Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	ZS 2018/2019
Autoři – jméno, příjmení, xname	Jakub Jaroš, jarj04 Filip Pazdera, pazf01 Marek Bernard, berm05
Téma	Automatizace testování v agilních přístupech
Datum odevzdání	6.12.2018

Abstrakt

Tato semestrální práce se zabývá automatizovaným testováním softwaru v agilních přístupech. V práci jsou uvedeny základní charakteristiky automatizovaného testování i agilních přístupů. Dále jsou popsány přínosy spojení těchto dvou technik, včetně překážek při implementaci. Jedna kapitola je věnována příkladům několika konkrétních agilních přístupů a využití automatizovaných testů v nich. Poslední kapitola je věnována kritickým faktorům úspěchu implementace automatizovaného testování v agilních projektech a popisu možného workflow v rámci delivery pipeline, včetně příkladů dostupných nástrojů.

Klíčová slova

Automatizované testování, agilní přístupy, FDD, BDD, DevOps

Obsah

Seznam obrázků

Úvod

Informační technologie jsou dnes již nedílnou součástí běžného života. Život bez nich si lze jen těžko představit. Tak, jak se IS/IT stále více integruje do dennodenního fungování společnosti, narůstají i požadavky na funkcionalitu a komplexnost IS/IT. S rostoucími nároky roste i potřeba ověřování kvality vyvíjeného softwaru. V důsledku toho vznikají nové technologie a přístupy jak k vývoji, tak k testování softwaru. Potřeba rychlého vývoje softwaru a jeho testování tak vedlo ke vzniku automatizovaného testování a agilních přístupů. Automatizované testování je dnes již nedílnou součástí agilních přístupů, což ještě více zvyšuje produktivitu, avšak to sebou nese i jistá rizika, kterých je třeba si být vědom a eliminovat je.

Cílem této práce je popsání výhod automatizované testování a agilních přístupů a přínosy spojení těchto dvou termínů. Dále pojednat o překážkách a kritických faktorech úspěchu při implementaci automatizovaného testování v rámci agilních projektů a uvést dostupné nástroje. Rovněž také uvést příklady agilních přístupů využívajících automatizované testování, s důrazem na DevOps.

Informace v této práci byly získávány převážně z dostupné literatury, ale vlastních zkušeností z oblasti testování jak manuálního, tak automatizovaného a z implementace praktiky kontinuálního dodávání, jenž je součástí DevOps. Práce je koncipována tak, aby vedla čtenáře od vysvětlení základních termínů přes popis jejich přínosů a úskalí, až k praktickým příkladům metodik a dostupných nástrojů pro jejich podporu.

1 Automatizované testování v agilních přístupech

Automatizované testování i agilní přístupy jsou již nějaký čas známy. Spojením těchto dvou praktik však může jejich přínos posunout ještě dále, což s sebou však přináší i jistá úskalí. V této kapitole jsou stručně popsány charakteristiky obou termínů a dále popsány přínosy a překážky při začlenění automatizovaného testování do agilních přístupů.

1.1 Přístupy k testování softwaru

K testování lze přistupovat dvěma hlavními způsoby, buď to manuálně, kdy testeři provádějí testy podle připravených testovacích případů, nebo automatizovaně, kdy jsou jednotlivé testovací případy prováděny automaticky pomocí skriptů.

Automatizované testování oproti manuálnímu testování s sebou přináší celou řadu výhod, mezi něž patří zejména úspora času, rychlejší detekce chyb, snadná opakovatelnost a nižší míra chybovosti během testování. Tyto uvedené výhody vyplývají převážně z toho, že stroj dokáže udělat práci rychleji než člověk, a to i s menší chybovostí než člověk.

Automatizované testování má ovšem i jisté nevýhody, jakými je zejména vyšší počáteční investice a náročnější údržba testů. Proto typicky není vhodné automatizovat často se měnící testy a testy pokrývající nové funkcionality. Další nevýhodou je, že při vynechání lidského faktoru dojde i k vynechání tvorby okamžitých rozhodnutí nad neočekávanou situací, kdy člověk by byl schopen najít nějaký způsob, jak vzniklé překážky obejít (tzv. workaround), ale stroj ne.

1.2 Agilní přístup

Pro řízení projektu rozlišujeme dva hlavní přístupy, rigorózní a agilní přístup. Rigorózní přístup je zaměřen na dopředné plánování a obsáhlé specifikace, vychází z principu, ve kterém výsledný produkt je nejprve celý navržen, pak vytvořen a následně otestován. Z toho tedy vyplývá, že pro rigorózní přístup je poměrně obtížné vypořádat se s častými změnami.

