Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	zimní 2014/2015
Autoři – jméno, příjmení, xname	Heřmanský, Michal, xherm22 Mladý, Lukáš, xmlal00, Müller, Petr, xmulp06 Němec, Milan, xnemm34
Téma	Nepřetržitá integrace při vývoji webových aplikací
Datum odevzdání	19. 12. 2014

Abstrakt

Nepřetržitá integrace (CI) je trendem v oblasti vývoje softwaru. Z průzkumů vyplývá, že ačkoliv je CI široce implementováno ve vývojářských týmech, v oblasti webových aplikací je přejímání této agilní metody stále v začátcích. Práce se zabývá praktickými nástroji CI a metodami jejich aplikace. Popisuje frontend i backend částí webových aplikací včetně potřeby udržování jednotného stylu kódu. Práce uvádí porovnání integračních serverů a uvádí konkrétní řešení integrace nástrojů pro podporu CI.

Klíčová slova

Nepřetržitá integrace, integrační server, backend, frontend, sestavení, testování, nasazení

Obsah

1. Úvod

Při vývoji softwaru dodržuje vývojový tým pro úspěšné vytvoření produktu stanovenou metodiku, její techniky a doporučení. Snaží se co nejrychleji a nejlevněji splnit stanovené požadavky s cílem dosažení nejvyšší kvality výstupu. A právě tato práce se zabývá technikou (praktikou) využívanou v agilních metodách umožňují vývojářům dosáhnout požadovaných cílů. Tato praktika se nazývá nepřetržitá integrace (v angličtině Countinuous Integration, dále jen CI) a tato práce se hlavně zaměřuje na použití CI při vývoji webových aplikací.

Cílem práce je definovat CI a používané nástroje se zaměřením na praktické a programátorské činnosti v prostředí vývoje webových aplikací.

Na začátku práce jsou definovány principy CI a jsou představeny dostupné nástroje a přínosy, kterých by vývojový tým měl dosáhnout při integrování CI do svého pracovního postupu. Jsou uvedeny statistiky používání CI k roku 2013. Nástroje jsou dále porovnány mezi sebou a závěr kapitoly uvádí doporučení konkrétních nástrojů.

Třetí kapitola je především zaměřena na backendovou část aplikace. Je zde prakticky ukázáno sestavení integračního skriptu a spouštění testů. Dále je nastíněn problém s ukazatelem pokrytí kódu testy a způsob řešení závislostí na modulech třetích stran. V závěru kapitoly jsou popsány možnosti nasazení nové verze aplikace.

Další kapitola pojednává o uživatelské, frontend části aplikace. Jsou zde popsána, především z praktického hlediska, řešení optimalizace aplikace a možnosti automatizace. Ke všem jednotlivým částem jsou doporučeny odpovídající nástroje.

Pátá kapitola popisuje problém s udržením jednotného stylu psaní kódu při vývoji a následně dává v příklad konkrétní nástroje, kterými je možné tento problém řešit.

Šestá kapitola představuje praktickou ukázku integrovaného řešení CI na groupwarovém nástroji JIRA a integračním serveru Bamboo od společnosti Atlassian.

2. Obecné představení nástrojů pro podporu nepřetržité integrace a jejich srovnání

Kapitola obsahuje popis principů a přínosů nástrojů podporující CI. Na základě popsané metodiky je sestaven seznam několika v současnosti nejrozšířenějších nástrojů a vypracován přehled jejich základních vlastností. V závěru kapitoly jsou diskutovány výsledky porovnání nástrojů.

Princip nástrojů CI

CI se řadí do jedné z dvanácti praktik agilní metodiky extrémního programování, avšak samostatnou praktiku můžeme zařadit a používat i v jiných agilních metodách.

Nepřetržitou integrace lze definovat jako praktiku softwarového vývoje, ve které jednotliví členové týmu svou práci integrují často, obvykle nejméně denně. Každá integrace je verifikována automatizovaným buildem (dále jen sestavením) včetně testu, za účelem odhalení integračních chyb co nejrychleji. (Fowler, 2006)

Toho je dosahováno díky rychlému poskytnutí zpětné vazby v případě nalezení chyby. Tak může být vadná část kódu velmi rychle odhalena a opravena. Nástroje pro nepřetržitou integraci slouží především k zautomatizování činností, které jsou k tomu nezbytné.

