

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Kontinuální integrace při vývoji webových aplikací v PHP

BAKALÁŘSKÁ PRÁCE

Student : Martin Hujer

Vedoucí : Ing. Jan Mittner

Oponent : Ing. Luboš Pavlíček

2012

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 18. 4. 2012

.....
Martin Hujer

Poděkování

Rád bych tímto poděkoval vedoucímu své práce, Ing. Janu Mittnerovi, za cenné podněty a připomínky, které mi poskytl při tvorbě této práce.

Abstrakt

Tato práce se zabývá kontinuální integrací webových aplikací, především těch v jazyce PHP. Hlavním cílem je výběr vhodného serveru pro kontinuální integraci, dále jeho nasazení a konfigurace pro integraci webových aplikací v jazyce PHP.

První kapitola popisuje koncept kontinuální integrace a její jednotlivé techniky.

Druhá kapitola se zabývá výběrem serveru pro kontinuální integraci a jeho základním nastavením.

Třetí kapitola obsahuje přehled jednotlivých nástrojů, které jsou využívány v rámci kontinuální integrace webových aplikací v jazyce PHP.

Čtvrtá kapitola se zabývá tvorbou skriptu pro sestavení, nasazením a konfigurací jednotlivých komponent.

Klíčová slova

Kontinuální integrace, PHP, webové aplikace, verzovací systémy, statická analýza, Jenkins.

Abstract

This work deals with continuous integration of web applications, especially those in PHP language. The main objective is the selection of the server for continuous integration, its deployment and configuration for continuous integration of PHP web applications.

The first chapter describes the concept of continuous integration and its individual techniques.

The second chapter deals with the choice of server for continuous integration and its basic settings.

The third chapter contains an overview of the different tools that are used for the continuous integration of web applications in PHP.

The fourth chapter deals with build script creation and configuration of individual tools.

Keywords

Continuous integration, PHP, web applications, version control systems, static analysis, Jenkins.

Obsah

Úvod	8
Vymezení tématu práce a důvod výběru tématu	8
Cíle práce	8
Předpoklady a omezení práce	8
Očekávané přínosy práce	8
Rešerše zdrojů pojednávajících o kontinuální integraci	9
1 Charakteristika konceptu kontinuální integrace	10
1.1 Přehled technik a zásad kontinuální integrace	11
1.1.1 Jednotné úložiště zdrojových kódů	11
1.1.2 Automatizované sestavení aplikace	12
1.1.3 Testovatelný build	13
1.1.4 Každý vývojář ukládá kód do úložiště alespoň jednou za den	13
1.1.5 Po každém commitu proběhne sestavení aplikace na integračním stroji	14
1.1.6 Sestavení musí být rychlé	14
1.1.7 Testování by se mělo provádět v prostředí co nejpodobnějším produkčnímu	15
1.1.8 Každý má snadný přístup k informacím	15
1.1.9 Automatizované nasazení	15
1.2 Typický průběh práce vývojáře v prostředí kontinuální integrace	15
1.3 Testování software	16
1.3.1 Manuální testování	16
1.3.2 Automatizované testování	17
1.3.3 Metriky – výsledky automatizovaných testů, pokrytí kódu testy	17
1.4 Statická analýza kódu, kontrola kvality	17
1.4.1 Počet řádků kódu, programových struktur	18
1.4.2 Složitost (komplexita) kódu	18
1.4.3 Standardy pro psaní kódu	19
1.4.4 Duplicitní kód	20
2 Výběr vhodného integračního serveru a jeho konfigurace	21
2.1 Přehled trhu serverů pro kontinuální integraci	21
2.1.1 Příbuzný software	21
2.2 Jenkins / Hudson	22
2.3 Instalace serveru Jenkins	23
2.4 Možnosti konfigurace serveru Jenkins	24
2.4.1 Zabezpečení serveru	25
3 Přehled nástrojů pro kontinuální integraci v PHP	26
3.1 PEAR	26
3.2 PHP Lint	27
3.2.1 Kontrola dopředné kompatibility	28
3.2.2 Pre-commit hook	28
3.3 PHP_CodeSniffer	29
3.4 PHP CPD	30
3.5 PHPLOC	30

3.6	PHP Depend	31
3.7	PHP MD	32
3.8	PHPUnit	33
3.9	Generování API dokumentace	34
3.9.1	PhpDocumentor	34
3.9.2	DocBlox	35
3.9.3	ApiGen	37
3.9.4	Výběr nástroje pro generování dokumentace	38
3.10	Automatizace sestavení	39
3.10.1	Výběr vhodného nástroje pro automatizaci sestavení	40
4	Praktická implementace platformy pro kontinuální integraci v malé firmě	41
4.1	Skript pro sestavení aplikace	41
4.2	PHP Lint	41
4.3	Vytvoření projektu na integračním serveru	42
4.4	Nasazení PHPUnit	45
4.4.1	Pokrytí kódu testy	46
4.5	Nasazení PHP_CodeSniffer	46
4.6	Nasazení ApiGen	47
4.7	Nasazení PHPCPD	48
4.8	Nasazení PHPLOC	49
4.9	Nasazení PDepend	50
4.10	Nasazení PHPMD	51
4.11	Nasazení rozšíření Violations	51
4.12	Nasazení PHP_CodeBrowser	52
4.13	Sledování změn v úložišti, automatické spouštění sestavení	53
4.14	Informace o proběhnutých sestaveních	54
4.15	Odstraňování informací o proběhlých sestaveních	55
4.16	Reálné nasazení	56
	Závěr	57
	Terminologický slovník	58
	Seznam použité literatury a zdrojů	59
	Seznam obrázků, tabulek a ukázek kódu	62
	Seznam obrázků	62
	Seznam tabulek	62
	Seznam ukázek kódu	62
	Příloha 1: Skript pro sestavení ukázkové aplikace (build.xml)	64
	Příloha 2: Úsek XML souboru pro nastavení rozšíření Plot	66

Úvod

Vymezení tématu práce a důvod výběru tématu

Velkou výhodou webových aplikací je možnost pružné reakce na změnu požadavků. Nicméně, při časté změně požadavků a tlaku trhu na jejich rychlou implementaci a nasazení do provozu, dochází ke snížení kvality programového kódu (tzv. technologický dluh), což se v dlouhodobém horizontu často projevuje zvýšením nákladů na jeho úpravy a také zvýšením chybovosti.

Během své praxe PHP a Zend Framework vývojáře jsem na tento jev postupně narážel, a proto jsem se začal o problematiku kvality programového kódu hlouběji zajímat. Zjistil jsem, že techniky a nástroje, které se již delší dobu běžně využívají při vývoji aplikací v Javě, začínají teprve postupně pronikat mezi vývojáře webových aplikací.

Proto jsem se rozhodl, jako svou bakalářskou práci, toto téma zpracovat z pohledu vývojáře webových aplikací v jazyce PHP.

Cíle práce

Hlavním cílem této práce je implementace kontinuální integrace do procesu vývoje webových aplikací v jazyce PHP v malém webovém studiu.

Díličními cíli, jejichž splněním toho bude dosaženo, jsou:

- charakterizovat koncept kontinuální integrace
- vybrat a nasadit vhodný server pro kontinuální integraci
- vytvořit přehled nástrojů využitelných pro kontinuální integraci projektů v jazyce PHP
- vytvořit skript pro automatizaci sestavení
- automatizovat sestavení s využitím serveru pro kontinuální integraci

Předpoklady a omezení práce

Práce se zaměřuje na kontinuální integraci webových aplikací v jazyce PHP, nicméně principy mohou být snadno přenositelné i na webové aplikace vyvíjené v jiných programovacích jazycích.

Cílovou skupinou jsou spíše menší webová studia, takže se práce nezabývá využitím více integračních serverů a distribuovaným spouštěním sestavení.

Očekávané přínosy práce

Bakalářská práce by měla popsat nasazení kontinuální integrace do procesu vývoje webových aplikací natolik prakticky, aby podle ní mohla jednotlivá webová studia postupovat při vlastní implementaci.

Rešerše zdrojů pojednávajících o kontinuální integraci

Prvním zdrojem zabývajícím se kontinuální integrací byl [FOWLER, 2006]. Většina dalších prací, které se kontinuální integrací zabývají, cituje jeho původní definici. Mnoho informací o kontinuální integraci shrnul [DUVALL, 2007] (autorem předmluvy je výše zmíněný Fowler). Oba tyto zdroje se zabývají kontinuální integrací obecně, případně pro projekty vyvíjené v jazyce Java nebo frameworku .NET.

Důležitým zdrojem byl [BERGMANN, 2011], který se zabývá kvalitou kódu a testováním aplikací v jazyce PHP.

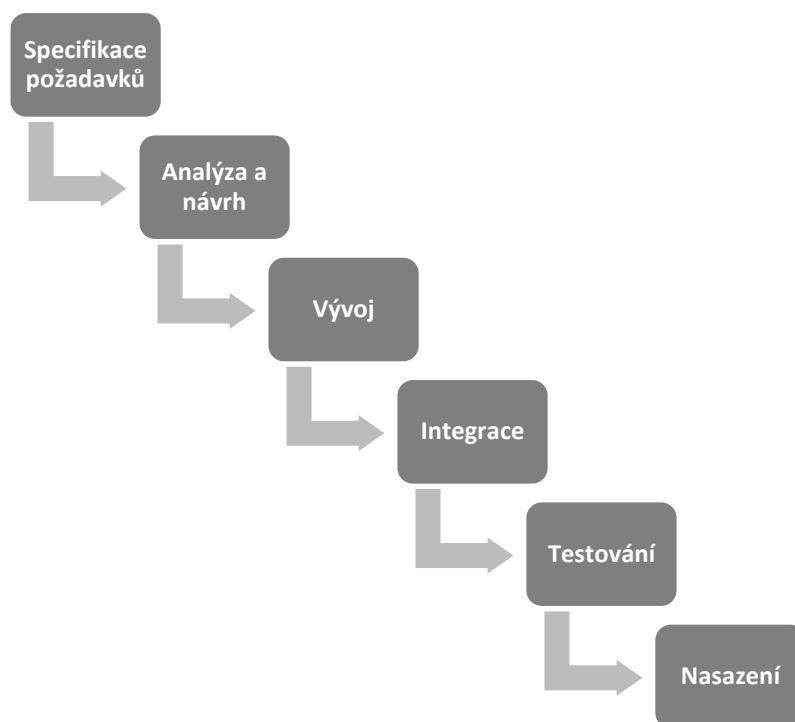
Dalším informačním zdrojem je web [BERGMANN, 2010], který obsahuje ukázkovou šablonu pro nastavení kontinuální integrace pro PHP projekty na integračním serveru Jenkins.

V průběhu mé práce vyšla kniha [SMART, 2011], která se detailně zabývá instalací, konfigurací a správou integračního serveru Jenkins.

Dále vyšla kniha [BERGMANN, 2011b], která se zabývá instalací a konfigurací integračního serveru Jenkins přímo pro webové aplikace v PHP. Autor nicméně zvolil některé jiné nástroje než já.

1 Charakteristika konceptu kontinuální integrace

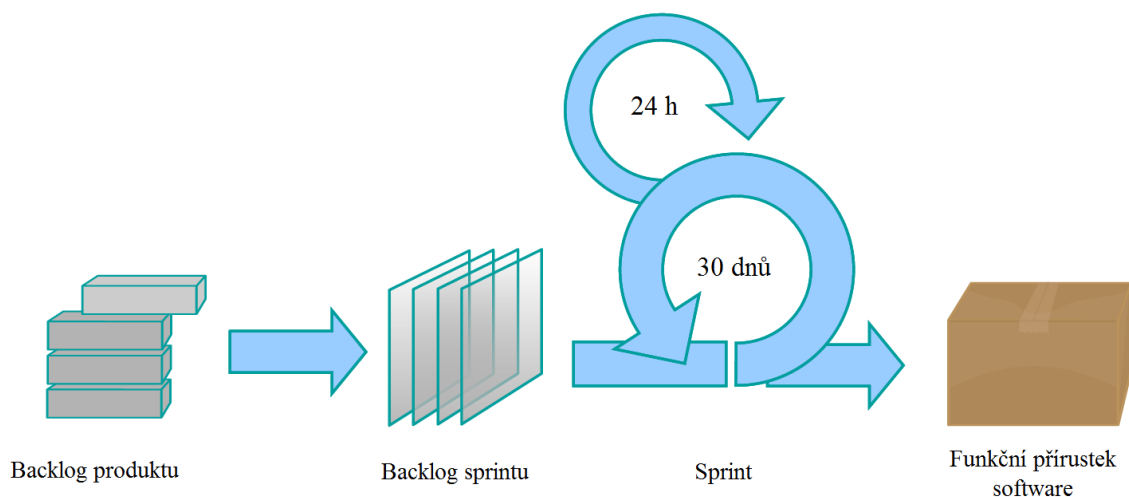
Při vývoji softwarového projektu většinou jednotliví vývojáři nebo jednotlivé vývojářské týmy pracují na samostatných subsystémech, které je poté nutné propojit dohromady. Tento proces se označuje jako integrace. Tradiční metodiky vývoje software, jako je například Vodopádový model, zařazují fáze integrace, testování a kontroly kvality až po dokončení fáze vývoje (viz Obrázek 1). Doba samotného vývoje software je často odhadnuta nesprávně, i když tomu lze částečně předejít. [MCCONNELL, 2006]



Obrázek 1: Schéma vodopádového modelu vývoje software [MALL, 2004]

Odhad doby trvání integrační fáze a fáze kontroly kvality je vzhledem k jejich neurčitosti prakticky nemožný. Fáze s neznámou dobou trvání v závěru projektu, kdy už ostatní fáze mohly překročit odhadované doby trvání, může způsobit překročení plánované doby trvání celého projektu a tedy i rozpočtu.

S řešením problému neurčitosti integrační fáze přichází technika tzv. kontinuální (neboli průběžné) integrace. Fáze integrace, testování a kontroly kvality se rozloží do celého průběhu fáze implementace a jsou prováděny průběžně, vždy po každé změně programového kódu. Kontinuální integrace je často využívána současně s tzv. agilními metodikami vývoje, jako je například Scrum (viz obrázek Obrázek 2), kdy je software vyvíjen v krátkých iteracích.



Obrázek 2: Schéma průběhu vývoje software pomocí metodiky Scrum [LAKEWORKS, 2009]

Kontinuální integrace zavádí proces průběžné kontroly kvality kódu – často prováděné nenáročné úkony, místo kontroly kvality až po dokončení vývoje.¹

[FOWLER, 2006] definici dále upřesňuje:

Kontinuální integrace je technika používaná při vývoji software, kdy členové týmu integrují svůj kód často, obvykle aspoň jednou za den, což vede k mnoha integracím každý den. Každá integrace je ověřena automatickým sestavením (včetně testů), aby byly případné problémy objeveny co nejdříve. Mnoho týmů zjistilo, že tento přístup vede k výraznému snížení problémů při integraci a umožňuje jim rychleji dodávat kvalitnější software.²

Kontinuální integrace byla poprvé popsána³ jako jedna z praktik tzv. Extrémního programování⁴, nicméně je ji samozřejmě možné využít i bez implementace ostatních praktik EP.

1.1 Přehled technik a zásad kontinuální integrace

Kontinuální integrace je souhrnem několika technik, které lze do týmové práce zahrnout samostatně (a tedy postupně). V této kapitole a jejích podkapitolách shrnu jednotlivé techniky podle [FOWLER, 2006] a [DUVALL, 2007].

1.1.1 Jednotné úložiště zdrojových kódů

Každý softwarový projekt se skládá z mnoha souborů různých typů – soubory se zdrojovým kódem, konfigurační soubory, multimediální data (například obrázky, které jsou součástí webu). Při spolupráci více lidí na projektu může být problém zajistit, aby všichni měli všechny soubory v aktuální verzi. Dříve bylo běžné na sdílení projektových souborů používat sdílené síťové disky, ale toto řešení neumožňuje snadno získat starší verzi souboru se zdrojovým kódem.

¹ Volně podle [WIKIPEDIA CONTRIBUTORS, 2011]

² Přeloženo z [FOWLER, 2006]

³ Viz <http://www.extremeprogramming.org/rules/integrateoften.html>

⁴ Zkracováno jako EP - viz [BECK, 2005]

Tento problém řeší nástroje pro správu zdrojového kódu⁵, které kromě úložiště nabízejí i přístup ke starším verzím souborů. Úložiště může být buď centrální (CVCS) anebo distribuované (DVCS). Mezi nejznámější open-source nástroje pracující s centrálním úložištěm patří Subversion⁶ nebo dnes již zastaralé CVS⁷. Z komerčních lze jmenovat Rational Team Concert⁸ od IBM nebo Team Foundation Server⁹ od společnosti Microsoft. Mezi nejznámější open-source nástroje pracující s distribuovaným úložištěm patří Git¹⁰ (je používán pro verzování jádra operačního systému Linux), Mercurial¹¹ nebo Bazaar¹². Z komerčních lze uvést BitKeeper¹³ od společnosti BitMover, Inc. (používaný pro verzování jádra operačního systému Linux v letech 2002-2005¹⁴).

Jakmile tým používá jednotné úložiště zdrojového kódu, odpadají problémy s dohledáváním chybějících souborů. To předpokládá, že do úložiště se ukládá opravdu vše – i konfigurační soubory, databázová schémata, instalační skripty a knihovny třetích stran. [FOWLER, 2006] doporučuje jednoduché pravidlo: Pokud na čistě nainstalovaném počítači provedete checkout¹⁵ z úložiště, tak by mělo být možné provést sestavení aplikace¹⁶. Výjimku tvoří například operační systém, JDK¹⁷ nebo databázový systém. Fowler také doporučuje do úložiště ukládat i další soubory, které nejsou nezbytně nutné k sestavení aplikace, ale mohou ušetřit práci ostatním členům týmu – například konfigurační soubory pro IDE¹⁸.

V úložišti by mělo být uloženo vše, co je potřeba pro sestavení aplikace, ale nic z toho, co vznikne při sestavení – podle Fowlera to většinou znamená, že není snadné provést spolehlivé sestavení aplikace.

1.1.2 Automatizované sestavení aplikace

Převod zdrojových kódů do funkční aplikace je často složitý proces, který zahrnuje kompilaci zdrojových kódů, přesouvání souborů, nahrávání schémat do databáze a další. Vzhledem k tomu, že tento proces lze automatizovat, tak by také automatizovaný měl být. Pokud ho provádí člověk ručně, tak zbytečně ztrácí čas a může se stát, že udělá chybu.

Mezi systémy pro automatizované sestavení aplikace lze zařadit make¹⁹ používaný v operačních systémech Linux a Unix, Ant²⁰ používaný pro projekty psané v jazyce Java nebo Phing²¹ používaný pro projekty psané v jazyce PHP.

⁵ SCM - Source Code Management Tools

⁶ Viz <http://subversion.tigris.org/>

⁷ Concurrent Versions System – viz <http://savannah.nongnu.org/projects/cvs>

⁸ Viz <https://jazz.net/projects/rational-team-concert/>

⁹ Viz <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/team-foundation-server/overview>

¹⁰ Viz <http://git-scm.com/>

¹¹ Viz <http://mercurial.selenic.com/>

¹² Viz <http://bazaar.canonical.com/>

¹³ Viz <http://www.bitkeeper.com/>

¹⁴ Viz <http://kerneltrap.org/node/4966>

¹⁵ Získání kopie všech souborů z repositáře

¹⁶ Tzv. build

¹⁷ Java Development Kit <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

¹⁸ Integrated development environment neboli Vývojové prostředí

¹⁹ Viz <http://www.gnu.org/software/make/>

²⁰ Viz <http://ant.apache.org/>

Častou chybou je, že automatizovaný build nezahrnuje všechny nutné kroky a je nutné manuálně provést některé další úkony po skončení automatizovaného sestavení. Fowler zde doplňuje svoje pravidlo: Pokud na čistě nainstalovaném počítači provedete checkout z úložiště, tak by mělo být možné pomocí jednoho příkazu získat běžící aplikaci.

Vývojová prostředí často obsahují nástroje na provádění sestavení, ale jejich použití je problematické, protože soubory s popisem procesu sestavení jsou většinou proprietární pro dané IDE a není je možné bez něj spustit. Samozřejmě je možné, aby jednotliví vývojáři spouštěli lokální sestavení přímo z IDE, ale vždy je potřeba mít skript pro sestavení, který lze spustit na serveru a je nezávislý na IDE.

