**Vysoká škola polytechnická Jihlava**

Studijní program

**Využití verzovacího softwaru v IT firmě**

Seminární práce

Autor práce: Jakub Halama

Předmět: Uživatelský software

Vyučující: Mgr. Hana Vojáčková, Ph.D.

Jihlava 24. 11. 2024

Obsah

[Seznam obrázků 3](#_Toc183355241)

[Seznam tabulek 3](#_Toc183355242)

[Seznam zkratek a výrazů 3](#_Toc183355243)

[Úvod 4](#_Toc183355244)

[1 GitLab obecně 5](#_Toc183355245)

[1.1 Pipelines 5](#_Toc183355246)

[1.2 Tagy (verze) 7](#_Toc183355247)

[1.3 Citace 8](#_Toc183355248)

[Závěr 9](#_Toc183355249)

[Seznam použité literatury 10](#_Toc183355250)

[Přílohy 11](#_Toc183355251)

# Seznam obrázků

Obrázek 1: [Doby trvání jednotlivých testů](#_Tagy_(verze)) 7

# Seznam tabulek

[Tabulka 1:](#_Toc105128914) [Výhody a nevýhody](#_Výhody_a_nevýhody) 6

# Seznam zkratek a výrazů

VŠPJ Vysoká škola polytechnická Jihlava

Repozitář Uložiště

Commit Editační změna

Push Nahrání změn

Větev Pevná část kódu

Merge request Žádost připojení větve do hlavní větve

Merge Spojení větví

Pipeline Balíček pravidel

Breaking change Změna, která není zpětně kompatibilní se starší verzí

Bug Chyba zanešená do programu

Release Vydání nové verze

# Úvod

Cílem práce je přiblížit využití verzovacího softwaru, konkrétně GitLab a popsat jeho využití v praxi.

Motivací pro výběr tohoto tématu je moje aktuální zaměstnání, kde s GitLabem pracuji na denní bázi.

Součástí práce bude doprovodný dokument s obrázky, které budou sloužit jako dodatečné vysvětlení.

# GitLab obecně

GitLab je verzovací software, který neslouží pouze k zaznamenávání změn, ale také jako cloudové uložiště. Poskytuje webový repozitář, do kterého mohou vývojáři vkládat své změny, následně je dodatečně upravovat a validovat. Vývojáři tvoří takzvané commity, které následně pushují ze svého vývojového prostředí na svém lokálním uložišti do GitLabu, kde tyto změny seskupují do větví a následně tyto větve připojují do hlavní větve s názvem Master nebo Main. Taktéž je myšleno na bezpečnost, takže GitLab disponuje možností nastavovat oprávnění a role, juniorští vývojáři nemají oprávnění na schvalování merge requestů. Ty je třeba prvně nechat schválit seniornějším kolegou, který merge request projde a následně schválí a zamerguje.

## Pipelines

Při větším počtu vývojářů již není možné kontrolovat správnost, čistotu a estetiku jednotlivých úprav, pro tyto účely je možné používat takzvané Pipelines. Jedná se o souhrn pravidel, které dokážou odhalit nejčastější chyby. Využíváme je zejména na odhalení estetických, logických, běhových chyb nebo udržování pořádku.

1. Estetické

Do této kategorie spadá špatné odsazení jednotlivých řádků, prázdné mezery, části kódu bez dokumentace.

1. Logické

Slouží k tomu, aby se v kódu neobjevovaly víckrát stejné části, nekonečné cykly, nesprávně zpracovávané datové typy či návratové hodnoty, nebo zbytečné podmínky, které například vždy skončí pravdou / nepravdou.

1. Hygiena repozitáře

S dlouhodobým vývojem produktu a vydáváním jeho novějších verzí dochází přirozeně v souborech ke změnám a jejich přidávání, pokud se stane, že je nějaký nový soubor redundantní, pipeline nás na to upozorní a selže, to výrazně pomáhá k udržování pořádku v repozitáři a nedochází k zaplevelení.

