UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

DEVOPS VO VÝVOJI APLIKÁCIÍ

2016 **Patrik PEKARČÍK**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

DEVOPS VO VÝVOJI APLIKÁCIÍ

BAKALÁRSKA PRÁCA

Študijný program:	Informatika
Pracovisko (katedra/ústav):	Ústav Informatiky
Vedúci diplomovej práce:	RNDr. JUDr. Pavol Sokol, PhD.

Košice 2016

Patrik PEKARČÍK

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Patrik Pekarčík
Študijný program:	Informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	Bakalárska práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: DevOps vo vývoji aplikácií

Názov EN: DevOps in the development of application

Cieľ: (1) Analyzovať možnosti DevOps (development and operations) so zameraním na priebežnú integráciu (continuous integration) a plynule nasadzovanie (continuous delivery).
(2) Porovnať aktuálne prístupy k jednotlivým prvkom DevOps.
(3) Analyzovať možnosti DevOps pri vývoji aplikácii v univerzitnom prostredí.
(4) Navrhnúť a implementovať DevOps pre konkrétny projekt.

Literatúra:
[1] Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering), Addison-Wesley Professional, 2015. ISBN-13: 978-0134049847
[2] Duvall, P.M.: Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.ISBN-13: 978-0321336385.
[3] Humble, J.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler)), Addison-Wesley Professional, 2010.ISBN-13: 978-0321601919.
[4] Swartout, P.: Continuous Delivery and DevOps: A Quickstart Guide - Second Edition, Packt Publishing - ebooks Account. 2014.ISBN-13: 978-1784399313
[5] Berg, A.M.: Jenkins Continuous Integration Cookbook - Second Edition, Packt Publishing - ebooks Account. 2015.ISBN-13: 978-1784390082.

Vedúci: RNDr. JUDr. Pavol Sokol, PhD.
Ústav : ÚINF - Ústav informatiky
Riaditeľ ústavu: prof. RNDr. Viliam Geffert, DrSc.

Dátum schválenia: 05.05.2016

prof. RNDr. Viliam Geffert, DrSc.
riaditeľ ústavu

Abstrakt v štátnom jazyku

So zrýchľujúcou sa dobou sa vytvára čoraz väčší tlak na vývoj softvérových projektov. Ten by mal byť rýchly a flexibilný. Avšak v tradičných prístupoch, akým je napríklad Waterfall, to nie je možné. Z tohto dôvodu vznikajú nové prístupy, ktoré umožňujú zrýchlenie pri vývoji a nasadení projektov. DevOps je jedným z týchto prístupov, ktorý sa v poslednej dobe začína nasadzovať vo veľkých medzinárodných spoločnostiach a umožňuje veľmi rýchle reakcie na meniace sa požiadavky počas vývoja aplikácií. DevOps prináša a popisuje dôležité automatizácie, ktoré v pôvodných prístupoch zaberali príliš veľa času počas tvorby projektov. Medzi tieto spôsoby automatizácie patrí integrácia projektu (continuous integration) a jeho nasadenie do prevádzky (continuous delivery). To umožňuje po každom vykonanom cykle vývoja automaticky zostaviť projekt a tým otestovať, či práca každého z vývojárov je v spoločnom celku funkčná, a to bez akéhokoľvek zásahu človeka. Nasadenie nových verzií projektu je zvyčajne veľmi zdĺhavá činnosť. Avšak pomocou tohto prístupu to odpadá a nasadenie je automaticky vykonané počítačom.

V práci sme navrhli spôsob, akým takýto postup využívať a implementovať v projektoch vznikajúcich na univerzite. Pripravili sme infraštruktúru, na ktorej demonštrujeme navrhovaný spôsob v niekoľkých rôznych projektoch, s ktorými sa študenti stretnú počas štúdia na univerzite.

Kľúčové slová: DevOps, nepretržitá integrácia, nepretržité nasadenie, repozitár

Abstrakt v cudzom jazyku

With the time speeding up, the pressure on development of software projects is also increasing as it should be fast and effective. However, it is not possible to reach using the traditional approaches. This is the reason for inventing new approaches which make it possible to speed up the development and delivery of the project. DevOps is one of these approaches which have been started putting on recently in big international companies and makes it possible to realize fast reactions on changing requirements during the development of applications. DevOps brings and describes important automation which used to take too much time in traditional approaches. Among these ways of automation are continuous integration and continuous delivery. Using them, it becomes possible to build the project automatically after each finished cycle and to test whether the work of each developer is functional without any human intervention. Delivery of new versions of the projects is usually an action requiring a lot of time. However, this time shortens thanks to the approach mentioned above and the delivery is automatically made by the computer.

In this thesis, we designed: a way how to use this approach in University's projects; and an infrastructure on which we demonstrate the suggested methods to be used in several different projects that are being made by students within their studies.

Key words: DevOps, continuous integration. continuous delivery, repository, university development

Obsah

Zoznam ilustrácií

Zoznam tabuliek

Úvod

V posledných rokoch sa informačné technológie stále viac presadzujú v rôznych odvetviach. Tieto odvetvia prinášajú aj rôzne nároky na tvorbu projektov. Medzi najčastejšie nároky patria možnosti rýchlych úprav a flexibilita vývoja. Dnes je už bežné, že zákazník priebežne zmení svoje požiadavky na projekt. Zmeny v požiadavkách nastávajú pomerne často, a preto je potrebné nájsť iný spôsob. Práca vývojárov sa však nekončí s odovzdaním projektu. S vývojom projektov je dnes už pevne spätá aj údržba, kde kontrolujeme, či je projekt dostatočne rýchly a odchytávame priebežne vzniknuté chyby. Počas prevádzky teda vznikajú stále nové požiadavky od výkonnostných zlepšení až po nové požiadavky zákazníka. Aby sme boli schopní udržať takýto projekt pri živote, potrebujeme pripraviť postup, akým by mali pracovať všetci členovia tímu. Týmto spôsobom je DevOps (Development and Operation).

Úlohou DevOps je zozbierať všetky procesy, ktoré prebiehajú pri vývoji projektov. Súčasťou je tiež ukázať, aké závislosti sú medzi jednotlivými procesmi a nakoniec zautomatizovať tie, ktoré sa opakujú. V tejto práci sa zameriavame najmä na podstatnú časť automatizácie procesov: continous integration (priebežnú integráciu) a continous delivery (priebežné dodanie).

Cieľom práce je analyzovať možnosti DevOps a navrhnúť jeho možné využitie pri projektoch vznikajúcich v univerzitnom prostredí. Porovnáme tento prístup s ďalšími prístupmi k vývoju projektov a pripravíme jeho konkrétnu implementáciu na univerzite. Aby bol prechod pre študentov a vyučujúcich hladký, uvedieme niekoľko ukážkových projektov, s ktorými sa študenti stretnú počas svojho štúdia.

Prácu sme rozdelili do šiestich kapitol. V prvej kapitole sa venujeme DevOps. Budeme sa v nej detailnejšie venovať automatizácii continuous integration a jej následnosti continuous delivery. Popíšeme tiež niekoľko základných pojmov, ktoré budeme v práci potrebovať.

V ďalšej kapitole popisujeme podobné prístupy k vývoju projektov, akými sú Waterfall, Agile a pod. Tieto prístupy porovnáme s DevOps a poukážeme na jeho výhody oproti ostatným prístupom.

V tretej kapitole sa venujeme analýze univerzitného prostredia. Ukážeme, čo rozumieme pod pojmom „projekt v univerzitnom prostredí“ a navrhneme spôsob, ako aplikovať DevOps v takýchto projektoch.

V štvrtej kapitole porovnáme dostupné nástroje pre nasadenie DevOps, vyberieme konkrétnu sadu nástrojov a implementujeme ju v prostredí našej univerzity.

Piata kapitola sa venuje trom ukážkovým projektom, v ktorých ukážeme spôsob, ako aplikovať techniky DevOps. V prvom projekte demonštrujeme Java projekt, kde výsledkom úspešnej integrácie je spustiteľný súbor. V druhom projekte vytvoríme webovú stránku pomocou jazyka PHP, ktorá po integrácii bude automaticky nasadená v prostredí Internetu. V poslednom projekte si ukážeme generovanie PDF výstupu z projektov tvorených v jazyku LaTeX.

V poslednej kapitole opisujeme vytvorenú aplikáciu Manažment, ktorá má slúžiť vyučujúcim pri výuke a používaní repozitára zdrojových kódov pri výukových predmetoch. Pomocou tejto aplikácie učiteľ vytvorí nový akademický rok výučby predmetu a priradí k predmetu študentov. Jej súčasťou je tiež kontrola zadaní, ktorá automaticky skontroluje repozitáre študentov a stav ich odovzdania.