1.1.2.1 Upřesnění pro projekty vyvíjené v jazyce PHP

Skripty psané v jazyce PHP se většinou nekompilují předem, ale až při zpracování v interpreteru. Proto skript pro sestavení aplikace nebude, na rozdíl například od projektů psaných v jazyce Java, obsahovat kompilaci, ale jen kontrolu syntaktické správnosti zdrojových kódů.

1.1.3 Testovatelný build

Úspěšné sestavení neznamená, že aplikace bude fungovat správně. Pro odchyčení chyb je vhodné do procesu sestavení zařadit i automatizované testování. Vhodným, nicméně ne nezbytným, přístupem je proto Test Driven Development²², kdy se nejdříve píše test a teprve poté kód aplikace. TDD je jednou z klíčových praktik EP [BECK, 2005].

Nejčastěji používanými testovacími nástroji pro automatizované testování jsou tzv. xUnit²³ frameworky, kde x značí programovací jazyk. Existuje tedy JUnit²⁴ pro Javu, NUnit²⁵ pro .NET, PHPUnit²⁶ pro PHP a další.

Testy jsou většinou sdružovány do sad testů a po jejich proběhnutí je vygenerována zpráva, která informuje o tom, zda některé testy selhaly.

[FOWLER, 2006] upozorňuje na to, že testy neobjeví všechny chyby. Nicméně nedokonalé testy, které jsou spouštěny často, jsou podle něj mnohem lepší než dokonalé testy, které nejsou nikdy napsány.

1.1.4 Každý vývojář ukládá kód do úložiště alespoň jednou za den

Nejsnazším způsobem, jak může dát vývojář vědět ostatním kolegům v týmu o změnách v nějaké části aplikace, je uložení změn do úložiště. Do úložiště je samozřejmě vhodné ukládat jen funkční kód, což nutí vývojáře rozdělit velký úkol na mnoho dílčích podúkolů, které lze řešit samostatně. To, že je kód funkční, ověří vývojář sestavením aplikace a spuštěním automatizovaných testů. Těsně před uložením změn si musí vývojář stáhnout případné změny, které provedli ostatní členové týmu a testy ověřit, že jeho kód stále funguje i s těmito změnami. Teprve poté může kód uložit.

²¹ Viz <http://www.phing.info>

²² TDD neboli vývoj řízený testy [BECK, 2002]

²³ Viz <http://www.martinfowler.com/bliki/Xunit.html>

²⁴ Viz <http://www.junit.org/>

²⁵ Viz <http://www.nunit.org/>

²⁶ Viz <http://www.phpunit.de/>

Pokud by při stažení nejnovějších změn z úložiště narazil na nějaký konflikt, tak většinou nebude problém ho vyřešit, protože během několika hodin od minulého stažení nemohlo změn být mnoho.

1.1.5 Po každém commitu proběhne sestavení aplikace na integračním stroji

Některé společnosti provádějí sestavení aplikace v naplánovaných intervalech (např. každou noc), ale to není úplně ideální, protože cílem kontinuální integrace je odhalit chyby, co nejdříve je to možné. Pokud sestavení probíhá jednou za 24 hodin, může se stát, že chyby v systému zůstanou dlouhou dobu (a budou působit problémy ostatním členům týmu). Proto je mnohem lepší provádět sestavení po každé změně v úložišti. To může probíhat buď manuálně, nebo automaticky.

Při manuálním sestavení si vývojář po uložení změn do úložiště sedne k speciálnímu počítači, který je vyhrazen jen na integraci²⁷. Stáhne si aktuální verzi z úložiště, spustí sestavení, a pokud to proběhne správně, tak je jeho úkol hotový.

Druhou variantou je využití integračního serveru, který sleduje úložiště (v pravidelných intervalech se dotazuje úložiště – tzv. polling), a po každé zjištěné změně spustí sestavení a autorovi zašle e-mail s výsledkem.

Integrační server není nezbytně nutný, ale jeho použití je pohodlnější. Jeho další výhodou je, že pokud součástí sestavení je i statická analýza zdrojového kódu²⁸, je možné na výstupu integračního serveru sledovat dlouhodobé trendy v kvalitě kódu.

1.1.6 Sestavení musí být rychlé

Cílem kontinuální integrace je rychle poskytnout vývojářům zpětnou vazbu. Proto by sestavení na integračním serveru mělo být co nejrychlejší. Kent Beck [BECK, 2005] ve své metodice extrémního programování doporučuje, aby sestavení netrvalo déle než 10 minut. Pokud by trval déle, výrazně se zvyšuje šance, že ho vývojáři nebudou spouštět lokálně.

Často je sestavení zpomalováno testy, které pracují s externími zdroji, jako je databáze nebo služby. V takovém případě je možné sestavení rozdělit na dvě části – primární (spouští se po uložení změn do úložiště), sekundární (spouští se po úspěšném doběhnutí primárního). Výhoda tohoto přístupu je v tom, že vývojáři stačí, když počká na výsledek primárního sestavení. Pokud by sekundární sestavení odhalilo nějaké chyby, tak to není tak velký problém, protože testy důležité funkčnosti jsou zahrnuté v primárním sestavení.

Pokud by sekundární sestavení trvalo příliš dlouho, je možné využít například více serverů, na kterých testy poběží (integrační servery často funkčnost pro distribuované sestavení a testování obsahují). V tu chvíli musí být testy schopny běžet izolovaně. Hlubší rozbor možností paralelního spouštění testů na více strojích nicméně přesahuje rozsah této práce.

²⁷ Je důležité, aby integraci neprováděl na svém počítači, protože například pokud zapomene uložit nějaký soubor do úložiště, sestavení by prošlo (soubor má k dispozici), ale u ostatních členů týmu by selhalo (neměli by k dispozici chybějící soubor).

²⁸ Viz kapitola 1.3

1.1.7 Testování by se mělo provádět v prostředí co nejpodobnějším produkčnímu

Cílem testování je odhalit problémy, které by mohly nastat v produkčním prostředí. Proto je vhodné, aby testovací prostředí bylo co nejvíce podobné produkčnímu. Pokud se budou lišit, tak se může stát, že se některé chyby nepodaří odhalit nebo naopak bude docházet k planým poplachům.

Například pokud by vývojáři pracovali na počítačích s operačním systémem Microsoft Windows, integrační server by také používal Windows, ale na produkčním serveru by aplikace běžela na operačním systému Linux, neprojevily by se problémy s rozdílnými znaky pro konce řádků nebo oddělovači adresářů v cestě.

Proto by integrační server měl běžet na stejném operačním systému, používat stejné verze aplikací, běhových prostředí i knihoven.

1.1.8 Každý má snadný přístup k informacím

Při aplikaci kontinuální integrace je důležitá komunikace mezi členy týmu. Jednou z nejdůležitějších informací je stav posledního sestavení, protože pokud to bylo neúspěšné (což značí, že kód obsahuje chybu nebo chyby), tak není vhodné, aby si vývojář aktualizoval svoji kopii zdrojových kódů z úložiště na tuto verzi.

Pokud se využívá manuální integrace na integračním stroji, tak by na jeho monitoru mělo vždy být vidět stav posledního sestavení.

Automatické nástroje pro kontinuální integraci umožňují snadnější přístup k této informaci. Členové geograficky distribuovaných vývojových týmů mohou dostávat tuto informaci pomocí e-mailu, SMS, aplikace běžící na počítači vývojáře nebo pomocí rozšíření do IDE. Týmy často experimentují s vizuálními pomůckami, které ukazují stav posledního sestavení. Některé používají například červenou a zelenou lávovou lampu²⁹, LED diody řízené Arduinem³⁰ nebo detailnější tabuli s přehledem stavu jednotlivých sestavení³¹.

1.1.9 Automatizované nasazení

Využívání kontinuální integrace umožňuje častěji získávat stabilní sestavení aplikace, čehož lze využít pro zvýšení frekvence vydávání a nasazování nových verzí. Tato technika se označuje jako Continuous Delivery. Pokud by systém automaticky nasadil do provozu každé úspěšné sestavení, které projde automatizovanými testy, označovali bychom to jako Continuous Deployment.

Podrobný rozbor tématu však přesahuje rozsah této práce. Více informací lze nalézt například v [HUMBLE, 2010], případně v [ZIKMUND, 2011] se zaměřením jen na PHP.

1.2 Typický průběh práce vývojáře v prostředí kontinuální integrace

1. Vývojář si z úložiště stáhne nejnovější verzi zdrojových kódů
2. Provede potřebné úpravy - většinou trvají jen několik hodin – všechny úkoly jsou rozloženy na dílčí podúkoly
3. Spustí soukromé sestavení aplikace (včetně testů) a ověří, že vše funguje

²⁹ Viz <http://www.pragmaticautomation.com/cgi-bin/pragauto.cgi/Monitor/Devices/BubbleBubbleBuildsInTrouble.rdoc>

³⁰ <http://dattein.com/blog/arduino-build-light/>

³¹ <http://fabiopereira.me/blog/2009/12/15/build-dashboard-radiator-your-build-light-2/>

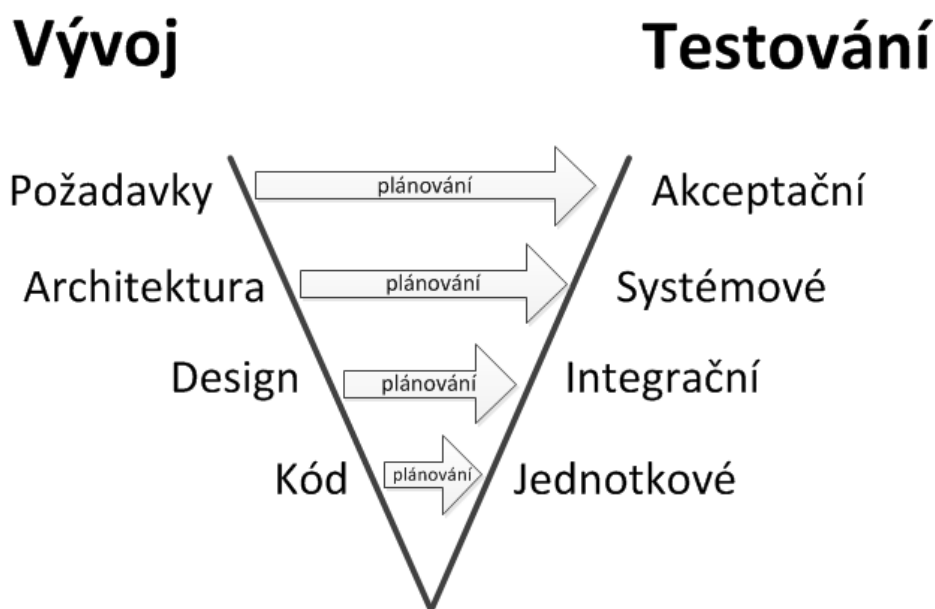
4. Stáhne si z úložiště nejnovější změny zdrojových kódů od ostatních členů týmu
5. Pomocí soukromého sestavení aplikace ověří, že vše stále funguje
6. Uloží své změny do úložiště
7. Integrační server provede sestavení aplikace a spuštění testů a zašle informační e-mail
8. Pokud nedojde k žádným chybám, vývojář si úkol může označit jako dokončený

1.3 Testování software

Jak již naznačuje kapitola 1.1.3, nedílnou součástí kontinuální integrace by mělo být testování software.

Testování je neustálý proces vývoje, využívání a udržování testovacích nástrojů za účelem měření a zlepšení kvality testovaného software. [CRAG, 2002]

Testování software lze rozdělit do několika úrovní, které lze znázornit pomocí tzv. „V“ Modelu (viz Obrázek 3). V dolní části obrácené pyramidy převládá automatizované testování, v horní části naopak manuální.



Obrázek 3: "V" model testování software [CRAG, 2002]

1.3.1 Manuální testování

Při manuálním testování procházejí testeři aplikaci podle instrukcí v testovacích scénářích a ověřují, zda chování a výsledky aplikace odpovídají očekávaným. Manuálním testováním webových aplikací se tato práce nezabývá.

Nicméně manuální testování může využívat výhod kontinuální integrace například tak, že každé úspěšné sestavení, které projde automatizovanými testy, bude automaticky zpřístupněno k manuálnímu testování na testovacím serveru.

1.3.2 Automatizované testování

Automatizované testování funkčnosti software většinou probíhá tak, že dochází k automatizovanému porovnávání očekávaných výstupů aplikace pro dané vstupy s aktuálními výstupy. Může probíhat na úrovni tříd (tzv. jednotkové testování), na úrovni komponent (integrační testování) nebo na úrovni celého systému (systémové).

1.3.3 Metriky – výsledky automatizovaných testů, pokrytí kódu testy

V kontextu kontinuální integrace jsou automatizované testy velmi užitečné. Díky tomu, že sestavení projektu (včetně spuštění automatizovaných testů) probíhá po každé změně zdrojových kódů, je v případě selhávajících testů snadné odhalit, která změna to způsobila.

Zjištění, zda nějaké testy selhávají, je většinou rozhodující pro další průběh sestavení. V takovém případě nemá smysl provádět například kontrolu dodržování standardů pro psaní kódu. Lze zjednodušeně říci, že pokud se některá část aplikace nechová podle očekávání, tak už informace, že některá místa v kódu nejsou elegantně formátovaná, není tak užitečná.

1.3.3.1 Míra pokrytí kódu testy

V souvislosti s automatizovaným testováním se často získává metrika Míra pokrytí kódu testy (code coverage). Tato metrika ukazuje, které řádky kódu byly provedeny během testování. Je možné si to snadno ukázat na příkladu Kód 1. Pokud funkce `checkSomething()` bude z testů zavolána jen s parametrem `true`, tak bude výsledné pokrytí kódu testy 50%.