1. Běhové chyby

Pokud projdou předchozí dva testy, následuje ještě otestování průběhu neboli testovací build. Na vývojovém serveru, kde jsou udržovány testovací a vývojové instance, tak dojde ke spuštění kompletního kódu z Master větve včetně aktuální úpravy. Tomuto testu se říká „Continuous integration“ v překladu „Průběžná integrace“. Tento běh může skončit třemi různými stavy:

1. Úspěch – Kód se podaří spustit bez chyb, v takovém případě pipeline vrátí úspěch.
2. Úspěch s varováním – Kód se podaří spustit, ale s varováními, které sice nejsou kritické pro běh programu, ale mohou způsobovat nepředvídatelné chování, nebo při některých vstupech vracet nesprávné výsledky, v případě front endu například nesprávné zobrazování některých komponent.
3. Neúspěch – Dojde k tomu, že s požadovanou úpravou není možné kód přeložit a následně spustit. Takovou úpravu nikdy nechceme zamergovat do Master větve, protože by došlo k rozbití celého programu. Avšak záleží na implementaci, to že spadne jakýkoliv z testů, nemusí znamenat, že se úprava nedá zamergovat.

### Zpracování jednotlivých stavů

Ačkoliv výsledné stavy jednotlivých testů vrací různé výsledky, nemusíme se jimi nutně řídit, vše záleží na implementaci, dokonce ani není povinné vůbec nějaké testy mít. Je ale silně doporučeno je používat, a to v takových implementacích, kdy GitLab vůbec nepovolí zamergovat úpravy, které neprojdou těmito testy.

### Výhody a nevýhody

Ačkoliv se zdá, že tento nástroj nemá žádné nevýhody, není tomu tak, při nesprávné implementaci, nedostatečně znalých uživatelích nebo velkých a složitých řešeních může docházet k problémům.

Tabulka : Výhody a nevýhody

|  |  |
| --- | --- |
| Výhody | Nevýhody |
| Automatizace vývojového procesu | Složitost při pokročilých nastaveních |
| Integrace s GitLabem | Přílišná složitost pro juniorní vývojáře |
| Flexibilita a přizpůsobitelnost | Problémy při ladění chyb |

Zdroj: Vlastní zpracování dle Borůvková a kol., (2021)

Výhody:

1. Automatizace vývojového procesu

Tento nástroj umožňuje výrazně urychlit proces schvalování úprav a zároveň šetří náklady tím, že odpadá povinnost ručně kontrolovat každý jeden merge request.

1. Integrace s GitLabem

Nejedná se o funkci třetí strany, proto je s GitLabem plně kompatibilní, snadno pracuje s uloženým kódem, správně, jednoduše a detailně loguje, kde nastala chyba a uživatel ji dokáže bez problémů identifikovat a opravit.

1. Flexibilita a přizpůsobitelnost

Veškerá pravidla a nastavení jsou uložena v souboru .gitlab-ci.yml, do kterého je možné volně přistupovat, již existující pravidla měnit nebo přidávat nové. Syntaxe je podobná .json souborům, což může některým uživatelům výrazně pomoci.

Nevýhody:

1. Složitost při pokročilých nastaveních

Při nastavování komplexnějších pravidel je časově náročné testy správně nastavit, je třeba důkladně promyslet, jak je správně integrovat, aby byly opravdu přínosem a vývojáře neomezovaly, například aby nezakazovaly podmíněné skoky, zkrácený zápis podmínek.

