílem je automatizované spouštění jednotlivých činností spadajících do procesu nasazení aplikace. Stručně jsem se již o těchto nástrojích zmínil v kapitole 3.2.2.5.

4.2.1 Kritéria výběru

Hlavním kritériem výběru nástroje pro automatizaci procesu nasazení je především to, zda daný nástroj poskytuje takovou funkcionalitu, která maximálně usnadní vývoj a budoucí údržbu systému. Firma, pro kterou je toto řešení připravováno, se zabývá vývojem aplikací čistě v jazyce PHP a vzhledem k požadavku na možnost efektivního rozšiřování systému a jeho bezproblémovou údržbu se zaměřím právě na ty nástroje, které jsou v jazyce PHP přímo vyvinuty a lze je v něm dále snadno rozšiřovat. S tímto požadavkem také úzce souvisí kritérium týkající se kompletnosti a kvality dokumentace jednotlivých nástrojů.

Dalším důležitým kritériem pro vybraný nástroj je také multiplatformnost řešení, díky kterému bude možné využít systém pro nasazení aplikací také na lokálních stanicích vývojářů nezávisle na provozovaném operačním systému.

Z nástrojů uvedených v kapitole 3.2.2.5 splňují výše uvedená kritéria pouze dva z nich - Phing a Pake.

4.2.1.1 Phing

Phing je, stejně jako tradiční nástroje pro sestavování aplikací v prostředí Linuxu (např. GNU Make[4]), nástroj umožňující automatizované spuštění veškerých činností, které je potřeba vykonat v rámci procesu nasazení aplikace. Ačkoliv vychází z jiného javovského nástroje Apache Ant[5], Phing je vyvinut a určen pro aplikace vyvíjené v prostředí jazyka PHP.

[4]GNU Make – *http://www.gnu.org/software/make/*

[5]Apache Ant – *http://ant.apache.org/*

K orchestraci jednotlivých činností Phing využívá XML soubor (tzv. *„buildfile“*). Tento soubor je vždy strukturován ve formě určitého počtu spustitelných cílů (*targets*), které dále obsahují volání elementárních úkolů (*tasks*), jako je například kopírování souborů, nastavení práv, spuštění databázového dotazu atp. Vykonání sady úkolů definovaných v orchestračním souboru `mybuildfile.xml` v cíli `mytarget` pak lze snadno spustit z příkazové řádky pomocí následujícího příkazu:

```
$ phing -f mybuildfile.xml mytarget
```

Důležitou vlastností Phingu je jeho nezávislost na konkrétní platformě. Jediná jeho závislost pak spočívá ve funkčním běhovém prostředí jazyka PHP a instalaci některých jeho dalších rozšíření, které jsou nutné v případě spouštění úkolů týkajících se například analýzy kódu či automatizovaného testování.

Ukázka 4.1: Příklad orchestračního XML souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="BuildApplication" default="build">
    <property name="build_dir" value="./build" />

    <target name="prepare">
        <echo msg="Making directory ${build_dir}" />
        <mkdir dir="${build_dir}" />
    </target>

    <target name="build" depends="prepare">
        <echo msg="Copying files to ${build_dir} directory..." />

        <echo msg="Copying ./about.php to ${build_dir} directory..." />
        <copy file="./about.php" tofile="${build_dir}/about.php" />

        <echo msg="Copying ./contacts.php to ${build_dir} directory..." />
        <copy file="./contacts.php" tofile="${build_dir}/contacts.php" />

        <echo msg="Creating archive..." />
        <tar destfile="${build_dir}/build.tar.gz" compression="gzip">
            <fileset dir="${build_dir}">
                <include name="*" />
            </fileset>
        </tar>

        <echo msg="Files copied and compressed in build directory OK!" />
    </target>
</project>
```

Jednoduchý příklad orchestračního XML skriptu (částečně převzatého z dokumentace Phingu [2]) demostruje ukázka 4.1. Kromě složení skriptu z jednotlivých spustitelných cílů (elementů `target`) příklad dále znázorňuje možnost definování závislostí mezi jednotlivými cíli pomocí atributu `depends`, způsob deklarace proměnných (element `property`) a jejich

následné použití pomocí výrazu `${název_proměnné}`. Úkolem tohoto skriptu by pak bylo sestavení aplikace ze souborů `about.php` a `contacts.php` a její uložení v komprimované podobě ve složce `build`.

Příklad dále ukazuje způsob volání elementárních úkolů (elementy `mkdir`, `copy`, `tar` atd.), z nichž každému přísluší stejnojmenná PHP třída odpovědná za jeho vykonání. Phing také umožňuje definování vlastních úkolů, které lze snadno implementovat vytvořením nové PHP třídy, která bude potomkem třídy `Task`. Výhodou takto vytvořených úkolů je pak především možnost jejich znovupoužití v různých orchestračních skriptech.

Předností nástroje Phing je také velice kvalitní dokumentace a velké množství již předpřipravených úkolů, které jsou již součástí instalace. Mezi ně patří například sada úkolů pro komunikaci s verzovacími systémy Subversion či Git, nástroji pro databázové migrace DbDeploy[6] a LiquiBase[7], úkoly pro spouštění automatizovaných testů aj.

4.2.1.2 Pake

Pake je (stejně jako Phing) nástroj, který umožňuje spuštění sady předdefinovaných úkolů z příkazové řádky. Tento nástroj vznikl v komunitě vývojářů kolem frameworku Symfony a na jeho vývoji se podílel také sám jeho tvůrce Fabien Potencier. Pake byl podobně jako Phing vyvinut v jazyce PHP, ve kterém ho také lze rozšiřovat, a ve značné míře vychází z podobného nástroje Rake[8] určeného pro aplikace vyvíjené v prostředí jazyka Ruby.

Příklad jednoduchého skriptu nástroje Pake, který byl převzat z oficiální dokumentace[9], je znázorněn na ukázce 4.2. Z ukázky je na první pohled patrné, že na rozdíl od Phingu je zde orchestrace celého procesu, stejně jako jednotlivé elementární úkoly (*tasks*), popsána v jazyce PHP. Výhoda využití jazyka PHP i na úrovni orchestrace spočívá v možnosti využití řídících funkcí jazyka (např. větvení, cyklů atp.) a také pro programátory mnohdy příjemnější a čitelnější formě zápisu než v případě jazyka XML. Vytvoření konkrétní úlohy pak v zásadě probíhá vždy ve třech krocích:

1. **Popis úkolu** - pomocí funkce `pake_desc()` je popsána funkce daného úkolu.

2. **Název úkolu a definice jeho závislostí** - definování unikátního názvu spustitelného úkolu je prováděno pomocí funkce `pake_task()`. Jejím prostřednictvím lze také volitelně nadefinovat závislosti daného úkolu na jiných, podobně jako tomu bylo v případě nástroje Phing. Způsob, kterým lze závislosti nadefinovat, je demonstrován na ukázce v případě úkolu „bar“, jehož vykonání je závislé na úkolu „foo“.

3. **Implementace úkolu** - implementace nového úkolu je definována ve formě standardní PHP funkce, jejíž jméno musí být ve tvaru `run_nazev_funkce()`.

[6] DbDeploy – *http://dbdeploy.com/*
[7] LiquiBase – *http://www.liquibase.org/*
[8] Rake – *http://rake.rubyforge.org/*
[9] Pake – *https://github.com/indeyets/pake/wiki*

Ukázka 4.2: Příklad orchestračního souboru nástroje Pake.

```
<?php
pake_desc("Description of foo task");
pake_task("foo");
function run_foo()
{
    pake_echo('I am the "FOO" task');
}

pake_desc("Description of bar task");
pake_task("bar", "foo");
function run_bar()
{
    pake_echo('I am the "BAR" task');
}
?>
```

V porovnání s nástrojem Phing však Pake zdaleka neobsahuje takové množství předpřipravených úkolů a velkou nevýhodou může být také podstatně skromnější dokumentace, která doposud není kompletně dokončena.

4.2.2 Porovnání nástrojů Phing a Pake

V obou případech se jedná o velice kvalitní nástroje, které nabízejí veškerou funkcionalitu potřebnou k plné automatizaci procesu nasazení. Z hlediska rozdílného způsobu zápisu orchestračního souboru Pake nabízí pro programátory přirozenější a flexibilnější prostředí jazyka PHP, avšak využití XML souborů v případě nástroje Phing naopak zajišťuje důsledné oddělení implementace elementárních úkolů od jejich orchestrace.

Především vzhledem k rozdílné úrovni kvality dokumentace, která je jedním z hlavních kritérií uvedených v kapitole 4.2.1, a podstatně větší komunitě vývojářů kolem nástroje Phing bude právě tento nástroj využit pro automatizaci procesu nasazení.

4.3 Aktualizace databázového schématu

Poslední neméně důležitou komponentou plně automatizovaného procesu nasazení je nástroj, umožňující bezproblémovou aktualizaci databázového schématu. O nástrojích, které pomáhají tuto problematiku řešit, jsem se již zmínil v kapitole 3.2.3.

Ve firmě, pro kterou je toto řešení připravováno, je naprostá většina aplikací postavena na dvou prakticky různých frameworcích - Symfony[10] verze 1 a Symfony2[11]. Zatímco Symfony verze 1 problematiku databázových migrací implicitně neřeší, v případě Symfony2 je situace o něco snazší, neboť jednou se základních komponent frameworku je také vrstva

[10]Symfony – *http://www.symfony-project.org/*
[11]Symfony2 – *http://symfony.com/*

pro databázovou abstrakci Doctrine2, pro kterou existuje velice kvalitní rozšíření - Doctrine Migrations[12].

4.3.1 Kritéria výběru

Základním požadavkem na tuto část procesu nasazení je dosažení alespoň čtvrté úrovně zralostního modelu uvedeném v kapitole 3.2.3. Vybraný nástroj by měl tedy umožňovat automatizovanou aktualizaci databázového schématu na základě připravených SQL příkazů spolu s možností návratu k původní verzi databázového schématu před jeho aktualizací.

4.3.2 DoctrineMigrations

Databázové migrace tak, jak je řeší rozšíření Doctrine Migrations, odpovídají dokonce páté úrovni zralostního modulu uvedeném v kapitole 3.2.3. Jedinou povinností vývojáře je při každé změně modelové vrstvy aplikace vygenerovat migrační delta soubor v podobě PHP třídy, která se vždy skládá ze dvou funkcí - `up()` a `down()`. Obě tyto funkce pak obsahují sadu příkazů v jazyce SQL, přičemž funkce `up()` je volána vždy při migraci databáze na novější verzi, zatímco funkce `down()` je vykonána při návratu změn databázového schématu do původního stavu. Po instalaci tohoto rozšíření prostřednictvím Git repozitáře lze migrační PHP třídu snadno vygenerovat zavoláním následujícího příkazu z adresáře projektu:

```
$ php app/console doctrine:migrations:diff
```

Doctrine2 automaticky porovná modelovou vrstvu aplikace se skutečnou podobou databázového schématu a na základě tohoto porovnání vygeneruje novou migrační PHP třídu s metodami `up()` a `down()`, které obsahují odpovídající sady SQL příkazů. Každá takto vygenerovaná třída je označena datem vytvoření, díky čemuž nástroj dokáže přesně určit pořadí spuštění jednotlivých delta souborů.

Aktualizaci databázového schématu na nejnovější verzi lze snadno spustit následujícím příkazem, kterému lze v případě potřeby předat jako další parametr označení požadované verze:

```
$ php app/console doctrine:migrations:migrate
```

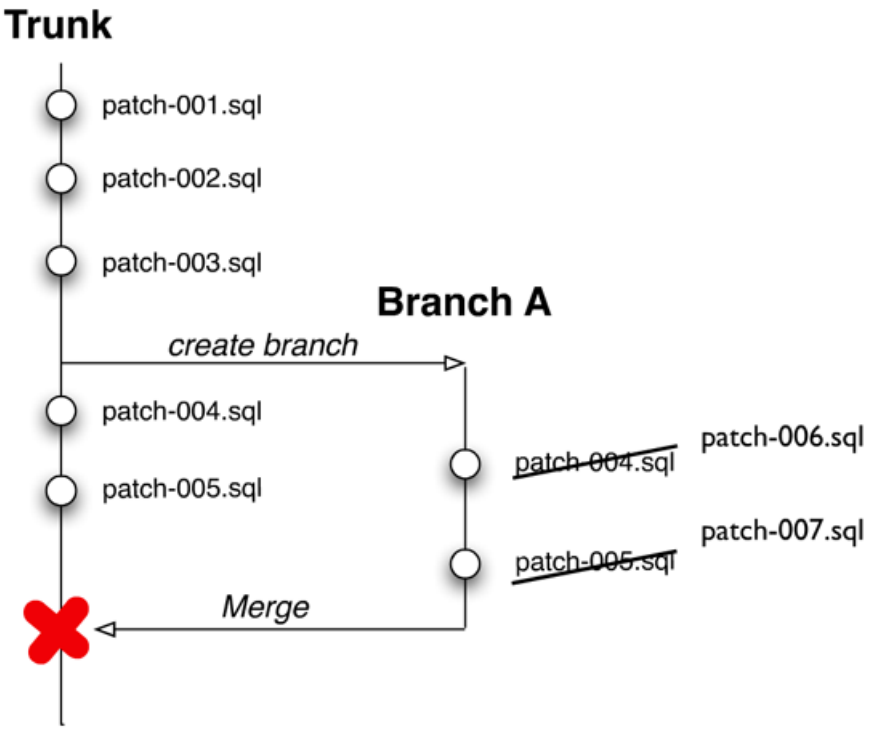
Po provedení prvotní migrace je vždy k databázi projektu navíc přidána tabulka, která obsahuje informace o všech aplikovaných migračních delta souborech aktuální verze databázového schématu. Při přechodu na jinou verzi pak lze pomocí této tabulky přesně určit sadu delta souborů, které je potřeba aplikovat.

[12]DoctrineMigrations – *http://www.doctrine-project.org/projects/migrations.html*

4.3.3 DbDeploy

Pro potřeby databázových migrací aplikací postavených na frameworku Symfony verze 1 připadá v úvahu samostatný nástroj DbDeploy[13], který jsem již představil v kapitole 3.2.3. V této kapitole byly uvedeny také některé jeho nedostatky týkající se především poměrně skromné dokumentace a faktu, že projekt je již několik let neaktivní.

Po dalším zkoumání tohoto nástroje jsem však narazil na několik dalších problémů týkajících se sekvenčního číslování jednotlivých verzí migračních delta souborů. Tento způsob číslování verzí způsobuje problémy především při vývoji aplikací za pomoci projektového repozitáře. Pokud totiž vývoj probíhá ve dvou různých větvích, v nichž v každé dojde k úpravám databázového schématu aplikace, pak je v případě spojení těchto dvou větví navíc nutné řešit přečíslování vytvořených delta souborů tak, aby jejich vykonání proběhlo vždy ve stejném pořadí. Tato situace je znázorněna na obrázku 4.1.



Obrázek 4.1: Problém spojení větví obsahující delta soubory se sekvenčním číslováním. [19]

Použití nástroje DbDeploy by tedy v každém případě zahrnovalo buď dodatečnou úpravu nástroje tak, aby pro číslování delta souborů využíval časové značky, nebo implementaci řešení pro přečíslování delta souborů při spojování větví projektového repozitáře. Vzhledem k těmto problémům jsem tedy od využití tohoto nástroje upustil.

[13] DbDeploy – *http://dbdeploy.com/*

4.3.4 sfPropelMigrationsPlugin

Framework Symfony verze 1 využívá Propel[14] jako vrstvu zajišťující objektově relační mapování. Ten sice na rozdíl od Doctrine2 neobsahuje žádné oficiální rozšíření řešící databázové migrace, nicméně součástí oficiálního repozitáře pluginů pro framework Symfony je kvalitní rozšíření sfPropelMigrationsLightPlugin.

Toto rozšíření však trpí stejnými problémy způsobenými sekvenčním číslováním migračních delta souborů jako v případě nástroje DbDeploy, avšak komunita vývojářů kolem frameworku Symfony dala vzniknout rozšíření tohoto pluginu - sfPropelMigrationsPlugin[15], který již podobně jako nástroj Doctrine Migrations využívá časové značky.

Funkcionalita nástroje je pak ve značné míře podobná nástroji Doctrine Migration s tím, že v tomto případě způsob aktualizace databázového schématu odpovídá čtvrté úrovni zralostního modelu uvedeného v kapitole 3.2.3.

Nástroj umožňuje vygenerovat nový migrační delta soubor příkazem

```
$ php symfony init-migration <název_migrace>
```

a po nadefinování sady SQL příkazů ve funkcích `up()` a `down()` lze migraci spustit následujícím příkazem:

```
$ php symfony migrate <název_aplikace>
```

Příkaz je opět možné doplnit cílovou verzí databázového schématu. Po prvotním vykonání migrace je, podobně jako v případě nástroje Doctrine Migration, v databázi vytvořena tabulka, obsahující informace o aplikovaných delta souborech.

4.4 Integrační server

Dodržování principů a zásad metody průběžné integrace (viz kapitola 2) lze snadno podpořit využitím tzv. integračního serveru. Protože metoda průběžné integrace není jen o automatizovaném sestavení aplikace, ale je také věcí automatizovaného testování, analýzy kódu, generování dokumentace atp., integrační server pak plní roli koordinátora všech podobných operací. Kromě toho vývojovému týmu navíc umožňuje sledovat související metriky (pokrytí kódu testy, počty neúspěšných sestavení aplikace aj.) nebo jej lze například integrovat s projektovým systémem pro správu úkolů.

4.4.1 CruiseControl

Jedním z nástrojů s nejdelší historií v této oblasti je integrační server CruiseControl[16], na jehož vývoji se podílel sám Fartin Fowler, který je jedním z autorů metody průběžné

[14] Propel – *http://www.propelorm.org/*

[15] sfPropelMigrationsPlugin – *https://github.com/jcoby/sfPropelMigrationsPlugin*

[16] CruiseControl – *http://cruisecontrol.sourceforge.net/*

integrace. Nástroj byl vyvinut společností *ThoughtWorks* a kromě integračního serveru ho lze použít jako rozšiřitelný framework pro tvorbu vlastních automatizovaných skriptů.

Nástroj CruiseControl je vyvíjen jako open-source řešení v jazyce Java. Základní funkcionalitu nástroje je možné snadno rozšířit některým ze zásuvných modulů (tzv. „*plug-in*“), jejichž nabídka je však v porovnání s nástrojem Jenkins výrazně menší. Jedním z rozšíření je také PHPUnderControl[17] usnadňující prvotní konfiguraci nástrojů průběžné integrace pro aplikace vyvinuté v jazyce PHP.

Oproti ostatním nástrojům vyniká CruiseControl především v širokých možnostech konfigurace (za pomoci XML souborů) a v jeho rozšiřitelnosti pomocí vlastních zásuvných modulů. I přesto bych ale jeho nasazení nedoporučoval vzhledem k užší nabídce rozšíření a faktu, že v současné době již projekt není aktivně vyvíjen.

4.4.2 Jenkins

Mezi nejpoužívanější open-source integrační servery současnosti patří nástroj Jenkins[18], který byl původně pod názvem Hudson vyvinut společnosti *Sun Microsystems* a uvolněn pod licencí MIT. K jeho přejmenování došlo v roce 2010 po akvizici společnosti *Sun Microsystems* společností *Oracle*, která jméno Hudson následně registrovala jako ochrannou známku. Zatímco vývoj nástroje Hudson[19] byl nadále zajišťován společností *Oracle* a zájem o něj značně opadl, v dalším vývoji tohoto nástroje, avšak pod jménem Jenkins, pokračovala také široká komunita vývojářů.

O značné oblíbenosti nástroje Jenkins svědčí také průzkum[20] společnosti *Sonatype* z roku 2011, ze kterého vyplynulo, že tento nástroj (původně Hudson) využívá více jak 71 % firem provádějících průběžnou integraci. Po následném přejmenování projektu pak lze soudit například z aktivity projektu na serveru GitHub.com, na kterém se v současné době podílí 369 vývojářů a sčítá na 753 repozitářů, v čemž značně převyšuje nástroj Hudson.

Jenkins byl původně vyvinut a určen pro projekty v jazyce Java, nicméně taktéž jej lze bez větších problémů použít v prostředí jiných jazyků. Jeho funkcionalitu lze rozšířit pomocí zásuvných modulů, kterých díky široké komunitě vývojářů, kteří nástroj vyvíjejí a používají, nabízí opravdu široké spektrum.

Na obrázku 4.2 je zobrazena ukázka úvodní obrazovky nástroje Jenkins, převzaté přímo ze serveru k průběžné integraci tohoto projektu[21]. V levé části obrazovky je kromě navigačního menu zobrazen seznam úloh (fronta), které z nějakého důvodu čekají na jejich vykonání. Takovým důvodem může být například dosažení limitu pro počet současně vykonávaných úloh nebo čekání na dokončení jiných úloh na základě definovaných závislostí.

Nástroj Jenkins navíc umožňuje řídit spouštění úloh na libovolném počtu podřízených (tzv. „*slave*“) serverů. Tyto informace nalezneme opět na levé straně obrazovky, kde je

[17] phpUnderControl – *http://phpundercontrol.org/*
[18] Jenkins – *http://jenkins-ci.org/*
[19] Hudson (Oracle) – *http://hudson-ci.org/*
[20] průzkum společnosti Sonatype – *http://go.sonatype.com/content/winter2011surveyresults*
[21] CI server projektu Jenkins – *http://ci.jenkins-ci.org/view/All/*

Obrázek 4.2: Úvodní obrazovka integračního serveru Jenkins.

na konkrétním příkladu patrné, že pro tento projekt jsou k dispozici čtyři podřízené servery, z nichž dva jsou momentálně nedostupné.

Dominantní část obrazovky je tvořena seznamem veškerých úloh týkajících se projektu. O každé z úloh pak lze zjistit některé základní informace. Zatímco symbol zbarvené kuličky vyjadřuje stav úlohy (např. modrá barva pro úspěch, červená pro selhání atp.), procentuální úspěšnost dokončení dané úlohy je vyjádřena ikonkou počasí, kde slunečno znamená 100% úspěšnost dokončení úlohy, naopak bouřka označuje úlohy, které nikdy nebyly úspěšně vykonány. Kromě názvů jednotlivých úloh lze dále o každé úloze získat informaci například o době, která uplynula od posledního úspěšného či neúspěšného vykonání úlohy nebo o délce jejího trvání.

Přednost nástroje Jenkins spatřuji, vedle intenzivního vývoje a široké nabídky rozšíření, také v možnosti způsobu jeho konfigurace, kterou lze kromě standardní editace XML souborů provádět také pomocí webového uživatelského rozhraní, kde je zároveň dynamicky prováděna kontrola správnosti zadaných informací.

Ačkoliv je automatizace sestavení a nasazení firemních aplikací pro metodu průběžné integrace základním krokem, vzhledem k absenci automatizovaného testování nebo standardů pro kontrolu kvality zdrojových kódů ve firmě, pro kterou je toto řešení připravováno, je kompletní zavedení metody průběžné integrace nad rámec této práce a nasazení integračního serveru ponechám jako jednu z možností budoucího rozšíření výsledného řešení.

5 Implementace řešení

První část této kapitoly se věnuje popisu architektury řešení a způsobu komunikace mezi jeho komponentami. Dále se kapitola věnuje nejzajímavějším částem konkrétní implementace či zavedení jednotlivých komponent ve firmě tak, aby byly splněny veškeré požadavky uvedené v kapitole 3.3. Závěr kapitoly pak demonstruje reálné použití výsledného řešení při úpravě internetové aplikace.

5.1 Architektura řešení

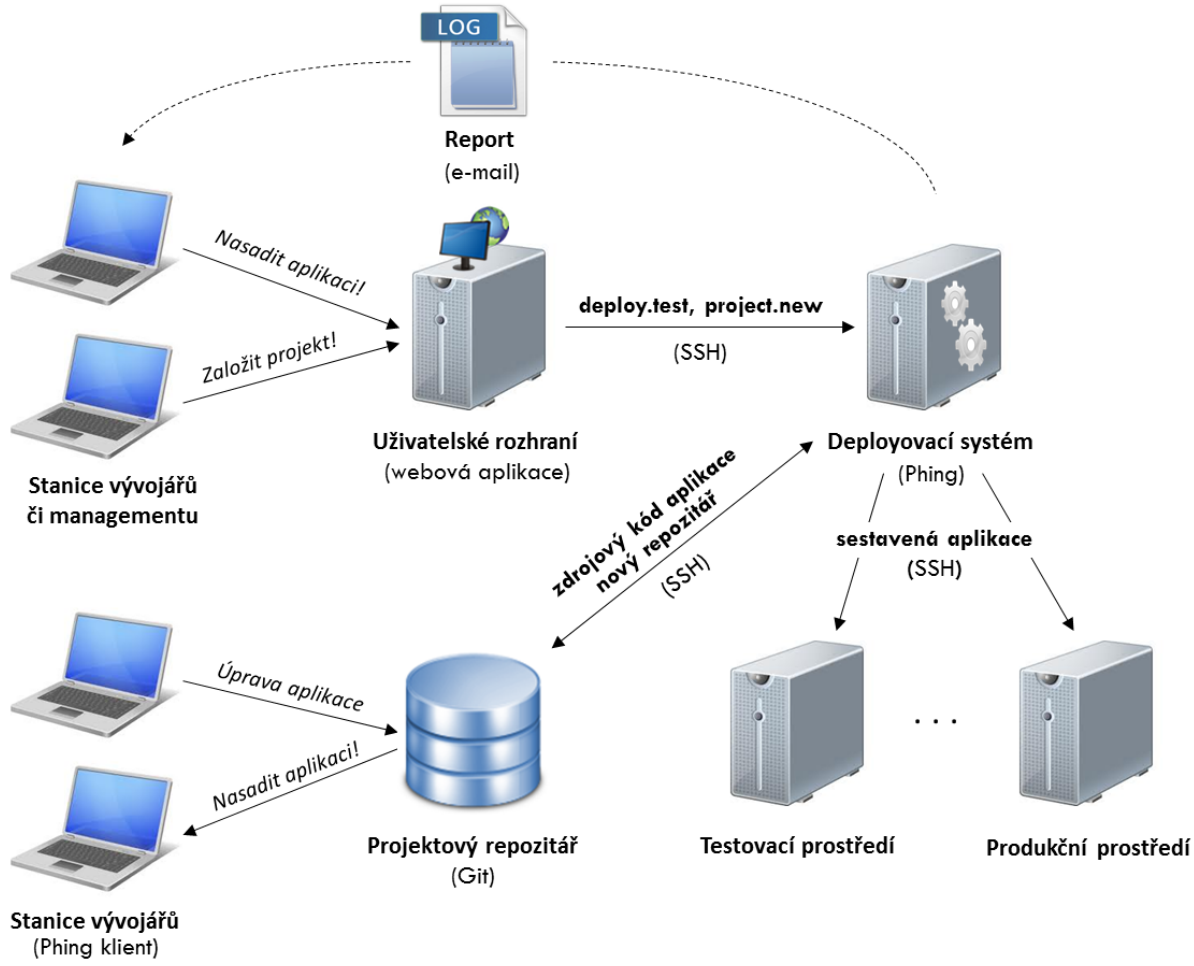
Řešení pro podporu automatizovaného nasazení internetových aplikací se skládá ze tří základních komponent - systému pro správu verzí, systému zajišťující automatizaci procesů a aplikaci, jejíž hlavním cílem je umožnit jejich pohodlné spouštění prostřednictvím webového uživatelského rozhraní. Architektura řešení je spolu se způsobem komunikace mezi jednotlivými komponentami znázorněna na obrázku 5.1.

Fyzické umístění jednotlivých komponent systému, tak jak je znázorněno na obrázku, je pouze jednou z mnoha možností. Díky jejich vhodnému oddělení a využití protokolu pro vzdálenou komunikaci je architektura celého řešení velice flexibilní. Pouhou úpravou konfigurace lze například libovolné komponenty sloučit a provozovat je na jediném fyzickém serveru.

Jednou z hlavních komponent celého řešení je systém pro správu verzí - v tomto případě nástroj Git. S ním nejčastěji komunikují členové vývojového týmu v průběhu vývoje aplikace (aktualizace kódu, stažení posledních změn atd.). Neméně důležitou komponentu však představuje i pro systém zajišťující automatizaci procesů (na obrázku deployovací systém), který v něm vytváří projektové repozitáře pro nové aplikace nebo z něj naopak získává konkrétní verze jejich zdrojového kódu.

Systém pro automatizaci procesů je navržen jako centralizované řešení, jehož hlavními výhodami je snadnější možnost správy zdrojového kódu (na jednom jediném místě) a v budoucnu efektivnější způsob rozšiřování funkcionality systému. Automatizace procesu nasazení na lokálních stanicích vývojářů je řešena klientskou verzí tohoto systému, která taktéž komunikuje s centrálním projektovým repozitářem, a jejíž pomocí lze pro účely vývoje nasadit aplikaci ve formě repozitáře.

Spouštění všech ostatních automatizovaných procesů pak probíhá prostřednictvím uživatelského rozhraní, které je realizováno formou webové aplikace postavené na frameworku Symfony2. Veškerá komunikace i přenos souborů mezi jednotlivými komponentami sys-



Obrázek 5.1: Architektura řešení pro automatizované nasazování aplikací.

tému a běhovými prostředími probíhá skrze zabezpečený komunikační protokol Secure Shell (SSH).

5.2 Systém pro správu verzí

Základním stavebním kamenem celého řešení, bez kterého proces nasazení nelze plně automatizovat, je systém pro správu verzí. V kapitole 4.1 jsem se již věnoval porovnání nejpoužívanějších systémů, z nichž byl pro účely tohoto řešení vybrán systém Git.

Zavedení systému pro správu verzí ve firmě je spojeno se zajištěním přístupu k centrálním projektovým repozitářům, které lze realizovat různými způsoby. Jednoduché techniky využívající tzv. Git démona pro „*read-only*“ či SSH pro plný přístup však nenabízejí bezpečný víceuživatelský přístup k centrálnímu repozitáři. Pro účely tohoto řešení jsou tedy vhodnější sofistikovanější nástroje, které vedle víceuživatelského přístupu umožňují také

daleko jemnější nastavení přístupových práv. Mezi nejpoužívanější nástroje z této oblasti patří nástroje Gitosis[1] a Gitolite[2].

Hlavní rozdíl mezi těmito nástroji spočívá v možnostech nastavení přístupových práv. U nástroje Gitosis lze pro daný repozitář definovat přístup více uživatelů (pro čtení nebo zápis), v případě nástroje Gitolite je pak navíc možné omezit právo zápisu až na úrovni jednotlivých větví repozitáře. Ačkoliv tato vlastnost není od tohoto řešení přímo požadována, použiji nástroj Gitolite, neboť v budoucnu nelze vyloučit rozšíření systému právě o tuto funkcionalitu.

5.2.1 Model větvení projektového repozitáře

Ve firmě, která doposud žádný verzovací systém aktivně nevyužívala, je součástí jeho zavedení také definování určité metodiky a pravidel týkajících se způsobu práce s verzovacím systémem při vývoji aplikace.

Jedna z nejpříjemnějších vlastností verzovacího systému Git je to, že díky jeho distribuované povaze a perfektní práci s větvemi může projektový tým používat takovou metodiku práce s verzovacím systémem (v anglickém jazyce označované termínem *„branching model“* či *„workflow“*), která nejlépe odráží jeho zvyklosti a způsob vývoje aplikací. Dodržování stanovených zásad pro větvení a následné spojování částí projektového repozitáře včetně určitých jmenných konvencí pomůže vývojářům udržet repozitář v takové podobě, ve které se v něm lze snadno zorientovat v případě budoucí práce s historií projektového repozitáře.

Jelikož ve firmě, pro kterou je toto řešení připravováno, bude verzovací systém teprve zaveden, zvolil jsem pro tyto účely model, který publikoval Vincent Driessen [4] a oblíbilo si ho široké spektrum vývojářů.

I přesto, že Git je distribuovaným verzovacím systémem a z technického pohledu jsou si všechny repozitáře rovnocenné, v tomto modelu je definován repozitář, který v něm hraje roli „centrálního“ repozitáře. Tento repozitář je pak označen jménem známým všem uživatelům Gitu - `origin`. [Obrázek 5.2]

Každý z vývojářů bude své úpravy nejčastěji synchronizovat právě s tímto repozitářem, nicméně v případě, že spolu bude spolupracovat více vývojových týmů, lze provádět synchronizaci také mezi jednotlivými vývojáři místo toho, aby byly nedokončené úpravy předčasně nahrávány do centrálního repozitáře. Přílohou této práce je také přehledné schéma tohoto modelu, které demonstruje způsob větvení projektového repozitáře. [Příloha A]

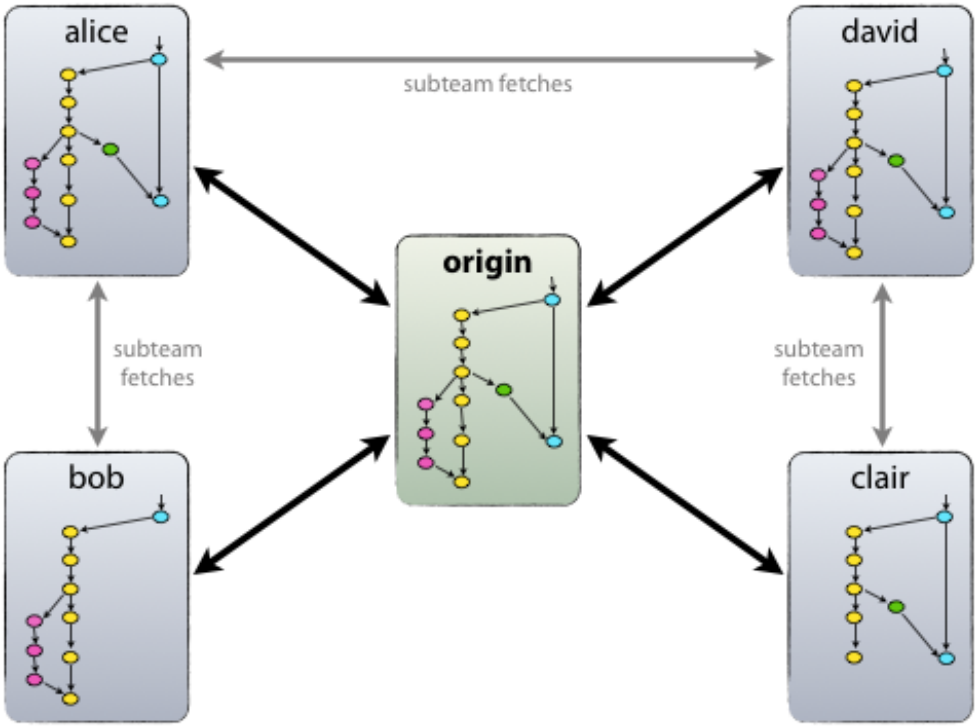
5.2.1.1 Hlavní větve repozitáře

Základním stavebním kamenem centrálního repozitáře `origin` jsou dvě větve s neomezenou životností - `master` a `develop`:

- `master` - V této větvi se nachází verze aplikace, která je vždy v takovém stavu, aby ji bylo možné kdykoliv nasadit do produkčního prostředí.

[1] Gitosis – *https://github.com/tv42/gitosis*
[2] Gitolite – *http://sitaramc.github.com/gitolite/*

- `develop` - Poslední revize této větve obsahuje vývojovou verzi aplikace s poslední dokončenou úpravou. Někdy je tato větev označována také jako integrační, neboť v této větvi dochází k integraci nových změn se zbytkem aplikace (spouštění testů atp.).



Obrázek 5.2: Způsob synchronizace dat mezi týmy a centrálním repozitářem. [4]

Ve chvíli, kdy aplikace ve vývojové větvi `develop` dosáhne stabilního stavu a je připravena k vydání, jsou veškeré poslední změny začleněny do hlavní větve `master` a označeny odpovídající verzí. Jak přesně tento proces probíhá, bude popsáno níže v rámci této kapitoly.

S ohledem na budoucí zapojení projektového repozitáře do procesu nasazení aplikací je třeba opravdu striktně dodržovat tu zásadu, že pokud jsou nahrány změny do `master` větve centrálního úložiště, jedná se vždy o novou verzi aplikace určenou k nasazení do produkčního prostředí.

5.2.1.2 Podpůrné větve repozitáře

Vedle hlavních větví `master` a `develop` tento model dále využívá sadu tzv. podpůrných větví, které vývojovému týmu umožňují paralelně upravovat aplikaci, jako například vyvíjet nové funkce aplikace, připravovat aplikaci k nasazení do produkčního prostředí, opravovat vzniklé chyby atp. Na rozdíl od hlavních větví projektového repozitáře mají tyto větve vždy pouze omezenou životnost a jsou po splnění svého účelu odstraněny.

Model pracuje s těmito podpůrnými větvemi:

- **Feature branches** - větve pro vývoj nových funkcí aplikace.
- **Release branches** - větve pro přípravu nasazení aplikace do produkčního prostředí.
- **Hotfix branches** - větve pro opravu vzniklých chyb.

Každý typ větve má svůj specifický účel a daná pravidla, která jasně definují, z které větve mohou být vytvořeny a do které větve naopak začleněny. Nutno připomenout, že z technického hlediska se tyto větve nijak neliší od jiných (např. `master` nebo `develop`), jedná se pouze o zásady, které musí dodržovat každý člen vývojového týmu.

Začleňování změn podpůrných větví repozitáře musí být prováděno pomocí příkazu:

```
$ git merge --no-ff <název_podpůrné_větve>
```

Důležitou roli zde hraje přepínač `--no-ff`, díky kterému bude v historii repozitáře zachována informace o existenci dané větve. V budoucnu pak lze velice snadno rozlišit, které revize jsou spojené s konkrétní větví repozitáře.

5.2.1.3 Feature branches

Tento typ větví slouží pro vývoj funkcí pro novou verzi aplikace. Vytváření tohoto typu větve probíhá výhradně z `develop` větve a její životnost je omezena dobou, po kterou probíhá vývoj nové funkcionality. V konečné fázi je větev začleněna zpět do `develop` větve nebo odstraněna v případě rozhodnutí, že nová funkcionalita nebude zahrnuta v nové verzi aplikace. Z hlediska jmenné konvence pro tyto větve platí, že mohou mít jakýkoliv název kromě názvů rezervovaných ostatními typy, tedy `master`, `develop`, `release-*` a `hotfix-*`. Tyto větve typicky existují pouze na lokálním repozitáři vývojářů, nikoliv v repozitáři `origin`.

5.2.1.4 Release branches

Větve typu release branches slouží k přípravě aplikace pro vydání nové verze. Příprava aplikace typicky spočívá v úpravě meta-dat aplikace (např. označení aplikace odpovídajícím číslem verze či datem atp.), je zde ale také možné provádět opravy posledních drobných chyb aplikace. Tyto činnosti jsou izolovány do samostatné větve především proto, aby došlo k uvolnění `develop` větve pro úpravy plánované pro nadcházející verzi aplikace.

Vytvoření `release` větve je prováděno výhradně z `develop` větve a právě tehdy, když došlo k začlenění veškerých úprav, které mají být vydány spolu s novou verzí aplikace. Teprve zároveň s vytvořením této větve dojde k označení aplikace odpovídající verzí, neboť `develop` větev do této doby shromažďovala pouze novou funkcionalitu pro novou verzi aplikace a ve většině případů do této chvíle není zřejmé, zda se bude jednat například o verzi 0.3 nebo 1.0. To je typicky odvozeno z množství změn obsažených v nové verzi aplikace a pravidly verzování daného projektu.

Pro tento typ větve platí jmenná konvence `release-*`, což znamená, že pokud novou verzí aplikace bude verze 1.0, bude vytvořena větev s názvem `release-1.0`. Životnost této větve je omezena jejím začleněním do `master` větve, kdy je nová verze nasazena do produkčního prostředí. Během této doby je v této větvi striktně zakázáno provádění rozsáhlejších změn. Ty by již měly být začleňovány do `develop` větve a zahrnuty až v další verzi aplikace.

Ve chvíli, kdy je nová verze aplikace připravena k vydání, dojde k začlenění této větve do `master` větve. Aby bylo možné se v budoucnu k této verzi aplikace vrátit, musí být tato akce zároveň opatřena značkou (tagem) s informací, o kterou verzi se jedná. Vzhledem k tomu, že tato větev může obsahovat opravy některých drobných chyb aplikace, je také potřeba pamatovat na její začlenění zpět do `develop` větve.

5.2.1.5 Hotfix branches

Posledním typem větví, které tento model využívá, jsou takzvané `hotfix` větve, které jsou velmi podobné výše popsaným `release` větvím, nicméně vznikají neplánovaně, a to při výskytu chyby v produkční verzi aplikace, kterou je potřeba okamžitě odstranit. Jsou vytvářeny výhradně z `master` větve a stejně jako u `release` větví je jim přiřazeno nové číslo verze aplikace, které se objeví v jejich názvu ve tvaru `hotfix-*` (hvězdička je nahrazena číslem verze).

Po dokončení opravy dané chyby je tato větev začleňována stejným způsobem jako `release` větev. Nejprve dojde k jejímu začleněné do `master` větve a aby byla oprava chyby zahrnuta i v příští verzi aplikace, je nutné provést začlenění také do `develop` větve (v případě její existence také do `release` větve).

5.3 Systém pro automatizaci procesu nasazení

Z hlediska implementace představuje komponenta pro automatizaci procesů nejnáročnější část tohoto řešení. Základním rysům a následně výběru nejvhodnějších nástrojů pro účely této práce se již věnovala kapitola 4.2 a jako nástroj pro automatizaci procesů kolem nasazování aplikací byl zvolen nástroj Phing. V rámci této kapitoly nejprve popíši základní vlastnosti této komponenty z hlediska návrhu a následně se budu věnovat nejzajímavějším částem automatizace jednotlivých procesů.

5.3.1 Struktura komponenty

Komponenta je tvořena sadou XML souborů interpretovatelných nástrojem Phing, které jsou pro větší přehlednost a snazší údržbu organizovány do několika modulů, z nichž každý se věnuje automatizaci určité části procesu. Díky této struktuře lze její funkcionalitu v budoucnu dále snadno rozšiřovat přidáním nových modulů či úpravou konkrétních částí komponenty.

Výjimkou z tohoto uspořádání je pak soubor `build.xml`, který zde plní funkci jakéhosi front-controlleru a představuje vstupní bod pro veškerou komunikaci s komponentou,

která probíhá vždy prostřednictvím příkazové řádky. Na rozdíl od ostatních XML souborů však neobsahuje žádnou logiku, ale pouze sadu všech spustitelných cílů (automatizovaných procesů) a jejich vykonání dále distribuuje mezi ostatní moduly komponenty. Tento návrh také činí soubor `build.xml` ideálním místem pro vykonání činností, které jsou součástí každého procesu. Příkladem může být import všech modulů komponenty či načtení systémové konfigurace (viz kapitola 5.3.2.1).

5.3.2 Konfigurace procesů

Jak již bylo zmíněno v kapitole 5.1 věnující se architektuře řešení, komponenta odpovědná za automatizaci procesů je navržena jako centralizovaný systém. Kromě zmíněných výhod je důsledkem takového návrhu také potřeba zajistit dostatečnou univerzálnost systému tak, aby bylo možné pracovat se všemi typy firemních aplikací a obsluhovat veškerá běhová prostředí pouhou změnou konfigurace.

Systém poskytuje v zásadě tři možnosti, jejichž pomocí lze průběh automatizovaných procesů ovlivnit či dokonce rozšířit:

- **parametrem volání procesu** - Využívá se pro drobné modifikace procesu, například předání verze nasazované aplikace nebo citlivé konfigurace, která není součástí jejího zdrojového kódu.

- **načtením konfiguračního souboru** - Systém pracuje se třemi typy konfiguračních souborů, jejichž pomocí lze ovlivnit průběh procesu způsobem, který je specifický pro konkrétní prostředí, typ aplikace atp. Systému konfiguračních souborů a způsobu jejich načítání se podrobněji věnuje kapitola 5.3.2.1.

- **rozšířením procesu** - Základní kostru procesu, která je vždy vykonána bez ohledu na typ aplikace či běhové prostředí, lze snadno rozšířit pomocí systému tzv. „*hooků*“, kterým se budu podrobněji věnovat později v kapitole 5.3.2.2. Typickým příkladem jejich využití je nasazení aplikací postavených na určitém frameworku, kdy je proces nasazení potřeba doplnit o činnosti specifické pro daný framework (smazání cache, rozdílné nastavení práv aj.).

5.3.2.1 Konfigurační soubory

Systém pracuje s třemi různými typy konfiguračních souborů. Každý typ konfiguračního souboru pak obsahuje informace specifické pro jednu z následujících úrovní:

- **systémová** - Systémová konfigurace, která je součástí systému, je načtena se spuštěním všech procesů a je umístěna v soubor `build.properties`. Obsahuje globální nastavení, jako je například umístění systému pro správu verzí, umístění adresáře pro sestavování aplikací atd.

- **běhové prostředí** - Sada konfiguračních souborů, z nich každý obsahuje nastavení specifické pro konkrétní běhové prostředí. Příkladem mohou být přístupové údaje k prostředí, umístění sdílených knihoven v daném prostředí atp. Konfigurační soubory pro jednotlivá běhová prostředí jsou součástí systému a jejich název je vždy ve tvaru `<env>.properties`, kde `<env>` je nahrazeno unikátním identifikátorem prostředí (např. „test“, „prod“ atp.).

- **konkrétní aplikace** - Konfigurace specifická pro konkrétní aplikaci je uložena společně s jejím zdrojovým kódem v projektovém repozitáři ve formě konfiguračního souboru `deploy.properties`. Její pomocí lze pro jednotlivé aplikace například definovat rozdílnou strukturu adresářů pro uživatelská data nebo způsob (hook), kterým má být rozšířena základní kostra procesů (viz kapitola 5.3.2.2).

Z popisu jednotlivých typů konfiguračních souborů je patrné, že pro každý typ konfigurace je jasně definována konvence z hlediska jeho názvu a umístění. Toto řešení zajišťuje efektivnější načítání konfiguračních informací, neboť zde odpadá potřeba neustálého předávání umístění konfiguračních souborů formou parametru při každém volání některého z procesů.

5.3.2.2 Rozšíření procesů

V předcházejících kapitolách jsem se zmínil o možnosti rozšíření základní kostry procesů, která je důležitá především z hlediska zajištění dostatečné univerzálnosti systému. Pro tyto účely systém nabízí možnost definování typu aplikace prostřednictví jejího konfiguračního souboru, na jehož základě jsou do libovolných částí procesů injektovány některé další činnosti specifické pro daný typ aplikace.

Systém může pracovat s libovolným počtem typů aplikací. Pro každý nový typ aplikace je potřeba systém rozšířit o nový XML soubor (umístěný ve složce `hooks`), ve kterém jsou definovány činnosti, o které chceme automatizovaný proces rozšířit. V současné době systém obsahuje rozšíření pro dva typy aplikací - aplikace založené na frameworku Symfony verze 1 (soubor `s1_hooks.xml`) a frameworku Symfony2 (soubor `s2_hooks.xml`). Odpovídající typ aplikace pak lze nastavit přímo v projektovém konfiguračním souboru `deploy.properties`, který je uložen spolu se zdrojovými kódy v projektovém repozitáři. [Ukázka 5.1]

Ukázka 5.1: Příklad projektového konfiguračního souboru `deploy.properties`.

```
# project deploy hook
deploy_hooks=s1_hooks

# list of uploads directories separated by comma
uploads_dirs=file,flash,picture,thumbnail
```

Příkladem takového rozšíření je příprava konfigurace aplikace. Součástí automatizovaného procesu nasazení typicky bývá předzpracování konfiguračních souborů aplikace, neboť citlivé nastavení (např. přístupové údaje k databázi) nesmějí být uloženy v projektovém repozitáři společně s aplikací. Protože pro každý typ aplikace mají konfigurační soubory jiné

umístění v rámci její adresářové struktury a jinou formu zápisu, je pro tyto účely rozšířena základní kostra procesu a předzpracování konfiguračních souborů probíhá dle konkrétního typu aplikace.

Způsob, kterým je rozšíření automatizovaného procesu realizováno, demonstrují ukázky 5.2 a 5.3.

Ukázka 5.2: Příklad rozšíření automatizovaného procesu.