1 Úvod do development a operations

Pojem **DevOps** [2] je spoločné označenie pre pojmy „vývoj“ (development) a „operácie“ (operations). V súčasnosti existuje niekoľko definícií DevOps. V širšom slova zmysle je DevOps metódou vývoja, ktorá zlepšuje komunikáciu medzi vývojom a samotnou produkciou. V užšom slova zmysle, DevOps vytvára a udržuje pracovnú infraštruktúru. Výsledkom použitia DevOps je popis práce softwarových inžinierov so zameraním na rýchle a kvalitné dodanie ich softwaru s maximálnou automatizáciou opakujúcich sa procesov.

Na realizáciu DevOps je potrebný balík nástrojov. Mal by obsahovať tieto kategórie procesov:

- **kód** – slúži na vývoj a kontrolu softwaru;
- **komunikácia** – medzi vývojármi počas vývoja;
- **zostavenie** – kontroluje verzie softwaru, stav zostavenia;
- **testovanie** – spúšťa automatické testy rôznych úrovní;
- **manažment verzií** – obsahuje históriu zmien a automatizácie tvorby verzií;
- **konfigurácia** – manažment konfigurácií infraštruktúry a
- **monitorovanie** – kontrola prevádzky softwaru a spokojnosť používateľov.

Vyššie uvedené procesy zabezpečujú, aby sme software vedeli bezproblémovo nasadzovať z vývojového prostredia do ostatných prostredí, ktorým sa budeme podrobnejšie venovať v nasledujúcej kapitole. V tejto časti vývoja systému už vstupuje aj proces automatizácie, a to z dôvodu, že procesy sa v postupe často krát opakujú bezo zmien.

Ciele využitia DevOps pri vývoji softwaru sú nasledovné:

- rýchle opravy problémov;
- kontinuálne nasadenie softwaru;
- rýchlejšie dodanie nových vlastností a
- stabilnejšie produkčné prostredia.

1.1 Prostredie

V práci je nutné venovať sa **prostrediu** (environment) [11]. Pod pojmom „prostredie“ rozumieme počítačový systém, na ktorom je náš software nasadený

a spúšťaný. V rámci práce budeme rozlišovať tri hlavné prostredia, a to vývojové, testovacie a produkčné prostredie. Rozdiely medzi týmito prostrediami spočívajú v ich veľkosti, výkone a samotnom využití.

Vývojové prostredie je prvé prostredie, v ktorom sa projekt nachádza. Je to priestor (napr. server, úložisko a pod.), v ktorom tvoria členovia vývojového tímu projekt. Malo by byť rovnako nakonfigurované pre všetkých členov. Súčasne by mali byť v dokumentácii zapísané všetky požiadavky na podporné programy. Všetci programátori by mali mať ľahký prístup ku zdrojovým kódom, na čo je ideálne využiť repozitár zdrojových kódov. Pomocou repozitára budeme vedieť prenášať nové zdrojové kódy na centrálny server, odkiaľ si ich všetci programátori budú môcť sťahovať a vyvíjať tak vždy na aktuálnej verzii programu.

Ďalším prostredím je **testovacie prostredie**, ktoré by malo simulovať väčšinu možností nasadenia systému v produkčnom prostredí. Prostredie by malo mať tiež zdokumentované požiadavky na konfiguráciu, ktoré musia byť nutne aj podmnožinou požiadaviek na vývojové prostredie. Testovacie prostredie napríklad nepotrebuje programy nevyhnutné pre vývoj, ktoré máme na vývojovom prostredí. Tu už zavádzame automatizáciu procesov nasadenia projektu. Medzi tieto procesy patrí stiahnutie zdrojových kódov z repozitára a spustenie testov projektu.

Produkčné prostredie je posledným miestom, kam sa program môže dostať. To, že je program v produkčnom prostredí znamená, že ho už používajú koncoví užívatelia. Produkčné prostredie by malo spĺňať požiadavky na konfiguráciu. Tieto požiadavky sú podmnožinou požiadaviek na testovacie prostredie s vynechanými testovacími nástrojmi. Produkčné prostredie je často distribuované na viacerých výpočtových zariadeniach, aby zvládlo nápor užívateľov.

1.2 Repozitár zdrojových kódov

Manažment zdrojových kódov a ďalších súčastí projektu je veľmi dôležitý. Tvorí ho **repozitár zdrojových kódov** [8], ktorý nám poskytuje databázu všetkých zmien vykonaných na projekte od jednotlivých vývojárov. Poskytuje vždy najaktuálnejšiu verziu zdrojových kódov projektu pre vývojárov. Tí si ho vedia stiahnuť vždy od svojich kolegov bez toho, aby si ich náročne kopírovali a spájali vo svojom prostredí. Repozitár

navyše umožňuje vrátiť sa k starým verziám zdrojového kódu projektu, ako aj porovnávať vykonané zmeny medzi jednotlivými verziami.

1.3 Continuous integration (priebežná integrácia)

V základnom ponímaní predstavuje **continuous integration** [3,7,12] postup, ktorý sleduje repozitár zdrojových kódov a v prípade zmeny automaticky stiahne, zostaví a otestuje aplikáciu. Tento automatizovaný postup ďalej využívame na udržiavanie kvality softwaru. Dosiahneme to špecifikovaním podmienok, ako by kód mal vyzerať, resp. akú štruktúru by mal dodržiavať. Pri každom cykle testovania projektu otestujeme aj dodržanie podmienok v samotnom zdrojovom kóde.

Výsledkom tohto postupu je obrazovka stavu projektu, na ktorej sa zobrazia informácie o úspešnosti testovania a ohodnotenie dodržaných podmienok zdrojových kódov. Hodnoty sú zobrazené v percentách a porovnané s predchádzajúcimi testovaniami. V prípade negatívnych výsledkov sa zobrazia presné informácie a pomocou komunikačných nástrojov sú informovaní zodpovední vývojári.

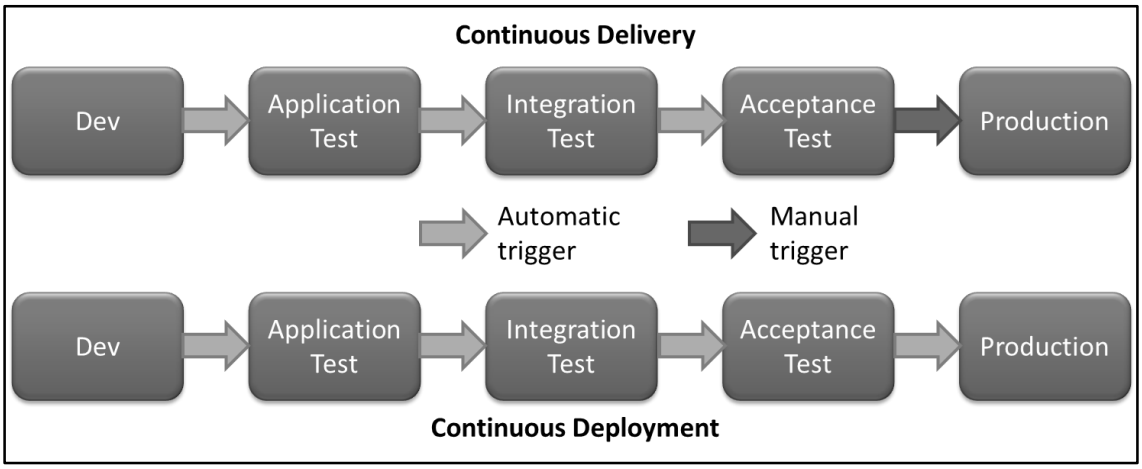
Podstata continuous integration spočíva v znížení rizika pomocou poskytnutia rýchlejšej spätnej väzby (feedback). V prvom rade pomáha rýchlejšie identifikovať a opraviť problém integrácie a regresie, čím sa dosiahne jednoduchšia a rýchlejšia prístupnosť a znížený počet chýb. Poskytnutím prehľadu pre technických i netechnických členov tímu, continuous integration dokáže otvoriť a spravovať komunikačné kanály medzi členmi tímu. Tým podporuje spoločné riešenie problémov a zlepšenie procesov. Automatizáciou vývojového procesu continuous integration pomáha v tom, aby sa software dostal k testerom a koncovým užívateľom rýchlejšie, spoľahlivejšie a s použitím menšieho úsilia.

1.4 Continuous delivery (priebežné dodanie) a continuous deployment (priebežné nasadenie)

Continuous deployment [4,7,13] považujeme za rozšírenie continuous integration zameriavajúce sa na minimalizáciu času medzi vývojom a samotným nasadením do produkcie. Aby vývojári získali continuous deployment, spoliehajú sa na infraštruktúru, ktorá automatizuje jednotlivé kroky nasadenia do produkčného prostredia.