Před tím, než je vůbec možné začít s nepřetržitou integrací, je potřeba zajistit tři základní prerekvizity:

- správa verzí
- automatizace sestavení
- přijetí praktiky týmem

Jakkoliv je nepřetržitá integrace především praktikou a nikoli nástrojem, a tudíž je spolupráce ze strany týmu nezbytná, bez vhodného nástroje se nelze obejít. (HUMBLE a FARLEY, 2010)

Přínosy nástrojů CI

Dle Humble a Farley (2010) existuje na trhu celá řada nástrojů, které poskytují infrastrukturu pro automatizaci sestavení a procesů testování kódu. Základním přínosem těchto nástrojů pro podporu nepřetržité integrace je kontrola systému pro správu verzí kódu, zdali nepřibyl nový commit (nová část kódu). K takové kontrole dochází obvykle každých pár minut (Duvall, Matyas a Glover 2007, s. 250). Pokud je detekován nový commit, pak má nástroj za úkol zkontrolovat poslední verzi software, spustit sestavovací skript, spustit testy a podat hlášení o výsledku.

Nástroje pro podporu nepřetržité integrace tak mají dvě základní komponenty, které zajišťují následující funkcionalitu:

1. dlouhodobě běžící proces, který provozuje jednoduché (výše popsané) workflow (pracovní postup) v pravidelných intervalech
2. zobrazení výsledků těchto procesů, vyrozumění o jejich úspěšném či neúspěšném výsledku a zajištění přístupu k reportům

Statistika používání CI ve světě

Podle průzkumu "8th Annual State of Agile Survey" (Versionone, 2013) mezi 3,501 respondenty z různých oblastí vývoje softwaru je CI na mírném vzestupu. Oproti roku 2012 využívání tohoto přístupu vzrostlo o 2 % na 56 %. Z průzkumu vyplývá, že v rámci agilních přístupů je CI na žebříčku priorit vývojářů na podobné pozici jako metoda jednotného psaní zdrojového kódu. Je třeba zdůraznit, že průzkum probíhal mezi vývojáři obecně. Jak je uvedeno v této práci dále, situace v praxi

vývoje webových aplikací je jiná. V této oblasti ze CI využívá spíše sporadicky a vývojářské týmy ji do svých postupů teprve integrují.

DevOps

Složenina slov Developer (Dev) & Operations (Ops). Tento přístup k vývoji zdůrazňuje komunikaci mezi vývojáři a odborníky na informační technologie. Cílem je maximalizovat předvídatelnost, účinnost, bezpečnost a udržovatelnost provozních procesů. DevOps podporují provázanost obchodních požadavků a odpovědí vývojářů. Evangelizaci DevOps jako nového přístupu rozšiřují v současné době především konference.

Porovnání nástrojů CI

Tato kapitola obsahuje popis metodiky, která byla použita pro výběr a porovnání nástrojů pro podporu nepřetržité integrace, seznam identifikovaných nástrojů, výčet jejich vlastností a diskuzi výsledků.

Metodika pro porovnání nástrojů

Vzhledem k tomu, že nelze zcela objektivně vyhodnotit nejlepší nástroj pro podporu nepřetržité integrace, není v tomto kontextu význam slova "porovnání" takto chápán. Cílem následujícího porovnání nástrojů je identifikovat nejrozšířenější nástroje a poskytnout jednoduchý výčet jejich základních vlastností pro zjednodušení rozhodování při výběru vhodného nástroje.

Pro naplnění tohoto cíle byl zvolen následující postup:

1. Identifikace nejrozšířenějších nástrojů byla provedena na základě uvedených zdrojů. Značně obsáhlý seznam příslušných nástrojů je uveden na Wikipedie (2014), nicméně kvůli jeho nedostatečné vypovídací hodnotě je omezen na základě Duvall (2011).
2. Ze seznamu byl vyřazen nástroj AntHill Pro, protože byl odkoupen firmou IBM a je nyní součástí obsáhlejšího řešení, u kterého není možné přesně zjistit jeho vlastnosti.
3. Ke každému nástroji byly zjišťovány základní vlastnosti, jejichž znalost může pomoci při rozhodování o výběru nástroje, a to v tomto rozsahu: název, webové stránky, platforma, licence, podporované jazyky, IDE integrace, další integrace.
4. Subjektivní vyhodnocení výsledků bylo provedeno z pohledu jednotlivých vlastností.

Přehled nástrojů

Dle uvedené metodiky je pro každý identifikovaný nástroj vytvořena samostatná tabulka s výčtem jeho vlastností.