Oproti tomu agilní přístup upřednostňuje inkrementální vývoj, časté iterace se zákazníkem, fungující software před rozsáhlou dokumentací a časté reakce na změny před striktním dodržováním plánu. Jinými slovy lze říci, že v agilních přístupech dochází velmi často ke změnám v návaznosti na nové požadavky od zákazníka, a tím dochází k jisté komplikaci při tvorbě automatizovaných testů (Collins a Ferreira, 2012).

1.3 Přínosy automatizovaného testování v agilních přístupech

Přínosy automatizovaného testování v agilních přístupech se zabývá například Sarah Elson (Elson, 2018). Představme si případ, kdy vyvíjíme software a každou novou funkcionalitu chceme kvůli vidině vyššího zisku okamžitě zpřístupnit zákazníkům. S každou novou funkcionalitou se však množství testů, které musíme provádět zvyšuje až do bodu, kdy je nemožné vše otestovat manuálně. Z toho plyne, že k naplnění potřeb rychlého vývoje, musejí být testovací metody rychlejší. Zde přichází na řadu automatizované testování.

Při implementaci automatizovaného testování, jako součásti kontinuálních procesů agilního vývoje, můžeme kompletní sady testů provádět opakovaně při každém novém přírůstku a ihned objevit případné chyby, které mohou být okamžitě řešeny, bez vážného dopadu na dobu vydání nové verze produktu.

1.4 Překážky při implementaci

Při automatizaci testování v agilních přístupech se může objevit mnoho problémů. Hlavním problémem je nutnost vytvořit spoustu testovacích skriptů. Automatizace testování je náročný úkol, časově náročný na plánování, což jde proti agilním přístupům, kde se snažíme plánování minimalizovat a co nejrychleji software dodat (Saleem et al., 2014).

Lisa Crispin a Janet Gregory (Crispin a Gregory, 2009) pak v knize "Agile Testing: A Practical Guide for Testers and Agile Teams" na základě jejich zkušeností z agilních týmů identifikovaly několik problémů, které ovlivňují úspěch automatizace testů:

1) Postoj programátorů, kdy se někteří příliš nezabývají testováním, jelikož mají QA tým, jako záchrannou síť.
2) Neznalost vhodných nástrojů.
3) Počáteční investice
4) Kód, který se neustále mění vede k neproveditelnosti automatizace
5) Legacy systémy, kde není starší kód navrhnut tak, aby se jeho testování dalo automatizovat
6) Strach testerů, kteří nemají mnoho zkušeností s programováním a rovněž programátorů, kteří naopak nemají zkušenosti s testováním.
7) Staré návyky, kdy se namísto automatizace regresní sady provádějí manuální regresní testy

2 Příklady agilních přístupů využívajících automatizované testování

Automatizované testování je přímo popsáno v několika agilních přístupech. Tato kapitola je věnována představení některých z nich, a to Test Driven Development, Behavior Driven Development a detailněji je pak popsáno DevOps, které je dnes velmi skloňováno.

2.1 Test Driven Development (TDD)

Test Driven Development je proces návrhu, psaní a exekuce testů, přičemž testy jsou navrhovány a psány před samotným zdrojovým kódem. Dopředný návrh testů napomáhá v odhadu průběhu následného vývoje, čímž v jisté míře mitiguje počet funkcionalit, na které je během psaní zdrojového kódu zapomenuto. Hlavním cílem tohoto přístupu je tedy prevence chyb a využití takto napsaných testů jako určité dokumentace toho, co se plánuje dosáhnout z hlediska pokrytí testů (Nair, 2018).

Test Driven Development proces je běžně složen z následujících šesti po sobě navazujících kroků (CodeUtopia, 2015):

1) Návrh a napsání testů
2) Spuštěním napsaných testů. V tomto kroku by všechny nově přidané testy, jež prozatím nemají k sobě napsaný zdrojový kód, měly skončit neúspěšně. Pokud by některý z nich skončil úspěšně, tak to indikuje na nesprávnost daného testu.
3) Napsání zdrojového kódu tak, aby splňoval minimální požadavky pro úspěšné projití k sobě přiřazených testů
4) Spuštění testů za účelem otestování nově přidaného zdrojového kódu z bodu 3.
5) Vyhodnocení testů a úprava zdrojového kódu, ve kterém testy z předešlého bodu skončily neúspěšně
6) Opakování celého procesu od bodu 1

Test Driven Development projekty obvykle dosahují velkého pokrytí zdrojového kódu testy (90 a více procent). Přidávání dalšího zdrojového kódu je tak poměrně snadné, neboť zdrojový kód je ihned ověřován testy, včetně kontroly, zda nedošlo k poruše ostatních funkcionalit softwaru.

