Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	LS 2016/2017
Autoři - jméno, příjmení, xname	Bc. Andrea Havlíčková, Bc. Jan Štěpán, Bc. Lukáš Pajma xhava23, xstej110, xpajl02
Téma	**Teaching Modern Software Development Techniques at University**
Datum odevzdání	14. 5. 2017 23:59

Abstrakt
Semestrální práce se věnuje tématu výuky technik moderního vývoje softwaru na vysokých školách a univerzitách. Obsahem práce jsou metodiky používané při vývoji softwaru aplikované na zahraniční i tuzemský model výuky, která se zabývá vývojem softwaru a následné aplikace

Klíčová slova
Moderní vývoj softwaru, software, vývoj softwaru, výuka, univerzita, výuka moderního vývoje softwaru, techniky moderního vývoje softwaru,

Obsah

Úvod

V dnešní době se mnoho z nás, studentů, střetává na vysokých školách v oblasti výpočetní techniky s pojmem moderní vývoj softwaru. Po absolvování studia na vysoké škole v oblasti výpočetní techniky se většina absolventů vydá právě tímto směrem. Problém však nastává v době, kdy po ukončení studia u absolventů převládají spíše teoretické znalosti nad praktickými zkušenostmi. Příčinou této problematiky, tj. výuky moderního vývoje softwaru na vysokých školách a jejím řešením se bude zabývat tato seminární práce.

Hlavním cílem práce je představit techniky vývoje v oblasti současného vývoje softwaru dostupné ze zahraničního modelu vytvořeného na základě této problematiky a u vybraných technik navrhnout jejich aplikaci v prostředí Vysoké školy ekonomické v Praze.

Autoři na základě článku od profesora Roberta Chatleyho na škole Imperial College v Londýně se nejdříve budou věnovat metodikám, které jsou během vyučování moderního vývoje softwaru na škole využívány, dále představí model, který na univerzitě využívají k výuce moderního vývoje softwaru a který má napomoci studentům již během studia získat znalosti, které mohou využívat v praxi, právě tím, že z reálných případů vycházejí. Následně se práce bude zabývat představením kurzů ze zmiňovaného oboru, které jsou nabízeny na univerzitě, na které autoři studují. V závěru práce bude pro kurzy týkající se moderního vývoje softwaru, které autoři absolvovali během bakalářského a magisterského studia jako povinné sestrojen návrh na jejich optimalizaci.

Kapitoly

1 Porovnání metodik moderního vývoje softwaru

Metodiky vývoje softwaru neboli metodologie vývoje softwaru v softwarovém inženýrství, je souhrn postupů, pravidel či nástrojů. Metodikou se také rozumí aplikační rámec, tzv. framework, který se používá ke strukturování, plánování a řízení procesu vývoje informačního systému.
Metodik pro vývoj procesů je mnoho a nejsou jednotně popsány. Proto je obtížné je vyhledávat, vzájemně porovnávat a zvolit nejvíce vhodnou metodiku. Mezi další nevýhody patří dostupnost několika metodik pouze v anglickém jazyce, či problém rigorózních metodik nevyhovující požadavkům současných projektů.
Pro lepší pochopení a orientaci velkého množství různých metodik se metodiky řadí alespoň do kategorií podle:

- zaměření - globální a projektové metodiky
- rozsahu - fáze - role - dimenze
- váhy - velikost metodiky x hustota metodiky
- typu řešení - vývoj nového řešení, úprava stávajícího řešení, integrace řešení, implementace typového řešení, atd.

- domény - ERP, CRM, SCM, atd.
- přístupu řešení - strukturovaný, objektové orientovaný, RAD

V této práci se budeme zabývat metodikami, které spadají do kategorie podle členění váhy metodiky, tj. agilní a rigorózní metodiky