Bamboo	https://www.atlassian.com/software/bamboo
Platforma	hostované nebo vlastní server
Licence	bezplatné pro open source projekty, bezplatná trial verze, předplatné
Podporované jazyky	nezávislé na jazyku
IDE integrace	Eclipse, Visual Studio, IntelliJ IDEA
Další integrace	Jenkins importer, JIRA, Selenium, Subversion, Mercurial, Git, Perforce, CVS

Codeship	https://codeship.com/

Platforma	hostované
Licence	bezplatné pro open source projekty, omezený počet projektů zdarma, předplatné
Podporované jazyky	Ruby (Rails), Node.js, PHP, Python, Java, Go
IDE integrace	ne
Další integrace	GitHub, Bitbucket, Capistrano

Jenkins	http://jenkins-ci.org/
Platforma	vlastní server
Licence	open source
Podporované jazyky	Java, nepřeberný seznam dalších jazyků pomocí pluginů (Ruby, Python, .NET, iOS,…) viz https://wiki.jenkins-ci.org/display/JENKINS/Plugins
IDE integrace	Eclipse, IntelliJ IDEA, NetBeans
Další integrace	nepřeberné množství pomocí pluginů viz https://wiki.jenkins-ci.org/display/JENKINS/Plugins

TeamCity	https://www.jetbrains.com/teamcity/
Platforma	vlastní server
Licence	bezplatné pro Open source projekty, předplatné
Podporované jazyky	Java, .Net, Ruby, Xcode, Node.js, pomocí pluginů: C++, Python, PHP a další
IDE integrace	Eclipse, Visual Studio,IntelliJ IDEA,RubyMine, PyCharm,PhpStorm, WebStorm
Další integrace	JIRA, bugzilla, AWS, YouTrack, NuGet

Travis	https://travis-ci.com/
Platforma	hostované (integrované s GitHub)
Licence	bezplatné pro Open source projekty, předplatné
Podporované jazyky	Andriod, C, C++, Clojure, Erlang, Go, Groovy, Haskell, Java, JavaScript (woith Node.js), Ojective-C, Perl, PHP, Python, Ruby, Rust, Scala
IDE integrace	ne
Další integrace	GitHub, Heroku

Vyhodnocení

Na základě platformy je nutné se rozhodnout dle specifik daného projektu. Na výběr je možnost hostování na vlastním serveru nebo využití hostovaného řešení ve formě SaaS. Nástroj Bamboo nabízí obě možnosti.

U hostovaných řešení je standardem forma předplatného. Nástroje Bamboo a Codeship nabízí bezplatné využití s omezeními. Všechny nástroje jsou bezplatné pro open source projekty. Nástroj Jenkins je sám o sobě open source řešením.

Z hlediska podporovaných jazyků je nutné zvažovat výběr nástroje dle požadavků projektu. Nelze tudíž tvrdit, že čím více podporovaných jazyků, tím lépe.

Integrace s IDE a další významné integrace s jinými nástroji slouží především pro informační přehled. Obě vlastnosti mohou znamenat významnou výhodu v rámci daného projektu. Opět však není možné hodnotit je kvantitativně a proto jsou uvedeny zejména z informačních důvodů.

Nelze zcela objektivně vybrat nejlepší nástroj. Z hlediska uvedených vlastností se jako nejpřístupnější jeví Bamboo (bezplatná trial verze, nezávislost na jazyku, možnost výběru mezi vlastním serverem a hostovaným řešením, řada integrací). Z pohledu možností rozšíření pomocí pluginů a podpory ze strany komunity, která je mezi nástroji pro podporu nepřetržité integrace nejrozšířenější (Jenkins, 2011), pak nejlépe vychází nástroj Jenkins.

3. Sestavení backend části

Za backend ve světě webových aplikací se považuje ta část aplikace, která se stará o zpracování dat a administraci. Oproti frontend části je běžným uživatelům skryta. Funguje tedy v pozadí aplikace a stará se o podporu funkčnosti ve frontend části. Aby backend fungoval a mohl co nejlépe podporovat frontend část, využívá technologie jako PHP, Java, Pyhton, Ruby, NodeJS a další pro práci s databázemi.