Jak z výše uvedeného vyplývá, nejdůležitější a nejpracnější částí tohoto přístupu není napsání zdrojového kódu, ale napsání testů, jež zdrojový kód kontrolují (CodeUtopia, 2015).

2.2 Behavior Driven Development (BDD)

Behavior Driven Development je stejně jako Test Driven Development proces, ve kterém jsou testy navrhovány a psány před zdrojovým kódem. Rozdíl mezi nimi tvoří perspektiva, z jaké je systém kontrolován. Test Driven Development je zaměřen na kontrolu implementace zdrojového kódu, oproti tomu Behavior Driven Development je zaměřen na testování softwaru z pohledu koncového uživatele (Nair, 2018).

Behavior Driven Development řeší běžný problém jednotkových testů, které jsou značně závislé na tom, jak je funkcionality implementována, tedy i při malé změně implementace, která nemění vstup ani výstup funkce, je potřeba upravit odpovídající test. Behavior Driven Development tedy neřeší, jak je funkcionalita implementována, ale jak by se měla chovat, jinými slovy za software na poskytnuté vstupy vrací očekávané výstupy, a to bez ohledu na detaily implementace (CodeUtopia, 2015).

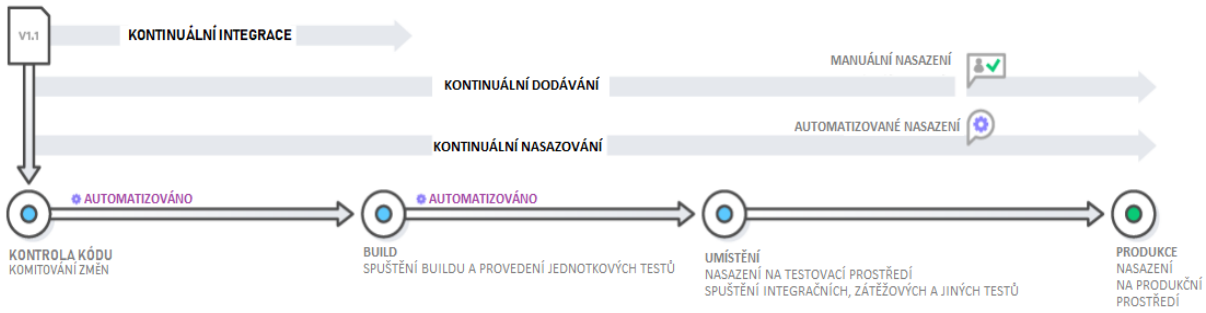
2.3 DevOps

Termín DevOps je spojením slov Development a Operations. DevOps popisuje postupy, které zefektivňují proces poskytování softwaru, zdůrazňující učení prostřednictvím zpětné vazby od produkce k vývoji a zkrácení doby cyklu od provedení změn v kódu do jejich uvolnění. DevOps umožňuje nejenom rychleji dodat software, ale také produkovat software vyšší kvality, který je lépe přizpůsoben definovaným požadavkům (Hüttermann, 2012).

Ačkoli tento termín vznikl spojením zmíněných slov, nedílnou součástí jsou týmy testující kvalitu softwaru.

DevOps není metodikou, nýbrž způsobem pohledu na vývoj softwaru a souborem doporučení a vzájemně provázaných kontinuálních praktik, které vedou ke zvýšení efektivity procesu vývoje. Je díky němu zkrácena doba nutná k uvedení softwaru do provozu díky automatizaci jednotlivých provázaných kroků a dobře organizované spolupráci napříč podílejícími se týmy.

DevOps je soustavou několika vzájemně provázaných praktik, z nichž nejznámější je kontinuální integrace, kontinuální dodávání a kontinuální nasazování, jejichž workflow je zobrazeno na Obrázku 1.