1.1 Rigorózní metodiky

Rigorózní metodiky řadíme podle kritéria podrobností metodik či váhy metodik do kategorie obtížných metodik. Rigorózní metodiky jsou těžké metodiky, které jsou vysoce podrobné, obsahují hodně formalit a mají direktviní řízení. Předpokládají možnosti nadefinovat si všechny požadavky na řešení předem a také zaručují opakovatelnost procesů. Příkladem je například metodiky Rational Unified Process (RUP).
Rigorózní metodiky však pro mnohé dnešní projekty nevyhovují. Jsou velmi náročné, čímž způsobují velké množství meziproduktů a hrozí ztráta cíle vývoje softwaru. Dalším problémem rigorózních metodik je tzv. standardizace lidí, kdy narozdíl od agilních metodik nevyužívají individualit lidí, ani nesdílejí znalosti řešení v týmu, to znamená, že čím více větší projekt, tím více specialistů je potřeba najmout.
Pro představu fungování rigorózních metodik je příklad, jenž je postaven na nedůvěře zákazníků a vývojářů. Nedůvěra ve vykonání správné práce směřuje k neustálé kontrole a sledování.

1.1.1 Metodika Rational Unified Process (RUP)

Rational Unified Process, dále jen RUP, je softwarový inženýrský proces, který představuje disciplinovaný přístup přiřazující úkoly a odpovědnosti v organizaci zabývající se vývojem softwaru. Metodika RUP byla původně vytvořena společností Rational Software Corporation, avšak jako samostatná divize náleží od roku 2003 firmě IBM.
RUP se snaží zachytit mnoho moderních postupů moderního vývoje softwaru, najít nejlepší praktiky vhodné pro širokou škálu projektů a organizací.

RUP obsahuje celkem čtyři základní fáze:

- Zahájení (Inception)
- Příprava (Elaboration)
- Konstrukce (Construction)
- Předávání (Transition)

Každá fáze obsahuje několik dalších iterací. Před započetím nové iterace musí být splněna dříve definovaná kritéria předchozí iterace. Fáze zahájení (inception) definuje účel, rozsah projektu a jeho obchodní kontext. Ve fázi projektování (elaboration) je potřeba analyzovat požadavky zákazníka, celého projektu a definovat základy architektury. Realizační fáze (construction) je nejdelší probíhá zde tvorba zdrojových kódů. V poslední fázi předání (transition) může být projekt předán zákazníkovi nebo do dalšího cyklu. (Buchalcevová, 2009)

Nejlepší praktiky a postupy při vývoji softwaru:

- Iterativní vývoj
- Aktivní správa požadavků

- Architektura založená na komponentách
- Vizuální modelování
- Ověřování kvality software
- Řízení změn software

1.2 Agilní metodiky

Agilní metodiky podle kritéria váhy metodiky spadají na rozdíl od rigorózních metodik do kategorie lehké metodiky. U agilních metodik jsou lidé prvořadým faktorem, je zde kladen důraz na týmovou práci a jejich individuální schopnosti. Agilní metodiky se zaměřují na iterativní vývoj s velmi krátkými iteracemi a také na fungující software, který je hodnotný pro zákazníka. Komunikace se zákazníkem je důležitým faktorem. Proto jsou agilní metodiky tolerantní ke změnám, tj. vytvářejí software, který je ochoten v budoucnosti akceptovat a reagovat na změny, které si zákazník vyžádá. Dalším kladem je agilních metodik je automatizované testování. Příkladem agilních metodik jsou metodiky SCRUM, Extreme Programming a Kanban.

1.2.1 Metodika Scrum

Metodika Scrum patří mezi nejpoužívanější agilní metodiky. Zaměřuje se hlavně na řízení projektu, kde hrozí riziko, kdy zákazníci mění během projektu názor na svůj původní cíl, tj. mění názor o tom, co chtějí a potřebují. Cílem metodiky Scrum je tedy rychle reagovat na nové požadavky zákazníků. Metodika Scrum má vždy určený tým, který se skládá ze tří rolí: role Product Owner (spravuje seznam požadavků), vývojový tým (skupina lidí s různou specializací, kteří mají za cíl včas dodat správně fungující software, či splnit jim určený požadavek) a Scrum Master (zodpovídá za metodiku, její správnou implementaci a maximalizaci užitku). Důležitou součástí Scrumu jsou tzv. Eventy neboli události. Hlavní událostí Scrumu je Sprint. Sprint neboli iterace je základní jednotka vývoje ve Scrumu. Sprint je vždy omezen na specifickou časovou dobu, obvykle jeden týden až měsíc. Doba Sprintu je vždy určena dopředu. Hlavním cílem sprintu je definovat jeho úkoly, tj, kde je přesně definována práce sprintu a jeho závazky. Každý Sprint končí tzv. Sprint Review, tj. recenzí či zpětnou vazbou o jeho průběhu. (Buchalcevová, 2009)