Kód 1: Míra pokrytí kódu testy

```
function checkSomething($condition) {  
    if ($condition) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Je důležité si uvědomit, že i když je řádek během testování zavolán, tak to neznamená, že nemůže obsahovat chyby. Dobře to ilustruje [WOOD, 2008], kdy v kódu, který dosahuje 100% pokrytí testy, postupně odhalí několik zásadních chyb. Dále mluví o tom, že není nutné se hnát za co nejvyšším pokrytím kódu testy:

Nicméně, je důležité si pamatovat, že cílem není dosáhnout 100% pokrytí, ale mít úplné a komplexní jednotkové testy. [WOOD, 2008]

Metriku lze ovšem využít na zjištění, které části aplikace jsou testovány výrazně méně než ostatní.

1.4 Statická analýza kódu, kontrola kvality

Statickou analýzou zdrojového kódu označujeme proces získávání metrik kvality kódu. V kontextu kontinuální integrace získáváme vybrané metriky automatizovaně a při využití vhodných nástrojů můžeme sledovat jejich dlouhodobé trendy. Platí zde známé:

Co se nedá měřit, to se nedá řídit. [DEMARCO, 1982]

Sledování a vyhodnocování metrik zdrojového kódu je důležité, protože mohou pomoci odhalit tzv. zatuchlý kód (code smell), který může způsobovat komplikace v pozdějších fázích softwarového projektu. O tom, jak psát kvalitní a čistý programový kód, detailně pojednává například [MARTIN, 2008].

Mezi často získávané metriky patří:

- Počet řádků kódu, programových struktur (viz kapitola 1.4.1)
- Složitost (komplexita) kódu (viz kapitola 1.4.2)
- Dodržování standardů pro psaní kódu (viz kapitola 1.4.3)
- Duplicitní kód (viz kapitola 1.4.4)

1.4.1 Počet řádků kódu, programových struktur

Zjišťování počtu řádků kódu a jednotlivých programových struktur (třídy, metody, rozhraní, ...) může být v rámci kontinuální integrace zajímavé pro sledování trendů. Nicméně je důležité, aby si řídicí pracovníci uvědomili, že produktivitu vývojáře není možné měřit podle přidáných řádků kódu. Dobře to vysvětluje [MCCONNELL, 2008]:

Pokud je produktivita měřena počtem řádků kódu, znamená to, že vývojář, který na vyřešení problému použije 10x více kódu je produktivnější než vývojář, který napíše desetinu kódu. To zjevně není v pořádku. [MCCONNELL, 2008]

Případně to lze ukázat na jiném příkladu. Vývojář provede refaktoring a vyčlení duplicitní kód do samostatné třídy, čímž sníží celkový počet řádků kódu (kód bude do budoucna lépe udržovatelný). Pokud by produktivita vývojáře byla měřena jen podle řádků kódu, tak i přes nezpochybnitelný přínos by jeho produktivita byla dokonce záporná.

Data jsou spíše podkladem pro měření komplexity kódu (viz následující kapitola).

1.4.2 Složitost (komplexita) kódu

V průběhu vývoje softwarového projektu je vhodné sledovat složitost jednotlivých částí zdrojového kódu. Vysoká složitost kódu může znamenat nižší přehlednost a často znesnadňuje testování kódu. Pro dlouhodobou udržitelnost rozvoje softwarového projektu je vhodné složitost sledovat a případně části kódu refaktorovat.

1.4.2.1 Cyklomatická složitost

Pojem cyklomatická složitost (označována jako CCN), tak jak ji popsal [MCCABE 1976], vychází z teorie grafů. Zjednodušeně lze říci, že jde o celkový počet míst, kde se kód větví (typicky podmínky nebo cykly).

1.4.2.2 Rozšířená cyklomatická složitost

Rozšířená cyklomatická složitost (CCN2) započítává jako další rozvětvení i boolovské výrazy v podmínkách. Některé nástroje tuto uvádějí jako běžnou cyklomatickou složitost.

1.4.2.3 Rizika vysoké cyklomatické složitosti

Jak již bylo zmíněno výše, vysoká cyklomatická složitost zhoršuje testovatelnost kódu a zároveň může zvyšovat riziko chyb (viz Tabulka 1). S rostoucí cyklomatickou komplexitou se zvyšuje riziko chybné opravy – při opravě jedné chyby dojde k vnesení jiné (viz Tabulka 2).

Tabulka 1: Cyklomatická složitost a riziko spolehlivosti [MCCABE JR., 2008]

Cyklomatická složitost	Riziko spolehlivosti
1 – 10	Jednoduchá metoda, malé riziko
11- 20	Složitější, střední riziko
21 – 50	Složitá, vysoké riziko
>50	Netestovatelné, VELMI VYSOKÉ RIZIKO

Tabulka 2: Cyklomatická složitost a pravděpodobnost chybné opravy [MCCABE JR., 2008]

Cyklomatická složitost	Pravděpodobnost chybné opravy
1 – 10	5%
20 –30	20%
> 50	40%
Blížící se 100	60%

1.4.2.4 NPath složitost

NPath složitost, tak jak ji definoval [NEJMEH 1988], označuje počet možných průchodů blokem zdrojového kódu. Autor doporučuje hodnotu NPath složitosti sledovat a pokud překročí 200, tak konkrétní úsek zdrojového kódu prověřit a případně ho refaktorovat.

1.4.3 Standardy pro psaní kódu

Na softwarových projektech většinou spolupracuje více vývojářů, z nichž každý může být zvyklý zapisovat zdrojový kód jiným způsobem. Pokud by i na společném projektu zapisovali kód různými styly, měli by ostatní členové týmu problémy mu snadno porozumět.

Tento problém řeší vytvoření standardů pro psaní kódu, které definují, jak má být zdrojový kód formátován. Většinou jde o dokument přístupný všem členům týmu. Skládá se z několika částí:

- pravidla formátování samotného souboru (odsazování kódu, maximální délky řádků, znak ukončení řádku)
- jmenné konvence – jak pojmenovávat třídy, rozhraní, funkce, metody, proměnné a konstanty, ...
- styl vlastního psaní – jak zapisovat jednotlivé programové struktury, kam umístit závorky uvozující jednotlivé části (třídy, metody, ...), jak zapisovat řídicí struktury (podmínky, cykly, ...)
- dokumentační standard

Výsledkem je poté konzistentní, snadno čitelný a pochopitelný kód. Není možné na první pohled rozpoznat, který vývojář psal danou část kódu. To je důležité zejména při uplatňování jedné z praktik extrémního programování, tzv. společného vlastnictví kódu (Shared code, viz [BECK, 2005])

Samozřejmě není nutné pro každý projekt vytvářet nové standardy, což ani není žádoucí. Mnohem vhodnější je zvolit již existující a zavedený standard pro formátování (například Code Conventions for the Java TM Programming Language³², PEAR Coding Standards³³ nebo Zend Framework Coding Standard for PHP³⁴). K jejich výhodám patří, že vývojáři už na ně budou pravděpodobně přivyklí a také existující podpora v různých nástrojích (jak editorech, tak nástrojích pro analýzu kódu).

O tom, proč je prakticky nutností dodržovat nějaké standardy pro psaní kódu, dále pojednává například [MYTTON, 2004] nebo [MARTIN, 2007].

1.4.4 Duplicitní kód

Duplicitní kód se v softwarových projektech může objevit například tak, že vývojář potřebuje metodu s velmi podobnou funkcionalitou, která už je obsažena v existující metodě, takže ji jen zkopíruje a drobně upraví. Kvůli tomu může v budoucnu dojít k problémům, kdy je upraven jen jeden výskyt tohoto kódu a ne všechny jeho kopie.

Řešením v kontinuuální integraci je nasazení nástroje, který tato místa vyhledává.

³² Viz <http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>

³³ Viz <http://pear.php.net/manual/en/standards.php>

³⁴ Viz <http://framework.zend.com/manual/en/coding-standard.html>

2 Výběr vhodného integračního serveru a jeho konfigurace

Jak již bylo zmíněno dříve, kontinuální integraci je možné provádět i bez využití integračního serveru, nicméně nenapadá mě žádný důvod proč. Mnoho serverů pro kontinuální integraci je k dispozici zdarma a jejich využití přináší další automatizaci úkonů, které by bylo jinak nutné dělat ručně. Zároveň zaručují, že sestavení je spouštěno vždy stejně.

2.1 Přehled trhu serverů pro kontinuální integraci

Na trhu jsou k dispozici různé servery pro kontinuální integraci. Ten nejstarší, CruiseControl³⁵, byl vyvinut ve společnosti ThoughtWorks, pro kterou pracuje Martin Fowler, autor známého článku [FOWLER, 2006]. Existuje pro něj rozšíření phpUnderControl³⁶, které usnadňuje prvotní nastavení nástrojů pro kontinuální integraci v PHP, nicméně již není aktivně vyvíjeno, takže jeho nasazení bych spíše nedoporučoval. Velmi známým integračním serverem je Jenkins, který je detailně popsán v kapitole 2.2. Kromě toho existují další méně známé integrační servery, například Xinc³⁷, který je napsaný v jazyce PHP.

Z komerčních serverů pro kontinuální integraci bych zde uvedl Bamboo³⁸ od společnosti Atlassian³⁹. Výhodou tohoto serveru je snadné propojení s dalšími nástroji pro projektové řízení od této společnosti. Dalším komerčním serverem je uBuild⁴⁰, který je nástupcem známého serveru AnthillPro. V neposlední řadě bych uvedl integrační server zaměřený především na jazyk Java a .NET, TeamCity⁴¹, jehož tvůrcem je původně česká společnost JetBrains⁴². Často používaný je také Team Foundation Server⁴³ od společnosti Microsoft. Nejedná se jen o integrační server, ale spíše platformu zastřešující celý proces vývoje software. Je zaměřený především na vývoj ve frameworku .NET a integraci s vývojovým prostředím Visual Studio.

2.1.1 Příbuzný software

S integračními servery úzce souvisí i další software, který je může doplňovat (ale funguje nezávisle na nich). Zařadil bych sem Sismo⁴⁴, což je Continuous Testing Server – zaměřuje se jen na spouštění testů. Díky tomu ho lze využít jako komplement k integračnímu serveru pro rychlé spouštění jednotkových testů lokálně, po uložení změn do úložiště ve verzovacím systému GIT. Podobně funguje i Travis CI⁴⁵, jehož cílem je vytvořit distribuovaný systém pro spouštění testů open-source software.

³⁵ Viz <http://cruisecontrol.sourceforge.net/>

³⁶ Viz <http://phpundercontrol.org/>

³⁷ Viz <http://code.google.com/p/xinc/>

³⁸ Viz <http://www.atlassian.com/software/bamboo/overview>

³⁹ Viz <http://www.atlassian.com/>

⁴⁰ Viz <http://www.urbancode.com/html/products/build/default.html>

⁴¹ Viz <http://www.jetbrains.com/teamcity/>

⁴² Viz <http://www.jetbrains.com/>

⁴³ Viz <http://msdn.microsoft.com/en-us/vstudio/ff637362>

⁴⁴ Viz <http://sismo.sensiolabs.org/>

⁴⁵ Viz <http://travis-ci.org/>

Sonar⁴⁶ je open-source nástroj na řízení kvality zdrojových kódů. V základu se zaměřuje na jazyk Java, ale pomocí open-source nebo i komerčních rozšíření je do něj možné doplnit podporu pro další jazyky (C, C#, PHP, ...). Je možné ho využít ve spojení s různými servery pro kontinuální integraci jako je Hudson nebo Jenkins⁴⁷.

2.2 Jenkins / Hudson

Velmi známým integračním serverem je Jenkins (resp. Hudson). Někdy dochází k zaměňování těchto názvů, proto je důležité si je upřesnit. Jeden ze zaměstnanců Sun Microsystems v roce 2006 vyvinul integrační server Hudson a uvolnil ho pod svobodnou licenci MIT. V následujících letech se vývoje účastnili i další vývojáři. V lednu 2010 společnost Oracle koupila⁴⁸ společnost Sun Microsystems, čímž mj. získala právo na využívání názvu Hudson, který si později registrovala jako ochrannou známku. Oracle se během sjednocování infrastruktury s ostatními open-source projekty, které zastřešují, dostal do sporu⁴⁹ s komunitou vývojářů Hudsonu a ti se rozhodli zbavit závislosti na společnosti Oracle⁵⁰ a začali Hudson dále vyvíjet pod názvem Jenkins⁵¹.

Společnost Oracle nadále vyvíjí Hudson, ale komunita se spolu s hlavními vývojáři přesunula k Jenkinsu, takže rozvoj Hudsonu se zpomalil. Oba projekty jsou v současné době hostovány na serveru GitHub.com, takže je možné snadno provést srovnání rychlosti vývoje a zájmu vývojářů o ně:

Tabulka 3: Aktivity u projektů Hudson a Jenkins na serveru GitHub.com (k 25. 2. 2012)

Projekt	URL	Sledování	Forky	Pull Requesty	Commity v roce 2012
Hudson	https://github.com/hudson/hudson/	159	34	0	6
Jenkins	https://github.com/jenkinsci/jenkins/	1112	398	16	257

Z těchto hodnot je vidět, že Jenkins je aktivně vyvíjen a zároveň se těší velkému zájmu uživatelů, což ho dělá vhodnější volbou než Hudson. Je to zajímavá ukázka toho, že vývoj software komunitou dobrovolníků může postupovat dopředu rychleji než vývoj software, který zaštiťuje velká společnost.

Při výuce vývoje aplikací v Javě na VŠE se dříve využíval integrační server Hudson, nicméně dnes (25. 2. 2012) již to je Jenkins⁵², což potvrzuje moji domněnku, že Jenkins je vhodnější volbou než Hudson. I vzhledem k tomu, že většina zdrojů, které se kontinuální integrací v prostředí PHP zabývají, zvolila jako integrační server Jenkins, jsem nenašel důvod, proč vybrat jiný.

⁴⁶ Viz <http://www.sonarsource.org/>

⁴⁷ Viz <http://docs.codehaus.org/display/SONAR/Hudson+and+Jenkins+Plugin>

⁴⁸ Viz http://news.cnet.com/8301-30685_3-20000019-264.html

⁴⁹ Viz <http://jenkins-ci.org/content/whos-driving-thing>

⁵⁰ Viz <http://jenkins-ci.org/content/hudsons-future>

⁵¹ Viz <http://jenkins-ci.org/content/jenkins>, <http://jenkins-ci.org/>

⁵² Viz <http://kitscm.vse.cz/>

2.3 Instalace serveru Jenkins

Pro ukázkovou instalaci integračního serveru jsem zvolil aktuální verzi serveru Jenkins (k 8. 3. 2012 1.454). Ty vycházejí velmi často (většinou každý týden), což není vhodné pro instalace, které musí být stabilní. Nasazení nové verze s sebou přináší nutnost otestovat, zda skript pro sestavení a veškerá rozšíření fungují jako dříve. Jenkins proto zavedl tzv. LTS verze (Long-Term Support)⁵³, které jsou založeny na některé starší verzi, která se osvědčila a jsou do nich zpětně zahrnovány jen opravy chyb a ne nová funkcionalita.

Jenkins je k dispozici jak ve formátu Java Web Archive (.war), tak i ve formě instalačních balíčků pro operační systém Windows, a běžné linuxové distribuce. Pro některé linuxové distribuce jsou k dispozici i úložiště balíčků⁵⁴, takže je možné Jenkins snadno aktualizovat na novou verzi.

Instalaci začneme stažením balíčku ve formátu .war, který umístíme do vybrané složky (například I:\BP-jenkins/). Jenkins si ve výchozím nastavení ukládá nastavení a soubory do domovského adresáře aktuálně přihlášeného uživatele. Vhodnější je data ukládat do nějakého jiného adresáře, což lze nastavit do hodnoty proměnné prostředí JENKINS_HOME. Samotný Jenkins lze poté spustit jako běžnou aplikaci v Javě pomocí příkazu `java -jar jenkins.war`.

Než zadávat tyto příkazy při každém spouštění Jenkinsu ručně, je vhodnější vytvořit spouštěcí dávkový soubor. Pojmenovat ho lze například jako `jenkins-start.cmd` a obsah bude tento:

```
set JENKINS_HOME=i:\BP-jenkins\data\  
java -jar jenkins.war
```

Po jeho spuštění začne startovat Jenkins. Z výpisu je zřejmé, že se použil námi určený adresář:

```
Jenkins home directory: i:\BP-jenkins\data found at:  
EnvVars.masterEnvVars.get("JENKINS_HOME")
```

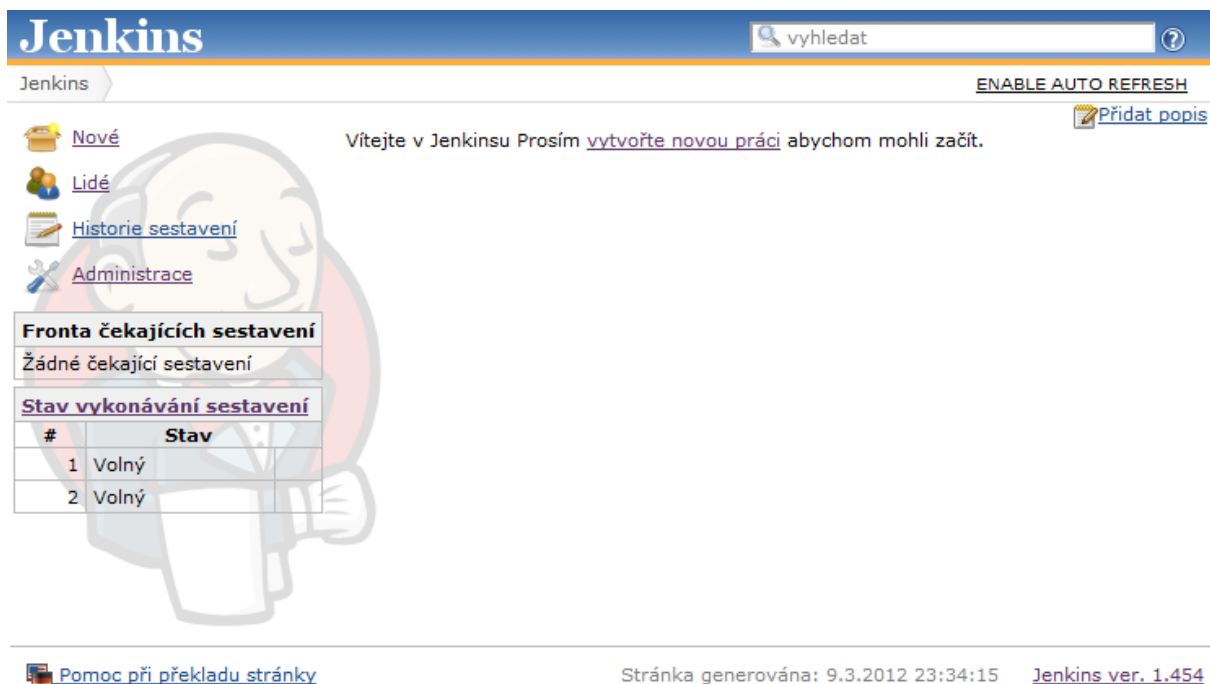
Posledním řádkem výpisu by měla být informace oznamující, že spuštění proběhlo v pořádku:

```
INFO: Jenkins is fully up and running
```

Ověřit to lze otevřením URL `http://localhost:8080/` ve webovém prohlížeči. Měla by se zobrazit hlavní stránka serveru Jenkins (viz Obrázek 4). Tím je dokončena základní instalace a je možné začít vytvářet projekt.

⁵³ Viz <https://wiki.jenkins-ci.org/display/JENKINS/LTS+Release+Line>

⁵⁴ package repository



Obrázek 4: Hlavní stránka Jenkinsu po prvním spuštění (zdroj: autor)

2.4 Možnosti konfigurace serveru Jenkins

Jedna z možností, jak ovládat a konfigurovat Jenkins, je přes webové rozhraní, jak je naznačeno výše. Další možností je pomocí příkazové řádky. Výhodou je možnost snadno spustit více příkazů za sebou, bez nutnosti manuálního procházení webových stránek. Nevýhodou je nemožnost ovládat vše, co je možné z webového rozhraní. Potřebnou verzi klienta lze snadno získat přímo ze serveru Jenkins pomocí příkazu:

```
wget http://localhost:8080/jnlpJars/jenkins-cli.jar
```

Bylo možné si všimnout, že se Jenkins automaticky načetl v jazyce, který je nastavený jako výchozí ve webovém prohlížeči. S největší pravděpodobností to tedy byla čeština. Český překlad Jenkinsu není kompletní, což by znepříjemňovalo práci a zároveň mohlo být v některých situacích matoucí. Lze to vyřešit nastavením webového prohlížeče tak, aby požadoval webové stránky v angličtině, což by ale museli provést všichni, kteří budou Jenkins využívat. Vhodnějším řešením je instalace rozšíření Locale⁵⁵, které umožňuje nastavit jazyk napevno, bez ohledu na konfiguraci webového prohlížeče. Rozšíření lze nainstalovat takto:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin locale
```

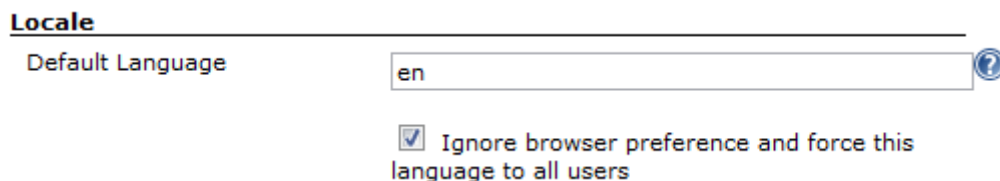
Server je po instalaci rozšíření vždy nutné restartovat, což lze provést příkazem:

```
java -jar jenkins-cli.jar -s http://localhost:8080 safe-restart
```

(Nicméně pokud je server spuštěný pomocí `java -jar jenkins.war`, tak restart nelze provést a je nutné server ukončit a poté opět spustit.)

⁵⁵ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Locale+Plugin>

Po restartu už je rozšíření aktivní, takže je možné přejít do nastavení systému⁵⁶ a sekci „Locale“ upravit takto:



Locale

Default Language ?

☒ Ignore browser preference and force this language to all users

Obrázek 5: Jenkins - nastavení anglického prostředí (zdroj: autor)

Z historických důvodů⁵⁷ Hudson a Jenkins používají pro označení úspěšných sestavení modrou barvu (resp. modré kuličky), což nemusí být na první pohled zřejmé (uživatelé spíše čekají zelenou barvu). Lze to vyřešit instalací rozšíření Green Balls⁵⁸.

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin greenballs
```

2.4.1 Zabezpečení serveru

Integrační server Jenkins není ve výchozím nastavení nijak zabezpečený. Je proto nutné na to před zpřístupněním nezapomenout. Je možné vytvořit databázi uživatelů⁵⁹ přímo na serveru, skrýt integrační server za Apache server⁶⁰ (a využít jeho nastavení) nebo využít některé z řady rozšíření⁶¹, které umožňují autentizaci proti dalším službám.

Pro případ, že by se během nastavování zabezpečení něco nepodařilo nastavit podle očekávání a došlo by ke ztrátě přístupu k Jenkins serveru, je k dispozici návod⁶², jak zabezpečení vypnout.

⁵⁶ <http://localhost:8080/configure>

⁵⁷ Viz <http://jenkins-ci.org/content/why-does-jenkins-have-blue-balls>

⁵⁸ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Green+Balls>

⁵⁹ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Standard+Security+Setup>

⁶⁰ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Apache+frontend+for+security>

⁶¹ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Plugins#Plugins-Authenticationandusermanagement>

⁶² Viz <https://wiki.jenkins-ci.org/display/JENKINS/Disable+security>

3 Přehled nástrojů pro kontinuální integraci v PHP

Nástroje použitelné pro kontinuální integraci projektů v jazyce PHP lze rozdělit do tří velkých skupin:

- nástroje na kontrolu kvality zdrojových kódů pomocí statické analýzy
- nástroje na automatizované testování
- podpůrné nástroje pro instalaci, automatizaci, tvorbu dokumentace, apod.

3.1 PEAR

PEAR⁶³ neboli PHP Extension and Application Repository je systém pro distribuci a instalaci balíčků komponent pro jazyk PHP. Tento balíčkovací systém je využit v dalších kapitolách pro instalaci jednotlivých nástrojů. Detailní popis instalace samotného systému PEAR přesahuje rozsah této práce. Instalační instrukce pro jednotlivé operační systémy lze nalézt v oficiální dokumentaci⁶⁴. Pro správné pochopení použití PEAR je nutné definovat základní pojmy. Jsou to tyto: balíček a kanál.

Balíček lze popsat jako množinu souborů a složek, které tvoří logický celek (komponentu). Součástí balíčku jsou zároveň další metainformace, jako je jeho verze a závislosti na dalších balíčcích.

Kanál je běžná webová stránka, která navíc obsahuje metainformace o balíčcích, které jsou tam k dispozici (informace jsou umístěny v souboru `channel.xml`).

PEAR po instalaci obsahuje jen oficiální kanál `pear.php.net`. Pro instalaci balíčků z jiných umístění je nutné nejprve další kanály inicializovat. To lze provést pomocí příkazu `pear channel-discover`.