1. Přílišná složitost pro juniorní vývojáře

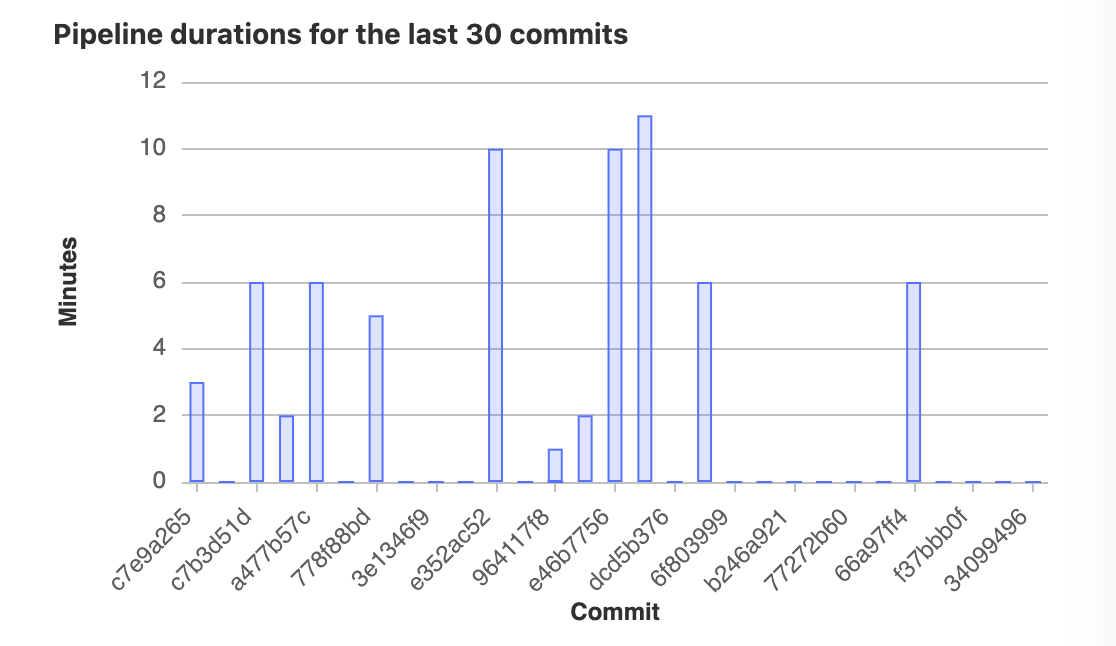
Syntaxe může být pro začínající uživatele matoucí, nastavování nejrůznějších testů bývá často z tohoto důvodu problematické.

1. Problémy při ladění chyb

V praxi se často stává, že při vývoji úprava neprojde jedním z testů a při nesprávné implementaci není možné chybu opravit, protože metoda může být v pravidlech zakázána.

V takovém případě je třeba buď najít další možnost, jak kód upravit, tak, aby byla logika zachována, nebo upravit soubor s pravidly.

Obrázek 1: Doby trvání jednotlivých testů

Zdroj: GitLab dokumentace

## Tagy (verze)

GitLab je verzovacím nástrojem, proto umožňuje uživatelům skládat změny a číslovat je. Tím lze docílit strukturovaného verzování, díky kterému je software přehledný, aktualizovatelný. V praxi to funguje tak, že se neústále vyvijí jedna hlavní větev, která ale není nasazena u žádného zákazníka a jednou za určitý časový interval vyjde nová verze. Verzování dělíme do různých skupin podle velikosti změn, které jsou obsaženy. Číslo verze může vypadat takto: 1.0.12

1. Hlavní verze (Major)

Hlavní verze je značená první číslicí tedy v tomto případě jedničkou **1**.0.12. Typicky se jedná o verze, které obsahují takzvané „breaking changes“, tedy změny, které nejsou zpětně kompatibilní. Jedná se o například uvedení softwaru na trh po alfa verzi (značí se číslem 0.x.x) nebo kompletním rebrandingu, redesignu.

1. Vedlejší verze (Minor)

Vedlejší verze vznikají přirozeně v životním cyklu každého vyvíjeného softwaru. V těchto verzích jsou přidávány nové funkce, nebo vylepšované již existující. Jsou také zpětně kompatibilní a je v zájmu každého zákazníka, aby si držel co možná nejaktuálnější verzi.