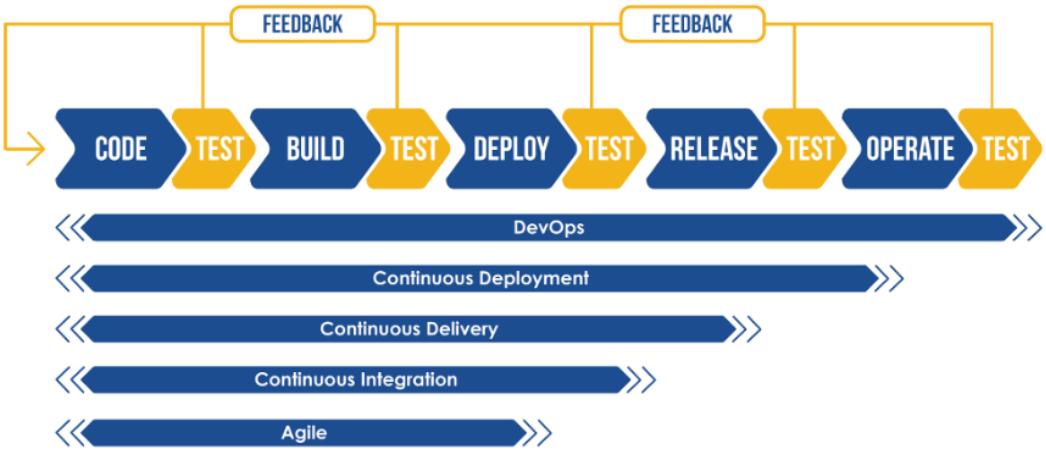


Obrázek 1 Porovnání praktik kontinuální integrace, kontinuální dodávání a kontinuální nasazování (Upraveno dle (Amazon Web Services, Inc., 2018))

2.3.1 Kontinuální testování

Kontinuální testování je proces provádění automatizovaných testů jako součásti kontinuálního procesu dodávání softwaru, s cílem co nejrychleji získat zpětnou vazbu o byznys rizicích spojenými s kandidátem na release (Tricentis, 2017).

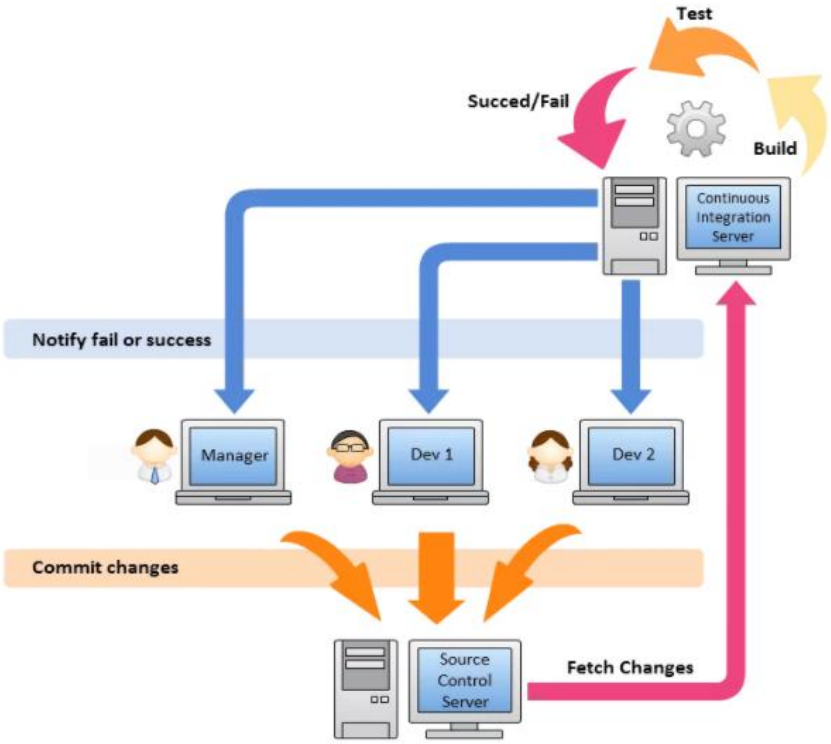
V podstatě jde o to, že po každém milníku v procesu vývoje provádíme sadu testů, abychom mohli rozhodnout, zda se můžeme pustit do dalšího kroku, nebo je třeba se vrátit zpět, pokud byly nalezeny chyby. Milníky se zde myslí jednotlivé kroky v procesu vývoje od nahrání zdrojového kódu do repozitáře až po jeho uvolnění. Celá myšlenka, včetně milníků je znázorněna na Obrázku 2.