```
> pear channel-discover pear.phpunit.de
Adding Channel "pear.phpunit.de" succeeded
Discovery of channel "pear.phpunit.de" succeeded
```

Poté už je možné instalovat jednotlivé balíčky i z tohoto kanálu. V následujícím příkladu je `phpunit` označení pro PEAR kanál a `phpcpd` je konkrétní balíček:

```
> pear install phpunit/phpcpd
Starting to download phpcpd-1.3.5.tgz (8,746 bytes)
.....done: 8,746 bytes
install ok: channel://pear.phpunit.de/phpcpd-1.3.5
```

Tento základní popis funkčnosti by měl být dostatečný pro využití PEARu pro instalaci balíčků potřebných pro zavedení kontinuální integrace. Detailní informace o funkčnosti nástroje PEAR lze nalézt v oficiální dokumentaci⁶⁵.

⁶³ Viz <http://pear.php.net/>

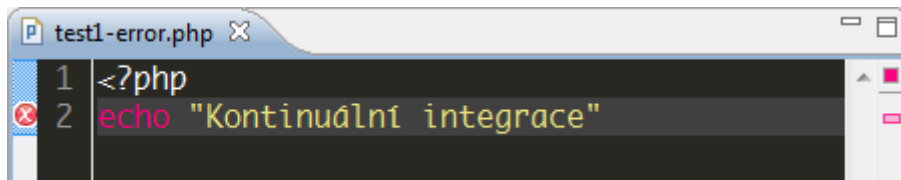
⁶⁴ Viz <http://pear.php.net/manual/en/installation.php>

⁶⁵ Viz <http://pear.php.net/manual/en/>

3.2 PHP Lint

Kontrolu kvality kódu nemá smysl provádět dříve, než si jsme jisti, že napsaný kód je syntakticky v pořádku (to, že kód nelze zkompileovat je větší problém než nedodržování stanovených standardů pro psaní kódu). Zároveň by kvůli tomu mohly nástroje pro kontrolu kvality vracet nepřesné výsledky.

Některá moderní IDE jako Zend Studio⁶⁶, Eclipse PDT⁶⁷ nebo Netbeans⁶⁸ už sice kontrolu syntaktických chyb obsahují (viz Obrázek 6), ale slouží spíše jako upozornění pro vývojáře. Průběh integračního procesu nemůže spoléhat na to, že si vývojář chyby všimne a opraví ji.



Obrázek 6: Kontrola syntaktických chyb v editoru Zend Studio 8 (zdroj: autor)

Samotné PHP umožňuje provést kontrolu syntaxe PHP souboru pomocí přepínače '-l' (lint):

```
> php -help
-l          Syntax check only (lint)
```

Pokud je kontrola spuštěna na skriptu test1-error.php (viz Kód 2), který obsahuje syntaktickou chybu, vrátí PHP chybový výstup (viz Kód 3).

Kód 2: Soubor test1-error.php

```
<?php
echo "Kontinuální integrace" //chybí středník
```

Kód 3: Chybový výstup z PHP Lint

```
> php -l test1-error.php

Parse error: syntax error, unexpected $end, expecting ',' or ';' in test1-error.php on line 2
Errors parsing test1-error.php
```

Pokud je soubor syntakticky v pořádku (viz Kód 4), vrátí PHP Lint výstup (Kód 5)

Kód 4: Zdrojový kód souboru test2-ok.php

```
<?php
echo "Kontinuální integrace";
```

Kód 5: Výstup z PHP Lint, když je kód v pořádku

```
> php -l test2-ok.php
No syntax errors detected in test2-ok.php
```

⁶⁶ Viz <http://www.zend.com/en/products/studio/>

⁶⁷ Viz <http://eclipse.org/pdt/>

⁶⁸ Viz <http://netbeans.org/features/php/>

3.2.1 Kontrola dopředné kompatibility

PHP Lint lze také použít pro kontrolu, zda zdrojové kódy budou zkompileovatelné i v další verzi PHP. Problém může nastat například, pokud je jako identifikátor použité slovo, které je v další verzi zařazeno mezi klíčová slova.

Při provedení kontroly pomocí PHP Lint (verze PHP 5.3) na souboru (viz Kód 6), je vše v pořádku (Kód 7).

Kód 6: Soubor test3-trait.php

```
<?php
class Trait {}
```

Kód 7: Výstup z PHP Lint (verze PHP 5.3) na souboru test3-trait.php

```
>php -l test3-trait.php
No syntax errors detected in test3-trait.php
```

Pokud je ovšem použito PHP ve verzi 5.4.0alpha2, kde trait je klíčovým slovem, vrátí chybový výstup (viz Kód 8)

Kód 8: Chybový výstup z PHP Lint (verze PHP 5.4.0alpha2) na souboru test3-trait.php

```
> php -l test3-trait.php

PHP Parse error:  syntax error, unexpected 'Trait' (T_TRAIT), expecting identifier
(T_STRING) in test3-trait.php on line 2
Parse error: syntax error, unexpected 'Trait' (T_TRAIT), expecting identifier (T_STRING) in
test3-trait.php on line 2
Errors parsing test3-trait.php
```

Pro účely kontinuální integrace je důležité spouštět kontrolu syntaktické správnosti pro aktuální verzi PHP, ale může být užitečné spouštět i kontrolu, zda skripty bude možné spustit v novější verzi PHP. Vzhledem k tomu, že novější verze PHP jsou většinou rychlejší⁶⁹, je možné díky zajištění dopředné kompatibility snadno získat výkonové zlepšení. Podle [BERGMANN, 2008] je PHP 5.3 1,2x rychlejší než PHP 5.2, takže pokud by projekt vyvíjený v době PHP 5.2 měl podobně kontrolovanou kompatibilitu s PHP 5.3, mohl by jen pomocí aktualizace na novou verzi PHP získat 20% zrychlení běhu.

3.2.2 Pre-commit hook

Systémy pro správu verzí jako Subversion nebo GIT umožňují navázat spouštění skriptů na události vyvolané v různých fázích ukládání souborů do úložiště. Pro kontrolu syntaktické správnosti souborů ještě před jejich přijetím do úložiště využít pre-commit hook, který může probíhající commit zrušit. Lze využít už hotové skripty pro SVN⁷⁰ nebo GIT⁷¹. Oba fungují tak, že získají seznam změněných souborů s příponou PHP, spustí na nich kontrolu syntaktické správnosti a pokud jsou všechny soubory v pořádku, tak commit umožní dokončit. V opačném případě vrátí textovou informaci, které soubory obsahují syntaktické chyby.

⁶⁹ Viz [BERGMANN, 2008]

⁷⁰ Viz <http://blueparabola.com/blog/subversion-commit-hooks-php>

⁷¹ Viz <http://phpadvent.org/2008/dont-commit-that-error-by-travis-swicegood>

PHP kód nemusí být ovšem uložený jen v souborech s příponou PHP. Například často používaný PHP framework Zend Framework⁷² využívá jako šablonovací jazyk přímo PHP a šablony jsou ve výchozím nastavení uloženy v souborech s příponou PHTML (viz Kód 9). Proto by bylo vhodné, aby se během commitu kontrolovala i jejich syntaktická správnost, což výše uvedené skripty neřeší.

Kód 9: Příklad šablony v PHTML souboru (test4-template.phtml)

```
<h1><?php echo "Kontinuální integrace"; ?></h1>
```

3.3 PHP_CodeSniffer

PHP_CodeSniffer⁷³ je nástroj určený na analýzu zdrojových kódů aplikace a kontrolu, zda odpovídají zvoleným standardům pro formátování souborů se zdrojovým kódem.

Je možné zvolit již nějaký předdefinovaný standard pro formátování (k dispozici jsou například standardy používané v knihovně PEAR, Zend Frameworku a dalších), případně je možné si vytvořit pravidla na míru pro svůj projekt.

Pro aplikace postavené nad Zend Frameworkem je samozřejmostí zvolit předdefinovaný standard. Nicméně provedl jsem jednu úpravu a upravil jsem pravidlo na omezení délky řádků zdrojového kódu. Výchozích 80 znaků je podle mého názoru přežitek z doby, kdy byla omezením šířka terminálu. Dnes, kdy je běžné vyvíjet software na monitorech s velkým rozlišením (fullHD – 1920x1080 bodů), nemá smysl omezovat délku řádku na méně než 120 znaků.

PHP_CodeSniffer lze nainstalovat pomocí nástroje PEAR následujícím příkazem:

```
pear install PHP_CodeSniffer
```

Pokud provedeme kontrolu dodržování standardu Zend⁷⁴ na ukázkovém kódu, který porušuje několik pravidel:

```
<?php
class ArticlesController extends Zend_Controller_Action{
    /**
     * Index action
     */
    public function indexAction(){
        $a = new ArticleModel();    }
}
```

Dostaneme výstup s přehledem chyb:

```
> phpcs --standard=Zend ArticlesController.php
```

```
FILE: php-cs\ArticlesController.php
```

```
-----
FOUND 4 ERROR(S) AFFECTING 4 LINE(S)
-----
```

```
2 | ERROR | Opening brace of a class must be on the line after the definition
6 | ERROR | Opening brace should be on a new line
```

⁷² Viz <http://framework.zend.com/>

⁷³ Vit http://pear.php.net/package/PHP_CodeSniffer/

⁷⁴ Viz <http://framework.zend.com/manual/en/coding-standard.html>

```
7 | ERROR | Closing brace must be on a line by itself
8 | ERROR | Closing brace indented incorrectly; expected 0 spaces, found 4
-----
```

```
Time: 0 seconds, Memory: 2.00Mb
```

PHP_CodeSniffer je možné podobně jako PHP Lint spouštět jako pre-commit hook⁷⁵ (a nedovolit do verzovacího systému uložit změny, které porušují standardy pro psaní kódu). Nicméně při větším množství změněných souborů v rámci jednoho commitu může kontrola trvat dlouho, takže doporučuji provádět kontrolu dodržování standardů pro psaní kódu až v rámci kontinuální integrace.

CodeSniffer samozřejmě podporuje výstup ve formátu vhodném pro zařazení do kontinuální integrace. Takový příkaz může vypadat například takto:

```
phpcs --standard=Zend --report=checkstyle --report-file=checkstyle-phpcs.xml .
```

3.4 PHP CPD

PHP Copy/Paste Detector⁷⁶ je nástroj, který odhalí zkopírované nebo duplicitní bloky kódu. Do systému ho lze nainstalovat pomocí příkazu

```
pear install phpunit/phpcpd
```

Jeho následné použití je jednoduché (viz Kód 10).

Kód 10: Ukázka použití PHP CPD

```
> phpcpd .
phpcpd 1.3.2 by Sebastian Bergmann.

Found 1 exact clones with 24 duplicated lines in 1 files:

- ArticlesController.php:14-38
  ArticlesController.php:67-91

20.34% duplicated lines out of 118 total lines of code.
```

Samozřejmostí je výstup do XML použitelného pro integrační server, který lze aktivovat přepínačem `--log-pmd <file>`.

3.5 PHPLOC

PHPLOC⁷⁷ je jednoduchý nástroj, který umožňuje získat přehled o počtu řádků kódu v projektu a počítá některé další metriky.

Instalaci lze provést pomocí nástroje PEAR:

```
pear install phpunit/phploc
```

Jeho následné použití je jednoduché (viz Kód 11).

⁷⁵ Viz <http://pear.php.net/manual/en/package.php.php-codesniffer.svn-pre-commit.phpz>

⁷⁶ Viz <https://github.com/sebastianbergmann/phpcpd>

⁷⁷ Viz <https://github.com/sebastianbergmann/phploc>

Kód 11: Ukázkový výstup z aplikace phplloc

```
> phplloc ./application
phplloc 1.6.1 by Sebastian Bergmann.

Directories:                               59
Files:                                     290

Lines of Code (LOC):                       44338
  Cyclomatic Complexity / Lines of Code:    0.05
Comment Lines of Code (CLOC):              8176
Non-Comment Lines of Code (NCLOC):         36162

Namespaces:                               0
Interfaces:                               0
Classes:                                  290
  Abstract:                               6 (2.07%)
  Concrete:                              284 (97.93%)
  Average Class Length (NCLOC):            129
Methods:                                  1182
  Scope:
    Non-Static:                           1164 (98.48%)
    Static:                               18 (1.52%)
  Visibility:
    Public:                               796 (67.34%)
    Non-Public:                           386 (32.66%)
  Average Method Length (NCLOC):            31
  Cyclomatic Complexity / Number of Methods: 2.48

Anonymous Functions:                       0
Functions:                                 10

Constants:                                 280
  Global constants:                        2
  Class constants:                        278
```

Samozřejmostí je opět výstup do formátu použitelného pro integrační server, který lze aktivovat přepínačem `--log-csv <file>`.

3.6 PHP Depend

PHPDepend nebo také PDepend⁷⁸ je nástroj, který získává různé metriky zdrojových kódů (viz kapitola 1.4.2). Snaží se být PHP obdobou nástroje JDepend⁷⁹.

Nainstalovat ho lze opět pomocí nástroje PEAR:

```
pear channel-discover pear.pdepend.org
pear install pdepend/PHP_Depend-beta
```

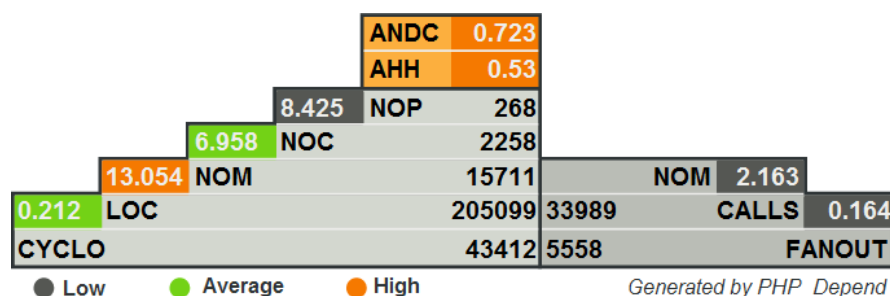
Spustit ho lze pomocí příkazu

```
pdepend --jdepend-chart=jdepend.svg --jdepend-xml=jdepend.xml --summary-xml=pdepend-summary.xml --overview-pyramid=pyramid.svg ./application
```

Výsledkem je graf shrnující metriky získané z analýzy zdrojových kódů do přehledné pyramidy (viz Obrázek 7). Detailní popis jednotlivých metrik lze nalézt v oficiální dokumentaci⁸⁰ nebo v [LANZA, 2006].

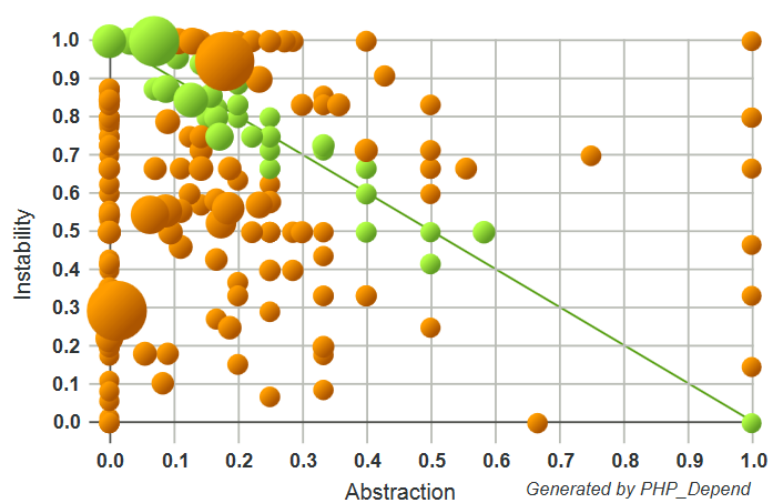
⁷⁸ Viz <http://pdepend.org/>

⁷⁹ Viz <http://clarkware.com/software/JDepend.html>



Obrázek 7: Pyramida s přehledem metrik kódu pro Zend Framework 1.11.11 (zdroj: autor)

Dalším výstupem je graf ukazující míru abstrakce a stability (viz Obrázek 8). Více informací lze nalézt v oficiální dokumentaci⁸¹ nebo v [MARTIN, 1994]. Tento graf může být užitečný například pro srovnání různých PHP frameworků - viz například [PRSKAVEC, 2009]



Obrázek 8: Graf míry abstrakce a stability pro Zend Framework 1.11.11 (zdroj: autor)

3.7 PHP MD

PHP MD (Mess Detector)⁸² je nástroj na analýzu zdrojových kódů v PHP a detekci potenciálně problematických míst ve zdrojových kódech. Snaží se být pro PHP podobným nástrojem, jako je PMD⁸³ pro Javu. Je závislý na nástroji PDepend, jehož výstup používá jako vstup pro vlastní analýzu. Kontroluje jak přílišnou komplexitu kódu (viz kapitola 1.4.2), tak i některá další porušení standardů pro psaní kódu (viz 1.4.3).

Je vhodné vytvořit si vlastní XML soubor s pravidly⁸⁴, která bude PHPMD ověřovat. Výhodou tohoto přístupu je, že pokud v budoucnu bude potřeba některá pravidla doplnit nebo upravit, nebude nutné upravovat skript pro sestavení. Soubor pravidel je typicky uložený v souboru `phpmd.xml` v kořenovém adresáři projektu.

Instalaci lze provést pomocí nástroje PEAR:

⁸⁰ Viz <http://pdepend.org/documentation/handbook/reports/overview-pyramid.html>

⁸¹ Viz <http://pdepend.org/documentation/handbook/reports/abstraction-instability-chart.html>

⁸² Viz <http://phpmd.org/>

⁸³ Viz <http://pmd.sourceforge.net/>

⁸⁴ Viz <http://phpmd.org/documentation/creating-a-ruleset.html>


```
pear channel-discover pear.phpmd.org
pear channel-discover pear.pdepend.org
pear install --alldeps phpmd/PHP_PMD
```

Spustit ho lze poté pomocí příkazu:

```
> phpmd ./application text codesize,design,naming,unusedcode
```

kdy „text“ označuje výstupní formát (pro kontinuální integraci je nutné nahradit za „xml“) a následující parametry určují, jaká pravidla se mají použít (lze nahradit za soubor s pravidly - „phpmd.xml“)

3.8 PHPUnit

PHPUnit⁸⁵ je testovací framework pro jazyk PHP. Kromě podpory běžného jednotkového testování nabízí podporu také pro testování komponent závislých na databázi a testování pomocí nástroje Selenium⁸⁶.

PHPUnit lze nainstalovat pomocí příkazů

```
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

Popis postupu vytváření testů s nástrojem PHPUnit přesahuje rozsah této práce, v následujících krocích se předpokládá, že jsou již vytvořené. Více informací o tvorbě testů pomocí PHPUnit lze nalézt například v [BERGMANN, 2005] nebo [BERGMANN, 2011].

Spuštění hotových testů je snadné. V adresáři, kde se nachází soubor `phpunit.xml` s konfigurací testů stačí zavolat příkaz `phpunit` a testy proběhnou podle nastavení v souboru `phpunit.xml`.

```
> phpunit
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from zf-tutorial\tests\phpunit.xml

..

Time: 1 second, Memory: 3.25Mb

OK (2 tests, 3 assertions)
```

Samozřejmostí je výstup ve formátu JUnit, se kterým umí pracovat integrační servery. Ten lze zajistit pomocí přepínače `--log-junit phpunit-report.xml`. Pomocí přepínače `--coverage-clover` lze z PHPUnit exportovat i statistiky pokrytí kódu testy. Opět jde o formát, který umí integrační servery zpracovávat.

V případě, že tým teprve začíná automatizované testování využívat, tak bych míru pokrytí kódu testy doporučoval nesledovat, neboť nízké hodnoty by mohly vývojáře spíše demotivovat (více viz kapitola 1.3.3.1).

⁸⁵ Viz <https://github.com/sebastianbergmann/phpunit/>

⁸⁶ Viz <http://seleniumhq.org/>

3.9 Generování API dokumentace

Při psaní zdrojového kódu je běžné do něj doplňovat tzv. dokumentační komentáře. Lze z nich totiž pak vygenerovat programátorskou dokumentaci (také nazývanou API dokumentace), kterou lze procházet pohodlněji než zdrojový kód.

Způsob zápisu používaný v jazyce PHP je založený na syntaxi používané v jazyce Java a nástroji Javadoc⁸⁷, jen je upravený kvůli odlišnostem jazyka PHP.

Kód 12: Ukázka možnosti využití PHPDoc (soubor phpdoc01.php)

```
<?php
/**
 * Ukázková třída
 *
 * @author Martin Hujer
 */
class MyClass
{
    /**
     * Funkce foo nic nedělá, jen vrátí předaný parametr.
     *
     * @param boolean $param Předaný parametr
     * @return boolean
     */
    public function foo($param)
    {
        return $param;
    }
}
```

3.9.1 PhpDocumentor

PhpDocumentor⁸⁸ je původní nástroj na generování API dokumentace pro PHP skripty. Ovládá se pomocí příkazové řádky, takže ho není problém zařadit do skriptu pro sestavení. Jeho spuštění může vypadat například takto:

```
phpdoc -f phpdoc01.php -t doc
```

Obrázek 9 naznačuje, jak vypadá výsledná dokumentace ve výchozí HTML šabloně.

⁸⁷ Viz <http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html>

⁸⁸ Viz <http://www.phpdoc.org/> nebo <http://pear.php.net/package/PhpDocumentor>

default

Description
Class trees
Index of elements
Classes
MyClass
Files
phpdoco1.php
phpDocumentor v 1.4.4

Class MyClass

Description

Description | [Methods \(details\)](#)

Ukázková třída

- author:** Martin Hujer

Located in [/phpdoco1.php](#) (line 7)

Method Summary

Description | [Methods \(details\)](#)

boolean **foo** (*boolean* \$param)

Methods

Description | [Methods \(details\)](#)

foo (line 15)

Funkce foo nic nedělá, jen vrátí předaný parametr.

- access:** public

boolean **foo** (*boolean* \$param)

- boolean* \$param:** Předaný parametr

Documentation generated on Sat, 03 Mar 2012 21:03:01 +0100 by [phpDocumentor 1.4.4](#)

Obrázek 9: Ukázka dokumentace vygenerované pomocí PhpDocumentor (zdroj: autor)

Nevýhodou PhpDocumentor je nepodpora⁸⁹ kódování UTF-8. Nicméně je možné ji doplnit úpravou⁹⁰ šablon pro generování dokumentace.