1. Opravná verze (Patch)

Ačkoliv při vývoji dochází vždy k testování nových funkcionalit, není však v lidských silách, aby se neobjevovaly v releasech nových verzí chyby. Tyto chyby jsou zpravidla sbírány ze zpětné vazby zákazníků, kteří se softwarem pracují, tedy od koncových uživatelů. Ty jsou následně projednávány na interních vývojářských statusech, kde je posuzováno, jestli se opravdu jedná o chybu (bug) nebo je to vlastnost, ale koncový uživatel jí nechápe. Pokud je nahlášená situace shledána chybou, je opravena a vychází v takzvané patch verzi, která zastřešuje opravy těchto chyb. Patch verze jsou nejfrekventovanější, vychází 1 – 2x za měsíc.

1. Hotfix

Speciální případ, který není verzí, nastává tehdy, když je objevena chyba, která je kritická a je potřeba jí okamžitě opravit a nelze s opravou čekat do dalšího releasu. V takovém případě vzniká patch soubor s opravou, který je zanesen do GitLabu a slouží jako dočasné záplata. Oprava vyjde v dalším releasu a zmíněný patch soubor se odstraní.

# Závěr

V této práci jsem se pokusil co nejvíce přiblížit nástroj GitLab a jeho základní funkcionality tak, aby i nezkušený čtenář získal hrubou představu o tom, jak nástroj funguje, k čemu ho lze využít, zároveň jsem poměrně detailně rozvinul možnost integrace CI nástrojů, tak aby bylo patrné k čemu slouží, proč se používají a jaké benefity přinášejí. I tak se ale jednalo o poměrně povrchový popis nástroje jako takového. Dále by se daly porovnávat rozdíly mezi verzí, která je zdarma a prémiovou placenou verzí, která přináší různé další funcionality, jako například fulltextové vyhledávání mezi projekty uloženými v GitLabu, různé statistiky, reporty aktivit jednotlivých uživatelů.

# Seznam použité literatury

Borůvková, Jana, Stanislava Dvořáková a Hana Vojáčková. *Jak psát práce na VŠPJ: Typografická pravidla pro studenty VŠPJ*. Jihlava: Vysoká škola polytechnická Jihlava, 2021. ISBN 978-80-88064-54-1

*Citace.com* [online]. [cit. 2020-12-21]. Dostupné z: citace.com

*GitLab dokumentace*. Online. Https://docs.gitlab.com. Dostupné z:<https://docs.gitlab.com/ee/user/analytics/ci_cd_analytics.html>. [cit. 2024-11-24].

*Jak často vychází patch verze*. Online. Https://community.spiceworks.com. Dostupné z: <https://community.spiceworks.com/t/how-often-do-you-patch-your-system/679323>. [cit. 2024-11-24].

*Verzování softwaru*. Online. Https://semver.org/. Dostupné z: <https://semver.org/>. [cit. 2024-11-24].

*CI Pipelines dokumentace*. Online. Https://docs.gitlab.com. Dostupné z: <https://docs.gitlab.com/ee/user/project/releases/release_cicd_examples.html>. [cit. 2024-11-24].

*What are pipelines?* Online. Https://about.gitlab.com. Dostupné z: <https://about.gitlab.com/topics/ci-cd/cicd-pipeline/>. [cit. 2024-11-24].

*GitLab CI / CD tests*. Online. Https://blog.josefjebavy.cz. Dostupné z: <https://blog.josefjebavy.cz/cs/unix/gitlab-cicd>. [cit. 2024-11-24].

*Co je to GitLab?* Online. Https://cloudfresh.com. Dostupné z: <https://cloudfresh.com/cs/blog/co-je-gitlab/>. [cit. 2024-11-24].

# Přílohy

Jako přílohu, ve které vysvětluji jednotlivé pojmy užité v této semestrální práci, včetně screenshotů z prostředí GitLabu připojuji soubor: **Příloha A - GitLab ukázky**