1.2.2 Metodika Extreme Programming (XP)

Metodika extrémního programování vznikla počátkem 90. let minulého století ve skupině Kenta Becka, Warda Cunninghama a Rona Jeffriese. Metodika extrémního programování je vhodná pro malé až střední týmy, kde je zhruba od dvou do desíti programátorů. Vhodná je také pro rychle se měnící požadavky či nejasné požadavky zákazníků na vývoj softwaru. Díky této metodice je software v budoucnosti schopen akceptovat změny požadavků a dodává tak softwaru vysokou kvalitu. Mezi principy extrémního programování patří například neustálé testování (testování před změnou kódu, po změně kódu a v průběhu vývoje). Pokud jsou osvědčeny krátké iterace, pak v extrémním programování jsou iterace zkráceny na minuty, hodiny či dny místo týdnů, měsíců či roků. (Buchalcevová, 2009)
Základní principy extrémního programování:

- Komunikace

- Jednoduchost
- Zpětná vazba
- Odvaha

Pravidla postupu vývoje extrémního programování:

- Zadání
- Plánování
- Design
- Programování
- Testování
- Dodávka a akceptace

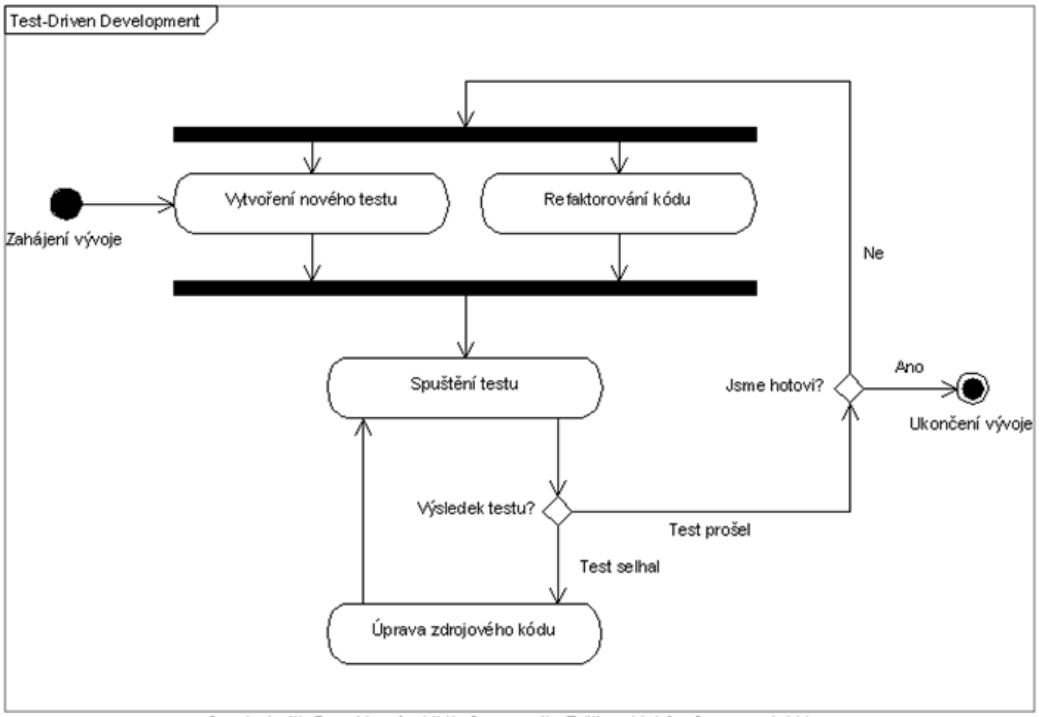
1.2.3 Kanban

Kanban nelze přímo považovat za metodiku, jedná se spíše o nástroj pro zefektivnění procesu a je úzce spjat s Lean Managementem.
Kanban narozdíl od metodiky Scrum neobsahuje tolik pravidel a všechno si víceméně můžeme zvolit sami. U Kanbanu však je potřeba dodržovat aspon tři základní principy, kterými jsou work in progress (omezení rozpracované práce), lead time (dodací lhůta) a progress visualization (vizuallizace progresu). Aplikováno na vývoji software, jedná se o vytvoření přehledné tabule, kde si označíme tři oblasti - backlog (nevyřízené), in progress (v progresu) a done (vyřízené). Práci omezíme, tj. omezíme oblast na počet lístečků s úkoly. Pokud už jsme omezeni přidáním lístečků s úkoly, nezbývá nám nic jiného než dané úkoly v dané oblasti vyřešit či posunout dál. Nástroj Kanban však dokáže být i více propracován, avšak nejlepší je jej využívat spolu s metodikou Extremního programování nebo metodou Scrum. (Bruckner, Buchalcevová, 2012)

1.2.4 Test Driven Development (Refactoring)

Metodika Test Driven Development byla od roku 1999 součástí agilní metodiky Extrémního programování, avšak v roce 2003 se od tradičních metodik TDD metodika odtrhla a je přisuzována autorovi Kentu Beckovi.
Metodika Test Driven Development neboli Programování řízené testy, je přístup k vývoji softwaru, který je založen na malých, ale stále se opakujících krocích, jenž vedou ke zefektivnění celého vývoje.