3.9.2 DocBlox

DocBlox⁹¹ je nový nástroj na generování API dokumentace. Jednou z jeho předností je rychlost generování. To je dobře vidět například na dokumentaci Zend Frameworku, která se s využitím nástroje PhpDocumentor generovala 100 minut, tak DocBlox ji zvládne vygenerovat za přibližně 10 minut⁹². Další zajímavou funkcí je generování schématu tříd ve formátu SVG. Výhodou nástroje DocBlox je možnost inkrementálního generování dokumentace, což může generování ještě zrychlit.

⁸⁹ Viz <http://pear.php.net/bugs/bug.php?id=12128>

⁹⁰ Viz <http://www.beranek.de/node/73>

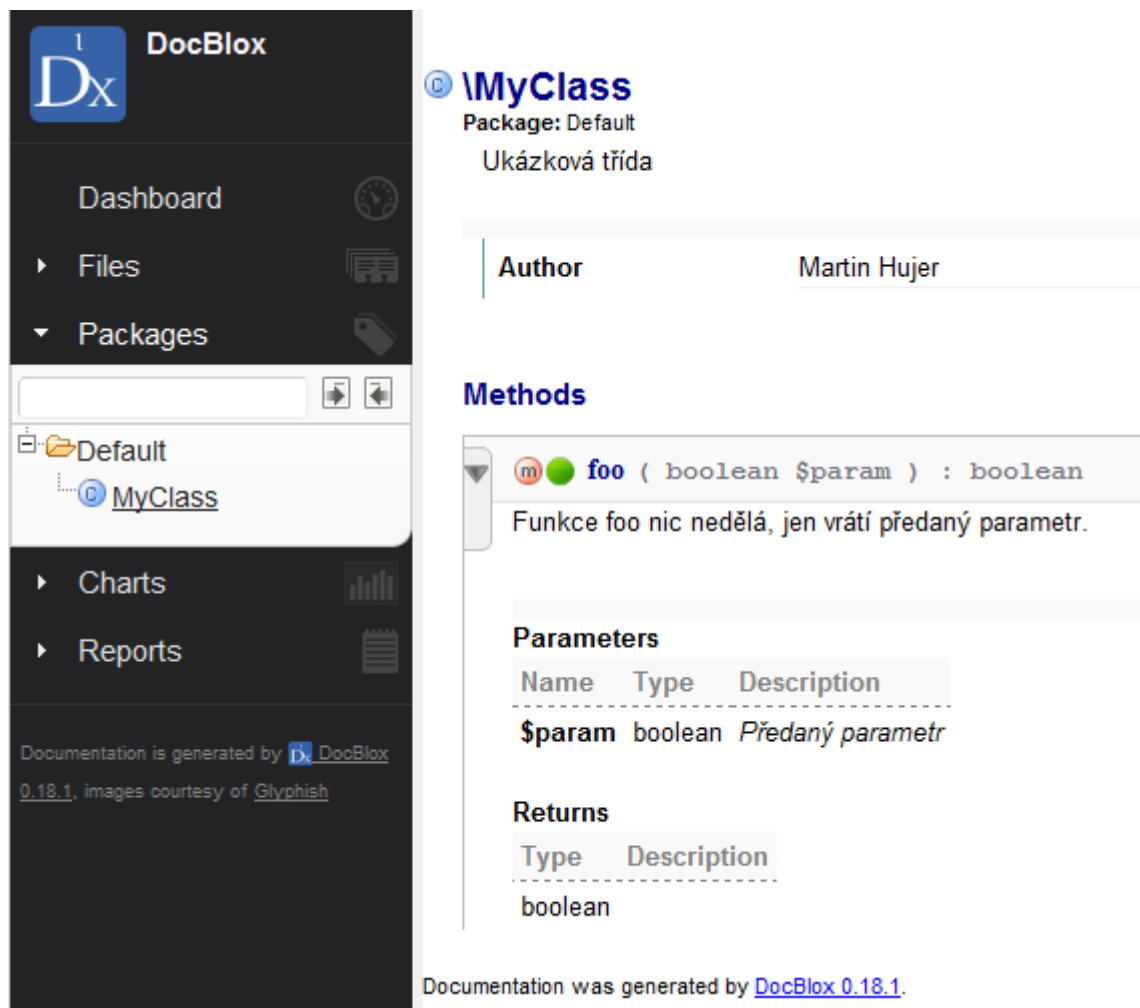
⁹¹ Viz <http://www.docblox-project.org/>

⁹² Viz <http://zend-framework-community.634137.n4.nabble.com/ZF-API-documentation-td3451968.html#a3452171>

Generování dokumentace v DocBloxu se spouští velmi podobně jako v případě nástroje PhpDocumentor:

```
docblox -f phpdoc01.php -t docblox
```

Jak vypadá vygenerovaná dokumentace ukazuje Obrázek 10.



Obrázek 10: Ukázka dokumentace vygenerované nástrojem DocBlox (zdroj: autor)

3.9.2.1 Validace dokumentačních komentářů

Docblox při generování dokumentace zároveň kontroluje, zda dokumentační komentáře u zdrojového kódu obsahují potřebné informace. Případné nedostatky standardně vypisuje do sekce v HTML dokumentaci. Pro snadnější napojení na kontinuální integraci existuje šablona „checkstyle“⁹³, která přehled problematických míst vygeneruje do XML souboru.

Šablonu je možné nainstalovat přímo přes Docblox:

```
docblox template:install checkstyle
```

Upravený příkaz pro kontrolu dokumentace vypadá takto:

⁹³ Viz <http://docs.docblox-project.org/for-users/validating-documentation-in-your-code.html#as-checkstyle-log-file>

```
docblox -f phpdoc01.php -t docblox --template checkstyle
```

3.9.3 ApiGen

ApiGen⁹⁴ je další nový nástroj na generování dokumentace ze zdrojových kódů v jazyce PHP. Je zajímavý například tím, že podporoval generování dokumentace pro PHP 5.4 už dříve, než bylo vydáno, a také tím, že je vyvíjen v České republice.

Instalaci je možné snadno provést pomocí nástroje PEAR:

```
pear config-set auto_discover 1  
pear install pear.apigen.org/apigen
```

V případě problémů s instalací pomocí PEAR je možné si stáhnout⁹⁵ instalační balíček, který již obsahuje všechny potřebné knihovny. Stačí ho jen rozbalit a jeho umístění přidat do systémové cesty.

Spustit ho lze opět snadno pomocí příkazové řádky:

```
apigen -s phpdoc01.php -d apigen
```

Jak vypadá vygenerovaná dokumentace ukazuje Obrázek 11.

⁹⁴ Viz <http://apigen.org/>

⁹⁵ Viz <https://github.com/apigen/apigen/downloads>

Overview

Classes

MyClass

Overview **Class**

Tree

Class MyClass

Ukázková třída

Author: Martin Hujer
Located at [phpdoc01.php](#)

Methods summary

public	<code>foo (boolean \$param)</code>	#
boolean	Funkce foo nic nedělá, jen vrátí předaný parametr.	
	Parameters	
	<code>\$param</code>	
	boolean	
	Předaný parametr	
	Returns	
	boolean	

API documentation generated by [ApiGen 2.5.0](#)

Obrázek 11: Ukázka dokumentace vygenerovaná nástrojem ApiGen (zdroj: autor)

3.9.3.1 Validace dokumentačních komentářů

ApiGen stejně jako DocBlox provádí kontrolu, zda jsou dokumentační komentáře vyplněny a případné chyby také umožňuje exportovat do formátu pro Checkstyle:

```
apigen -s phpdoc01.php -d apigen --report apigen/checkstyle.xml
```

3.9.4 Výběr nástroje pro generování dokumentace

Pro volbu nástroje pro generování dokumentace ze zdrojových kódů máme na výběr z několika možností. Vzhledem k tomu, že ostatní součásti kontinuální integrace na vygenerované dokumentaci nezávisí, je možné nástroj snadno vyměnit i později.

Nakonec jsem pro využití v kontinuální integraci zvolil nástroj ApiGen, protože má subjektivně přehlednější dokumentaci než DocBlox. PhpDocumentor jsem nevybral, protože neumožňuje vytvoření reportu pro Checkstyle.

3.10 Automatizace sestavení

Kontinuální integrace vyžaduje pro své fungování automatizované sestavení (viz kapitola 1.1.2). Integrační server Jenkins, zvolený pro implementaci (viz kapitola 2.2), podporuje pomocí rozšíření různé nástroje na automatizaci sestavení. Například Apache Ant⁹⁶, Phing⁹⁷, Rake⁹⁸, Maven⁹⁹, NAnt¹⁰⁰ a další.

Pro tuto implementaci integračního serveru jsem zvažoval nástroje Apache Ant a Phing. Ant je nástroj typicky využívaný pro sestavování projektů napsaných v jazyce Java. Phing vznikl jako převod principů využívaných v nástroji Ant do jazyka PHP.

Oba nástroje využívají k popisu procesu sestavení XML soubory, které obsahují jednotlivé úkoly. Kořenovým elementem dokumentu je tzv. projekt (project), který obaluje jednotlivé úkoly (target). Úkoly se skládají z jednotlivých úloh (task).

Ukázkový soubor může vypadat takto:

```
//build-example.xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="Ukazkove sestaveni" default="build">
  <target name="cleanup">
    <echo message="Zde by byly dalsi ulohy pro vycistení pracovního adresáře"></echo>
  </target>

  <target name="build" depends="cleanup">
    <echo message="Zde by byly dalsi ulohy pro sestavení"></echo>
  </target>
</project>
```

Projekt má v atributu `default` určený výchozí úkol, který se provede, pokud nebude při spuštění určeno jinak. U úkolu `build` je uvedena závislost na úkolu `cleanup`, což znamená, že úkol `build` se neprovede dříve, než bude vyčištěn pracovní adresář.

Detailní popis nástrojů Ant a Phing přesahuje rozsah této práce, více informací lze nalézt v [THE PHING PROJECT, 2011], [ZANDSTRA, 2007] nebo [LOUGHRAN, 2007].

Tento jednoduchý skript pro sestavení využívá jen vlastnosti společné jak pro Ant, tak pro Phing, takže je možné ho spustit pomocí obou:

Phing:

```
> phing -f build-example.xml
Buildfile: I:\_BP\phing\build-example.xml

Ukazkove sestaveni > cleanup:

    [echo] Zde by byly dalsi ulohy pro vycistení pracovního adresáře

Ukazkove sestaveni > build:

    [echo] Zde by byly dalsi ulohy pro sestavení
```

⁹⁶ Viz <http://ant.apache.org/>

⁹⁷ Viz <http://www.phing.info/>

⁹⁸ Viz <http://rake.rubyforge.org/>

⁹⁹ Viz <http://maven.apache.org/>

¹⁰⁰ Viz <http://nant.sourceforge.net>

```
BUILD FINISHED
```

```
Total time: 0.2688 seconds
```

Ant:

```
> ant.bat -f build-example.xml
Buildfile: I:\_BP\phing\build-example.xml

cleanup:
    [echo] Zde by byly dalsi ulohy pro vycistení pracovního adresare

build:
    [echo] Zde by byly dalsi ulohy pro sestavení

BUILD SUCCESSFUL
Total time: 0 seconds
```

Jak je vidět, tak výstupy obou nástrojů se u takto jednoduchého skriptu liší jen v drobných detailech.

3.10.1 Výběr vhodného nástroje pro automatizaci sestavení

Pro účely nasazení kontinuální integrace jsem zvolil nástroj Phing a to z několika důvodů. Je implementovaný v jazyce PHP a je zaměřený na automatizaci úloh, které mohou být při jeho využití potřebné. Pro většinu z nich už obsahuje vytvořené šablony úloh, takže jejich parametry lze zapisovat přímo jako elementy a atributy v XML a ne jen jako parametry příkazové řádky. Díky tomu, že celá implementace je v PHP, tak je možné si snadno dopsat vlastní šablony úloh. V neposlední řadě jsem ho zvolil proto, že už se mi osvědčil při automatizaci nasazování PHP aplikací.

Instalace nástroje Phing je snadná, lze ji provést pomocí již dříve popsaného nástroje PEAR (viz kapitola 3.1):

```
pear channel-discover pear.phing.info
pear install phing/phing
```


4 Praktická implementace platformy pro kontinuální integraci v malé firmě

V této kapitole se zaměřím na postup instalace platformy pro kontinuální integraci pro malou firmu, nebudu se tedy zabývat tématy, která se netýkají kontinuální integrace v malé firmě, jako je koordinace spolupráce více integračních serverů, distribuované spouštění testů a další.

4.1 Skript pro sestavení aplikace

Vývoj a testování skriptu pro sestavení je vhodné provádět spíše na menším projektu než na rozsáhlé aplikaci. Důvodem je, že při volbě rozsáhlé aplikace se stovkami PHP souborů, by bylo nutné zbytečně dlouhou dobu čekat na proběhnutí jednotlivých částí sestavení.

Proto jsem pro vytvoření skriptu pro sestavení aplikace zvolil ukázkovou aplikaci ZFTutorial, která je výsledkem návodu Getting Started with Zend Framework¹⁰¹, pro úvod do funkčnosti Zend Frameworku¹⁰². Protože aplikace je k dispozici jen jako archiv ve formátu ZIP a pro kontinuální integraci je důležité využívat verzovací systém, vytvořil jsem GIT repositář¹⁰³ na serveru GitHub.com a aplikaci tam uložil.

Jako první krok tvorby skriptu pro sestavení aplikace je nutné si v kořenovém adresáři projektu vytvořit prázdný soubor build.xml a do něj vložit kostru projektu a jednoduchý „task“ echo, pro ověření, že Phing funguje správně.

```
<project name="zf-tutorial" default="main">
  <target name="main">
    <echo message="Phing works!"/>
  </target>
</project>
```

Výsledkem zavolání příkazu phing v adresáři, kde je uložen soubor build.xml, by měl být tento výstup:

```
> phing
Buildfile: I:\_BP\zf-tutorial\build.xml

zf-tutorial > main:

    [echo] Phing works!

BUILD FINISHED

Total time: 0.3852 seconds
```

4.2 PHP Lint

Prvním krokem, který je nutné při sestavení udělat, je kontrola syntaktické správnosti, pro jazyk PHP je to tzv. PHP Lint (viz kapitola 3.2).

¹⁰¹ Viz <http://akrabat.com/zend-framework-tutorial/>

¹⁰² Viz <http://framework.zend.com/>

¹⁰³ Viz <https://github.com/mhujer/zf-tutorial>

Do build skriptu je nutné přidat nový target:

```
<target name="lint" description="Kontrola pomocí PHP Lint">
  <phplint haltonfailure="true" level="info">
    <fileset dir="${project.basedir}/application">
      <include name="**/*.php"/>
      <include name="**/*.phtml"/>
    </fileset>
  </phplint>
</target>
```

Je zde několik věcí, které je potřeba vysvětlit. Phing již v základu obsahuje task `phplint`¹⁰⁴, který na pozadí spouští samotnou kontrolu pomocí `php -l`.

Atribut `haltonfailure="true"` znamená, že pokud selže syntaktická kontrola jakéhokoliv souboru, tak ihned selže celé sestavení a nebudou se provádět další kroky.

Proměnná `${project.basedir}` se automaticky nastaví na adresář, který je nadřazený souboru `build.xml`.

Sada souborů (fileset) popisuje množinu souborů, která se předají úloze (task). Bylo by možné stejný fileset zkopírovat do dalších částí skriptu pro sestavení, protože na zdrojových kódech samotné aplikace se spouští i další úlohy. Nicméně Phing toto umožňuje řešit elegantněji. Je možné vytvořit fileset i uvnitř tagu `<project>`, přiřadit mu identifikátor a na něj pak jen odkazovat.

Filesety jsou označené jako „src“ a „templates“:

```
<fileset id="src" dir="${project.basedir}/application">
  <include name="**/*.php"/>
</fileset>
<fileset id="templates" dir="${project.basedir}/application">
  <include name="**/*.phtml"/>
</fileset>
```

A target je pak možné zpřehlednit takto:

```
<target name="lint" description="Kontrola pomocí PHP Lint">
  <phplint haltonfailure="true" level="info">
    <fileset refid="src"/>
    <fileset refid="templates"/>
  </phplint>
</target>
```

Podobně lze ve skriptu pro sestavení definovat adresáře `tests` a `library` a ty doplnit do úkolu `phplint`.

Jak vypadá výsledný skript, je možné vidět v Příloze 1.

4.3 Vytvoření projektu na integračním serveru

Integrační server je nainstalovaný a nastavený podle postupu v kapitolách 2.3 a 2.4. Vše je tedy připravené a je možné se pustit do přípravy automatizovaného sestavování. Je nutné nainstalovat dvě rozšíření - GIT¹⁰⁵ a Phing¹⁰⁶.

¹⁰⁴ Viz <http://www.phing.info/docs/guide/stable/chapters/appendixes/AppendixC-OptionalTasks.html#PhpLintTask>

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin phing
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin git
```

Instalace verzovacího systému GIT přesahuje rozsah této práce, další postup předpokládá, že je k dispozici v systémové cestě. Více informací lze nalézt například v [CHACON, 2009].

Vytvoření nového projektu se provede volbou možnosti „New Job“ („Job“ je označení, které Jenkins používá pro něco, co lze chápat jako „projekt“), vyplněním jména (v tomto případě `zf-tutorial`), a dále vybráním volby „Build a free-style software project“.

Job name

☒ **Build a free-style software project**

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Obrázek 12: Vytvoření nového projektu na serveru Jenkins (zdroj: autor)

V dalším kroku v sekci „Source Code Management“ je nutné vložit URL repozitáře s projektem (`https://mhujer@github.com/mhujer/zf-tutorial.git`) a do pole „Branches to build“ zadat „master“ (sestavení se bude provádět jen na hlavní větvi projektu).

Source Code Management

☐ CVS
☒ **Git**

Repositories Repository URL

Advanced...
Delete Repository

Branches to build Branch Specifier (blank for default):

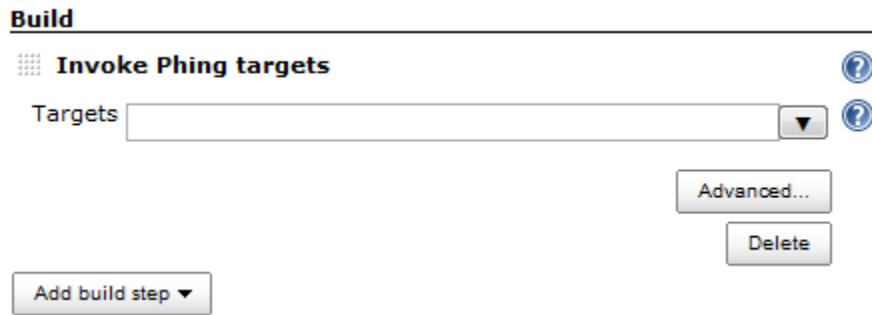
Delete Branch

Obrázek 13: Nastavení cesty k repozitáři s projektem (zdroj: autor)

V sekci „Build“ je nutné přidat krok pro sestavení pomocí „Add build step“ a „Invoke Phing targets“. Nic dalšího není nutné nastavovat, protože soubor s instrukcemi pro sestavení je pojmenovaný standardně, a zároveň je nastavený výchozí „target“.

¹⁰⁵ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin>

¹⁰⁶ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Phing+Plugin>



Obrázek 14: Nastavení skriptu pro sestavení aplikace (zdroj: autor)

Stačí tedy změny uložit pomocí volby „Save“ a spustit sestavení pomocí „Build Now“ v nabídce vlevo. Poté je možné vidět, že sestavení proběhlo v pořádku.



Obrázek 15: Úspěšné sestavení v Jenkinsu (zdroj: autor)

Do detailních informací o sestavení se lze dostat kliknutím na datum a čas konkrétního sestavení v části „Build History“. V sekci „Console Output“ je vidět průběh sestavení (pokud by trvalo déle, tak tam lze sledovat aktuální průběh).

Dále je vidět, že se spustil target `main`, který jen vypsal informační hlášku.