Prepojenie continuous deployment s continuous integrations spočíva v tom, že do produkčného prostredia nasadíme len funkčné a kvalitné verzie softwaru.

Continuous delivery [14] na rozdiel od continuous deployment pred nasadením vyhodnocuje aj ďalšie faktory. V tomto postupe nie je verzia softwaru automaticky nasadená do produkčného prostredia, ale je označená za kandidáta na nasadenie. Vieme ju jednoducho nasadiť kliknutím tlačidla „Spusti“. Mohli by sme povedať, že to znamená krok späť oproti continuous deployment. Avšak tu už do rozhodovania vstupuje aj vyšší manažment, ktorý rozhodne o tom, kedy je ten správny čas na vydanie novej verzie.



Obrázok 1 Rozdiel medzi Continuous Delivery a Deployment [21]

1.5 Ďalšie možnosti DevOps

Keďže DevOps je prístup, tak to samozrejme nekončí pri continuous integration a continuous delivery. DevOps prakticky zastrešuje všetky postupy, ktoré prebiehajú pri vývoji. V nasledujúcom texte spomenieme 2 postupy, ktoré DevOps zastrešuje.

Continuous security [15] je postup, ktorý do projektu vnáša aj bezpečnostné kontroly softwaru. Vykonávajú sa testovaním softwaru voči známym bezpečnostným chybám. Prináša aj mnohé pravidlá pre kontroly zdrojových kódov, ktoré majú zabezpečiť software pred možným zneužitím implementácií alebo pred použitím nebezpečných knižníc.

Monitoring sa aplikuje na produkčné rozhranie a prináša vývojovému tímu hodnotné štatistiky o efektívnosti vyvinutého softwaru. Pri výskyte neočakávaných chýb je tím automaticky informovaný komunikačným nástrojom o presných informáciách

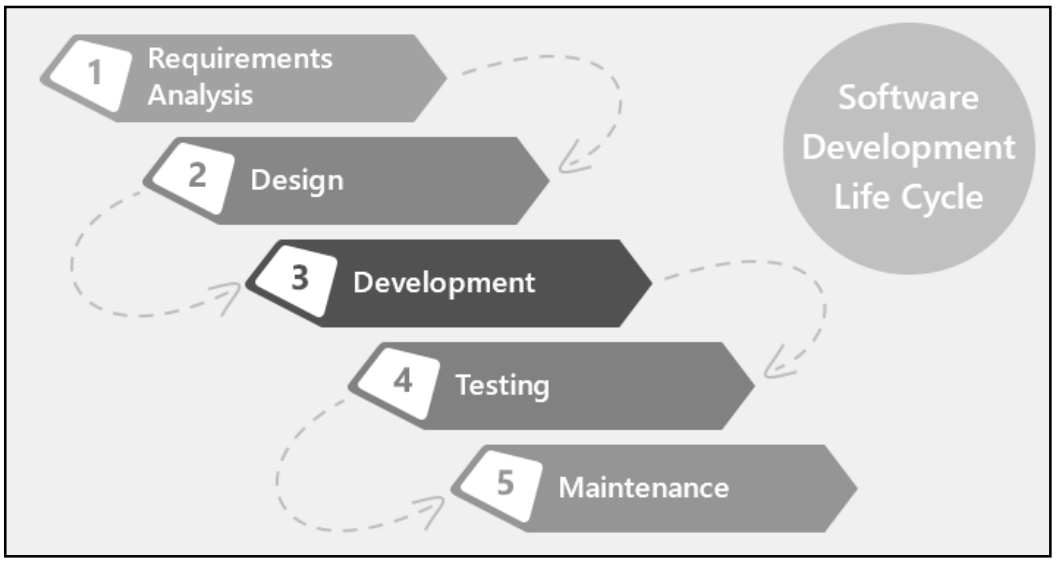
o chybe. Výsledkom monitoringu je predvídanie nedostatočného výkonu prostredia a samozrejme pomoc pri riešení problémov z produkčného prostredia.

2 Podobné prístupy

Existuje niekoľko prístupov ako tvoriť projekty. V nasledujúcej časti sa pozrieme na niektoré z nich a porovnáme si ich s našim prístupom DevOps. Medzi najznámejšie prístupy patria hlavne prístupy Waterfall, Agile a Lean.

2.1 Waterfall

Model Waterfall [1,16] je často považovaný za klasický prístup k vývoju softwaru. Tento model je označovaný ako lineárny a sekvenčný. Má určené samostatné ciele pre každú fázu vývoja. Ďalší krok je vždy možné začať až vtedy, keď ten predchádzajúci je úspešne dokončený. Akonáhle sme sa už dostali do ďalších krokov, tak sa nesmieme vrátiť späť.



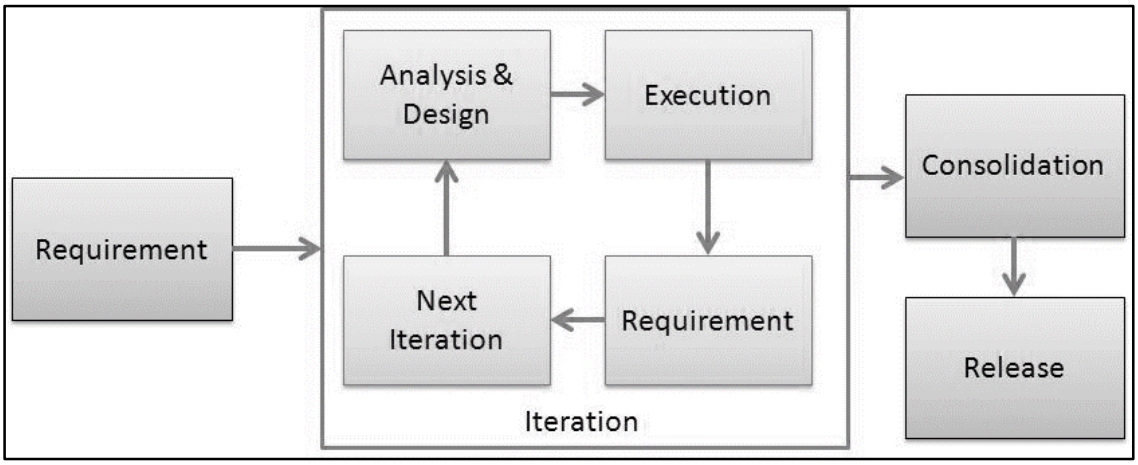
Obrázok 2 Waterfall model [22]

Výhodou tohto modelu je možnosť pevného plánovania a určenia termínov dokončenia pre každú fázu vývoja a tak vydať software načas. Vývoj prebieha postupne cez tieto kroky: koncept, dizajn, implementácia, testovanie, nasadenie, úpravy a nakoniec údržba.

Nevýhodou takého vývoja je, že nedovoľuje meniť výsledky predchádzajúcich krokov, a teda vývoj nie je flexibilný voči zmenám. Ak sa v testovacej fáze objavia chyby v analýze, tak je ich pomerne náročné opraviť.

2.2 Agile

Agilný spôsob vývoja [1,17] je prístup určený primárne pre kreatívny prístup k softwaru, ktorý potrebuje flexibilitu konceptu a aplikuje istú úroveň pragmatizmu vývoja. Dôsledkom toho je dodávanie hotových častí softwaru koncovým užívateľom, a to v krátkych pravidelných časových intervaloch. Agilný spôsob sa zameriava na udržiavanie prehľadného zdrojového kódu a pravidelné testovanie. Cieľom použitia agilného spôsobu je nasadzovanie malých funkčných časti softwaru v stanovených časových iteráciách, namiesto dodania celého veľkého softwaru na konci vývoja.



Obrázok 3 Agile Model [23]

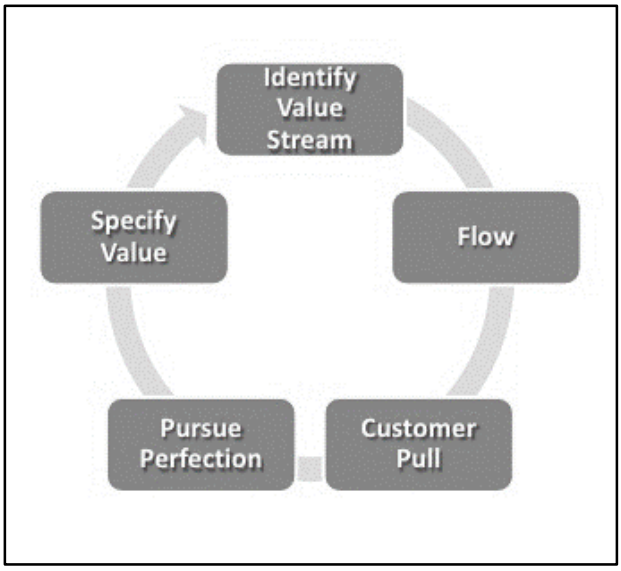
Jednou z hlavných výhod agilného vývoja je dosiahnutie správneho projektu na konci vďaka tomu. že je prispôsobiteľný zmenám, ktoré nastali počas vývoja. Zbytočné požiadavky je možné včas zrušiť a taktiež je možné rozširovať projekt o nové, hodnotnejšie požiadavky.