```
<!-- ============================================ -->
<!-- Target: deploy -->
<!-- ============================================ -->
<target name="deploy" depends="deploy.prepare, deploy.loadProjectProperties">
    <!-- post-cache hook -->
    <echo msg="===HOOK=== Executing ${deploy_hooks} post-cache hook ===POST-CACHE===" />
    <phingcall target="${deploy_hooks}.postCache" />

    ... zbytek zdrojového kódu ...
</target>
```

Ukázka 5.3: Příklad rozšíření automatizovaného procesu.

```
<!-- ============================================ -->
<!-- Target: postCache -->
<!-- ============================================ -->
<target name="postCache"
    depends="s2_hooks.createParameters"
    description="Post-cache deploy hook" />

<target name="createParameters" description="Environment configuration" >
    <echo msg="[Symfony2 hook] Creating environment specific parameters.." />
    <copy
        file="${deploy.repositoryDirectory}/app/config/parameters.ini.dist"
        tofile="${deploy.repositoryDirectory}/app/config/parameters.ini"
        overwrite="true">
            <filterchain>
                <replacetokens begintoken="##" endtoken="##">
                    <token key="DB_HOSTNAME" value="${db_hostname}" />
                    <token key="DB_PORT" value="${db_port}" />
                    <token key="DB_USERNAME" value="${db_username}" />
                    <token key="DB_PASSWORD" value="${db_password}" />
                    <token key="DB_DATABASE" value="${db_database}" />
                </replacetokens>
            </filterchain>
    </copy>
</target>
```

Ukázka 5.2 zobrazuje tu část automatizovaného procesu nasazení, kdy již úspěšně proběhla příprava konkrétní verze zdrojových kódů aplikace a načtení projektové konfigurace. Z projektové konfigurace systém získal informaci o konkrétním typu aplikace (proměnná `${deploy_hooks}`), na jejímž základě tuto část procesu rozšíří o další činnosti definované

v cíli `postCache`.

Konkrétní příklad rozšíření procesu je znázorněn na ukázce 5.3. Vždy se jedná o prázdný cíl (v tomto případě cíl `postCache`), který musí být definován v každém rozšiřujícím XML souboru. Cíli `postCache` je pak pomocí atributu `depends` definován libovolný počet závislostí na činnosti specifické pro daný typ aplikace, které v daném místě rozšiřují základní kostru procesu.

Podobným způsobem lze rozšířit libovolnou část každého z automatizovaných procesů, díky čemuž systém zajišťuje takřka neomezené možnosti jejich přizpůsobení a umožní tak obsluhovat všechny stávající i budoucí typy aplikací.

5.3.3 Nasazení aplikace

Nejkomplexnějším procesem z hlediska automatizace je bezesporu proces nasazení aplikace do některého z běhových prostředí. Rozboru jednotlivých činností, které je potřeba vykonat při manuálním nasazení aplikace, jsem již věnoval v kapitolách 3.1 a 3.2. Na jeho základě byla navržena základní kostra automatizovaného procesu nasazení obohacená o některé další aspekty (např. práce s verzovacím systémem) a jeho výsledná podoba je znázorněna na obrázku 5.3.

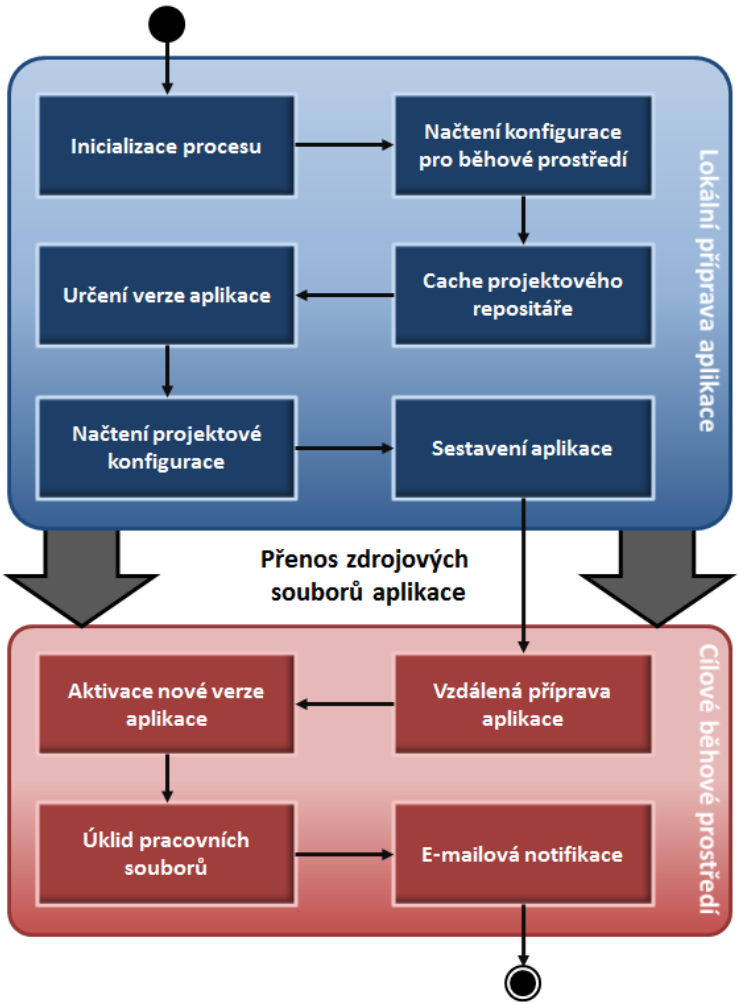
Z obrázku je patrné, že systém provádí převážnou část činností souvisejících se sestavením aplikace lokálně v rámci svých pracovních adresářů. Výsledkem sestavení aplikace je vždy archiv obsahující veškeré zdrojové kódy aplikace, který je bezpečně přenesen do cílového běhového prostředí.

1. **inicializace procesu** - Předchází spuštění každého procesu a spočívá především v načtení veškerých součástí systému a systémové konfigurace.

2. **načtení konfigurace pro běhové prostředí** - Dle zvoleného cílového prostředí dojde k načtení konfiguračního souboru, který obsahuje nastavení specifické pro dané běhové prostředí (viz kapitola 5.3.2).

3. **cache projektového repozitáře** - Abych předešel neustálému stahování kompletního zdrojového kódu aplikace při každém jejím nasazení, systém si ve svém pracovním adresáři při prvotním nasazení projektu vytvoří aktivní kopii projektového repozitáře. Ta je následně využita při každém dalším nasazení aplikace k získání konkrétní verze zdrojového kódu, díky čemuž lze výrazně snížit množství přenášených dat a dobu sestavení aplikace.

4. **určení verze aplikace** - Získání konkrétní verze zdrojových kódů aplikace. Pokud cílovou verzi aplikace neurčí sám uživatel, systém nasadí poslední verzi aplikace dle zvoleného běhového prostředí. V případě nasazení aplikace do produkčního prostředí je zvolena poslední verze vývojové větve `master`, v ostatních případech systém pracuje s větví `develop`.



Obrázek 5.3: Schéma automatizovaného procesu nasazení.

5. **načtení projektové konfigurace** - V této fázi procesu je načten konfiguračního soubor odpovídající zvolené verzi aplikace, který je důležitý zejména pro určení typu aplikace a tedy způsobu rozšíření procesu nasazení (viz kapitola 5.3.2.2).

6. **sestavení aplikace** - V této části již dochází k rozšíření procesu o činnosti specifické pro daný typ aplikace, po jejichž dokončení jsou od zdrojových kódů aplikace odděleny všechny soubory související s jejím projektovým repozitářem. Následuje označení sestavení aplikace unikátním identifikátorem a příprava archivu pro přenos do cílového běhového prostředí.

Přenos zdrojových kódu aplikace do cílového běhového prostředí probíhá prostřednictvím zabezpečeného protokolu Secure Copy (SCP), kde jsou následně pomocí vzdáleného komunikačního protokolu SSH provedeny následující činnosti:

1. **vzdálená příprava aplikace** - Před samotnou aktivací nové verze jsou v rámci procesu nasazení vykonány ty činnosti, které je vhodné nebo přímo nezbytné provést až v cílovém běhovém prostředí (např. nastavení práv, smazání cache aj.). V této fázi nasazení aplikace dochází také k zálohování kompletní databáze projektu a aktualizaci databázového schématu. Ta je dalším z příkladů rozšíření automatizovaného procesu nasazení, neboť způsob jejího provedení je svázán s konkrétním typem aplikace.

2. **aktivace nové verze aplikace** - V této fázi procesu dochází k nalinkování těch adresářů, které jsou sdílené mezi všemi verzemi nasazené aplikace (např. adresář s uživatelskými daty). Samotná aktivace nové verze aplikace pak spočívá ve vytvoření symbolického linku, který odkazuje na adresář s novou verzí aplikace.

3. **úklid pracovních souborů** - Po aktivaci nové verze aplikace se systém postará o úklid projektového adresáře, který obnáší například smazání přeneseného archivu aplikace nebo odstranění starších historických verzí aplikace dle konfigurace systému.

4. **e-mailová notifikace** - Všichni členové vývojového týmu jsou po dokončení procesu informováni o nasazení nové verze aplikace v daném běhovém prostředí formou e-mailového reportu, který navíc obsahuje podrobný log popisující veškeré činnosti provedené v rámci automatizovaného procesu nasazení.

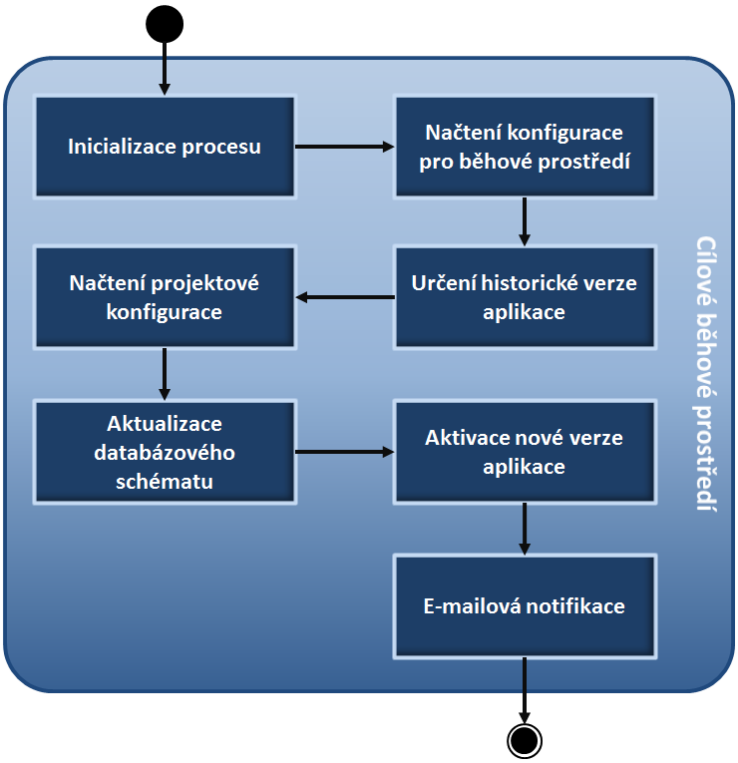
Systém navíc umožňuje nasazení aplikace ve formě kompletního projektového repozitáře, která je využívána v lokálním vývojovém prostředí programátorů. K tomuto účelu je možné využít odlehčenou verzi výše uvedeného procesu nasazení, kterou lze snadno spustit pomocí příkazové řádky na lokální stanici vývojáře. Systém poté uživatele interaktivně vyzve k zadání potřebných parametrů, kterým je například identifikátor projektového repozitáře nebo cílové umístění adresáře projektu, a provede automatizované nasazení aplikace včetně aktualizace databázového schématu.

5.3.4 Návrat k historické verzi

Návrat k historické verzi projektu (tvz. „*roll-back*“) lze samozřejmě provést opětovným nasazením starší verze aplikace. V takovém případě ovšem znovu dojde k jejímu kompletnímu sestavení, které může být mnohdy časově náročné a zbytečné obzvláště v případě, kdy jsme již stejnou verzi aplikace v minulosti nasazovali a máme její sestavu k dispozici. K tomuto účelu systém nabízí historii nasazených verzí aplikace, díky které lze návrat k historické verzi provést v řádu několika vteřin.