```
zf-tutorial > main:
[echo] Phing works!
```

Dalším krokem je přechod do nastavení projektu a sekci „Build“ změna pole „Targets“ na hodnotu „lint“, což je název targetu, který byl vytvořen v kapitole 4.2. Teď je možné spustit další sestavení pomocí „Build Now“.

Výstup v konzoli ukazuje, že se sestavení provedlo tak, jak mělo:

```
zf-tutorial > lint:

[phplint] I:\BP-jenkins\data\jobs\zf-tutorial\workspace\application\Bootstrap.php: No
syntax errors detected
.....
[phplint] I:\BP-jenkins\data\jobs\zf-tutorial\workspace\tests\library\bootstrap.php: No
syntax errors detected

BUILD FINISHED

Total time: 1.0731 second

Finished: SUCCESS
```

Dále je vhodné přidat další target, který nebude vykonávat žádnou činnost, ale bude záviset na ostatních, takže zajistí, že ty se spustí dříve. Zároveň je nutné ho nastavit jako hlavní pro spouštění na integračním serveru.

```
<target name="build" depends="lint" description="Meta target, spouští ostatní targety"/>
```

4.4 Nasazení PHPUnit

Nástroj pro automatizované testování PHPUnit je popsán v kapitole 3.8. Do skriptu pro sestavení aplikace ho lze přidat pomocí:

```
<target name="phpunit" depends="prepare" description="PHPUnit testy">
  <phpunit printsummary="true" haltonfailure="true" haltonerror="true">
    <formatter todir="${project.basedir}/build" outfile="phpunit-report.xml"
type="xml"/>
    <batchtest>
      <fileset refid="tests"/>
    </batchtest>
  </phpunit>
</target>
```

Je vhodné ho zařadit ihned po kontrole pomocí PHPLint, protože opět platí, že pokud automatizované testy odhalí chyby ve funkčnosti, tak další metriky kódu nemá smysl získávat. Důležité jsou pro to atributy `haltonfailure` a `haltonerror`, které v případě selhání testu nebo chyby ve skriptu zastaví zpracování sestavení.

Vzhledem k tomu, že PHPUnit generuje tzv. artefakt (soubor, který je výsledkem sestavení), je nutné při dalších spuštěních zajistit, aby tento soubor neexistoval a vygeneroval se znovu. Toho lze dosáhnout pomocí dalších targetů, jeden pro smazání adresáře a druhý pro jeho znovuvytvoření. Je vhodné upravit všechny ostatní targety, aby závisely na targetu `prepare`, neboť je pak bude možné spouštět i samostatně.

```
<target name="cleanup" description="Vyčištění workspace">
  <delete dir="${project.basedir}/build"/>
</target>

<target name="prepare" depends="cleanup" description="Příprava workspace">
  <mkdir dir="${project.basedir}/build"/>
</target>
```

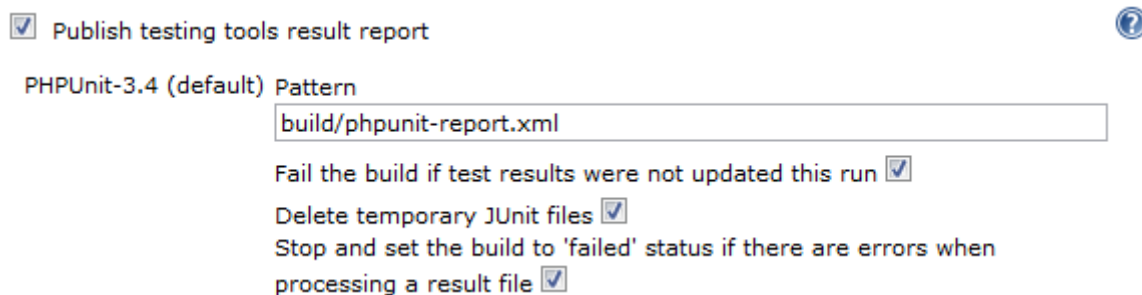
Jenkins sice ve výchozí instalaci umožňuje zobrazení výsledků proběhnuvších testů, ale je vhodnější využít rozšíření, které umožňuje zpracovávat výsledky testů z nástrojů z rodiny xUnit¹⁰⁷. Zároveň umí totiž navíc graficky zobrazovat historii průběhu testů a případně sestavení v případě selhávajících testů označit jako neúspěšné.

Nainstalovat ho lze pomocí příkazu:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin xunit
```

Po restartu serveru je nutné rozšíření nakonfigurovat ve vlastnostech projektu. Prvním krokem je zaškrtnutí pole „*Publish testing tools result report*“, dalším přidání „*PHPUnit-3.4*“ pomocí tlačítka „*Add*“. Do pole „*Pattern*“ je nutné vyplnit relativní cestu k výsledku proběhnutí testů - `build/phpunit-report.xml`.

¹⁰⁷ Viz <https://wiki.jenkins-ci.org/display/JENKINS/xUnit+Plugin>



Obrázek 16: Konfigurace nástroje PHPUnit na serveru Jenkins (zdroj: autor)

Po sestavení projektu na úvodní stránce přibude položka „*Latest Test Result*“, kde je možné procházet výsledky testů, a zároveň graf proběhnutí testů.

4.4.1 Pokrytí kódu testy

Zjišťování metriky pokrytí kódu testy jsem se rozhodl nenasazovat z důvodů popsanych v kapitole 1.3.3.1. Více informací o zařazení této metriky do kontinuální integrace lze nalézt v [BERGMANN, 2011b]

4.5 Nasazení PHP_CodeSniffer

Kontrolu dodržování standardů pro psaní kódu s použitím standardu pro Zend Framework a exportem chyb ve formátu pro Checkstyle lze do skriptu pro sestavení přidat pomocí:

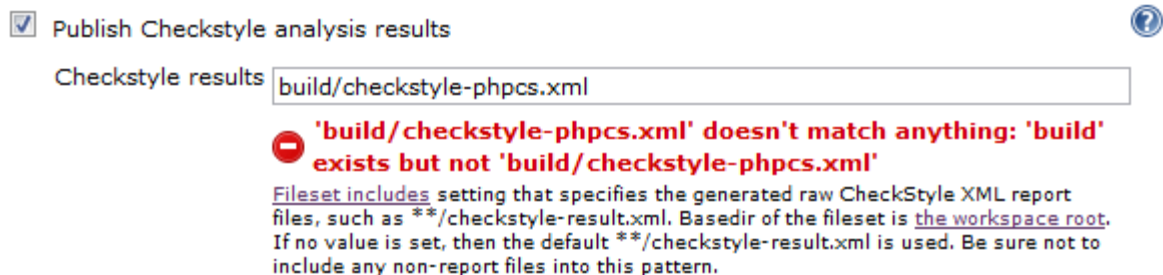
```
<target name="phpcs" depends="prepare" description="Kontrola Standardů pro psaní kódu">
  <phpcodesniffer standard="Zend">
    <fileset refid="src"/>
    <fileset refid="tests"/>
    <formatter type="default" usefile="false"/>
    <formatter type="checkstyle" outfile="${project.basedir}/build/checkstyle-
phpcs.xml"/>
  </phpcodesniffer>
</target>
```

Zároveň je nutné upravit zdrojový kód ukázkové aplikace, aby tam existovalo nějaké porušení těchto standardů.

Integrační server Jenkins umožňuje pomocí rozšíření Checkstyle zpracovat XML soubor se zjištěnými chybami. Rozšíření lze nainstalovat pomocí:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin checkstyle
```

Po restartu serveru je dále nutné v editaci projektu v sekci „*Post-build Actions*“ vybrat volbu „*Publish Checkstyle analysis results*“ a do pole „*Checkstyle results*“ vyplnit `build/checkstyle-phpcs.xml`. Jenkins vypíše chybové hlášení, že soubor neexistuje, což je správně, protože bude vygenerován až při dalším sestavení.



Obrázek 17: Nastavení zpracování výstupu z nástroje PHP_CodeSniffer (zdroj: autor)

Po dvou následujících sestaveních se na hlavní stránce projektu začne zobrazovat graf ukazující trend v množství porušení standardů pro psaní kódu.

4.6 Nasazení ApiGen

Dalším krokem je přidání generování dokumentace pomocí nástroje ApiGen popsáno v kapitole 3.9.3.

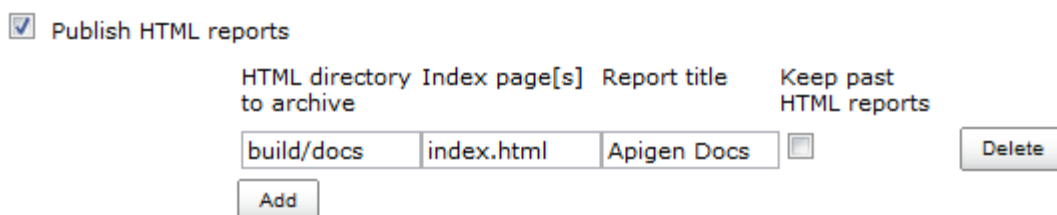
Nejprve je nutné přidat target pro ApiGen. Lze využít přímo task ApiGen, který je součástí nástroje Phing od verze 2.4.10:

```
<target name="apigen" depends="prepare" description="Generování dokumentace">
  <apigen source="${project.basedir}/application"
    destination="${project.basedir}/build/docs" report="${project.basedir}/build/checkstyle-
    apigen.xml"/>
</target>
```

Jenkins je schopen na hlavní stránku projektu doplnit odkazy na další HTML výstupy, toho lze dosáhnout rozšířením HTML Publisher¹⁰⁸, které lze nainstalovat pomocí příkazu:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin htmlpublisher
```

Po restartu je třeba HTML Publisher nakonfigurovat. V editaci nastavení projektu v sekci „Post-build Actions“ musí být zaškrtnuté pole „Publish HTML reports“. Adresář s daty („HTML directory to archive“) je build/docs. „Report title“ je vhodné nastavit například na „API Docs“. Ostatní volby je možné ponechat ve výchozím nastavení.



Obrázek 18: Doplnění odkazů na HTML výstupy (zdroj: autor)

Na hlavní stránce projektu se po proběhnutí sestavení objeví možnost procházet vygenerovanou dokumentaci.

¹⁰⁸ Viz <https://wiki.jenkins-ci.org/display/JENKINS/HTML+Publisher+Plugin>



Obrázek 19: Odkaz na vygenerovanou dokumentaci (zdroj: autor)

ApiGen generuje i soubor s chybami v dokumentačních komentářích ve formátu Checkstyle, který je možné agregovat s ostatními porušeními standardů pro psaní kódu. Stačí upravit pole „*Checkstyle results*“ na `build/checkstyle-*.xml` (hvězdička znamená, že se budou načítat všechny XML soubory z daného adresáře začínající na `checkstyle-`).

4.7 Nasazení PHPCPD

Nástroj na kontrolu duplicitního kódu PHPCPD je detailně popsán v kapitole 3.4. Je nutné ho zařadit do skriptu pro sestavení. Opět je možné využít již předpřipravený task v rámci nástroje Phing:

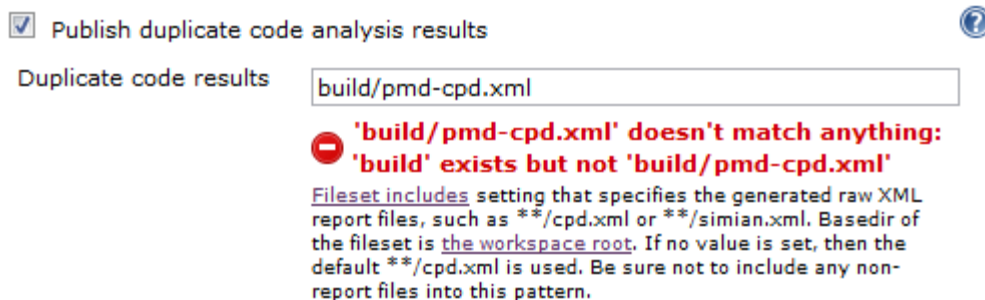
```
<target name="phpcpd" depends="prepare" description="Kontrola CPD">
  <phpcpd>
    <fileset refid="src"/>
    <fileset refid="tests"/>
    <formatter type="pmd" outfile="${project.basedir}/build/pmd-cpd.xml"/>
  </phpcpd>
</target>
```

Aby bylo možné ověřit, že nástroj kontroluje kód správně, je vhodné zduplikovat nějaký PHP soubor v testovací aplikaci (bude nutné přejmenovat třídu v něm, aby nedošlo k problémům při generování dokumentace).

Pro Jenkins existuje rozšíření, které umí zpracovat výsledky analýzy duplicitního kódu, DRY (zkratka pro Don't Repeat Yourself). Instalaci lze provést pomocí příkazu:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin dry
```

Zároveň je nutné nastavit projekt v Jenkinsu. Zaškrtnutím pole „*Publish duplicate code analysis results*“ v sekci „*Post-build Actions*“ se aktivují další pole. Do pole „*Duplicate code results*“ patří cesta k souboru, který PHPCPD generuje: `build/pmd-cpd.xml`.



Obrázek 20: Zpracování výstupu z analýzy výskytu duplicitního kódu (zdroj: autor)

Po provedení sestavení jsou k dispozici i informace o duplicitním kódu a s dalšími sestaveními je opět možné sledovat trend v jeho množství.

4.8 Nasazení PHPLOC

PHPLOC je nástroj, který počítá různé metriky zdrojových kódů a výsledky umí exportovat do CSV souboru využitelného v Jenkinsu. Detailně byl popsán v kapitole 3.5.

Do skriptu pro sestavení se spouštění PHPLOC doplní přidáním další targetu:

```
<target name="phploc" depends="prepare" description="Analýza PHPLOC">
  <exec command="phploc --log-csv ${project.basedir}/build/phploc.csv
${project.basedir}/application" logoutput="true" />
</target>
```

Do Jenkinse je nutné nainstalovat rozšíření Plot¹⁰⁹, které umí vykreslovat grafy z hodnot získaných v jednotlivých sestaveních.

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin plot
```

Bylo by možné vybrat jednotlivé metriky a postupně je zadat do sekce „Plot build data“, jak naznačuje Obrázek 21.

☒ Plot build data

Delete Plot

Plot group:

Plot title:

Number of builds to include:

Plot y-axis label:

Plot style:

Build Descriptions as labels: ☐

Data series file:

☐ Load data from properties file

☒ Load data from csv file

Include all columns	Include columns by name	Exclude columns by name	Include columns by index	Exclude columns by index
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

CSV Exclusion values:

URL:

Display original csv above plot: ☐

☐ Load data from xml file using xpath

Delete Data Series

Add

A new data series definition

Obrázek 21: Jenkins - konfigurace rozšíření Plot (zdroj: autor)

¹⁰⁹ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Plot+Plugin>

Mnohem vhodnějším způsobem, než manuálně zadávat hodnoty, které budou pro většinu projektů stejné, je využít toho, že integrační server Jenkins ukládá konfiguraci projektu do souboru ve formátu XML, který lze upravovat samostatně.

Stačí soubor `JENKINS_HOME/jobs/zf-tutorial/config.xml` otevřít běžným textovým editorem a nahradit celou konfiguraci rozšíření Plot v sekci `<hudson.plugins.plot.PlotPublisher>` kódem, je k dispozici v Příloze 2 nebo online¹¹⁰ (konfigurace je založená na konfiguraci z [BERGMANN, 2010]).

Poté je ještě nutné oznámit Jenkinsu, že jeho nastavení bylo změněno externě, pomocí voleb „Manage Jenkins“ a „Reload Configuration from Disk“.

4.9 Nasazení PDepend

PDepend je nástroj umožňující získávat různé metriky zdrojových kódů. Jeho detailní popis je možné nalézt v kapitole 3.6.

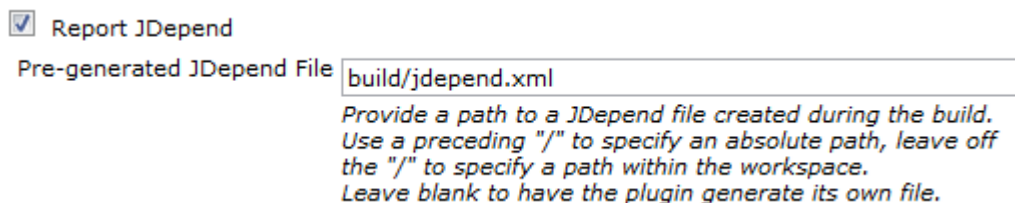
Do skriptu pro sestavení ho lze přidat opět pomocí předpřipraveného tasku v nástroji Phing. Jako parametr mu jsou předány jen soubory, které jsou přímo součástí aplikace, nebudeme metriky získávat pro použité knihovny apod. PDepend generuje jak data pro rozšíření JDepend, tak i dvě přehledová schémata.

```
<target name="pdepend" depends="prepare" description="Analýza nástrojem PDepend">
  <phpdepend>
    <fileset refid="src"/>
    <logger type="jdepend-xml" outfile="${project.basedir}/build/jdepend.xml"/>
    <logger type="jdepend-chart" outfile="${project.basedir}/build/dependencies.svg"/>
    <logger type="overview-pyramid" outfile="${project.basedir}/build/overview-
pyramid.svg"/>
  </phpdepend>
</target>
```

Do Jenkinse je nutné nainstalovat rozšíření JDepend¹¹¹:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin jdepend
```

Po restartu serveru je nutné mu nastavit cestu k souboru, který se generuje během sestavení. V sekci „Post-build Actions“ je nutné zaškrtnout pole „Report JDepend“ a do pole „Pre-generated JDepend File“ zadat cestu: `build/jdepend.xml`. Po provedení sestavení bude možné procházet zpracované informace na stránce s jeho výsledky.



Obrázek 22: Jenkins - konfigurace rozšíření JDepend (zdroj: autor)

¹¹⁰ <https://github.com/mhujer/bakalarka/blob/master/jenkins/plot.xml>

¹¹¹ Viz <https://wiki.jenkins-ci.org/display/JENKINS/JDepend+Plugin>

Na zobrazení vygenerovaných schémat není nutné do Jenkinse instalovat žádné rozšíření. Vzhledem k tomu, že jsou to obrázky ve formátu SVG, které umí dnešní běžně využívané prohlížeče webových stránek bez problémů zobrazit, tak stačí na hlavní stránce projektu kliknout na odkaz „add description“ a do textového pole vložit tento kód:

```


```

Schémata se poté budou zobrazovat na hlavní stránce.

4.10 Nasazení PHPMD

PHPMD (Mess Detector) je nástroj na odhalení potenciálních problémů ve zdrojových kódech. Jeho detailnější popis lze nalézt v kapitole 3.7.

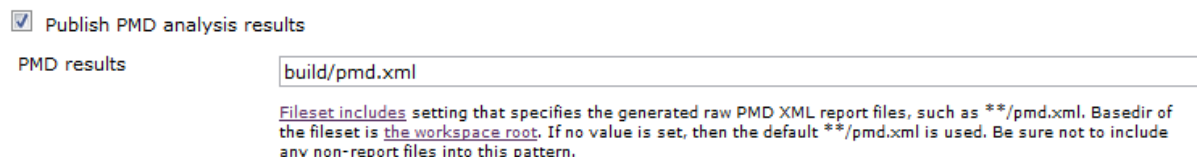
Skript pro sestavení je nutné doplnit o následující kód:

```
<target name="phpmd" depends="prepare" description="PMD analýza">
  <phpmd rulesets="${project.basedir}/phpmd.xml">
    <fileset refid="src"/>
    <fileset refid="tests"/>
    <formatter type="xml" outfile="${project.basedir}/build/pmd.xml"/>
  </phpmd>
</target>
```

Podporu pro zpracování výstupů z PHPMD lze do Jenkinse doplnit pomocí příkazu:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin pmd
```

Po restartu je nutné nastavit načítání výstupů do projektu. Dosáhneme toho vložení cesty k souboru `build/pmd.xml` do pole „PMD results“ v sekci „Publish PMD analysis results“. Výsledky analýzy je poté možné procházet v sekci „PMD Warnings“.



Obrázek 23: Zpracování výstupu z analýzy pomocí nástroje PHPMD (zdroj: autor)

4.11 Nasazení rozšíření Violations

Jednotlivé nástroje zařazené do skriptu pro sestavení generují reporty, které pak jednotlivá rozšíření Jenkinse zpracovávají. Bylo by vhodné mít ucelený pohled na problémy ve zdrojových kódech. Pro Jenkins existuje rozšíření Violations¹¹², které tento problém řeší.