2.3 Lean

Lean software development [18] je metodika prebraná z Lean manufakturing vyvinutým firmou Toyota. Je to metodika, ktorej cieľom je uspokojiť zákazníkove požiadavky v maximálnej možnej miere, a to spôsobom, že bude vyrábať len to, čo zákazník požaduje so zameraním na rýchlosť a kvalitu. Lean software development bol spísaný v rovnomennej knihe a môže byť sumarizovaný siedmimi princípmi:

1. **minimalizácia odpadu** – nevykonávať na softwari veci, ktoré neposkytujú zákazníkovi žiadnu pridanú hodnotu;

2. **zlepšiť vzdelanie** – vývojári by sa mali kontinuálne vzdelávať, čoho výsledkom je kvalitnejší a efektívnejší kód;
3. **robiť rozhodnutia najneskôr, ako je to možné** – aby sme mohli robiť rýchle rozhodnutia aj počas tvorby softwaru;
4. **nasadenie najskôr, ako to je možné;**
5. **upevniť kolaboráciu tímu;**
6. **stavať integritu** – jedná sa o vnímanie softwaru a o jeho dodaní zákazníkovi, aký je intuitívny, atď.;
7. **pohľad na software ako na celok** – definovanie vzťahov medzi jednotlivými jednotkami vývoja.



Obrázok 4 Lean model [24]

Vplyvom tejto metodiky sa proces vývoja neustále zrýchľuje vďaka primárnemu zameraniu na používateľove požiadavky. To má vplyv aj na náklady na vývoj. Využitím tohto postupu prinášame používateľom viac najhodnotnejších požiadaviek v kratšom čase, čo má za dôsledok silnejšiu motiváciu samotných vývojárov.

2.4 Porovnanie prístupov

Porovnanie techník uvedených v predchádzajúcich podkapitolách (Waterfall, Lean a Agile) si ukážeme na príklade pridania novej požiadavky používateľa v závislosti od času, kedy mu bude poskytnutá nová verzia. Predpokladáme, že požiadavka

od používateľa je aktuálne najhodnotnejšia (najpotrebnejšia) a taktiež, že projekt je v pokročilejšej fáze prípravy nových funkcionalít.

V prípade Waterfall je potrebné projekt dostať znova do prvej fázy, pridanú požiadavku zanalyzovať a zaradiť, kde v projekte bude. Takáto zmena fázy však úplne zastaví prebiehajúcu fázu programovania, prípadne už testovania, ku ktorej sa môžeme vrátiť až po dopracovaní novej požiadavky do každej nadradenej fázy. Toto môže trvať aj niekoľko mesiacov a zmena príde až so všetkými ďalšími pripravovanými požiadavkami. V prípade Agile a Lean by sa táto požiadavka zapracovala hneď na začiatku ďalšieho vývojového cyklu, čo je spravidla začiatok týždňa a zapracovaná vie byť ešte v ten týždeň, a poskytnutá bude na konci aktuálneho vývojového cyklu spolu s niekoľkými ďalšími požiadavkami, na ktorých sa pracovalo (vývojový cyklus v Agile trvá 1-2 týždne).

V prípade DevOps bude požiadavka dopracovaná a poskytnutá používateľovi oveľa rýchlejšie. Pretože DevOps, ktoré využíva praktiky Agile a Lean, navyše definuje aj automatizácie v podobe continuous integration a continuous delivery. Continuous integration zabezpečí, aby nenastali triviálne chyby a projekt robil to, čo zákazník požaduje. Continuous delivery poskytne novú verziu používateľovi hneď, ako to bude možné.

3 Analýza a návrh v univerzitnom prostredí

V tejto časti sa zameriame na projekty, na akých študenti pracujú počas svojho štúdia v spolupráci so svojimi vyučujúcimi. Následne navrhneme spôsob, ako v týchto projektoch využiť poznatky získané z metódy DevOps.

3.1 Projekty

Existujú rôzne typy projektov, s ktorými sa študenti zaoberajú počas svojho štúdia. Prvým typom projektov sú domáce úlohy (zadania), ktoré sú priamo späté so štúdiom daného študenta. Ich podstata spočíva v tom, že vyučujúci očakáva od študenta na konci akademického semestra odovzdanie vlastnej práce, v ktorej študent využije poznatky, ktoré sa naučil počas štúdia predmetu. Nazvime tento typ projektu **výučbovými projektmi**.

Ako ďalší typ projektu, s ktorým sa stretávame, sú **záverečné práce**. Využitie DevOps pri tvorbe záverečnej práce zlepší komunikáciu medzi vedúcim a študentom, a to hlavne zjednodušením prístupu k aktuálnej verzii práce, na ktorej spoločne pracujú. Využitie DevOps by nemalo spočívať len v samotnej vytváranej aplikácii. Využijeme ho aj na tvorbu textového obsahu práce. Príkladom použitia môže byť LaTeX, u ktorého je možné priebežne vytvárať súbory vo formáte PDF, a to len zo spoločného miesta pre verzie práce. Medzi záverečné práce zaraďujeme:

- Bakalárske práce (magisterské, rigorózne, ...) a
- Vedecké články

Netreba však zabudnúť na vlastnú iniciatívu študentov. Je potrebné im ponúknuť podporu pri tvorbe **vlastných projektov**, a to hlavne poskytnutím vybudovanej infraštruktúry pre školské projekty.

3.2 Návrh nasadenia DevOps

Ako sme uviedli v úvodnej kapitole, DevOps je metódou a nie konkrétnym nástrojom. Preto je nutné venovať pozornosť tomu, aké nástroje budeme využívať a ako ich budeme využívať.

Najdôležitejším nástrojom je **repozitár dokumentov** (nástroj na centrálne uloženie zdrojových kódov s históriou zmien). V nasledujúcich podkapitolách si ukážeme, ako

tento repozitár využívať pri projektoch v univerzitnom prostredí. Navrhneme si jednu štruktúru, na ktorej si ukážeme výhody, ktoré prináša.

3.2.1 Výučbové projekty

Pri návrhu štruktúry využitia repozitára vo výučbových projektoch je nutné hľadieť na viacero špecifík, ktoré sú s projektmi pevne späté. Repozitáre zdrojových kódov nám ponúkajú viacero úrovní členenia projektov. Týmito úrovňami sú:

Skupina → Projekt → Branch.

Vnímajme vyučovaný predmet ako jednu skupinu s rovnakým názvom. Počas výučby predmetu sa od každého študenta očakáva vlastná práca, prípadne skupinová práca od viacerých študentov. Preto bude každému študentovi pridelený vlastný, rovnomenný projekt. Z pohľadu vyučujúceho je to priestor, kde budú študenti odovzdávať vypracované zadania. Vyučujúci takto bude mať prístup ku všetkým vypracovaným zadaniam, navyše vždy k ich aktuálnej verzii.

Pokiaľ chceme, aby študenti využívali tento nový spôsob, je potrebné priniesť výhody aj pre nich. Takou výhodou je poskytovanie výučbových materiálov cez repozitár, a to vytvorením projektu „vyucba“ v skupine vyučovaného predmetu. Okrem výučbových materiálov by sa tu zverejňoval harmonogram predmetu, domáce zadania, prípadne zadania záverečných projektov.

Vieme, že jeden predmet sa opakovane vyučuje novým študentom v ročnom intervale. Preto si definujme rozdelenie, ako presne identifikovať rok výučby. Túto identifikáciu urobíme pomocou posledného členenia, ktorým je branch. V každom projekte máme dokumenty/materiály zverejnené v branchi s názvom master/rok. Názov branch využijeme kvôli známej konvencii využívania repozitárov a pripísaním roku za lomku presne identifikujeme rok, ktorému príslušné materiály prislúchajú. Toto delenie má najväčší význam pri výučbovom projekte.

Nasledujúca tabuľka zobrazuje štruktúru vytvorenú podľa vyššie uvedeného formátu, určenú pre predmet Algoritmy a štruktúry údajov, dvoch študentov a s dvomi rokmi výučby vo výučbovom branchi.