Součástí konfigurace systému je také možnost nastavení počtu historických verzí aplikace, které systém uchovává. Historické verze jsou umístěny přímo v adresáři projektu daného běhového prostředí. Každý záznam je opatřen unikátním identifikátorem tvořeným datem nasazení, který je při spuštění procesu předán prostřednictvím parametru.

Schéma návratu do historické verze je znázorněno na obrázku 5.4. Vzhledem k tomu, že historická verze aplikace se již připravena pro nasazení, jediným úkolem, který je potřeba provést před její aktivací, je aktualizace databázového schématu. Systém tedy nejprve



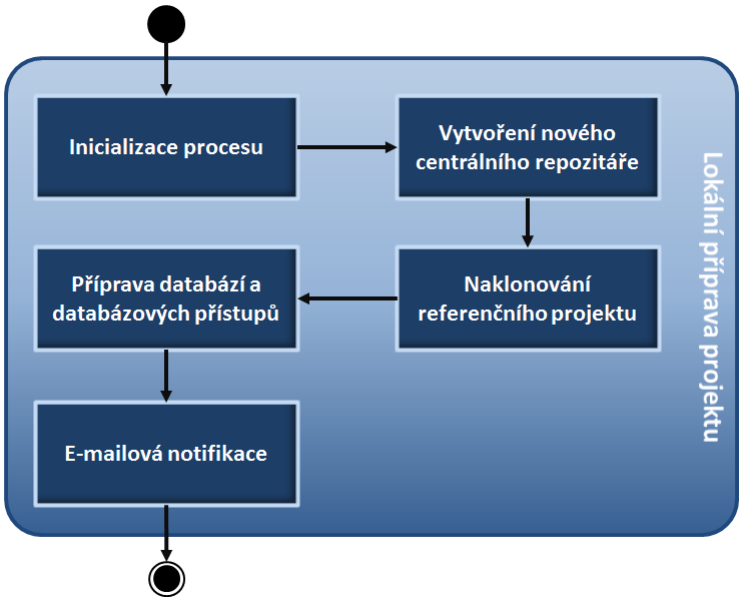
Obrázek 5.4: Schéma procesu automatizujícího návrat k historické verzi aplikace.

porovná aktuální verzi databázového schématu s cílovou verzí aplikace a za použití obousměrných migračních delta souborů uvede databázové schéma do odpovídajícího stavu. Následuje samotná aktivace historické verze aplikace vytvořením symbolického linku do příslušného adresáře a zaslání e-mailového reportu spolu s HTML logem, stejně jako v případě nasazení nové verze aplikace.

5.3.5 Založení projektu

Jednou z častých činností, která zahrnuje značné množství rutinních kroků, je založení nového projektu. Ve firmě, pro kterou je toto řešení připravováno, navíc tento proces zesložiťuje fakt, že nové projekty jsou v drtivé většině založeny na jiných tzv. referenčních projektech (např. referenční projekt pro e-shop, volební systém atp.). Pro založení nového projektu je tedy kromě vytvoření nového repozitáře navíc potřeba naklonovat a vhodně upravit nastavení některého z referenčních projektů.

Podoba kompletního automatizovaného procesu odpovídajícího za založení nového projektu je znázorněna na obrázku 5.5. Po inicializaci procesu dojde k vytvoření nového centrálního projektového repozitáře. Při použití nástroje Gitolite je k vytvoření nového repozitáře potřeba naklonovat repozitář `gitolite-admin`, který mimo jiné obsahuje konfigurační soubor nesoucí informace o všech repozitářích a jejich přístupových právech. Systém do to-



Obrázek 5.5: Schéma procesu automatizujícího založení nového projektu.

hoto konfiguračního souboru přidá informace o novém projektovém repozitáři a výsledné úpravy nahraje na server.

V případě, že uživatel zvolí některý z referenčních projektů, systém se postará o naklonování odpovídajícího repozitáře a úpravou konfigurace týkající se vzdáleného centrálního repozitáře nahraje veškerý jeho obsah do repozitáře, který byl vytvořen v předcházejícím kroku.

Po úspěšném vytvoření projektového repozitáře následuje příprava databáze v každém běhovém prostředí, která kromě vytvoření nové databáze obnáší také vytvoření nových databázových přístupů pro aplikaci spolu s vhodně nastavenými právy. Celý proces je, stejně jako v předcházejících případech, zakončen zasláním e-mailového reportu spolu s podrobným HTML logem.

5.3.6 Nedostupnost aplikace

Automatizovaný proces nasazení aplikace trvá o poznání déle než automatizovaný proces návratu k historické verzi. Doba, po kterou lze aplikaci uvést do nekonzistentního stavu, je však u obou procesů přibližně stejná, neboť při nasazení nové verze aplikace dochází k jejímu sestavení lokálně (nezávisle na „ostré“ verzi). U obou těchto procesů je pro aplikaci nejrizikovější časový úsek mezi aktualizací zdrojového kódu (aktivace nové verze aplikace) a aktualizací databázového schématu. Během této doby může aplikace například obdržet požadavek do zatím neexistující tabulky v databázi, čímž může dojít k pádu aplikace a ztrátě či poškození dat.

Ochrana aplikace před podobnými problémy spočívá v jejím odstavení po dobu, kdy ji lze uvést do nekonzistentního stavu. Systém pro automatizaci procesů k tomuto účelu

využívá soubor `.htaccess`, jehož pomocí jsou veškeré požadavky uživatelů v tomto časovém úseku přesměrovány na stránku `maintenance.html` informující o údržbě aplikace. Systém navíc obsahuje standardní stránku informující o údržbě, kterou před odstavením nahraje do kořenového adresáře aplikace v případě, že aplikace nedisponuje vlastní verzí této stránky.

Z hlediska návrhu automatizovaných procesů je pořadí provedení těchto dvou rizikových kroků irelevantní, nicméně vzhledem k minimalizaci času, po který je aplikace nedostupná, je vhodné jejich vykonání bezprostředně za sebou.

5.3.7 Ostatní funkce systému

Kromě komplexních automatizovaných procesů, kterým se věnovaly předcházející kapitoly, systém navíc poskytuje některé další funkce, mezi které patří například opětovné zaslání některého z e-mailových reportů, získání seznamu revizí některého z projektů nebo informací o jeho stavu v daném běhovém prostředí.

Výměnu informací mezi systémem pro automatizaci procesů a webovou aplikací usnadňuje také možnost získání odpovědi v jazyce XML, kterou lze následně snadněji zpracovat a nabídnout uživatelům. Způsobu komunikace mezi těmito komponentami se podrobněji věnuje kapitola 5.4.1.

5.4 Webová aplikace

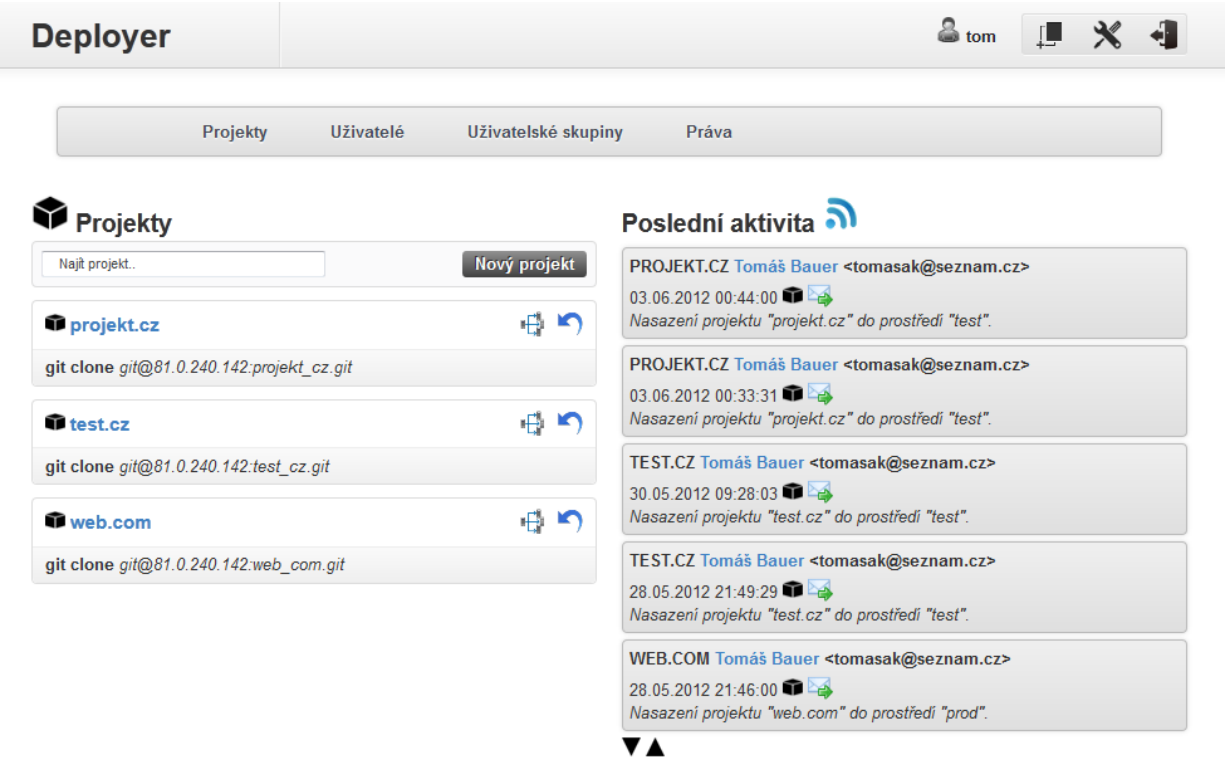
Hlavním úkolem webové aplikace je poskytnout uživatelům grafické rozhraní k systému pro automatizaci procesů. Uživatelské rozhraní pak zaručí vždy stejný způsob spouštění automatizovaných procesů a navíc usnadňuje práci se systémem nejen vývojovému týmu, ale umožní spustit proces nasazení aplikace například i členům managementu, pro které příkazová řádka nepředstavuje příliš vhodný způsob ovládání.

Na obrázku 5.6 je ukázka úvodní obrazovky webového uživatelského rozhraní. Zatímco v levé části má uživatel k dispozici seznam projektů, u kterých došlo v nedávné době k nějakým úpravám, pravá strana informuje o poslední aktivitě ostatních členů vývojového týmu, díky čemuž uživatel získá větší přehled o aktuálním dění ve firmě.

Kromě možnosti pohodlného spuštění některého z automatizovaných procesů aplikace dále nabízí přehled nejdůležitějších informací týkající se správy firemních projektů a jejich stavu v konkrétních běhových prostředích, řízení přístupu formou uživatelských práv, logování činnosti uživatelů aj.

5.4.1 Způsob komunikace

Webová aplikace komunikuje výhradně se systémem pro automatizaci procesů. Jedná se tedy o standardní webovou aplikaci obohacenou o servisní třídu, která je odpovědná za správné sestavení příkazů včetně všech důležitých parametrů na základě činnosti uživatele webové aplikace a jejich předání systému pro automatizaci procesů.



Obrázek 5.6: Ukázka hlavní obrazovky webového uživatelského rozhraní.

Na základě typu požadavku servisní třída rozhodne o tom, zda konkrétní akci vykonat na pozadí (typicky spuštění automatizovaných procesů) nebo počkat na odpověď systému. V případě, že se jedná o požadavek, na který uživatel očekává okamžitou odpověď, aplikace požádá systém (volbou vhodného parametru) o odpověď ve formátu XML, kterou následně zpracuje a nabídne uživateli. Příkladem takových požadavků může být vyžádání dostupných historických verzí aplikace v daném běhovém prostředí, seznamu revizí projektu atp.

Druhým typem požadavku jsou požadavky na spuštění automatizovaných procesů, jejichž vykonání trvá řádově až několik minut. Tyto akce jsou spouštěny na pozadí a jejich činnost je důsledně logována. O dokončení automatizovaného procesu je uživatel informován prostřednictví e-mailového reportu, jehož součástí je také uvedený log.

5.5 Reálné použití systému

Způsob použití výsledného řešení demonstruji na příkladu, ve kterém je členu vývojového týmu zadán požadavek na úpravu zdrojového kódu a databázového schématu některého z firemních projektů postavených na frameworku Symfony2.

5.5.1 Získání projektu

V případě, že programátor s projektem v poslední době nepracoval, jeho prvním krokem je získání zdrojových kódů aplikace z projektového repozitáře a její nasazení na své lokální stanici, která zde představuje vývojové prostředí. K tomuto účelu je k dispozici speciální verze automatizovaného procesu nasazení, která pro účely vývoje umožňuje nasazení aplikace ve formě kompletního projektového repozitáře.

Ukázka 5.4: Ukázka konfiguračního souboru `dev.properties`.