Obrázek 2 Kontinuální testování (Zdroj: (Nguyen, 2018))

2.3.2 Kontinuální integrace

Dle Martina Fowlera (Fowler 2006) je kontinuální integrace praktika vývoje softwaru, kde členové týmu integrují svou práci často, alespoň jednou denně, což vede k více integracím zdrojové kódu za den. Každá integrace je pak ověřena automatizovaným buildem kódu (včetně testování), což umožňuje rychleji detekovat integrační chyby. Každá chyba je okamžitě reportována zpět na vývoj, kde může být okamžitě řešena. Workflow kontinuální integrace je znázorněno na Obrázku 3.



Obrázek 3 Kontinuální integrace wrokflow (Zdroj: (Pecanac 2016))

2.3.3 Kontinuální dodávání

Martin Fowler (Fowler, 2013) definuje kontinuální dodávání jako disciplínu vývoje softwaru, při níž se software staví takovým způsobem, že může být kdykoli uvolněn na produkci. Avinash Pawar (Pawar, 2017) pak definuje kontinuální dodávání způsobem, který poskytuje lepší představu o jednotlivých krocích, a to jako vývojovou praxi, při níž jsou změny kódu automaticky sestaveny, otestovány a připraveny k uvolnění na produkci. Ke kontinuální integraci jsou tak přidány další kroky, které zahrnují nasazení zdrojového kódu na testovací prostředí, kde je podroben dalším testům. Po úspěšném testování je tak určen kandidát na nasazení na produkční prostředí.

2.3.4 Kontinuální nasazování

Pojmy kontinuální dodávání a kontinuální nasazování bývají často nesprávně zaměňovány, což je zřejmě způsobeno nešťastným pojmenováním předchozí praktiky. Slovo delivery (dodávka) v člověku evokuje finální dodání produktu.

Awinash Pawar (Pawar, 2017) definuje kontinuální nasazování jako praxi vývoji softwaru, při níž je každá změna kódu automaticky nasazena až na produkci.

Kontinuální nasazování je tedy dalším vývojovým stupněm kontinuálního dodávání. Jediný rozdíl je v tom, že při kontinuálním dodávání zůstává nasazení softwaru na produkci v režii vývojářů, kdežto při kontinuálním nasazování je i tento krok automatizován. To umožňuje každou změnu kódu ihned zpřístupnit reálným uživatelům, pokud projde automatizovanými testy, k čemuž může dojít i několikrát za den.

3 Implementace automatizovaného testování v agilních přístupech

Tato kapitola je zaměřena na praktickou implementaci automatizovaného testování v agilních projektech. Jsou zde rozebrány kritické faktory úspěchu, dle prezentace Lukasze Grabinskio (Grabinski, 2014) na Agile Business Conferenece. Také je zde rozebráno možné workflow delivery pipeline, včetně dostupných nástrojů.

3.1 Kritické faktory úspěchu

U agilních projektů není přesně definovaný postup, kde začít ani návod, jak postupovat krok po kroku při zavádění automatického testování. Automatické testování je velmi důležité pro agilní projekty, protože vývoj jde velmi rychle, a proto je důležité na začátku projektu rozhodnout, kde začít.

Je možné začít například s identifikováním opakujících se úkolů, identifikace problémových oblastí aplikace nebo určit výzvy při testování. Testeři mohou současně začít na smoke testech a regresi. Postupně dochází k automatizaci opakujících se testů.