Tabuľka 1 Štruktúra výučbových projektov

Skupina	*Projekt*	*Branch*
UI_ASU	/ Vyuka	/ master/2015
		/ master/2016
	/ pekarcik_patrik	/ master/2016
	/ mrkvicka_jozko	/ master/2016

3.2.2 Záverečné práce

Podobne, ako pri výučbových projektoch, aj pri záverečných prácach využijeme úrovne členenia repozitára. V tomto prípade už nie je potrebné vytvárať skupinu, pretože využijeme tú vlastnosť repozitára, že každý užívateľ má vlastnú rovnomennú skupinu. Podľa konvencie repozitára zdrojových kódov sú úrovne nasledovné:

Používateľ → Projekt → Branch.

Tvorca záverečnej práce teda vytvorí pod svojím menom príslušný projekt (bakalarska_praca, magisterska_praca, ...), v ktorom bude uchovávať všetky dokumenty potrebné pri tvorbe záverečnej práce. Do vytvoreného projektu má prístup aj školiteľ, ktorý môže tiež zasahovať do tvorby tejto práce.

Rozdeľme projekt záverečnej práce na dva branche: master a publish. **Branch master** je miesto, v ktorom budeme uchovávať rozpracovanú prácu, ktorú priebežne upravujeme spolu so školiteľom práce. Obsahom **branchu publish** budú dokumenty nutne potrebné pre zverejnenie práce. Týmito dokumentmi sú:

- práca v PDF formáte;
- výstup práce (EXE, JAR) – spustiteľná verzia a
- prílohy – zdrojové kódy, návody, ilustrácie.

Branch publish bude generovaný automaticky počas fázy continuous delivery. To znamená, že pri tvorbe projektu už začneme využívať automatizácie, počas ktorých sa vytvoria dokumenty z našich zdrojových súborov v branchi master. V ďalšej kapitole si na príklade ukážeme spôsob, akým to docielime.

Nasledujúca tabuľka zobrazuje štruktúru vytvorenú podľa vyššie uvedeného formátu u dvoch študentov.

Tabuľka 2 Štruktúra projektu záverečných prác

Používateľ		*Projekt*		*Branch*
pekarcik_patrik	/	bakalarska_praca	/	master
			/	publish
	/	magisterska_praca	/	master
mrkvicka_jozko	/	bakalarska_praca	/	master

3.2.3 Vlastné projekty

Študenti majú počas štúdia aj veľa vlastných nápadov - či už sú to malé nástroje, ktorými si uľahčujú každodenný život, alebo ich osobná stránka. Môžu prísť aj s komplexnou myšlienkou, ktorá má potenciál stať sa v budúcnosti úspešnou. Cieľom každej školy by mala byť podpora iniciatívnych študentov. Poskytnutie infraštruktúry je dobrou cestou podpory týchto študentov. Práca na vlastných projektoch je veľmi podobná práci na záverečnej práci, a preto odporúčame využiť štruktúru, ktorú sme navrhli práve pre záverečné práce.

4 Implementácia navrhovaného riešenia

DevOps je metódou popisujúcou viaceré postupy, ktoré sme spomenuli v prechádzajúcich kapitolách. Pre tieto postupy sa však vyžaduje využitie istej sady nástrojov, ktorá nám pomôže úspešne splniť ciele DevOps. Nástroje, ktoré budeme potrebovať v tejto kapitole, sú:

- repozitár zdrojových kódov;
- nástroj zostavenia a testovania projektu a
- server nasadenia projektu (produkčné prostredie).

Pre tieto nástroje existuje veľa implementovaných riešení. Z tohto dôvodu sme určili kritériá, podľa ktorých vyberieme konkrétne implementované riešenie. Prvým kritériom je **komplexnosť systému**, čo znamená pokryť čo najviac nástrojov jedným riešením. Ďalšie kritérium je riešenie s aktívnou komunitou vývojárov. Toto je veľmi dôležité z hľadiska údržby našej implementácie. Potrebujeme, aby bolo použité riešenie funkčné na dlhé obdobie, počas ktorého sú možné opravy chýb. Zároveň vybrané riešenie musí zvládnuť veľké množstvo používateľov. Posledným kritériom sú možnosti zabezpečenia projektov v danom riešení.

V prípade repozitára zdrojových kódov sme sa rozhodli porovnať tri aktuálne najpoužívanejšie riešenia: GitHub, GitLab a Bitbucket. [20]

Tabuľka 3 Porovnanie repozitárov zdrojových kódov

	GitLab	**GitHub**	**Bitbucket**
Nástroje	Repozitár (technológia GIT)		
Continuous integration	vstavané	Travis[1]	Bamboo[2]
Open source	áno	nie	nie
Posledná verzia	Apríl 2016	Marec 2016	Apríl 2016
Zabezpečenie projektov	súkromné[3], verejné, interné[4]		
Vlastná infraštruktúra	áno		
OAuth 2.0	áno		

[1] Nástroj inej spoločnosti
[2] Formou dokúpenia ďalšieho nástroja
[3] Len pre určenú skupinu užívateľov
[4] Len registrovaným užívateľom

Nástroje zostavenia a testovania projektu sme vyberali pomocou podobných kritérií. Pozreli sme sa tiež na rozsah projektov, ktoré vieme nástrojom testovať a možnosti testovania na rôznych operačných systémoch.

Tabuľka 4 Porovnanie continuous integration nástrojov

	GitLab CI	**Jenkins**	**Bamboo**	**Travis CI**
Open source	áno	áno	nie	nie
Vlastná infraštruktúra	áno	áno	áno	nie
Podporované OS[5]	Linux Max OSX Windows	Linux Max OSX Windows	Linux Max OSX Windows	Linux Max OSX
Podpora virtualizácie	Docker VirtualBox Parallels	Docker[6]	Docker	neznáme
Posledná verzia	Apríl 2016	Apríl 2016	Marec 2016	Apríl 2016

Na základne porovnaní sme sa rozhodli použiť open source riešenie **GitLab**, ktoré vieme nasadiť na vlastnej infraštruktúre. Pre continuous integration sme sa rozhodli použiť vstavené riešenie GitLab-u, **GitLab CI**. Ako jediné z porovnávaných riešení nám ponúka širokú rôznorodosť nasadenia na rôznych virtualizačných prostrediach.

[5] Operačné systémy
[6] Formou rozšírenia

5 Ukážkové projekty

V predchádzajúcich kapitolách sme navrhli spôsob prístupu vývojárov s cieľom aplikovať DevOps. Tiež sme vybrali a implementovali konkrétne riešenie GitLab. V tejto kapitole sa venujeme trom ukážkovým projektom. V prvom projekte demonštrujeme Java projekt, kde výsledkom úspešnej integrácie je spustiteľný súbor. V druhom projekte vytvoríme webstránku pomocou jazyka PHP, ktorá po integrácii bude automaticky nasadená na Internet a v poslednom projekte si ukážeme generovanie PDF výstupu z projektov vytvorených v LaTeX-u.

5.1 Kompilovateľné jazyky

Na demonštráciu sme si zvolili jazyk Java, kde s pomocou zastavovacieho nástroja Maven [32] vytvoríme aplikačný súbor Jar. Projekt je jednoduchá kalkulačka (trieda Kalkulacka) s testom jej metód (trieda KalkulackaTest). Na testovanie použijeme knižnicu z Javy JUnit, ktorú pridáme do závislostí projektu Maven.

```
<dependency>
<groupId>junit</groupId><artifactId>junit</artifactId>
<version>4.12</version><scope>test</scope>
</dependency>
```

Takto vytvorený projekt môžeme testovať na vlastnom vývojárskom prostredí, avšak to pre požiadavky DevOps nestačí. Aby sme ich splnili, je potrebné uložiť projekt v repozitári zdrojových kódov. Vo webovom rozhraní GitLab vytvoríme nový privátny projekt (javademo) pod užívateľom (patrik.pekarcik). Uloženie zdrojových kódov z počítača na repozitár vykonáme z priečinka projektu nasledovnými príkazmi:

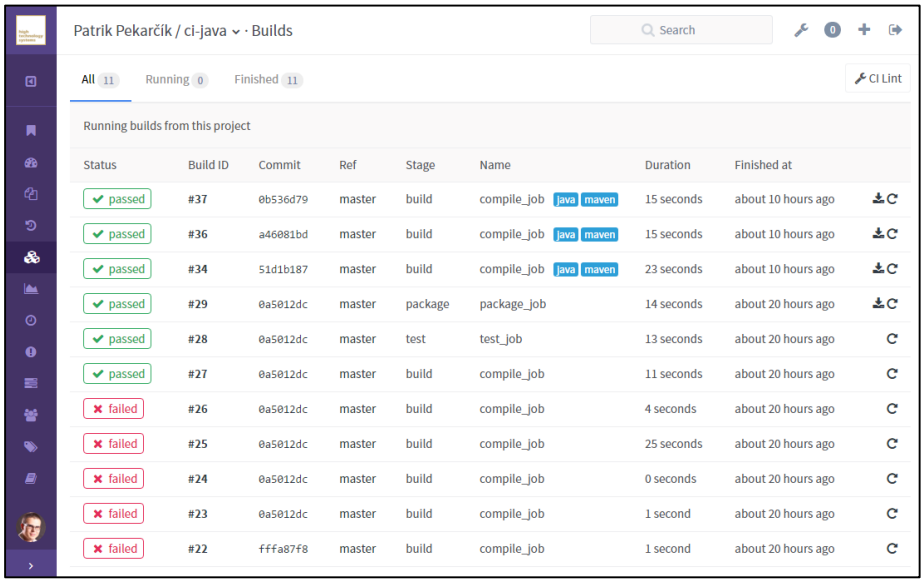
```
git init
git remote add origin
git@gitlab.science.upjs.sk:patrik.pekarcik/javademo.git
git add .
git commit –m initial
git push -u origin master
```

Posledným krokom je spustenie continuous integration na vytvorenom projekte. To spravíme pridaním súboru **.gitlab-ci.yml** do projektu s obsahom:

```
stages:
  - build
```

```
job:
  stage: build
tags:
  - java
  - maven
script:
  - mvn compile
  - mvn test
  - mvn package
artifacts:
  paths:
    - target/javademo-1.0.jar
```

Takto upravený projekt bude po každom uložení na repozitár zostavený a otestovaný na serveri. Po úspešnom dokončení všetkých cieľov bude aplikačný súbor Jar zverejnený na webovom rozhraní repozitára.



Obrázok 5 Webové rozhranie repozitára. Zostavenia projektu v Jave

5.2 Interpretované jazyky

Na demonštráciu sme zvolili jazyk PHP, kde za použitia nástroja composer označíme závislosti tvorenej webstránky (použité knižnice, rozšírenia PHP). V projekte sme využili framework FuelPHP [26] vhodný na tvorbu rôznych webových projektov. Na testovanie projektu použijeme knižnicu PHPUnit [33], ktorú pridáme do závislostí projektu nasledovne:

```
composer global require "phpunit/phpunit=4.*"
```

Okrem zostavenia a testovania v tomto projekte ukážeme automatické nasadenie aj v prípade, že projekt bude úspešne otestovaný. Na nasadenie do produkčného prostredia použijeme cloud službu Heroku [34]. Na službe sme si vytvorili používateľské konto a server Heroku nám vygeneroval špeciálny API kľúč, pomocou ktorého budeme aplikáciu automaticky nasadzovať do produkčného prostredia. Vložíme ho do projektu pomocou webového rozhrania GitLab ako premennú pod názvom **HEROKU_API_KEY**.

Zdrojové kódy - rovnako ako v prvom projekte - uložíme do repozitára (projekt: phpdemo, užívateľ: patrik.pekarcik). Pre naplnenie určených cieľov projektu, vytvoríme súbor .**gitlab-ci.yml** s obsahom:

```
stages:
  - build
  - deploy
build_job:
  stage: build
tags:
  - php
  - composer
script:
  - composer install
  - php oil test
deployment_job:
  stage: deploy
tags:
  - php
  - composer
  - dpl
dependencies:
  - build_job
script:
  - dpl --provider=heroku --app=phpdemo--api-key=$HEROKU_API_KEY
```

5.3 LaTeX projekt

Posledným projektom demonštrujeme generovanie PDF zo zdrojových súborov LaTeXu. Zdrojové kódy - rovnako ako v prvom projekte - uložíme do repozitára (projekt:

latexdemo, užívateľ: patrik.pekarcik). Náš cieľ automatického generovania dosiahneme pridaním continuous integration v projekte. Vytvoríme súbor **.gitlab-ci.yml** s obsahom:

```
stages:
  - build
compile_job:
  stage: build
tags:
  - tex
  - latex
script:
  # príprava bibliografie
  - pdflatex zaverecna-praca.tex
  # opakované zostavenie pri použití krížových referencii
  - pdflatex zaverecna-praca.tex
artifacts:
  paths:
    - zaverecna-praca.pdf
```

6 Manažment výučbových predmetov

V kapitole 3.2.1 sme navrhli nový spôsob používania repozitára zdrojových kódov. V aktuálne nasadenom riešení by vytvorenie takej štruktúry vyžadovalo zdĺhavú prácu s konfiguráciou. Preto sme sa rozhodli vytvoriť administračné rozhranie s názvom Manažment výučbových predmetov. Pomocou neho zjednodušíme prácu vyučujúcim pri konfigurácii ich predmetu.

Manažment rozdelíme na tri časti, ktorým sa musíme špeciálne venovať. Časti sme navrhli tak, aby sa vyučujúcim čo najviac minimalizoval počet krokov, ktoré musia vykonať na vytvorenie svojho výučbového predmetu.

Prvou časťou je autentifikácia do nami vytvoreného Manažmentu. Tu sme sa rozhodli nevytvárať nanovo všetkých používateľov, ale využiť štandard Oauth2 [27], pomocou ktorého vieme používateľa bezpečne autentifikovať voči repozitáru zdrojových kódov. Tento štandard nám ponúka väčšina webových rozhraní repozitára zdrojových kódov.

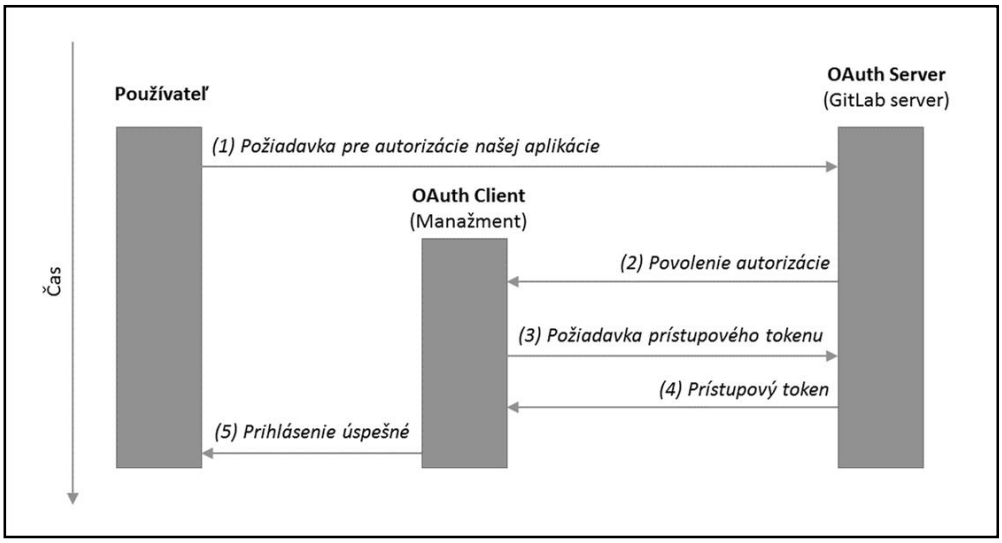
V druhej časti sa venujeme navrhnutému spôsobu používania. Autentifikovaný používateľ vytvára výučbové projekty, otvára aktuálny výučbový rok a prideľuje študentov do výuky. Okrem vytvárania projektov a študentských účtov pridáme aj funkcionalitu zadaní. Tá bude podľa konfigurácie sledovať repozitáre študentov, či odovzdali zadanie a bude obsahovať presmerovanie priamo na študentovo vypracovanie.

Posledná časť slúži na samotné prepojenie medzi Manažmentom a repozitárom zdrojových kódov. Pomocou API rozhrania repozitára zdrojových kódov bude automaticky vytvorená štruktúra, ktorú sme nastavili v druhej časti, priamo do repozitára zdrojových kódov. Taktiež sa tu bude vykonávať kontrola repozitárov študentov na vypracované zadania.

Manažment sme sa rozhodli vyvíjať v jazyku PHP 5.6 [25] s využitím continuous integration a continuous delivery, ktoré sme navrhli v kapitole 5.2. Okrem samotného jazyka využívame aj nadstavbu v podobe frameworku FuelPHP [26], ktorý nám udáva Model-View-Controller štruktúru a niekoľko interface-ov potrebných pre tvorbu našej aplikácie.

6.1 Autentifikácia

OAuth 2.0 [27,30] je otvorený protokol, ktorý dovoľuje bezpečnú autentifikáciu rôznych aplikácií voči webovému serveru. Tento spôsob nevyžaduje zadávanie prihlasovacích údajov od používateľa do našej aplikácie, avšak presmeruje používateľa do repozitára, voči ktorému bude autentifikovaný do našej aplikácie.