TDD na rozdíl od tradičních metodik využívá přístupu Test-First, který vytváří testy dříve než existuje dostatek kódu, který by testy prošel. Tradiční metodiky využívají přístupu Test-Last, kdy nejprve předpokládají existenci kódu, a teprve poté kód testují a následně kontrolují. TDD však přístup Test-First nevyužívá pro sebekontrolu, ale využívá ho jako nástroj pro návrh systému.
Metodika TDD vyžaduje psaní automatických unit testů. Unit testy jsou testy, které testují nejmenší jednotku programu. kterou může být třída v objektově orientovaném programování, funkce nebo metoda. Pro svoji složitost však bývají spíše náplní vývojářů.

Test Driven Development má svůj vývojový cyklus, který obsahuje tyto fáze (viz obrázek):



- **Napsat test** - Testy mohou být napsány podle use-case diagramů, user-stories, či jiných materiálů. Napsání testu slouží k ujištění, že vývojář pomocí ověřování kódu chápe správně funkcionalitu a požadavky na testovanou komponentu, čímž zabrání a eliminuje odchýlení se od původního cíle
- **Spustit testy a kontrola, že neprojdou** - Všechny testy v testu nesmí projít. Tato část cyklu slouží k sebekontrole, kdy si vývojář ověří, že neexistuje žádná nová funkcionalita dříve než implementace. Projití testů pak znamená, že jsou špatně napsané.
- **Napsat vlastní kód** - Cílem napsání vlastního kódu je úspěšné splnění testů
- **Kontrola kódu** - Kontroluje se, zda kód plní definované požadavky
- **Refaktorace** - Refaktorace je posledním krokem vývojového cyklu TDD. Zde je předmětem elegance a efektivnost napsaného kódu, tj. odstraňují se duplicity a kód se co nejefektivněji upravuje.
- **Opakování** - opakování vývojového cyklu slouží k ověření funkcionality testů

Přínosem TDD je možnost opětovného spouštění velkého množství automatických testů, které kontrolou a neustálým ověřováním předem definovaných požadavků a funkcionalit, odhalují možné chyby. Omezením TDD je výskyt chyb v testech nebo kódu, které naléhají na jejich opravu. Při velkém výskytu chyb však může docházet k demotivaci a neschopnosti chyby napravit. Dalším omezením TDD je také špatná testovatelnost určitých částí programů, jako například uživatelského rozhraní. Nelze zde zapisovat akceptační protokoly, které jsou nutné pro vypuštění programu. (Buchalcevová, 2009; Bruckner, 2012)