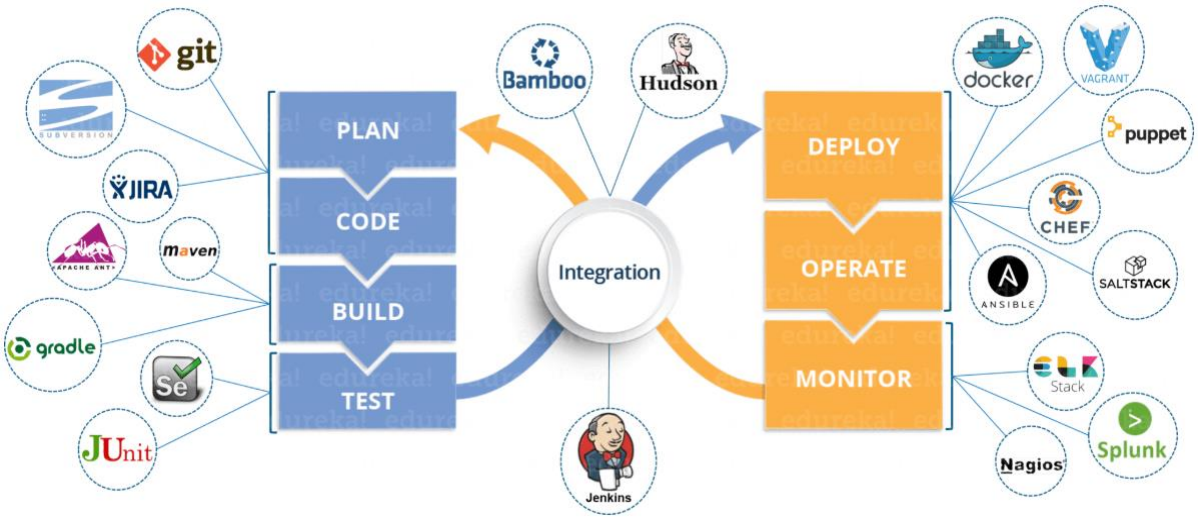
Důležité je myslet na rychlý vývoj, takže se nesnažit od začátku automatizovat 100 %, ale postupovat po malých krocích. Postupovat podle stejných postupů jako agilní projekt. Pokud automatické testování nebude dobře naplánované a rozvržené, může dojít k prodlužení testování. Hlavní je neautomatizovat z důvodu automatizace, je potřeba rozvrhnout si testy, tak aby se to agilnímu projektu přizpůsobilo a pomohlo při vývoji a testování. Také používání vhodného vývojového prostředí a nástrojů, které poskytne rychlejší tvorbu automatických testů a neukládání kódu na lokální uložiště, ale sdílet si kód mezi sebou pro efektivnější tvorbu.

Udržovat věci jednoduché a postupovat po menších krocích je dobrý krok k úspěšné implementaci. Nezapomínat by se mělo na to, že se projekt vyvíjí v agilních metodikách, takže do automatizace se může zapojit celý tým. Tím dojde ke zrychlení a každý bude mít povědomí o to, co se automatizuje a ze zpětné vazby bude jasné, zda je automatizace smysluplná. Velmi podstatné je udržet testy navzájem nezávislé a udržitelné, protože prostředí v agilním vývoji se mění rychle a je potřeba tomu přizpůsobovat i automatické testy. Protože se snažíme o velmi rychlé dodání kódu, je v agilním vývoji uvažovat o automatickém testování tak, aby nedošlo k přeplánování.

V agilním vývoji je důležité automatizovat více než smoke a regresní testy, což znamená, že testeři musí mít větší zkušenosti a cíle než pouze nalézt chyby a defekty. Testeři musí více spolupracovat a rozšířit své dovednosti v oblastech programování a automatizace testů. Čím více testů je automatizováno, tím více času mají testeři v agilním vývoji na náročnější a neobvyklé úkoly.

3.2 Popis delivery pipeline a dostupných nástrojů

Znalost a využití správných nástrojů je jedním z kritických faktorů pro úspěšnost implementace automatizovaných testů v agilním prostředí, jak bylo uvedeno v kapitole *1.4 Překážky při implementaci*. Níže je dle vlastních zkušeností uveden popis implementace automatizovaných testů v rámci kontinuálního dodávání, včetně možných nástrojů. Popis nástrojů v rámci celého DevOps je pak možné najít například v článku „How To Orchestrate DevOps Tools Together To Solve Our Problems?“ (Vardhan, 2017) na webu Eduraka.co, odkud je převzat Obrázek 4.



Obrázek 4 Dostupné nástroje v rámci DevOps (Zdroj: (Vardhan, 2017))

1) Plan

Plánování zahrnuje činnosti pro určení toho, co se má udělat a kdy. Tato činnost se nedá příliš automatizovat a stále zůstává v režii lidí, avšak existují nástroje pro podporu plánování, kterými jsou různé ticketové systémy, jako například Jira, které umožňují vytvářet požadavky a přiřazovat je řešitelům.

2) Code

Kódování je činností, kde se tvoří vlastní kód. V dnešní době již existuje spousta nástrojů, které zjednodušují správu kódu. Jedná se o verzovací systémy, bez kterých se dnes neobejde žádný vývojář. Umožňují totiž uchovávat historii kódu, tvorbu větví, kdy může mít každý vývojář svou vlastní větev, kde vytváří nové funkcionality bez dopadu na stabilní verzi aplikace a v neposlední řadě umožňují také tvorbu reportů. Verzovací systémy pak běží na platformě, která slouží jako úložiště kódu, tzv. repozitář, odkud mají ke kódu přístup nejen samotní vývojáři, ale rovněž další aplikace, které navazují na celý kontinuální proces dodávání softwaru. V základu můžeme rozlišit dva druhy verzovacích systémů, a to centralizované a decentralizované. Zástupcem centralizovaných systémů je například Subversion a decentralizovaných pak Mercurial nebo velmi oblíbený Git.