```
# hostname git serveru
git_hostname=192.168.10.41

# cesta k web root adresari
deploy.path=C:\xampp\htdocs

# path to PHP interpreter
php_path=php

# Symfony lib a data adresare
symfony_lib_path=C:\xampp\php\symfony
symfony_data_path=C:\xampp\symfony

# DB konfigurace
db_hostname=localhost
db_port=
db_username=root
db_password=
```

Před prvotním spuštěním procesu na lokální stanici vývojáře je navíc potřeba upravit konfiguraci v souboru `dev.properties`, který obsahuje konfigurační informace specifické pro vývojové prostředí uživatele. [Ukázka 5.4]

Automatizovaný proces nasazení pak lze snadno spustit z příkazové řádky pomocí příkazu:

```
$ phing deploy.dev
```

Po úvodní inicializaci procesu je uživatel interaktivně vyzván k zadání potřebných parametrů (např. identifikátor projektového repozitáře nebo jeho cílové umístění), po jejichž správném zadání je aplikace automaticky nasazena (včetně aktualizace databázového schématu) na lokální stanici vývojáře. [Obrázek 5.7]

```
Slim@Slim-DELL /cygdrive/c/xampp/htdocs/deployer
$ phing.bat deploy.dev
Buildfile: C:\xampp\htdocs\deployer\build.xml
 [property] Loading C:\xampp\htdocs\deployer\conf\build.properties

deployment > dev.properties:

     [echo] Loading environment specific properties..
 [property] Loading C:\xampp\htdocs\deployer\.\conf\dev.properties

deployment > deploy.repository:

Identifikátor projektu: [weby_drosera_cz] test_cz
Název adresáře projektu: [test_cz]
Název databáze: [test_cz]
     [echo]
     [echo]
     [echo] ============================
     [echo] Executing deployment ...
     [echo] ============================
     [echo] Web root directory resolved as:   C:\xampp\htdocs
     [echo] Repository resolved as:   git@81.0.240.142:test_cz.git
[phingcall] Calling Buildfile 'C:\xampp\htdocs\deployer\build.xml' with target 'deploy.repositoryDeploy'
 [property] Loading C:\xampp\htdocs\deployer\conf\build.properties

deployment > deploy.repositoryPrepare:

    [mkdir] Created dir: C:\xampp\htdocs\test_cz
     [echo] Cloning into 'C:\xampp\htdocs/test_cz'...
     [echo] DONE
     [echo]
     [echo] Checkout develop branch..
     [echo] Branch develop set up to track remote branch develop from origin.
Switched to a new branch 'develop'

deployment > deploy.loadProjectProperties:

     [echo] Loading project specific properties..
 [property] Loading C:\xampp\htdocs\test_cz\deploy.properties

deployment > deploy.repositoryDeploy:

     [echo] ==============================
     [echo] CREATING UPLOADS DIRECTORIES ...
     [echo] ==============================
  [foreach] Calling Buildfile 'C:\xampp\htdocs\deployer\build.xml' with target 'createDevUploadsSubdirs'
 [property] Loading C:\xampp\htdocs\deployer\conf\build.properties

deployment > createDevUploadsSubdirs:
```

Obrázek 5.7: Ukázka nasazení projektu na lokální stanici vývojáře.

5.5.2 Úprava aplikace

Po provedení potřebných úprav zdrojového kódu aplikace (včetně její modelové vrstvy) dle zadání požadavku je před nahráním změn do projektového repozitáře navíc potřeba vytvořit odpovídající migrační delta soubor v podobě PHP třídy. Ta popisuje obousměrné změny databázového schématu a bude společně se zdrojovými kódy aplikace uložena v projektovém repozitáři.

Způsobu vytvoření migračních tříd dle typu aplikace se již věnovala kapitola 4.3. Tento příklad demonstruje úpravu aplikace založené na frameworku Symfony2, kde vytvoření migrační třídy spočívá ve spuštění jediného příkazu:

```
$ php app/console doctrine:migrations:diff
```

Ukázka 5.5: Příklad vygenerované migrační PHP třídy pro framework Symfony2.

```
<?php

namespace Application\Migrations;

use Doctrine\DBAL\Migrations\AbstractMigration,
    Doctrine\DBAL\Schema\Schema;

/**
 * Auto-generated Migration: Please modify to your need!
 */
class Version20120509164837 extends AbstractMigration
{
    public function up(Schema $schema)
    {
        // this up() migration is autogenerated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() != "mysql");

        $this->addSql("ALTER TABLE log ADD revision VARCHAR(255) DEFAULT NULL");
    }

    public function down(Schema $schema)
    {
        // this down() migration is autogenerated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() != "mysql");