Automatizace sestavení

Při vývoji aplikace nebo jen při změně konfigurace musí vývojář v backendu udělat spoustu kroků, aby dostal danou změnu ven k uživatelům na produkční server. Jedná se zejména o nahrání všech potřebných zdrojových souborů na server. Při větším rozsahu postižených souborů je nutné odeslat celou aplikaci. Dále je nutné zapsat novou verzi aplikace, zkompilovat zdrojové kódy, pokud to příslušný jazyk vyžaduje, provést konfigurační změny a samozřejmě otestovat aplikaci. Je to velký počet netriviálních kroků, které je nutné udělat a právě jejich množství a manuální vykonávání často svadí k chybám. Proto se CI snaží tyto kroky co nejvíce zjednodušit a zautomatizovat.

Při každé zjištěné změně spustí integrační server sestavení, které vykoná nadefinované úkoly a pošle výsledek vývojářovi. Sestavení by mělo být co nejrychlejší, aby vývojář dostal rychle zpětnou vazbu. Navíc při úspěšném sestavení se spouští testy, které v případě nalezení chyby nedovolí nahrání aplikace na produkční server.

Pro vytvoření automatizovaného sestavení kromě integračního serveru a verzovacího systému je nutné použít systém pro automatické sestavení aplikace a to např. systém GNU Make[1] pro operační systémy Linux a Unix, Apache Ant[2] zejména pro programy v Javě, Phing[3] používaný pro aplikace v PHP a Capistrano[4] napsané v jazyce Ruby. (HUJER, 2012)

Phing

Pro demonstraci sepsání integračního skriptu pro sestavení byl vybrán oblíbený nástroj Phing, ve kterém se všechny úkoly, které se mají vykonat, píší do jednoho XML souboru. Tento soubor je součástí zdrojových souborů. Struktura souboru není příliš složitá. Kořenovým elementem je `project`, ten obsahuje jednotlivé úkoly (`target`), které se ještě skládají z jednotlivých úloh (`task`). Zadání úkolu pro kontrolu syntaktické správnosti pak může být zapsána takto.

```
<target name="PHPlint" description="Kontrola syntaxe">
  <phplint haltonfailure="true"
    <fileset dir="${project.basedir}/application">
      <include name="**/*.php"/>
    </fileset>
  </phplint>
</target>
```

Pro kontrolu syntaxe souborů v PHP je možné využít PHPLint[5]. Místo elementu `task` se zapíše název prováděného úkolu, atribut `haltonfailure` říká, pokud dojde při vykonávání tohoto úkolu k selhání syntaktické kontroly, proces se automaticky ukončí. V elementu `fileset` je specifikováno umístění zdrojových souborů aplikace, které mají být zařazeny do kontroly. A element `include` říká, že se mají zkontrolovat všechny soubory s koncovkou php. Tímto způsobem se specifikují jednotlivé úkoly, které se mají provést při sestavení aplikace. (PHING, 2014)

[1] http://www.gnu.org/software/make/
[2] http://ant.apache.org/
[3] http://www.phing.info/
[4] http://capistranorb.com/
[5] http://www.icosaedro.it/phplint

Spouštění unit testů

Testování je velice důležitá část procesu sestavení a celého CI. Pokud dojde jen k jediné chybě během testování, sestavení se ihned ukončí a nedojde k odeslání na produkční server. Za velice oblíbené nástroje pro testování lze zařadit frameworky z rodiny xUnit, kde x představuje programovací jazyk. Pro PHP je tedy k dispozici PHPUnit, pro Javu je to JUnit apod. Sadu připravených testů pak lze ve Phingu spustit následovně.

```
<target name="phpunit" description="PHPUnit testy">
  <phpunit haltonfailure="true" haltonerror="true">
    <formatter todir="reports" type="xml"/>
    <batchtest>
      <fileset dir="tests">
        <include name="**/*Test*.php"/>
      </fileset>
    </batchtest>
  </phpunit>
</target>
```

Výsledný report ve formátu xml se bude nacházet ve složce reports. Testy se spustí pro všechny soubory obsahující slovo „Test“ s koncovkou php a budou ve složce tests. (PHING, 2014)

Pokrytí testy

Důležitým ukazatelem, který se uplatňuje nejen v CI ale ve vývoji software obecně, je pokrytí kódu testy. Díky němu má vývojář přehled, kolik procent kódu je pokryto testy, popřípadě které řádky kódu nejsou vůbec testovány. Dále je nutné brát v úvahu, že i když je kód během testování zavolán, tak to neznamená, že nemůže obsahovat chyby. Proto také bývá uváděno, že cílem není dosáhnout 100% pokrytí, ale mít úplné a komplexní jednotkové testy. (WOOD, 2008)