3) Build

Jak bylo uvedeno, z repozitáře kódu je kód stahován dalšími aplikacemi v rámci delivery pipeline. Aby se z kódu stala funkční aplikace, musí se nejdříve provést build kódu. K tomu lze použít nástroje, jako např. Maven, nebo Apache Ant. Součástí tohoto kroku je také provedení unit testů.

4) Test

Po úspěšném provedení unit testů a sestavení aplikace je možno přistoupit k dalším typům testů, zpravidla to bývají funkční testy. Součástí tohoto kroku může být i nasazení aplikace na testovací prostředí. Můžeme totiž spustit určitou sadu testů na jednom prostředí a po úspěšném otestování aplikaci nasadit na jiné prostředí, třeba s jinou konfigurací, kde můžeme spustit stejnou, či rozdílnou sadu testů, nebo dokonce úplně jiný typ testů. Právě v této části má největší podíl automatizované testování. Výběr nástroje záleží na typu testů, programovacím jazyku atd. Dnes se těší velké oblibě například Selenim Framework, který je dostupný pro řadu programovacích nástrojů a je zdarma.

5) Integration

Všechny předtím zmíněné kroky je třeba provázat tak, aby nevyžadovali zásah člověka. K tomu slouží integrační servery, kde je možno libovolně provázat všechny činnosti pomocí tzv. jobů a triggerů. Job definuje určitou sadu kroků a trigger jejich spouštěč. Když to vezmeme od začátku, prvním triggerem může být nahrání nové verze kódu do repozitáře, který spustí job, jenž stáhne zdrojový kód, spustí unit testy a vytvoří spustitelný balík. V každém jobu musí být stanoveno, co se stane při úspěchu a co při neúspěchu. Pokud celý job proběhne bez chyb, měl být triggerem pro další job, tedy například nahrání aplikace na testovací prostředí a spuštění dalších testů, například funkčních. V opačném případě by se měla rozeslat informace o chybě zainteresovaným osobám, aby mohla být chyba co nejrychleji opravena. Celé workflow na integračním serveru může vést až k nasazení aplikace na produkční prostředí, což ale málokterá firma dokáže implementovat. Je totiž potřeba mít vše dokonale navržené, což se týká zejména automatizovaných testů, které musejí mít takové pokrytí a přesnost, aby panovala důvěra v to, že pokud testy proběhnou bez chyb, chyby se opravdu nevyskytují a aplikaci je možno zpřístupnit zákazníkům. V softwaru budou chyby samozřejmě vždy, není v lidských silách odhalit každou chybu, avšak měly by být odhaleny všechny kritické chyby s velkým dopadem na kvalitu aplikace.

Velké oblibě se těší integrační server Jenkins. Dalšími zástupci jsou například Travis CI nebo TeamCity.

Závěr

Cílem této práce byl popis automatizovaného testování a agilních přístupů a přínosy spojení těchto dvou termínů. A dále popis možných překážek a kritických faktorů úspěchu při implementaci automatizovaného testování v agilních projektech a dostupných nástrojů pro její podporu. Informace, popisující vše výše uvedené bylo zpracovány do tří kapitol.

První část práce byla věnována stručnému popisu automatizovaného testování i agilních přístupů pro uvedení do tématu. Rovněž byly pospány přínosy automatizovaného testování v agilním prostředí, a také překážky při jeho implementaci.

Druhá kapitola byla věnována příkladům konkrétních agilních přístupů, jež v sobě mají automatizované testování zabudováno, a to Feature Driven Development, Behaviour Driven Development a DevOps, kterému byla věnována největší pozornost.

Třetí kapitola byla věnována kritickým faktorům úspěchu implementace automatizovaných testů v agilním prostředí, a taktéž bylo popsáno možné workflow v rámci delivery pipeline, včetně uvedení možných nástrojů pro její realizaci.