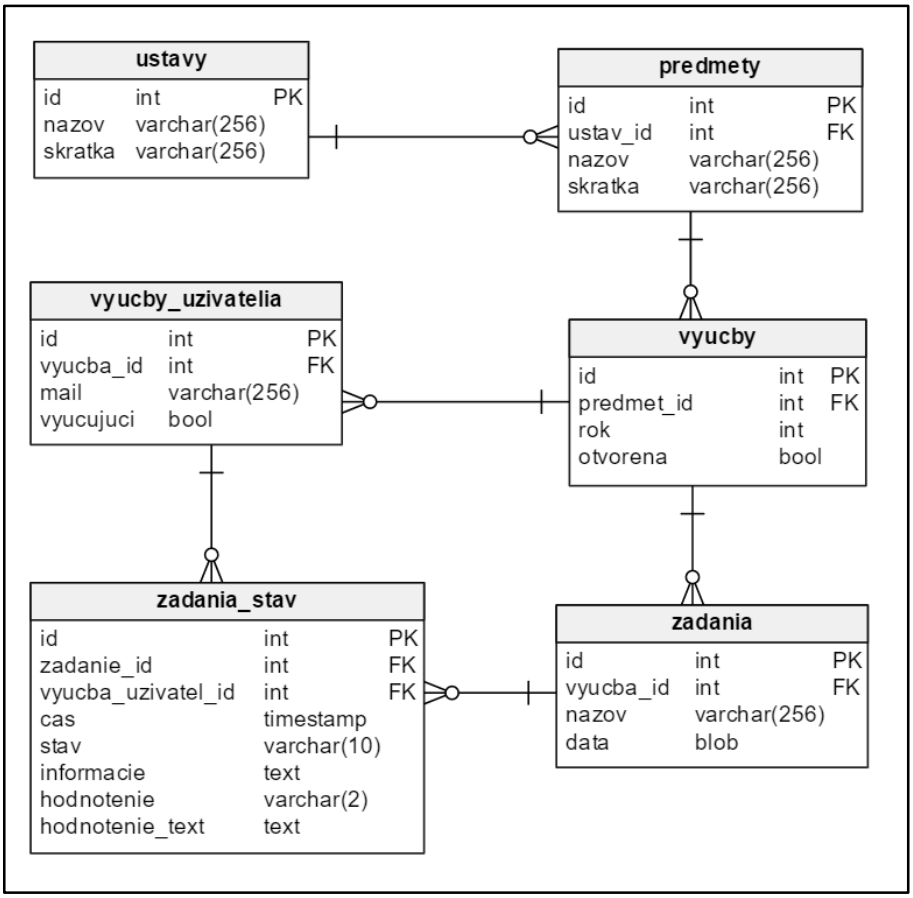
Obrázok 6 Schéma prihlásenia pomocou OAuth 2.0 v aplikácii Manažment

Na obrázku je znázornený princíp prihlásenia pomocou protokolu OAuth 2.0. V prvom kroku používateľ kliknutím na odkaz v aplikácii Manažmentu požiada OAuth Server o autorizáciu aplikácie Manažment. OAuth server odpovedá aplikácii Manažment kódom autorizácie. Aplikácia následne posiela požiadavku na prístupový token OAuth serveru. Táto požiadavka obsahuje kód povolenia autorizácie a špeciálny kód, ktorým je aplikácia Manažment registrovaná na OAuth serveri. OAuth server následne odpovedá prístupovým tokenom úspešného prihlásenia a následne aplikácia už len informuje používateľa o úspešnom prihlásení.

Na implementáciu autentifikácie do Manažmentu sme využili vstavaný interface Auth použitého frameworku FuelPHP, podporujúci OAuth pomocou open source projektu **Opauth** [28]. Projekt opauth tvorí interface na prihlasovanie k rôznym službám, akými sú Facebook, Twitter, Gmail atď. V práci sme vytvorili implementáciu tohto interface-u aj pre nami použitý repozitár zdrojových kódov a tiež sme zverejnili túto implementáciu ako open source projekt [29].

6.2 Správa predmetov, študentov a zadaní

V tejto časti Manažmentu vytvoríme databázovú štruktúru, v ktorej budeme uchovávať informácie o výukových projektoch navrhnutých v kapitole 3.2.1. Na nasledujúcom obrázku je znázornený navrhnutý **databázový model.**



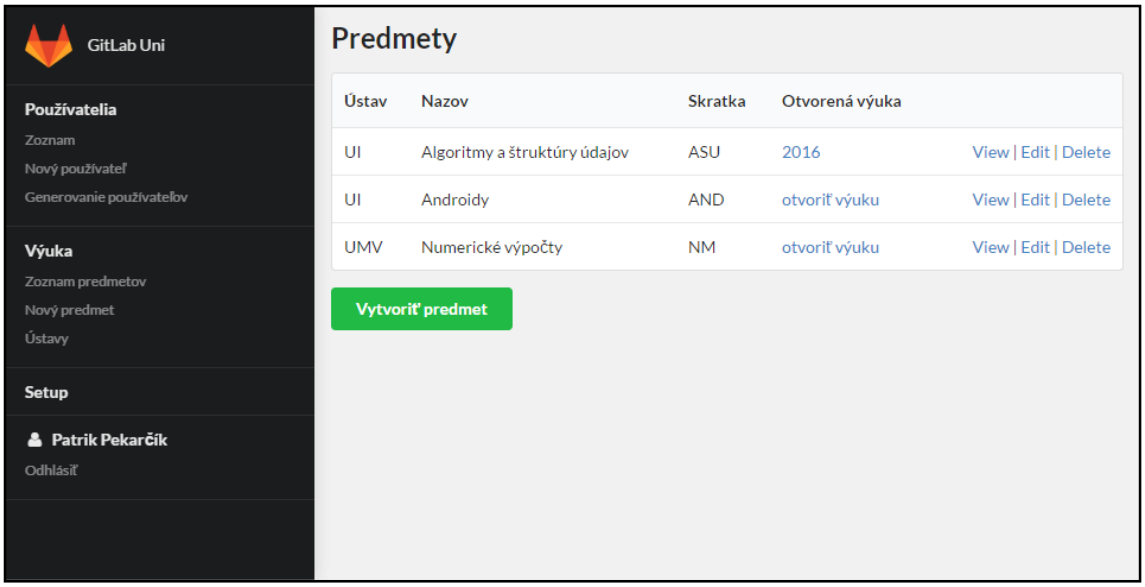
Obrázok 7 Databázový model

Tabuľky *ustavy* a *predmety* uchovávajú informácie o dostupných predmetoch a ústavoch, pod ktoré patria. Z týchto tabuliek bude vytvorený názov skupiny v repozitári zdrojových kódov, a to spojením stĺpcov skratka z oboch tabuliek.

Tabuľka *vyucba* je určená na vytváranie aktuálneho vyučovacieho obdobia, obsahuje stĺpce *rok* a *otvorena*. Prvý stĺpec slúži na rok, v ktorom sa predmet vyučuje a druhý identifikuje, či výuka aktuálne prebieha alebo je už ukončená. S ukončenou výučbou už nebude možné pracovať a jej obsah bude archivovaný pre použitie v ďalšej výučbe. V podradenej tabuľke *vyuky_uzivatelia* je zoznam študentov a vyučujúcich, ktorí sú súčasťou vytvorenej výučby. Rozdelenie, či je to študent alebo vyučujúci, je uvedené v stĺpci *vyucujuci*.

Poslednou časťou schémy sú tabuľky *zadania* a *zadania_stav*, pomocou nich vytvoríme zadania na vypracovanie pre študentov. V stĺpci *data* sa nachádzajú inštrukcie, podľa ktorých bude Manažment sledovať repozitáre študentov, či odovzdali vypracované zadania. Informácie o vypracovaných zadaniach sa uchovávajú v tabuľke *zadania_stav*. V tejto tabuľke je možné okrem automaticky vygenerovaných informácií o stave zadania pridať aj informáciu od vyučujúceho, na čo slúžia stĺpce *hodnotenie* a *hodnotenie_text*.

Pre navrhnutú databázovú štruktúru sme pomocou spomenutého frameworku FuelPHP vytvorili jednoduché používateľské rozhranie. V rozhraní sme využili podobné GUI (graphical user interface) ako vo zvolenom repozitári zdrojových kódov, aby bola práca s Manažmentom intuitívna.