Pro získání informace o pokrytí kódu unit testy, stačí ve Phingu přidat atribut `codecoverage` s hodnotou `true` do elementu `phpunit`.

```
<phpunit codecoverage="true">
  <batchtest>
    <fileset dir="tests">
      <include name="*Test.php"/>
    </fileset>
  </batchtest>
</phpunit>
```

Řešení závislostí

Webová aplikace se během vývoje a běhu na serveru neobejde bez podpory balíčků (modulů) třetích stran. Závisí na frameworcích a nástrojích, které již někdo napsal a dal vývojářům k dispozici. Ve většině případů, pokud webová aplikace potřebuje nějaký modul, musíme zajistit stažení stabilní verze, umístění do projektu, řádné nakonfigurování a v budoucnu aktualizace těchto nástrojů. V CI se však využívá pomoci balíčkovacích systémů (package management, dependency management), které velice usnadňují vývoj a pomáhají řešit závislosti na modulech.

V projektech postavených na JavaScriptu se nejčastěji využívá pomoci systému npm[6]. V aplikacích na PHP se naopak používá Composer[7]. Vývojář po nainstalování tohoto systému pak jenom napíše jeden příkaz do příkazové řádky a potřebný modul se mu nainstaluje do kořenového adresáře projektu.
Vývojář dále definuje metainformace o projektu do souboru v jazyce JSON (pro npm package.json,

[6] https://www.npmjs.org/
[7] https://getcomposer.org/

pro Composer composer.json), kde kromě názvu a verze aplikace definuje závislost na modulech. Závislost se pak může definovat takto.

```
{
    "require": {
        "php": ">=5.3.2",
        "nette/nette": "~2.2.0"
    }
}
```

Zde říkáme, že aplikace vyžaduje PHP ve verzi 5.3.2 a vyšší, dále pak balíček Nette ve verzi 2.2.0. Nově příchozí vývojář k projektu si pak pouze stáhne zdrojové soubory z repozitáře a příslušný balíčkovací systém mu už doinstaluje všechny požadované moduly.

Způsoby deploymentu

Pokud máme novou verzi aplikaci sestavenou a otestovanou, můžeme ji nasadit na produkční server. Z historického hlediska rozeznáváme manuální nasazení, které se dnes již používá spíše pro menší projekty. Manuální nasazení je netriviální proces s velkou řadou úkolů, kdy vývojáři postupují podle podrobné dokumentace a plní dílčí kroky. Protože je tento krok závislý na lidském faktoru, vede často k chybám. Pro potřeby CI není tento způsob vhodný.

Daleko vhodnější je naopak Continuous Delivery (nepřetržitá příprava k nasazení), kdy po úspěšném sestavení a otestování se nová verze aplikace automaticky připraví k nasazení na produkční server. Pro tento krok je nutný příkaz od vývojáře. Pokud by se nová verze aplikace automaticky nahrála na produkční server, jednalo by se o nasazení zvané Continuous Deployment (nepřetržité nasazení). Díky těmto způsobům nasazení můžeme v CI nasazovat nové verze aplikací každých pár minut.

4. Sestavení frontend části

Frontend, tedy klientská část aplikace, se skládá z technologií HTML, CSS a JavaScript. Dále sem patří např. obrázky či písma.

Vývoj klientské části probíhá v uzavřeném vývojářském prostředí. V jeho rámci je nejčastější činností psaní a ladění kódu. Takový kód je běžně protkaný komentáři vysvětlujícími jeho funkcionalitu, případně je psán v tzv. preprocesoru, což je nadstavba jazyka s vylepšenou syntaxí či funkcionalitou. Pro JavaScript je to např. TypeScript nebo CoffeeScript a pro CSS existují SASS či LESS. Dále je běžně kód rozdělován do logických částí (modulů), které napomáhají k lepší orientaci a zapouzdření funkcionalit.

V prostředí, kdy se dostane kód webové stránky k uživateli, by však měl být cílem co nejmenší objem přenesených dat a také optimální výkon (rychlost jeho spuštění).