Automatizované testování je dnes velmi rozšířené. Testeři mají stále větší přesah do programování. Stejně tak agilní přístupy jsou stále rozšířenější, ať už jako celé metodiky, nebo jen jako prvky v rámci procesu vývoje. I korporátní firmy se dnes snaží alespoň částečně implementovat agilní praktiky alespoň do části projektů. S rostoucí důležitostí informačních technologií roste potřeba rychlého vývoje podpořeného rychlým a kvalitním testováním s adekvátní vypovídající hodnotou. Při implementaci těchto praktik je však třeba brát velký zřetel. Uspěchaná implementace bez jasné vize může přes všechny výhody vést k selhání projektu.

Použitá literatura

AMAZON WEB SERVICES, INC., 2018. What is Continuous Delivery? *AWS* [online] [vid. 2018-03-07]. Dostupné z: https://aws.amazon.com/devops/continuous-delivery/

CODEUTOPIA, 2015. What's the difference between Unit Testing, TDD and BDD? | CodeUtopia. *CodeUtopia* [online] [vid. 2018-12-02]. Dostupné z: https://codeutopia.net/blog/2015/03/01/unit-testing-tdd-and-bdd/

COLLINS, Eliane a Vicente FERREIRA, 2012. Software test automation practices in agile development environment: an industry experience report. *7th International Workshop on Automation of Software Test* [online]. **2012**. Dostupné z: doi:10.1109/IWAST.2012.6228991

CRISPIN, Lisa a Janet GREGORY, 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams* [online]. New Jersey: Pearson Education [vid. 2018-12-02]. ISBN 0-321-53446-8. Dostupné z: http://www.mypearsonstore.ca/bookstore/agile-testing-a-practical-guide-for-testers-and-agile-9780321534460?xid=PSED

ELSON, Sarah, 2018. Why Automation Testing Is Important In Agile Development? *Medium* [online]. [vid. 2018-11-14]. Dostupné z: https://medium.com/@sarahelson81/why-automation-testing-is-important-in-agile-development-919e8f185c06

FOWLER, Martin, 2006. Continuous Integration. *martinfowler.com* [online] [vid. 2018-03-14]. Dostupné z: https://martinfowler.com/articles/continuousIntegration.html

FOWLER, Martin, 2013. Continuous Delivery. *martinfowler.com* [online] [vid. 2018-03-14]. Dostupné z: https://martinfowler.com/bliki/ContinuousDelivery.html

GRABINSKI, Lukasz, 2014. How to Implement Efficient Test Automation in an Agile Project. In: *Agile Business Conference* [online]. B.m. Dostupné z: https://www.agileconference.org/wp-content/uploads/2014/10/How-to-Implement-Efficient-Test-Automation-on-an-Agile-Project-Lukasz-Grabinski-John-OHare.pdf

HÜTTERMANN, Michael, 2012. *DevOps for Developers*. 1. vyd. CA USA: Apress. ISBN 978-1-4302-4569-8.

NAIR, Jithin, 2018. TDD vs BDD - What's the Difference Between TDD and BDD? *TestLodge Blog* [online] [vid. 2018-12-02]. Dostupné z: https://blog.testlodge.com/tdd-vs-bdd/

NGUYEN, Tan, 2018. WEBINAR: HOW DOES TESTARCHITECT FIT INTO CONTINUOUS DELIVERY? *TestArchitekt* [online] [vid. 2018-12-04]. Dostupné z: https://www.testarchitect.com/how-does-testarchitect-fit-into-the-software-delivery-pipeline

PAWAR, Avinash, 2017. Devops: Continuous Integration vs Continuous Delivery vs Continuous Deployment. *DevOps Techie* [online]. [vid. 2018-03-14]. Dostupné z: http://devopstechie.com/continuous-integration-vs-continuous-delivery-vs-continuous-deployment/

PECANAC, Vladimir, 2016. What Is Continuous Integration and Why Do You Need It? *Code Maze* [online]. [vid. 2018-11-14]. Dostupné z: https://code-maze.com/what-is-continuous-integration/

SALEEM, Rana, Salman QADRI, Imran HASSAN, Rab BASHIR a Yasir GHAFOOR, 2014. Testing Automation in Agile Software Development. *International Journal of Innovation and Applied Studies*. **2014**(2), 9, 541–546. ISSN 2028-9324.

TRICENTIS, 2017. What is Continuous Testing? *Software Testing Tools for Continuous Testing | Tricentis* [online]. [vid. 2018-11-14]. Dostupné z: https://www.tricentis.com/what-is-continuous-testing/

VARDHAN, 2017. DevOps Tools: How To Orchestrate Them To Solve Our Problems. *Edureka Blog* [online]. [vid. 2018-12-02]. Dostupné z: https://www.edureka.co/blog/devops-tools