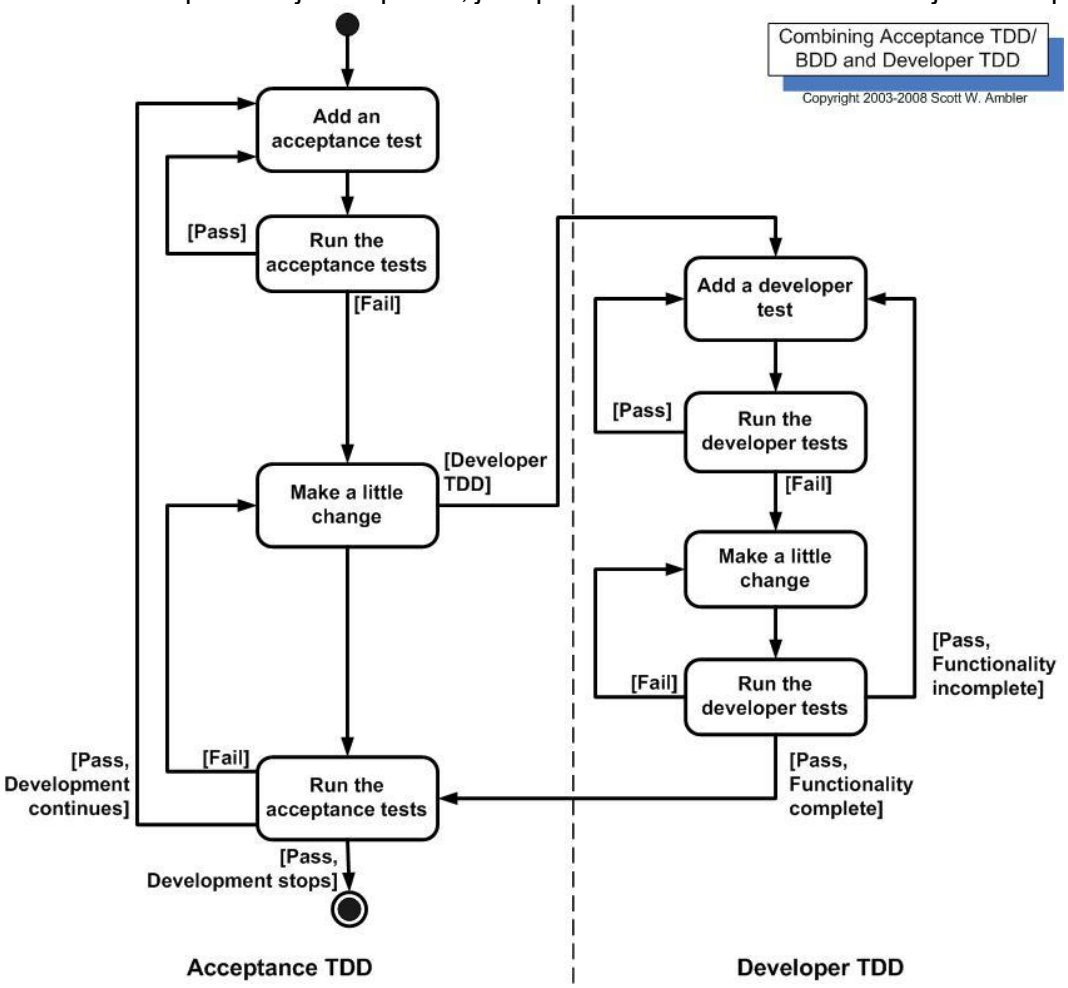
TDD vs ATDD

Metodika Test Driven Development obsahuje dvě úrovně, kterými jsou: Developer Test Driven Development a Acceptance Test Driven Development.

Acceptance Test Driven Development ATDD
Acceptance Driven Development neboli akceptační programování řízené testy je úroveň metodiky TDD nebo také nástroj, sloužící ke komunikaci mezi zákazníkem, vývojářem a testerem a zajišťuje správné a podrobné nadefinování požadavků.Z testů ATDD jsou často odvozovány TDD testy. Rozdílem testů ATDD a TDD je v tom, že testy ATDD musí být čitelné zákazníkem, narozdíl od testů TDD, které naopak být nemusí.