Optimalizace HTML

Základem každé webové stránky je HTML dokument popisující její obsah a další zdroje (obrázky, CSS či JavaScript soubory). Pro jejich důkladnou optimalizaci existuje řada nástrojů, jedním z nich je např. HTML Minifier[8]. Tento nástroj transformuje dokumenty těmito způsoby:

1. Odstraní komentáře.
2. Odstraní bílé znaky (mezery, tabulátory, nové řádky, …).
3. Minifikace CSS a JavaScriptu zapsaných přímo v dokumentu.

Optimalizace JavaScriptu a testování

V současné době je kladen velký důraz na interaktivitu a příjemné uživatelské rozhraní. K tomu je v dominantní míře používán JavaScript. To vede ke zvýšenému objemu dat, které si musí uživatel k prohlížení stránky stáhnout. Je proto zásadní používat optimalizační nástroje, které náš kód připraví pro produkční prostředí:

1. Spojení souborů (modulů) do co nejmenšího počtu souborů (ideálně právě do 1).
2. Použití nástroje pro minifikaci (např. UglifyJS).

UglifyJS[9] optimalizuje JavaScript mnoha způsoby, mimo jiné:

1. Odstraní komentáře.
2. Odstraní bílé znaky (mezery, tabulátory, nové řádky, …).
3. Přejmenuje proměnné a funkce tak, aby byly co nejkratší.
4. Optimalizuje strukturu kódu.

Kód se tak stane pro člověka téměř nečitelným, ale zároveň bude mít minimální velikost a optimální rychlost spuštění.

Abychom zajistili správnou funkcionalitu, snazší vývoj a odhalování případných chyb, můžeme psát jednotkové (unit) testy. V prostředí webových stránek má testování další dimenzi - testovat kód musíme napříč různými prohlížeči, jejich verzemi a platformami, na kterých jsou nainstalované. Můžeme využít služeb jako SauceLabs[10] či BrowserStack[11], které nám dovolí v rámci nepřetržité integrace spouštět naše testy na všech prohlížečích, které chceme podporovat.

[8] https://github.com/kangax/html-minifier
[9] http://lisperator.net/uglifyjs
[10] https://saucelabs.com
[11] http://www.browserstack.com

Optimalizace CSS a testování

Obzvláště rozsáhlé aplikace spoléhají na velké množství stylů, které jsou nejlépe rozkouskovány do mnoha samostatných souborů. Pro optimalizaci CSS kódu můžeme použít nástroje jako Clean CSS[12] či Minify CSS[13], které:

1. Odstraní komentáře.
2. Odstraní bílé znaky (mezery, tabulátory, nové řádky, …).
3. Restrukturalizují CSS selektory.

Přehled testů CSS optimalizátorů můžeme najít na http://goalsmashers.github.io/css-minification-benchmark/.

Testováním CSS kódu se v současné době rozumí převážně vizuální regrese, tj. změna stránky po změně CSS kódu. Např. nástroj PhantomCSS[14] nám umožní nadefinovat seznam URL, které má „vyfotit“ a použít pro porovnání vizuálních změn. Takové testování je využitelné především pro ověření toho, že jsme upraveným či novým CSS kódem „nerozbili“ nějaké jiné stránky webu, které s daným kódem zdánlivě nesouvisí.

Optimalizace obrázků

Obrázky obvykle obsahují určitá metadata, která nejsou na webu potřeba. Jedná se např. o EXIF u JPEG. Optimalizace obrázků tedy hlavně odstraní veškerá metadata a dle nastavení aplikuje ztrátovou či bezztrátovou kompresi.

Pro optimalizaci můžeme použít např. tyto nástroje:

1. Imagemin[15]
2. JPEGTran[16]
3. OptiPNG[17]
4. SVGO[18]

Imagemin je nástroj, který slučuje funkcionalitu ostatních zmíněných nástrojů a je tedy univerzální a nejčastěji používaný.

Automatizace

Hlavním cílem je tyto kroky zautomatizovat pomocí nějakého nástroje pro spouštění úkolů (task runner) a v rámci nepřetržité integrace tento nástroj spouštět. Mezi nejpoužívanější patří Gulp.js[19] a Grunt.js[20].

V daném nástroji si zjednodušeně můžeme nadefinovat sadu souborů a seznam transformací, které se na ně mají aplikovat. Např. pro všechny CSS soubory můžeme spustit nástroj pro minifikaci a spojení souboru pro jedno.