        $this->addSql("ALTER TABLE log DROP revision");
    }
}
```

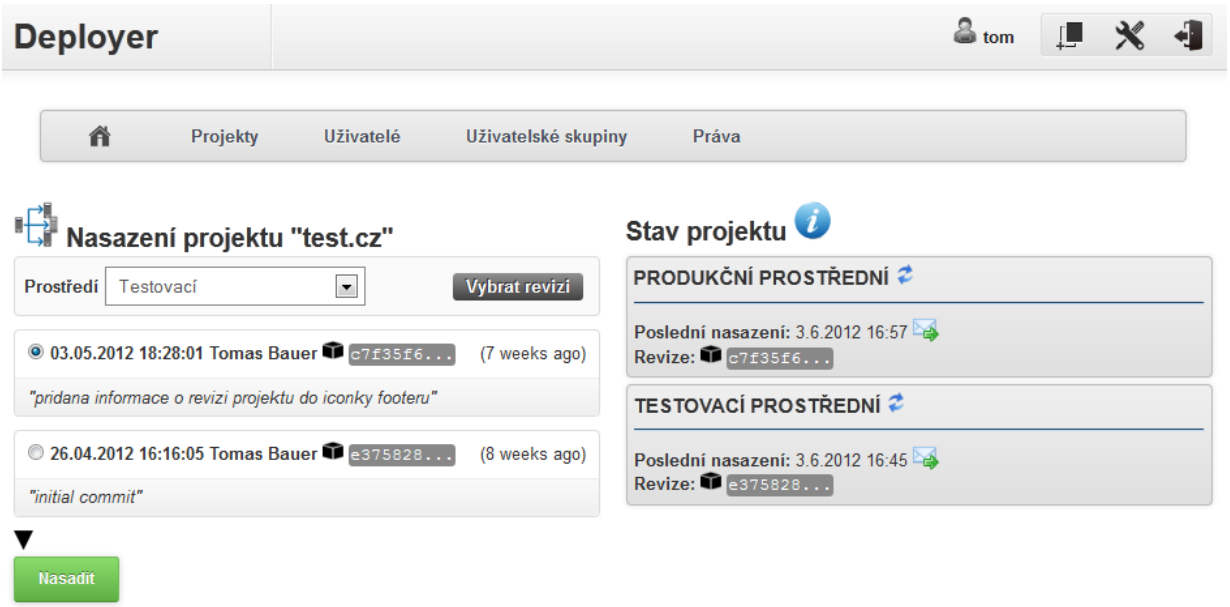
Výsledkem tohoto příkazu je nově vygenerovaná PHP třída s metodami `up()` a `down()`, které obsahují odpovídající sady SQL příkazů. [Ukázka 5.5]

Před jejím nahráním do projektového repozitáře je vždy vhodné její spuštění v obou směrech důkladně otestovat také v rámci vývojového prostředí, neboť úpravy těch, které již byly nahrány do projektového repozitáře, mohou následně způsobovat problémy při jejich vykonání v rámci nasazení aplikace. V případě výskytu chyby například v názvu sloupce databázové tabulky je pak potřeba, místo opravy chyby ve stávajícím souboru, vygenerovat nový migrační soubor, který dodatečně provede přejmenování sloupce.

5.5.3 Nasazení nové verze aplikace

Aby bylo možné označit úkol za dokončený, je třeba provedené změny nasadit do některého z běhových prostředí (typicky testovacího), kde může dále proběhnout akceptační testování a schválení nové funkcionality.

Spuštění automatizovaného procesu nasazení nové verze aplikace lze provést prostřednictvím uživatelského rozhraní webové aplikace. [Obrázek 5.8] Zatímco v pravé části obrazovky jsou uživateli nabídnuty informace o stavu projektu v jednotlivých běhových pro-



Obrázek 5.8: Nasazení aplikace prostřednictvím webového uživatelského rozhraní.

středích, v levé části uživatel volí cílové běhové prostředí (volitelně také konkrétní revizi projektu) a stisknutím tlačítka „Nasadit“ vyvolá spuštění procesu.

O dokončení automatizovaného procesu nasazení jsou členové projektového týmu informování e-mailovým reportem, jehož součástí je také podrobný HTML log. Příklad konkrétního HTML logu (jeho část) je součástí přílohy této práce. [Příloha B]

6 Závěr

6.1 Splnění cílů práce

Cílem této práce bylo navrhnout a vytvořit řešení pro automatizované nasazení internetových aplikací v konkrétní vývojářské firmě. Protože sám jsem měl mnohaleté zkušenosti pouze s ručním nasazováním internetových aplikací, samotnému návrhu řešení předcházelo důkladné seznámení s danou problematikou a přístupy jako *Continuous Integration*, *Continuous Delivery* a *Continuous Deployment*, které se na metodiku doručení softwarového produktu koncovým uživatelům přímo orientují. Získané znalosti jsem shrnul v kapitole 2, která popisuje základní principy výše uvedených přístupů a uvádí je v kontrastu s jednou z nejpoužívanějších metodik softwarového inženýrství.

V rámci kapitoly 3, taktéž spadající do přípravné fáze, byl proveden rozbor veškerých činností souvisejících s nasazením aplikací, které byly v dané firmě doposud prováděny manuálně. Na jeho základě jsem následně nastínil možné způsoby jejich automatizace a definoval základní požadavky na výsledné řešení.

Jedním z dílčích úkolů před zahájením samotné implementace byla mimo jiné analýza dostupných nástrojů, které se věnuje kapitola 4. Vstupem této analýzy byl popis požadavků, na jehož základě byla určena kritéria, podle kterých byly vybrány nejvhodnější nástroje tvořící jednotlivé komponenty celého řešení s cílem minimalizovat potřebu vlastního vývoje. Veškeré nabyté znalosti byly následně využity při návrhu a implementaci jednotlivých komponent, jejichž nejzajímavějším částem se věnuje kapitola 5. Vzhledem k tomu, že firma, pro kterou bylo toto řešení vytvořeno, dosud aktivně nevyužívala žádný verzovací systém, součástí této kapitoly je také popis vybraného modelu větvení projektových repozitářů, který jasně definuje pravidla práce s verzovacím systémem.

Výsledné řešení beze zbytku splňuje veškeré definované požadavky a jeho kvalitní návrh vytváří prostor dalšímu vývoji a budoucímu rozšíření jeho funkcionality. Aktuální verze již byla v dané firmě nasazena a postupně bude využívána pro nasazení všech firemních aplikací.

6.2 Zhodnocení řešení a možná rozšíření

Ačkoliv zavedení systému pro správu verzí je už pro firmu samo o sobě krokem vpřed, jeho využívání se již v dnešní době dá považovat za standard. V tomto případě však navíc

tvoří nezbytnou komponentu řešení, jehož hlavním přínosem je bezesporu minimalizace času stráveného nasazováním firemních aplikací. Vzhledem k tomu, že přibližná doba manuálního nasazení aplikace se ve firmě pohybovala kolem třiceti minut, při jediném využití automatizovaného procesu denně může roční úspora firemních zdrojů dosahovat až dvaceti člověkodní.

Absence automatizovaného řešení ve firmě navíc často vedla k odkládání nasazení nové funkcionality. Jako vedlejší efekt lze tedy očekávat určitou úsporu také díky možnosti snadného doručení nové funkcionality ihned po jejím dokončení. Při zvýšení frekvence doručování změn koncovým uživatelů by v souladu s přístupem *Continuous Delivery* nadále nemělo docházet k nasazování velkých změnových balíků, u kterých značně roste riziko výskytu nějakého problému stejně, jako množství času potřebného k jeho odstranění.

V budoucnu lze doporučit rozšíření tohoto řešení o některé další prvky, kterými jsou například spouštění automatizovaných testů nebo analýzy kódu, jejichž pomocí lze v rámci procesu nasazení také snadno zvýšit a kontrolovat kvalitu softwarových produktů. Ať už snížením reakční doby, úsporou firemních zdrojů nebo zvýšením kvality softwarových produktů, výsledné řešení napomůže ke zvýšení konkurenceschopnosti firmy, zpříjemní a zejména zefektivní práci členů vývojového týmu a zvýší jejich důvěru ve vyvíjený softwarový produkt.

Literatura

[1] ALDORF, Filip. *Metodika RUP*. Praha, 2007. 60 s. Diplomová práce. VŠE, Katedra informačních technologií.

[2] BLACK Alex, HOLTGREWE Manuel, LELLELID Hans, ROOK Michiel a ADERHOLD Andreas. Phing: User Guide. [online]. [cit. 2012-05-23]. Dostupné z: http://www.phing.info/docs/guide/stable/

[3] BRAVENEC, Petr. *Rsync - inteligentní kopírování souborů*. ABC Linuxu [online]. 2010, [cit. 2012-02-28]. Dostupné z: *http://www.abclinuxu.cz/clanky/rsync-inteligentni-kopirovani-souboru*

[4] DRIESSEN, Vincent. *A successful Git branching model*. NVIE [online]. 5.1.2010, [cit. 2011-12-20]. Dostupný z: *http://nvie.com/posts/a-successful-git-branching-model/*.

[5] DUVALL, Paul; MATYAS, Steve; GLOVER, Andrew. *Continuous Integration : Improving Software Quality and Reducing Risk*. 1st edition. Boston : Addison-Wesley Professional, 2007. 336 s. ISBN 03-2133-638-0.

[6] EVRON, Shahar. *Best Practices in PHP Application Deployment*. SlideShare.com [online]. 2010 [cit. 2012-02-28]. Dostupné z: *http://www.slideshare.net/shahar/best-practices-inphpappdeployment*

[7] FOWLER, Martin. Martin Fowler [online]. 2006 [cit. 2011-10-30]. *Continuous Integration*. Dostupné z: *http://martinfowler.com/articles/continuousIntegration.html*.

[8] HUMBLE, Jez; FARLEY, David. *Continuous Delivery : Reliable Software Releases through Build, Test, and Deployment Automation*. 1st edition. Boston : Addison-Wesley Professional, 2010. 512 s. ISBN 03-2160-191-2.

[9] JANSCH, Ivo. *Dealing with Deployment*. PHP Architect. Červen 2009, [cit. 2011-10-30], roč.8, č.6, s. 10-13. ISSN 1709-7169

[10] KUČERA, František. *Distribuované verzovací systémy*. ABC Linuxu [online]. 25.1.2011, [cit. 2011-12-20]. Dostupný z: *http://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1*.

[11] Malmö University [online]. 2001 [cit. 2011-10-30]. *Rational Unified Process: Disciplines*. Dostupné z: *http://www.ts.mah.se/RUP/RationalUnifiedProcess/process/workflow/ovu_core.htm*.

[12] MAY, Chris. *Deploy every 30 minutes: Redux*. [online]. 2005, 25. 8. 2005 [cit. 2012-05-17]. Dostupné z: *http://blogs.warwick.ac.uk/chrismay/entry/deploy_every_30/*

[13] MURRAY, Peter. *Creating Applications in Four Tiers*. Disruptive Library Technology Jester [online]. 2006, [cit. 2012-02-26]. Dostupné z: *http://dltj.org/article/software-development-practice/*

[14] Rational (IBM). *Rational Unified Process : Best Practices for Software Development Teams*. [online]. 2005, [cit. 2011-10-30]. Dostupný z: *http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf*.

[15] ROBBINS, J. *Analysis of Git and Mercurial*. Google Code [online]. 4.2.2010, [cit. 2011-12-20]. Dostupný z: *http://code.google.com/p/support/wiki/DVCSAnalysis*.

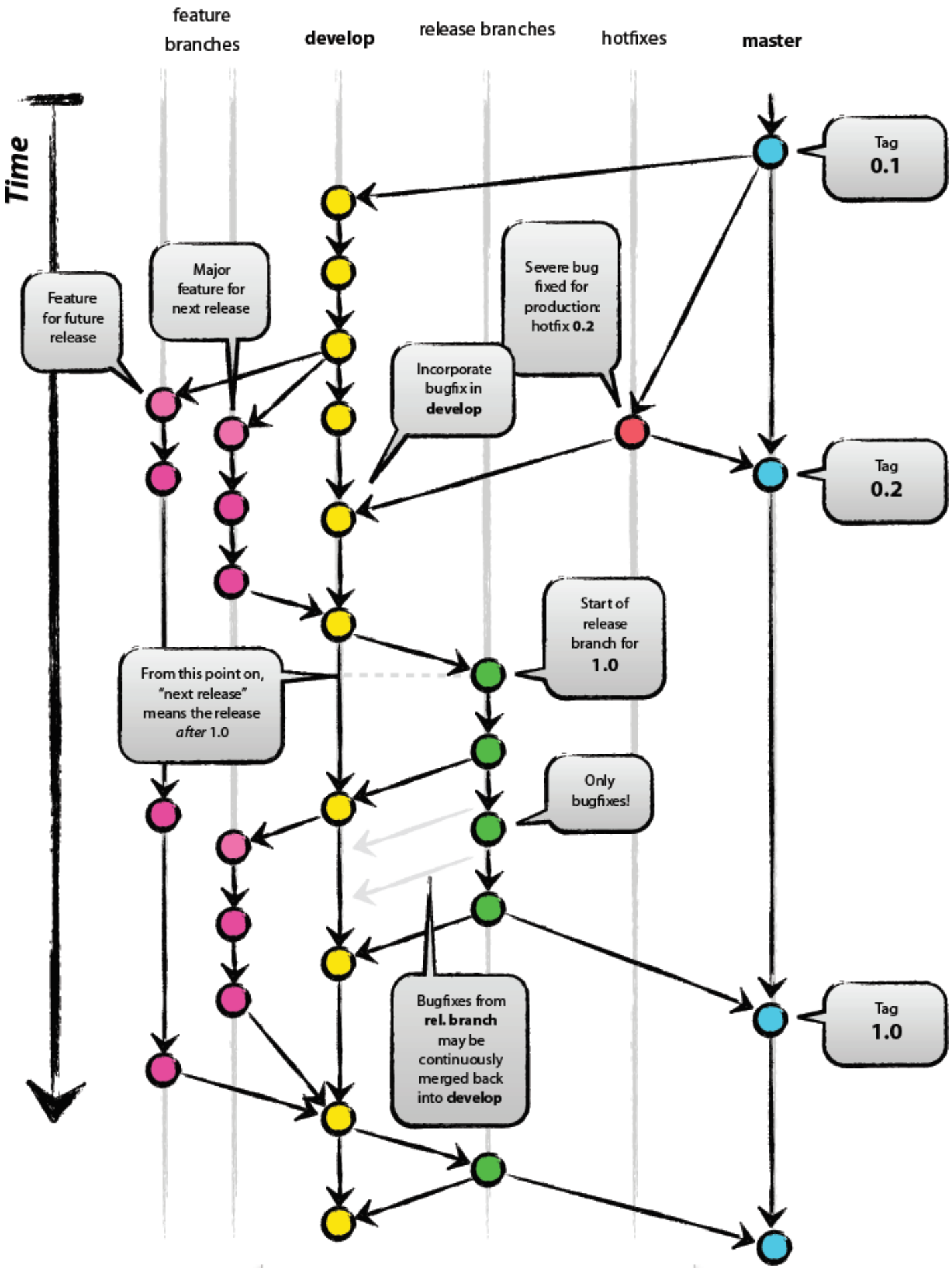
[16] SCHROEDER, Kevin. *Guidelines for deploying PHP applications*. ESchrade [online]. 2010, [cit. 2012-02-28]. Dostupné z: *http://www.eschrade.com/page/deployment-pear-4c228790*

[17] STANSBERRY, Glen. *7 Version Control Systems Reviewed*. Smashing Magazine [online]. 18.10.2008, [cit. 2011-12-20]. Dostupný z: *http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/*.

[18] SZALBOT, Petr. *PEAR*. ABC Linuxu [online]. 2005, [cit. 2012-02-27]. Dostupné z: *http://www.abclinuxu.cz/clanky/programovani/pear-i*

[19] VERVEER, Harrie. *Database version control DPC version*. SlideShare.com [online]. 2010 [cit. 2012-03-25]. Dostupné z: *http://www.slideshare.net/harrieverveer/database-version-control-without-pain-the-dpc-version*

Příloha A



Příloha A: Model větvení projektového repozitáře. [4]

Příloha B

Buildfile: /home/deployer/deployer/build.xml
[property] Loading /home/deployer/deployer/conf/build.properties

deployment > test.properties:
[property] Loading /home/deployer/deployer/../conf/test.properties

deployment > deploy.main:
[echo] ============================
[echo] Executing deployment ...
[echo] ============================
[echo] Target directory resolved as: /srv/www/deployment/env/test/weby2.drosera.cz
[echo] Repository resolved as: git@localhost:weby2_drosera_cz.git
[phingcall] Calling Buildfile '/home/deployer/deployer/build.xml' with target 'deploy.deploy'
[property] Loading /home/deployer/deployer/conf/build.properties

deployment > deploy.prepare:
[echo] Fetching from repository..
[echo]
[echo] Target revision was selected by the user: cca1434a2dd3d8911126e1f638400d47407547d6
[echo] Revision checkout: cca1434a2dd3d8911126e1f638400d47407547d6
[echo]
[php] Calling PHP function: strrpos()

deployment > deploy.loadProjectProperties:
[echo] Loading project specific properties..
[property] Loading /home/deployer/deployer/.builds/weby2.drosera.cz/cache/repository/deploy.properties

deployment > deploy.deploy:
[echo] =====HOOK===== Executing s2_hooks post-cache hook =====POST-CACHE=====
[phingcall] Calling Buildfile '/home/deployer/deployer/build.xml' with target 's2_hooks.postCache'
[property] Loading /home/deployer/deployer/conf/build.properties

deployment > s2_hooks.createParameters:
[echo]
[echo] [Symfony2 hook] Creating environment specific parameters..
[copy] Copying 1 file to /home/deployer/deployer/.builds/weby2.drosera.cz/cache/repository/app/config

deployment > s2_hooks.setCredentials:

deployment > s2_hooks.updateVendors:
[echo] [Symfony2 hook] Installing vendors..

deployment > s2_hooks.robots:

deployment > deploy.createArchive:
[echo] Creating tarball..
[tar] Building tar: /home/deployer/deployer/.builds/weby2.drosera.cz/tarballs/20120623003553_develop.tar.gz
[php] Evaluating PHP expression: floor(5158996/1024)
[php] Evaluating PHP expression: floor(5158996/1024/1024)
[echo] Archive 20120623003553_develop.tar.gz successfully created..
[echo] Filesize is: 5038 Kb
[echo] Filesize is: 4 Mb
[foreach] Calling Buildfile '/home/deployer/deployer/build.xml' with target 'deploy.deliver'
[property] Loading /home/deployer/deployer/conf/build.properties

Příloha B: Úvodní část HTML logu činnosti automatizovaného procesu nasazení.