Nejprve je nutné ho nainstalovat:

```
java -jar jenkins-cli.jar -s http://localhost:8080 install-plugin violations
```




Vytváření shrnujících reportů lze aktivovat v nastavení projektu zaškrtnutím pole „Report Violations“ a vyplněním nově zobrazených polí takto:

¹¹² Viz <https://wiki.jenkins-ci.org/display/JENKINS/Violations>

```
checkstyle: build/checkstyle-*.xml
cpd: build/pmd-cpd.xml
pmd: build/pmd.xml
```

Rozšíření zároveň umožňuje nastavit hranice počtů nalezených problémů a na jejich základě zobrazit ikonu u projektu v přehledu projektů.

☒ Report Violations

				XML filename pattern
checkstyle	10	999	999	build/checkstyle-*.xml
codenarc	10	999	999	
cpd	10	999	999	build/pmd-cpd.xml
cpplint	10	999	999	
csslint	10	999	999	
findbugs	10	999	999	
fxcop	10	999	999	
gendarme	10	999	999	
jcreport	10	999	999	
jslint	10	999	999	
pep8	10	999	999	
pmd	10	999	999	build/pmd.xml
pylint	10	999	999	
simian	10	999	999	
stylecop	10	999	999	

Obrázek 24: Jenkins - nastavení rozšíření Violations (zdroj: autor)

4.12 Nasazení PHP_CodeBrowser

Druhým nástrojem, který umožňuje generovat shrnující reporty problémů v kódu je PHP_CodeBrowser¹¹³. Je vyvinutý přímo pro zpracovávání výstupů z analýz zdrojových kódů v jazyce PHP. Výsledkem jeho činnosti je sada HTML souborů, které obsahují hierarchický přehled souborů s vyznačenými problematickými místy.

Nainstalovat ho je možné opět pomocí nástroje PEAR:

```
pear install phpunit/PHP_CodeBrowser
```

Do skriptu pro sestavení ho lze přidat takto:

¹¹³ Viz https://github.com/Mayflower/PHP_CodeBrowser

```
<target name="phpcb" depends="phpcs, phpcpd, phpmd" description="Vygeneruje souhrnný výstup
chyb v kódu pomocí PHP_CodeBrowser">
    <exec command="phpcb --log ${project.basedir}/build --source ${project.basedir} --
output ${project.basedir}/build/code-browser" logoutput="true" />
</target>
```

A podobně jako v případě vygenerované dokumentace z nástroje AgiGen (viz kapitola 4.6) je nutné ho doplnit do nastavení rozšíření HTML Publisher – v nastavení projektu v sekci „*Post-build Actions*“ v části „*Publish HTML reports*“ vytvořit nový řádek. Do pole „*HTML directory to archive*“ vložit `build/code-browser` a titulek („*Report title*“) nastavit například na „*Code Browser*“.

HTML directory to archive	Index page[s]	Report title	Keep past HTML reports	
build/docs	index.html	Apigen Docs	<input type="checkbox"/>	Delete
build/code-browser	index.html	CodeBrowser	<input type="checkbox"/>	Delete

Add

Obrázek 25: Přidání výstupu z nástroje PHP_CodeBrowser (zdroj: autor)

Není nutné instalovat oba nástroje - PHP_CodeBrowser a Violations. Záleží na tom, který bude subjektivně vhodnější pro danou instalaci serveru.

Další možností je nasazení pouze PHP_CodeBrowser a Violations, a nenasazení rozšíření DRY, Checkstyle a PMD.

4.13 Sledování změn v úložišti, automatické spouštění sestavení

V průběhu vytváření skriptu pro sestavení je vhodné spouštět sestavení na integračním serveru ručně - vždy, když je nutné otestovat novou část skriptu.

Oproti tomu v běžném provozu je toto nežádoucí - sestavení by mělo proběhnout automaticky po každé změně uložené ve verzovacím systému a ne záviset na manuálním spuštění. Integrační server Jenkins toto podporuje a umožňuje pravidelně kontrolovat, zda do úložiště nepříbyla nějaká změna. Nastavení je velmi podobné nastavení cronu na systémech linuxového typu (první hvězdička určuje minutu, druhá hodinu, třetí den v měsíci, čtvrtá měsíc a poslední den v týdnu).

Build Triggers

☐ Build after other projects are built

☐ Build periodically

☒ Poll SCM

Schedule

*** **

Obrázek 26: Nastavení kontroly změn každou minutu (zdroj: autor)

V detailu projektu se poté objeví záložka „*Subversion Polling Log*“, která obsahuje informaci o poslední kontrole změn v úložišti.

Polling Log

This page captures the polling log that triggered this build.

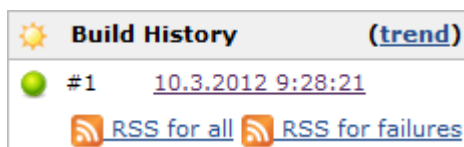
```
Started on Apr 7, 2012 7:22:46 PM
No existing build. Scheduling a new one.
Done. Took 3 ms
Changes found
```

Obrázek 27: Log kontroly změn v úložišti (zdroj: autor)

Jinou možností je vytvořit v úložišti post-commit hook¹¹⁴, který se spustí po uložení změn do úložiště a předá integračnímu serveru informaci, že do úložiště byla uložena nová verze.

4.14 Informace o proběhnutých sestaveních

Je důležité sledovat, jak proběhla jednotlivá sestavení. Jednou z možností je odběr RSS kanálu (viz Obrázek 28).



Obrázek 28: Odebírání výsledků sestavení pomocí RSS kanálů (zdroj: autor)

Nicméně informaci o selhaných sestaveních je potřebné dostat v reálném čase, je tedy vhodnější nastavit zasílání e-mailů. Nejdříve je nutné nastavit SMTP server (viz Obrázek 29).

E-mail Notification

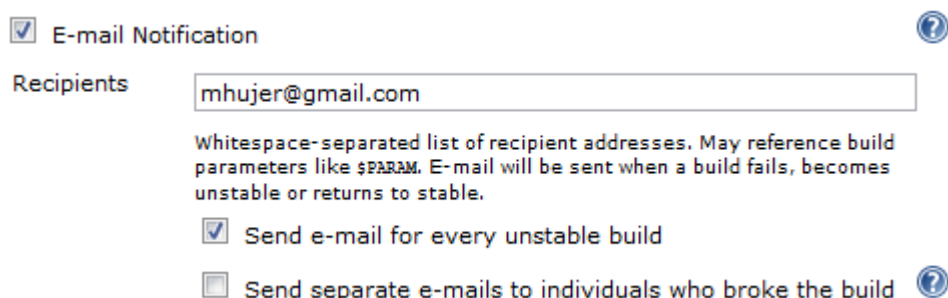
SMTP server	<input type="text" value="localhost"/>	?
Default user e-mail suffix	<input type="text"/>	?
Sender E-mail Address	<input type="text" value="jenkins@localhost"/>	?

☐ Test configuration by sending test e-mail

Obrázek 29: Nastavení SMTP serveru pro Jenkins (zdroj: autor)

¹¹⁴ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Plugin#SubversionPlugin-Postcommithook>

A poté jednotlivé e-mailové adresy v jednotlivých projektech (viz Obrázek 30).

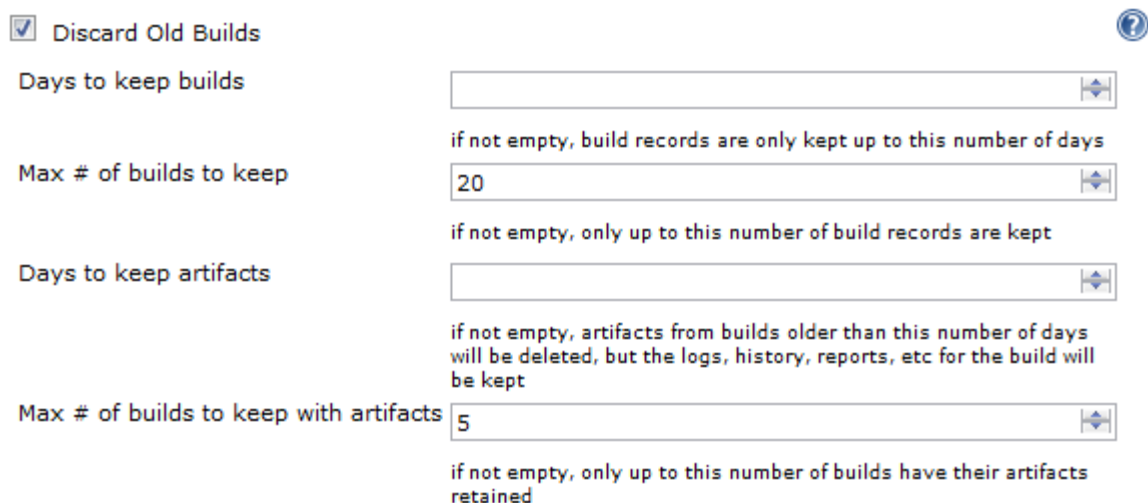


Obrázek 30: Nastavení zasílání e-mailů v projektu (zdroj: autor)

Velkou výhodou integračního serveru Jenkins je jeho popularita. Díky ní existuje mnoho rozšíření¹¹⁵, která umožňují zasílání informací o proběhlých sestaveních nejrůznějšími kanály.

4.15 Odstraňování informací o proběhlých sestaveních

Výsledkem sestavení jsou nejrůznější soubory, takže při mnoha sestaveních by jejich velikost mohla obsadit veškerý volný prostor na serveru. Je proto nutné v sekci „Discard Old Builds“ nastavit, kolik výsledků minulých sestavení bude na serveru ponecháno.



Obrázek 31: Nastavení, kolik výsledků sestavení bude na serveru ponecháno (zdroj: autor)

Konkrétní nastavení samozřejmě závisí na velikosti diskového prostoru na serveru.

¹¹⁵ Viz <https://wiki.jenkins-ci.org/display/JENKINS/Plugins#Plugins-Buildnotifiers>

4.16 Reálné nasazení

Vytvořený skript pro sestavení jsem jen s drobnými úpravami nasadil pro kontinuální integraci systému Shopio na jehož vývoji spolupracuji se společností w3w, s.r.o.¹¹⁶.

Samozřejmě bylo nutné změnit cesty k jednotlivým sadám souborů. Dále jsem v nastavení projektu změnil úložiště GIT na úložiště SVN. Po prvním sestavení jsem zjistil, že souhrnný přehled vygenerovaný nástrojem PHP_CodeBrowser byl značně nepřehledný, takže jsem upravil generování a v současné době se generují tři samostatné reporty - pro PHPMD, PHPCPD a PHP_CodeSniffer.

Kromě toho jsem vyřadil generování API dokumentace, neboť všichni vývojáři jsou zvyklí využívat procházení zdrojových kódů pomocí IDE, takže tato by byla zbytečná a jen by prodlužovala dobu běhu sestavení.

Shodné nastavení jsem poté využil i pro další PHP projekt s velmi podobnou strukturou.

¹¹⁶ Viz <http://w3w.cz/>

Závěr

Hlavním cílem mé bakalářské práce bylo implementovat kontinuální integraci do procesu vývoje webových aplikací v jazyce PHP v malém webovém studiu. Toho jsem se snažil dosáhnout splněním jednotlivých dílčích cílů.

V první kapitole jsem charakterizoval koncept kontinuální integrace a její jednotlivé techniky. Ve druhé jsem vybral vhodný integrační server a popsal jeho základní konfiguraci. V rámci třetí kapitoly jsem vytvořil přehled nástrojů vhodných pro zařazení do procesu kontinuální integrace webových aplikací v jazyce PHP. Ve čtvrté kapitole jsem shrnul způsob vytvoření skriptu pro automatizované sestavení a jeho nasazení na server pro kontinuální integraci.

Jednotlivé dílčí cíle se mi podařilo splnit, díky čemuž jsem splnil i cíl hlavní. Přínos práce spočívá především v souhrnném zpracování tématu kontinuální integrace pro webové aplikace v jazyce PHP, které zatím chybělo. Dalším přínosem je postup na samotné nasazení, který je podle mého názoru v některých aspektech dokonalejší, než již popsané způsoby nasazení a který jsem ověřil v praxi.

Nečekaným přínosem tvorby práce bylo nalezení a nahlášení různých chyb v použitých nástrojích. Na několik chyb¹¹⁷ jsem narazil v nástroji ApiGen, dvě¹¹⁸ drobné chyby jsem opravil v nástroji Phing a během testování nástroje na kontrolu standardů pro psaní kódu jsem opravil jejich různá porušení¹¹⁹ v připravovaném Zend Frameworku verze 2. Původně jsem také plánoval vytvořit modul do nástroje Phing pro snadnější konfiguraci analýzy pomocí nástroje PHPLoc, ale při analýze jsem zjistil, že takový modul už existuje¹²⁰, jen není dosud součástí distribuce nástroje Phing. Kontaktoval¹²¹ jsem tedy proto autora a ten modul zaslal jako opravu do nástroje Phing. Dále jsem ještě upravil nástroj PHP_CodeBrowser a autorovi zaslal patch¹²², který přidává možnost reportování jen souborů s chybami.

Až delší čas ukáže, zda mnou provedené nasazení kontinuální integrace přinese očekávané výsledky – tedy kvalitnější kód s menším množstvím chyb.

Zajímavým nástrojem, který usnadňuje automatizaci jednotlivých projektů je PHP Project Wizard¹²³, který [BERGMANN, 2011b] označuje jako automatizaci automatizace. Nicméně nástroj generuje konfiguraci právě podle [BERGMANN, 2011b], takže pro hromadné nasazování automatizace, jak je popsána v této práci, by bylo nutné nástroj upravit.

¹¹⁷ Viz <https://github.com/apigen/apigen/issues/113>, <https://github.com/apigen/apigen/issues/114> a <https://github.com/apigen/apigen/issues/133>

¹¹⁸ Viz <https://github.com/phingofficial/phing/pull/100> a <https://github.com/phingofficial/phing/pull/105>

¹¹⁹ Viz <https://github.com/zendframework/zf2/pull/992>, <https://github.com/zendframework/zf2/pull/993> a <https://github.com/zendframework/zf2/pull/996>

¹²⁰ Viz <https://github.com/raphaelstolt/phploc-phing>

¹²¹ Viz <https://github.com/raphaelstolt/phploc-phing/issues/7>

¹²² Viz https://github.com/Mayflower/PHP_CodeBrowser/pull/12

¹²³ <https://github.com/sebastianbergmann/php-project-wizard>

Terminologický slovník

Termín	Anglický termín	Zkratka	Význam [zdroj]
Systém pro správu verzí	Version control system	VCS	Nástroj umožňující snadno ukládat a procházet historii změn v souborech, které jsou součástí aplikace. [vlastní definice autora]
Sestavení	Build		Jednotlivé kroky pro převod zdrojových kódů do funkční aplikace [vlastní definice autora]
Úložiště	Repository		Místo, kam verzovací systém ukládá data [vlastní definice autora]
	Checkout		Proces vytvoření lokální kopie dat z verzovacího systému [vlastní definice autora]
	Commit		Uložení změn do verzovacího systému. [vlastní definice autora]
	hook		Událost, která je zavolána na úložišti při různých situacích. Typický je například pre-commit hook, který se spustí před uložením změn do verzovacího systému (a může tedy tuto změnu zamítnout). [vlastní definice autora]
Refaktoring	Refactoring		Změna (zlepšení) struktury zdrojového kódu při zachování funkcionality [vlastní definice autora]
	scm polling		Pravidelné zjišťování, zda do verzovacího systému nepříbyly nějaké změny. [vlastní definice autora]
Standardy pro psaní kódu	coding standards		Soubor pravidel, pro formátování souboru se zdrojovým kódem (viz kapitola 1.4.3)
Vývojové prostředí	Integrated development environment	IDE	Vývojové prostředí je software, který většinou obsahuje editor a různé podpůrné nástroje pro usnadnění vývoje software [vlastní definice autora]
Vývoj řízený testy	Test-driven development	TDD	Jde o proces vývoje software, kdy je nejprve vytvořen automatizovaný test a teprve poté samotná implementace. [vlastní definice autora]
Extrémní programování	Extreme programming	EP	Sada agilních technik pro zkvalitnění vývoje software (viz [BECK, 2005]) [vlastní definice autora]
	Fork		Oddělená větev vývoje software, často využíváno v souvislosti s DCVS. [vlastní definice autora]
	Pull Request	PR	Pull Request je požadavek na zahrnutí změn do hlavního vývojového stromu aplikace typicky využívaný na server GitHub.com [vlastní definice autora]

Seznam použité literatury a zdrojů

BECK, K. *Test-driven Development: By Example*. Boston: Addison-Wesley Professional, 2002. ISBN: 0321146530.

BECK, K. *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley, 2005. ISBN: 0321278658.

BERGMANN, S. *PHPUnit Pocket Guide*. O'Reilly Media, 2005. ISBN 978-0596101039.

BERGMANN, S. Benchmark of PHP Branches 3.0 through 5.3-CVS [online]. 07. 02. 2008 [cit. 2011-07-20]. Dostupné z: <http://sebastian-bergmann.de/archives/745-Benchmark-of-PHP-Branches-3.0-through-5.3-CVS.html>

BERGMANN, S. Template for Jenkins Jobs for PHP Projects [online]. © 2010, verze 2012-03-15 [cit. 2012-04-07]. Dostupné z: <http://jenkins-php.org/>

BERGMANN, S. a S. PRIEBSCH. *Real-World Solutions for Developing High-Quality PHP Frameworks and Applications*. Indianapolis: Wrox, 2011. ISBN 978-0470872499.

BERGMANN, S. *ZendCon Sessions Episode 039: Continuous Inspection and Integration of PHP Projects* [online]. © 2011, verze February 15, 2011 [cit. 2012-04-09]. Dostupné z: <http://devzone.zend.com/1865/zendcon-sessions-episode-039-continuous-inspection-and-integration-of-php-projects/>

BERGMANN, S. *Integrating PHP Projects with Jenkins*. Sebastopol: O'Reilly Media, Inc., 2011b. ISBN 978-1-4493-0943-5.

CRAG, R. D. a S. P. JASKIEL. *Systematic software testing*. Artech House, 2002. ISBN 1580535089.

DEMARCO, T. *Controlling software projects: management, measurement & estimation*. Michigan: Yourdon Press, 1982. ISBN 9780917072321.

DUVALL, P., S. MATYAS a A. GLOVER. *Continuous Integration: Improving Software Quality and Reducing Risk*. Upper Saddle River, NJ: Addison-Wesley Professional, 2007. ISBN: 978-0321336385.

FOWLER, M. *Continuous Integration* [online]. 1. 5. 2006 [cit. 2011-07-18]. Dostupné z: <http://martinfowler.com/articles/continuousIntegration.html>

HUMBLE, J. a D. FARLEY. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Upper Saddle River, NJ: Addison-Wesley, 2010. ISBN: 9780321601919.

CHACON, S. *Pro Git*. Apress, 2009. ISBN 978-1430218333.