Obrázok 8 Manažment GUI

6.3 Prepojenie s repozitárom zdrojových kódov

Prepojenie sa ako služba vykonáva na pozadí. Periodicky sa synchronizujú nastavenia z našej databázovej štruktúry do repozitára zdrojových kódov. Využívame aplikačné rozhranie API [31] repozitára zdrojových kódov. Synchronizácia postupne prebieha v nasledujúcich krokoch. Na začiatku skontrolujeme existenciu skupín a projektov v repozitári zdrojových kódov. Následne sa vytvoria neexistujúce skupiny a projekty. V poslednej fáze synchronizácie sa nastavia prístupy do vytvorených projektov, každému projektu sa priradia používatelia, ktorí môžu sledovať zmeny v projekte a tiež používatelia, ktorí môžu vykonávať zmeny v projekte.

Po dokončení synchronizácie sa vykoná kontrola zadaní podľa nastavenej konfigurácie. Automatizovaný systém sa postupne pripojí ku každému projektu študentov a skontroluje, či vložili vypracované zadanie na predpísané miesto. Ak tento systém nájde vypracované zadanie, otestuje ďalšie parametre zadania a výsledok zapíše do databázy. Stav zadaní je zobrazený v prehľadnej tabuľke vytvorenej aplikácie Manažment.

Záver

Vývoj softvérových projektov sa v posledných rokoch natoľko zrýchlil, že pôvodné prístupy už nie sú schopné efektívne reagovať. Preto vznikajú nové prístupy a jedným z nich je DevOps. V práci sme sa zamerali na spôsoby zavedenia nového prístupu v univerzitných projektoch. Navrhli sme postup používania repozitára zdrojových kódov s cieľom použitia continuous integration na projektoch.

Prvým cieľom práce bolo analyzovať možnosti DevOps so zameraním na priebežnú integráciu (continuous integration) a priebežné nasadzovanie (continuous delivery). Tomuto cieľu sme sa venovali v prvej kapitole, kde sme vysvetlili fungovanie DevOps a zaradili do tohto postupu prvky continuous integration a continuous delivey. Pomocou analýzy sme zistili, že je potrebné sa venovať repozitáru zdrojových kódov.

V ďalšej kapitole sme sa venovali existujúcim prístupom k vývoju projektov. Porovnali sme prístupy Waterfall, Agile a Lean. DevOps bol inšpirovaný práve prístupom Agile. V závere kapitoly sme na modelovej situácii ukázali nevýhodu prístupu Waterfall oproti ostatným prístupom. Touto kapitolou sme naplnili druhý cieľ práce.

Tretím cieľom bolo analyzovať možnosti využitia DevOps v univerzitnom prostredí, čomu sme sa venovali v tretej kapitole. V úvode kapitoly sme analyzovali projekty, ktoré vznikajú v univerzitnom prostredí. Určili sme tri druhy projektov a navrhli sme spôsob ako v nich použiť DevOps. Zamerali sme sa hlavne na štruktúru repozitára zdrojových kódov.

V štvrtej kapitole sme sa už venovali implementácii nástrojov potrebných pre DevOps. Okrem toho sme uskutočnili porovnanie existujúcich nástrojov podľa zvolených kritérií, implementovali sme existujúce riešenie GitLab. Výhodou tohto riešenia je komplexnosť ponúkaných nástrojov, pretože okrem repozitára zdrojových kódov nám ponúka aj integračný nástroj. Implementáciou sme naplnili aj náš posledný cieľ práce.

Pre uľahčenie používania navrhovaného riešenia sme vytvorili tri ukážkové projekty. Demonštrovali sme na nich automatické zostavovanie a testovanie v rôznych oblastiach. Venovali sme sa Jave, jazyku PHP a generovaniu PDF súborov z LaTeX projektov. V projekte webovej stránky v jazyku PHP sme ukázali aj spôsob automatického nasadenia projektu do produkčného prostredia.

V poslednej kapitole sme vytvorili aplikáciu Manažment. Hlavnou úlohou aplikácie je zjednodušiť vytváranie prostredí pre výukové predmety. Okrem toho sme do tejto

aplikácie vytvorili možnosť tvorby zadaní a automatickej kontroly vypracovania zadaní študentmi. Vyučujúci má tak možnosť jednoduchej kontroly ako študenti pracujú a udelenia hodnotenia vypracovaného zadania.

Zoznam použitej literatúry

[1] Sommerville, I: Software Engineering, Pearson, 2010. ISBN-13: 978-0137035151

[2] Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering), Addison-Wesley Professional, 2015. ISBN-13: 978-0134049847

[3] Duvall, P.M.: Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley Professional, 2007.ISBN-13: 978-0321336385.

[4] Humble, J.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley Signature Series (Fowler)), Addison-Wesley Professional, 2010.ISBN-13: 978-0321601919.

[5] Swartout, P.: Continuous Delivery and DevOps: A Quickstart Guide - Second Edition, Packt Publishing - ebooks Account. 2014.ISBN-13: 978-1784399313

[6] Berg, A.M.: Jenkins Continuous Integration Cookbook - Second Edition, Packt Publishing - ebooks Account. 2015.ISBN-13: 978-1784390082.

[7] Ferguson Smart, J.: Jenkins – The Definitive Guide, 2011. ISBN-13: 978-1449305352

[8] VCS Definition, http://www.yourdictionary.com/version-control, [Naposledy navštívené 23.01.2016]

[9] Benefits of DevOps, http://newrelic.com/devops/benefits-of-devops, [Naposledy navštívené 26.03.2016]

[10] DevOps, http://searchcloudcomputing.techtarget.com/definition/DevOps, [Naposledy navštívené 26.03.2016]

[11] Environments, http://dltj.org/article/software-development-practice/, [Naposledy navštívené 26.03.2015]

[12] Continuous integration, https://www.thoughtworks.com/continuous-integration, [Naposledy navštívené 26.03.2016]

[13] Continuous deployment, http://guide.agilealliance.org/guide/cd.html, [Naposledy navštívené 26.03.2016]

[14] Continuous delivery, https://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff, [Naposledy navštívené 26.03.2016]

[15] Continuous security & Monitoring, http://devops.com/2015/04/15/devops-makes-security-assurance-affordable/, [Naposledy navštívené 26.03.2016]

[16] Waterfall model, http://searchsoftwarequality.techtarget.com/definition/ waterfall-model, [Naposledy navštívené 26.03.2016]

[17] Agile development, https://dzone.com/articles/continuous-delivery-vs, [Naposledy navštívené 26.03.2016]

[18] Mary Poppendieck; Tom Poppendieck (2003). Lean Software Development: An Agile Toolkit. Addison-Wesley Professional. ISBN 978-0-321-15078-3

[19] Lean, Agile & DevOps, http://www.agileweboperations.com/lean-agile-devops-related, [Naposledy navštívené 26.03.2016]

[20] Best version control service, http://www.slant.co/topics/153/~hosted-version-control-services, [Naposledy navštívené 7.5.2016]

[21] CI/CD scheme, https://notafactoryanymore.files.wordpress.com /2014/08/cdvscd1.png, [Naposledy navštívené 7.5.2016]

[22] Waterfall model scheme, http://xbsoftware.com/wp-content/uploads/ 2014/10/software-development-life-cycle.png, [Naposledy navštívené 7.5.2016]

[23] Agile model scheme, https://techmytalk.files.wordpress.com/2013/05/agile-model2.jpg, [Naposledy navštívené 7.5.2016]

[24] Lean model scheme, http://image.slidesharecdn.com/introduction-to-lean-software-development-100/95/introduction-to-lean-software-development-9-728.jpg?cb=1247900638, [Naposledy navštívené 7.5.2016]

[25] Jazyk PHP, http://php.net, [Naposledy navštívené 8.5.2016]

[26] FuelPHP framework, http://fuelphp.com, [Naposledy navštívené 8.5.2016]

[27] OAuth 2.0, http://oauth.net, [Naposledy navštívené 8.5.2016]

[28] Opauth, http://github.com/opauth/opauth, [Naposledy navštívené 8.5.2016]

[29] Implementácia Opauth pre GitLab, http://github.com/ppatrik/opauth-gitlab, [Naposledy navštívené 8.5.2016]

[30] Špecifikácia OAuth 2.0, http://tools.ietf.org/html/rfc6749, [Naposledy navštívené 8.5.2016]

[31] Dokumentácia GitLab API, http://docs.gitlab.com/ce/api/README.html, [Naposledy navštívené 8.5.2016]

[32] Maven, https://phpunit.de/, [Naposledy navštívené 8.5.2016]

[33] PHPUnit, https://phpunit.de/, [Naposledy navštívené 8.5.2016]

[34] Heroku, https://www.heroku.com/, [Naposledy navštívené 8.5.2016]

Príloha A

Obsah CD:

- *./praca.pdf* – Bakalárska práca v elektronickej podobe
- *./demo/projekt-X/** – zdrojové kódy ukážkových projektov
- *./manazment/** – zdrojové kódy Manažment rozhrania