Důvody, proč by se měly všechny tyto činnosti zautomatizovat a spouštět v rámci nepřetržité integrace:

[12] https://github.com/jakubpawlowicz/clean-css
[13] http://www.minifycss.com
[14] https://github.com/Huddle/PhantomCSS
[15] https://github.com/imagemin/imagemin
[16] http://jpegclub.org/jpegtran
[17] http://optipng.sourceforge.net
[18] https://github.com/svg/svgo
[19] http://gulpjs.com
[20] http://gruntjs.com

1. Zmenšení rozdílu mezi vývojovou a produkční verzí aplikace.
2. Ověření, že lze vše úspěšně sestavit.
3. Spuštění testů na sestaveném kódu.

5. Nástroje pro udržování jednotného stylu psaní kódu

Každý programovací jazyk má svůj určitý základní styl psaní, který nastolil jeho tvůrce. V týmu je důležité dohodnout se na tom, jakým stylem se bude kód psát. Jedná se především o formátování bílých znaků, způsob pojmenovávání proměnných či způsob psaní konstrukcí jazyka. Výstupem by měl ideálně být nějaký dokument, který shrnuje všechna pravidla a ukazuje je na konkrétních příkladech.

Naučit se tato pravidla trvá určitý čas a i po jejich dlouholetém užívání může nastat situace, kdy nejsou dodržena. Styl můžeme kontrolovat v rámci tzv. code review (kontrola kódu), o který můžeme požádat kolegy před začleněním svého kódu do hlavního kódu. To je však většinou zbytečně strávený čas, který by měl být využit k ověření správnosti fungování a logiky kódu, ne jeho stylu, který nemá na fungování vliv.

Kontrola dodržování standardů psaní kódu je tedy ideální zautomatizovat. Pro téměř každý programovací jazyk existuje určitý nástroj, který nám s tím může pomoci. Běžně ho stačí nastavit dle našich standardů.

Tento nástroj by měl být spouštěn v rámci nepřetržité integrace. Jeho výstupem může být seznam nalezených chyb, které má daný vývojář opravit.

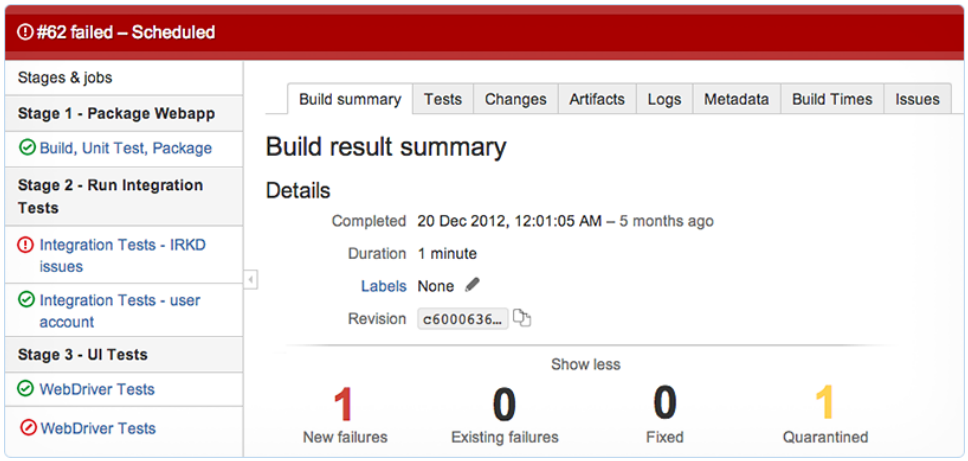
V rámci webových aplikací existuje mnoho takových nástrojů. Pro JavaScript je to nově ESLint[21] či JSHint[22]. CSS můžeme ověřit pomocí CSSLint[23] a pro preprocesor SCSS existuje nástroj SCSS-Lint[24]. Pro jazyk PHP je to např. PHPLint. Máme tedy na výběr a je na každém týmu zvolit si takový nástroj, který nejlépe vyhovuje jeho požadavkům.

[21] http://eslint.org
[22] http://jshint.com
[23] http://csslint.net
[24] https://github.com/causes/scss-lint

6. Nástroje pro podporu CI a týmové práce

Atlassian Bamboo a Jira

Příkladem nástrojů pro podporu CI a týmové spolupráce je integrované řešení od společnosti Atlassian. Sestává z groupwarového nástroje Jira[25] a integračního serveru Bamboo[26].