KOSKELA, L. *JavaRanch Journal Introduction to Code Coverage* [online]. © 2004 [cit. 2012-03-31]. Dostupné z: <http://www.javaranch.com/journal/2004/01/IntroToCodeCoverage.html>

LAKEWORKS. The Scrum project management method. [online]. © 2009 [cit. 2012-04-08]. Dostupné z: http://commons.wikimedia.org/wiki/File:Scrum_process.svg

- LANZA, M. a R. MARINESCU. *Object-Oriented Metrics in Practice. Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-oriented Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 978-3540244295.
- LOUGHRAN, S. a H. ERIK. *Ant in Action*. Greenwich: Manning, 2007. ISBN 1-932394-80-X.
- MALL, R. *Fundamentals of Software Engineering*. 2. Prentice-Hall of India, 2004. ISBN 9788120324459.
- MARTIN, R. *OO Design Quality Metrics: An Analysis of Dependencies*. 1994
- MARTIN, R. C. *Object Mentor Coding Standards* [online]. 18. 4. 2007 [cit. 2012-03-17]. Dostupné z: <http://blog.objectmentor.com/articles/2007/04/18/coding-standards>
- MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Boston: Prentice Hall, 2008. ISBN: 978-0132350884.
- MCCABE, T. J. A Complexity Measure. *IEEE Transactions on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society, 1976,(4), 308-320. ISSN 0098-5589.
- MCCABE JR., T. *Software Quality Metrics to Identify Risk* [prezentace]. 2008, verze Nov. 2008. Dostupné z: <http://www.mccabe.com/ppt/SoftwareQualityMetricsToIdentifyRisk.ppt>
- MCCONNELL, S. *Odhadování softwarových projektů*. Brno: Computer Press, 2006. ISBN 80-251-1240-3.
- MCCONNELL, S. 10x Software Development *Measuring Productivity of Individual Programmers* [online]. © 2008 [cit. 2012-04-01]. Dostupné z: <http://forums.construx.com/blogs/stevemcc/archive/2008/04/09/measuring-productivity-of-individual-programmers.aspx>
- MYERS, G. J. *The Art of Software Testing, Second Edition*. Wiley, 2004. ISBN 978-0471469124.
- MYTTON, D. SitePoint *Why You Need Coding Standards* [online]. 23. 9. 2004 [cit. 2012-03-17]. Dostupné z: <http://www.sitepoint.com/coding-standards/>
- NEJMEH, B. A. NPATH: a measure of execution path complexity and its applications. *Communications of the ACM*. New York, NY, USA: ACM, 1988,(31), 188--200. ISSN 0001-0782.
- THE PHING PROJECT *Phing 2.4 - User Guide* [online]. © 2011, verze 1b3461f9f8e5da4becf97e7c298e209f1e6c8c6c [cit. 2011-03-29]. Dostupné z: <http://www.phing.info/docs/guide/stable/>
- PICHLER, M. Cyclomatic Complexity *PHP Depend* [online]. © 2011, verze 24.1.2012 [cit. 2012-04-02]. Dostupné z: <http://pdepend.org/documentation/software-metrics/cyclomatic-complexity.html>
- SCHROEDER, K. a J. MARIEN. *Using Jenkins/Hudson for Continuous Builds* [online]. © 2011, verze March 29, 2011 [cit. 2012-04-09]. Dostupné z: <http://www.zend.com/en/resources/webinars/>
- SMART, J. F. *Jenkins: The Definitive Guide*. Sebastopol: O'Reilly Media, 2011. ISBN 978-1-4493-8959-8.

PRSKAVEC, L. phpDepend and php frameworks *Top Topic?* [online]. © 2009, verze 21 Mar 2009 [cit. 2012-04-03]. Dostupné z: <http://blog.prskavec.eu/2009/03/21/phpdepend-and-php-frameworks/>

VAN DAM, M. Zend.com *Improving QA on PHP development projects* [online]. 14. 4. 2011 [cit. 2012-02-29]. Dostupné z: <http://www.zend.com/en/resources/webinars/#QAPHP>

VAN DAM, M. *ZendCon Sessions Episode 043: Improving QA on PHP Projects* [online]. © 2011, verze March 11, 2011 [cit. 2012-04-09]. Dostupné z: <http://devzone.zend.com/1881/zendcon-sessions-episode-043-improving-qa-on-php-projects/>

WOOD, J. *Why Code Coverage Alone Doesn't Mean Squat* [online]. © 2008 [cit. 2012-03-31]. Dostupné z: <http://johnpwood.net/2008/12/30/why-code-coverage-alone-doesnt-mean-squat/>

WIKIPEDIA CONTRIBUTORS. Wikipedia, The Free Encyclopedia *Continuous integration* [online]. 20. 07. 2011 [cit. 2011-07-27]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Continuous_integration&oldid=440518476

ZANDSTRA, M. *PHP Objects, Patterns, and Practice*. Apress, 2007. ISBN 978-1590599099.

ZIKMUND, Š. *Diplomová práce: Deployment aplikací v PHP*. Praha: KIT FIS VŠE, 2011

Seznam obrázků, tabulek a ukázek kódu

Seznam obrázků

Obrázek 1: Schéma vodopádového modelu vývoje software [MALL, 2004]	10
Obrázek 2: Schéma průběhu vývoje software pomocí metodiky Scrum [LAKEWORKS, 2009]	11
Obrázek 3: "V" model testování software [CRAG, 2002]	16
Obrázek 4: Hlavní stránka Jenkinsu po prvním spuštění (zdroj: autor)	24
Obrázek 5: Jenkins - nastavení anglického prostředí (zdroj: autor)	25
Obrázek 6: Kontrola syntaktických chyb v editoru Zend Studio 8 (zdroj: autor)	27
Obrázek 7: Pyramida s přehledem metrik kódu pro Zend Framework 1.11.11 (zdroj: autor)	32
Obrázek 8: Graf míry abstrakce a stability pro Zend Framework 1.11.11 (zdroj: autor)	32
Obrázek 9: Ukázka dokumentace vygenerované pomocí PhpDocumentor (zdroj: autor)	35
Obrázek 10: Ukázka dokumentace vygenerované nástrojem DocBlox (zdroj: autor)	36
Obrázek 11: Ukázka dokumentace vygenerovaná nástrojem ApiGen (zdroj: autor)	38
Obrázek 12: Vytvoření nového projektu na serveru Jenkins (zdroj: autor)	43
Obrázek 13: Nastavení cesty k repositáři s projektem (zdroj: autor)	43
Obrázek 14: Nastavení skriptu pro sestavení aplikace (zdroj: autor)	44
Obrázek 15: Úspěšné sestavení v Jenkinsu (zdroj: autor)	44
Obrázek 16: Konfigurace nástroje PHPUnit na serveru Jenkins (zdroj: autor)	46
Obrázek 17: Nastavení zpracování výstupu z nástroje PHP_CodeSniffer (zdroj: autor)	47
Obrázek 18: Doplnění odkazů na HTML výstupy (zdroj: autor)	47
Obrázek 19: Odkaz na vygenerovanou dokumentaci (zdroj: autor)	48
Obrázek 20: Zpracování výstupu z analýzy výskytu duplicitního kódu (zdroj: autor)	48
Obrázek 21: Jenkins - konfigurace rozšíření Plot (zdroj: autor)	49
Obrázek 22: Jenkins - konfigurace rozšíření JDepend (zdroj: autor)	50
Obrázek 23: Zpracování výstupu z analýzy pomocí nástroje PHPMD (zdroj: autor)	51
Obrázek 24: Jenkins - nastavení rozšíření Violations (zdroj: autor)	52
Obrázek 25: Přidání výstupu z nástroje PHP_CodeBrowser (zdroj: autor)	53
Obrázek 26: Nastavení kontroly změn každou minutu (zdroj: autor)	53
Obrázek 27: Log kontroly změn v úložišti (zdroj: autor)	54
Obrázek 28: Odebírání výsledků sestavení pomocí RSS kanálů (zdroj: autor)	54
Obrázek 29: Nastavení SMTP serveru pro Jenkins (zdroj: autor)	54
Obrázek 30: Nastavení zasílání e-mailů v projektu (zdroj: autor)	55
Obrázek 31: Nastavení, kolik výsledků sestavení bude na serveru ponecháno (zdroj: autor)	55

Seznam tabulek

Tabulka 1: Cyklomatická složitost a riziko spolehlivosti [MCCABE JR., 2008]	19
Tabulka 2: Cyklomatická složitost a pravděpodobnost chybné opravy [MCCABE JR., 2008]	19
Tabulka 3: Aktivita u projektů Hudson a Jenkins na serveru GitHub.com (k 25. 2. 2012)	22

Seznam ukázek kódu

Kód 1: Míra pokrytí kódu testy	17
Kód 2: Soubor test1-error.php	27
Kód 3: Chybový výstup z PHP Lint	27
Kód 4: Zdrojový kód souboru test2-ok.php	27
Kód 5: Výstup z PHP Lint, když je kód v pořádku	27

Kód 6: Soubor test3-trait.php	28
Kód 7: Výstup z PHP Lint (verze PHP 5.3) na souboru test3-trait.php	28
Kód 8: Chybový výstup z PHP Lint (verze PHP 5.4.0alpha2) na souboru test3-trait.php	28
Kód 9: Příklad šablony v PHTML souboru (test4-template.phtml)	29
Kód 10: Ukázka použití PHP CPD	30
Kód 11: Ukázkový výstup z aplikace phploc.....	31
Kód 12: Ukázka možnosti využití PHPDoc (soubor phpdoc01.php)	34

Příloha 1: Skript pro sestavení ukázkové aplikace (build.xml)

```
<project name="zf-tutorial" default="build">
  <target name="build" depends="prepare, lint, phpunit, phpcs, apigen, phpcpd, phplloc,
pdepend, phpmd, phpcb" description="Meta target, spouští ostatní targety"/>

  <!-- Definice adresářů, níže na ně budu jen odkazovat -->
  <fileset id="src" dir="${project.basedir}/application">
    <include name="**/*.php"/>
  </fileset>
  <fileset id="templates" dir="${project.basedir}/application">
    <include name="**/*.phtml"/>
  </fileset>
  <fileset id="tests" dir="${project.basedir}/tests">
    <include name="**/*.php"/>
  </fileset>
  <fileset id="library" dir="${project.basedir}/library">
    <include name="**/*.php"/>
  </fileset>

  <target name="cleanup" description="Vyčistění workspace">
    <delete dir="${project.basedir}/build"/>
  </target>

  <target name="prepare" depends="cleanup" description="Příprava workspace">
    <mkdir dir="${project.basedir}/build"/>
  </target>

  <target name="lint" description="Kontrola pomocí PHP Lint">
    <phplint haltonfailure="true" level="info">
      <fileset refid="src"/>
      <fileset refid="templates"/>
      <fileset refid="tests"/>
      <fileset refid="library"/>
    </phplint>
  </target>

  <target name="phpunit" depends="prepare" description="PHPUnit testy">
    <phpunit printsummary="true" haltonfailure="true" haltonerror="true">
      <formatter todir="${project.basedir}/build" outfile="phpunit-report.xml"
type="xml"/>
      <batchtest>
        <fileset refid="tests"/>
      </batchtest>
    </phpunit>
  </target>

  <target name="phpcs" depends="prepare" description="Kontrola standardů pro psaní kódu">
    <phpcodesniffer standard="Zend">
      <fileset refid="src"/>
      <fileset refid="tests"/>
      <formatter type="default" usefile="false"/>
      <formatter type="checkstyle" outfile="${project.basedir}/build/checkstyle-
phpcs.xml"/>
    </phpcodesniffer>
  </target>

  <target name="apigen" depends="prepare" description="Generování dokumentace">
    <apigen source="${project.basedir}/application"
destination="${project.basedir}/build/docs" report="${project.basedir}/build/checkstyle-
apigen.xml"/>
  </target>
</project>
```



```

<target name="phpcpd" depends="prepare" description="Kontrola CPD">
    <phpcpd>
        <fileset refid="src"/>
        <fileset refid="tests"/>
        <formatter type="pmd" outfile="${project.basedir}/build/pmd-cpd.xml"/>
    </phpcpd>
</target>

<target name="phploc" depends="prepare" description="Analýza PHPLOC">
    <exec command="phploc --log-csv ${project.basedir}/build/phploc.csv
${project.basedir}/application" logoutput="true" />
</target>

<target name="pdepend" depends="prepare" description="Analýza nástroj PDepend">
    <phpdepend>
        <fileset refid="src"/>
        <logger type="jdepend-xml" outfile="${project.basedir}/build/jdepend.xml"/>
        <logger type="jdepend-chart"
outfile="${project.basedir}/build/dependencies.svg"/>
        <logger type="overview-pyramid" outfile="${project.basedir}/build/overview-
pyramid.svg"/>
    </phpdepend>
</target>

<target name="phpmd" depends="prepare" description="PMD analýza">
    <phpmd rulesets="${project.basedir}/phpmd.xml">
        <fileset refid="src"/>
        <fileset refid="tests"/>
        <formatter type="xml" outfile="${project.basedir}/build/pmd.xml"/>
    </phpmd>
</target>

<target name="phpcb" depends="phpcs, phpcpd, phpmd" description="Vygeneruje souhrnný
výstup chyb v kódu pomocí PHP_CodeBrowser">
    <exec command="phpcb --log ${project.basedir}/build --source ${project.basedir} --
output ${project.basedir}/build/code-browser" logoutput="true" />
</target>
</project>

```

Příloha 2: Úsek XML souboru pro nastavení rozšíření Plot

```
<hudson.plugins.plot.PlotPublisher>
  <plots>
    <hudson.plugins.plot.Plot>
      <title>A - Lines of code</title>
      <yaxis>Lines of Code</yaxis>
      <series>
        <hudson.plugins.plot.CSVSeries>
          <file>build/phploc.csv</file>
          <label></label>
          <fileType>csv</fileType>
          <strExclusionSet>
            <string>Lines of Code (LOC)</string>
            <string>Comment Lines of Code (CLOC)</string>
            <string>Non-Comment Lines of Code (NCLOC)</string>
          </strExclusionSet>
          <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
          <exclusionValues>Lines of Code (LOC),Comment Lines of Code (CLOC),Non-Comment
Lines of Code (NCLOC)</exclusionValues>
          <url></url>
          <displayTableFlag>>false</displayTableFlag>
        </hudson.plugins.plot.CSVSeries>
      </series>
      <group>phploc</group>
      <numBuilds>100</numBuilds>
      <csvFileName>123.csv</csvFileName>
      <csvLastModification>0</csvLastModification>
      <style>line</style>
      <useDescr>>false</useDescr>
    </hudson.plugins.plot.Plot>
    <hudson.plugins.plot.Plot>
      <title>B - Structures</title>
      <yaxis>Count</yaxis>
      <series>
        <hudson.plugins.plot.CSVSeries>
          <file>build/phploc.csv</file>
          <label></label>
          <fileType>csv</fileType>
          <strExclusionSet>
            <string>Functions</string>
            <string>Classes</string>
            <string>Namespaces</string>
            <string>Files</string>
            <string>Directories</string>
            <string>Methods</string>
            <string>Interfaces</string>
            <string>Constants</string>
            <string>Anonymous Functions</string>
          </strExclusionSet>
          <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
          <exclusionValues>Directories,Files,Namespaces,Interfaces,Classes,Methods,Functions,Anonymous
s Functions,Constants</exclusionValues>
          <url></url>
          <displayTableFlag>>false</displayTableFlag>
        </hudson.plugins.plot.CSVSeries>
      </series>
      <group>phploc</group>
      <numBuilds>100</numBuilds>
      <csvFileName>1107599928.csv</csvFileName>
      <csvLastModification>0</csvLastModification>
      <style>line</style>
    </hudson.plugins.plot.Plot>
  </plots>
</hudson.plugins.plot.PlotPublisher>
```

```

    <useDescr>false</useDescr>
</hudson.plugins.plot.Plot>
<hudson.plugins.plot.Plot>
  <title>C - Testing</title>
  <yaxis>Count</yaxis>
  <series>
    <hudson.plugins.plot.CSVSeries>
      <file>build/phploc.csv</file>
      <label></label>
      <fileType>csv</fileType>
      <strExclusionSet>
        <string>Functions</string>
        <string>Classes</string>
        <string>Methods</string>
        <string>Test Classes</string>
        <string>Test Methods</string>
      </strExclusionSet>
      <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
      <exclusionValues>Classes,Methods,Functions,Test Classes,Test
Methods</exclusionValues>
      <url></url>
      <displayTableFlag>false</displayTableFlag>
    </hudson.plugins.plot.CSVSeries>
  </series>
  <group>phploc</group>
  <numBuilds>100</numBuilds>
  <csvFileName>174807245.csv</csvFileName>
  <csvLastModification>0</csvLastModification>
  <style>line</style>
  <useDescr>false</useDescr>
</hudson.plugins.plot.Plot>
<hudson.plugins.plot.Plot>
  <title>D - Types of Classes</title>
  <yaxis>Count</yaxis>
  <series>
    <hudson.plugins.plot.CSVSeries>
      <file>build/phploc.csv</file>
      <label></label>
      <fileType>csv</fileType>
      <strExclusionSet>
        <string>Abstract Classes</string>
        <string>Classes</string>
        <string>Concrete Classes</string>
      </strExclusionSet>
      <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
      <exclusionValues>Classes,Abstract Classes,Concrete Classes</exclusionValues>
      <url></url>
      <displayTableFlag>false</displayTableFlag>
    </hudson.plugins.plot.CSVSeries>
  </series>
  <group>phploc</group>
  <numBuilds>100</numBuilds>
  <csvFileName>594356163.csv</csvFileName>
  <csvLastModification>0</csvLastModification>
  <style>line</style>
  <useDescr>false</useDescr>
</hudson.plugins.plot.Plot>
<hudson.plugins.plot.Plot>
  <title>E - Types of Methods</title>
  <yaxis>Count</yaxis>
  <series>
    <hudson.plugins.plot.CSVSeries>
      <file>build/phploc.csv</file>
      <label></label>
      <fileType>csv</fileType>

```

```

        <strExclusionSet>
          <string>Methods</string>
          <string>Static Methods</string>
          <string>Non-Static Methods</string>
          <string>Public Methods</string>
          <string>Non-Public Methods</string>
        </strExclusionSet>
        <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
        <exclusionValues>Methods,Non-Static Methods,Static Methods,Public Methods,Non-
Public Methods</exclusionValues>
        <url></url>
        <displayTableFlag>>false</displayTableFlag>
      </hudson.plugins.plot.CSVSeries>
    </series>
    <group>phploc</group>
    <numBuilds>100</numBuilds>
    <csvFileName>1019987862.csv</csvFileName>
    <csvLastModification>0</csvLastModification>
    <style>line</style>
    <useDescr>>false</useDescr>
  </hudson.plugins.plot.Plot>
  <hudson.plugins.plot.Plot>
    <title>F - Types of Constants</title>
    <yaxis>Count</yaxis>
    <series>
      <hudson.plugins.plot.CSVSeries>
        <file>build/phploc.csv</file>
        <label></label>
        <fileType>csv</fileType>
        <strExclusionSet>
          <string>Class Constants</string>
          <string>Global Constants</string>
          <string>Constants</string>
        </strExclusionSet>
        <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
        <exclusionValues>Constants,Global Constants,Class Constants</exclusionValues>
        <url></url>
        <displayTableFlag>>false</displayTableFlag>
      </hudson.plugins.plot.CSVSeries>
    </series>
    <group>phploc</group>
    <numBuilds>100</numBuilds>
    <csvFileName>217648577.csv</csvFileName>
    <csvLastModification>0</csvLastModification>
    <style>line</style>
    <useDescr>>false</useDescr>
  </hudson.plugins.plot.Plot>
  <hudson.plugins.plot.Plot>
    <title>G - Average Length</title>
    <yaxis>Average Non-Comment Lines of Code</yaxis>
    <series>
      <hudson.plugins.plot.CSVSeries>
        <file>build/phploc.csv</file>
        <label></label>
        <fileType>csv</fileType>
        <strExclusionSet>
          <string>Average Method Length (NCLOC)</string>
          <string>Average Class Length (NCLOC)</string>
        </strExclusionSet>
        <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
        <exclusionValues>Average Class Length (NCLOC),Average Method Length
(NCLOC)</exclusionValues>
        <url></url>
        <displayTableFlag>>false</displayTableFlag>
      </hudson.plugins.plot.CSVSeries>
    </series>
  </hudson.plugins.plot.Plot>
</hudson.plugins.plot.Plot>

```

```

    </series>
    <group>phploc</group>
    <numBuilds>100</numBuilds>
    <csvFileName>523405415.csv</csvFileName>
    <csvLastModification>0</csvLastModification>
    <style>line</style>
    <useDescr>>false</useDescr>
  </hudson.plugins.plot.Plot>
  <hudson.plugins.plot.Plot>
    <title>H - Relative Cyclomatic Complexity</title>
    <yaxis>Cyclomatic Complexity by Strcuture</yaxis>
    <series>
      <hudson.plugins.plot.CSVSeries>
        <file>build/phploc.csv</file>
        <label></label>
        <fileType>csv</fileType>
        <strExclusionSet>
          <string>Cyclomatic Complexity / Lines of Code</string>
          <string>Cyclomatic Complexity / Number of Methods</string>
        </strExclusionSet>
        <inclusionFlag>INCLUDE_BY_STRING</inclusionFlag>
        <exclusionValues>Cyclomatic Complexity / Lines of Code,Cyclomatic Complexity /
Number of Methods</exclusionValues>
        <url></url>
        <displayTableFlag>>false</displayTableFlag>
      </hudson.plugins.plot.CSVSeries>
    </series>
    <group>phploc</group>
    <numBuilds>100</numBuilds>
    <csvFileName>186376189.csv</csvFileName>
    <csvLastModification>0</csvLastModification>
    <style>line</style>
    <useDescr>>false</useDescr>
  </hudson.plugins.plot.Plot>
</plots>
</hudson.plugins.plot.PlotPublisher>

```