Obrázek 1- Výsledek sestavení aplikace v Bamboo

Jak lze vidět na obrázku 1, výhodou pravidelné integrace webových aplikací je především včasné odhalení a řešení chyb. Integrované nástroje jako právě Bamboo a Jira umožňují při integraci ve vývojářském týmu včas problém zachytit a komunikovat ho napříč celou organizací. Výsledkem je poté automatizovaný postup bugtrackingu (Jira) a nasazování aplikace (Bamboo).

[25] https://www.atlassian.com/software/jira
[26] https://www.atlassian.com/software/bamboo

7. Závěr

Práce byla úzce zaměřena na prostředí vývoje webových aplikací a s tím spojených činností. Důraz byl kladen spíše na praktickou a programátorskou část, která jsme čerpali právě z praktických zkušeností. Teoretická část tedy není na stejné úrovni jako ta praktická.

Cíle práce byly uspokojivě splněny a během jejího zpracování se nevyskytly zásadní problémy. Definovali jsme obecně nepřetržitou integraci a její propojení s běžnými vývojářskými nástroji, od verzovacích systémů až po nástroje pro řízení projektu.

Nástroje a postupy navržené v této práci nejsou v současné době plně využívány a praktikovány malými a středními firmami. V tomto odvětví je určitý trend požadavku na kvalitu a bezchybnost, který tlačí tyto firmy do takového jednání, které má za cíl zlepšení procesů vývoje. Doporučujeme popsané nástroje aplikovat iterativně a klást důraz na zpětnou vazbu, tedy přizpůsobit se jedinečným potřebám daného vývojářského týmu či celé firmy.

8. Zdroje

ATLASSIAN. *Bamboo* [online]. 2014 [cit. 2014-12-01]. Dostupné z: https://www.atlassian.com/software/bamboo

CODESHIP. *Codeship* [online]. 2014 [cit. 2014-12-01]. Dostupné z: https://codeship.com/
Dostupné z: http://johnpwood.net/2008/12/30/why-code-coverage-alone-doesnt-meansquat/

DUVALL, Paul M., MATYAS, Steve a GLOVER, Andrew. Continuous integration: improving software quality and reducing risk. Upper Saddle River: Addison-Wesley, 2007. xxxiii, 283 s. The Addison-Wesley signature series. ISBN 978-0-321-33638-5.

DUVALL, Paul. Continuous Delivery Tools List. In: [online]. 2011 [cit. 2014-12-01]. Dostupné z: http://www.stelligent.com/agile/continuous-delivery-tools-list/

FOWLER, Martin. Continuous Integration. In: [online]. 2006 [cit. 2014-12-03]. Dostupné z:http://www.martinfowler.com/articles/continuousIntegration.html

HUJER, Martin, Kontinuální integrace při vývoji webových aplikací v PHP (bakalářská práce). Martin Hujer o všem možném. [online]. 2012 [cit. 2014-12-10]. Dostupné z: http://blog.martinhujer.cz/bp/

HUMBLE, Jez a David FARLEY. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Upper Saddle River, NJ: Addison-Wesley, 2010, xxxiii, 463 p. ISBN 978-032-1601-919.

JENKINS, the Fastest Growing Continuous Integration Open Source Community: Ratifies Formal Governance Standards. *Computers, Networks & Communications* [online]. 2011 [cit. 2014-12-01]. Dostupné z: http://search.proquest.com/docview/897456595?accountid=17203

JENKINS. *Jenkins* [online]. 2014 [cit. 2014-12-01]. Dostupné z: http://jenkins-ci.org/

JETBRAINS. *TeamCity* [online]. 2014 [cit. 2014-12-01]. Dostupné z: https://www.jetbrains.com/teamcity/

PHING, Phing User Guide [online]. 2014 [cit. 2014-12-10]. Dostupné z: http://www.phing.info/docs/guide/stable/

The DevOps Lifecycle: Keep C.A.L.M. and Carry On. In: *New Relic* [online]. 2013 [cit. 2014-12-19]. Dostupné z: http://newrelic.com/devops/lifecycle

TRAVIS. *Travis* [online]. 2014 [cit. 2014-12-01]. Dostupné z: travis-ci.org
Versionone, 8th Annual State of Agile Survey [online]. 2013 [cit. 2014-12-19]. Dostupné z: http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

WIKIPEDIE. Comparison of continuous integration software. [online]. 2014 [cit. 2014-12-01]. Dostupné z:http://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

WOOD, J. Why Code Coverage Alone Doesn't Mean Squat [online]. © 2008 [cit. 2012-03-31].