Developer Test Driven Development TDD
Developer Driven Development neboli z názvu vyplývající vývojářské programování řízené testy je zaměřeno na vývojářský cyklus.

Obrázek níže poukazuje na způsob, jak spolu ATDD a TDD do sebe vzájemně zapadají.



1.3 Rigorózní metodiky vs Agilní metodiky

Rigorózní a agilní metodiky jsou od sebe velice odlišné, například v samotném chápání procesu vývoje softwaru. Předpokladem rigorózních metodik jsou procesy softwaru, které lze popsat a požadavky, které je možné definovat předem, zatímco u agilních metodik je možné předem definovat pouze hrubé požadavky a softwarové procesy popsat nelze. Obsahem rigorózních metodik jsou přesně definované procesy, činnosti a artefakty, narozdíl od agilních metodik, kde obsahem jsou jen generativní pravidla, praktiky a principy. Rigorózní metodiky se používají pro standardní projekty a velké projekty. Agilní metodiky se používají především pro výzkumné projekty, time-to-market a menší týmy. (Buchalcevová, 2009)

2 Zahraniční model výuky moderního vývoje softwaru

Na zahraniční škole Imperial College v Londýně je moderní vývoj softwaru vyučován ve čtyřech ročnících. Nový model výuky této oblasti zaváděli právě z důvodu nedostatečné možnosti integrace vystudovaných programátorů v praxi. Jedná se o tříletý nebo čtyřletý program s názvem Computing, kde studenti mají možnost získat dva druhy vysokoškolského titulu (bakalářský nebo magisterský). Oba programy se prolínají během studentských let, program na čtyři roky je však obohacen o půlroční pracovní stáž ve vybrané softwarové společnosti. Studenti vedle dalších předmětů jako matematiky, logiky nebo operačních systémů se věnují také softwarovému inženýrství, které na škole mají po celé čtyři roky a je obsahem různých předmětů(Chatley, 2017)

2.1 První ročník

V prvním ročníku si studenti osvojují základní dovednosti programování v oblasti funkčního objektově orientovaného a systémového programování. Výuka probíhá formou přednášek a následně také integrovanými počítačovými laboratorními cvičeními. Drobný problém může vznikat v odlišné úrovni zkušeností studentů v oblasti programování. Někteří na školu přicházeli s nulovými zkušenostmi, jiní již měli zkušenosti z oblasti programování ze střední školy nebo z jiných soukromých projektů, popř. se naučili programovat doma sami ve svém volném čase. Cílem je obě skupiny studentů motivovat a zároveň ani jednu z nich neznevýhoďnovat, proto se v úvodních kurzech prvního ročníku na Imperial College vyučuje jazyk Haskell, který je povětšinou neznámý pro celou skupinu studentů. (Chatley, 2017)
V prvním ročníku jsou vyučovány následující kurzy související s moderním vývojem softwaru:

- Programování I - vyučuje se v jazyce Haskell, kurz je zaměřený na objektově orientované programování a procedury a syntaxi a sémantiku zvoleného jazyka
- Programování II - vyučuje se v jazyce Java, kurz se zabývá kontrolami, datovými strukturami,jednoduchými konstrukcemi algoritmů, testováním a odladěním kódu, také abstraktními datovými typy
- Laboratorní cvičení - praktická cvičení zaměřená na vývoj softwaru v jazycích Haskell, Java, Assembler a C. Výstupem je umět testovat a implementovat algoritmy programů s využitím TDD.

- Architektura - předmět zaměřený na výuku prvků potřebných pro pochopení architektury počítačových systémů, jako např. kódování, normalizace, IEEE standardy, registry, komponenty CPU, programovancí modely atd.
- Databáze 1 - předmět zaměřený na základy modelování v jazyce SQL spojených s tvorbou relačních databází (Computing, 2017)

2.2 Druhý ročník

Druhý rok je zaměřený na návrh a rozvoj větších systémů. Pokračuje se ve výuce programování v běžných jazycích a dále se více rozvíjí dovednosti z oblasti grafiky. Jelikož každá oblast programování softwaru se zaměřuje trochu na jiné postupy vývoje softwaru a jsou při vývoji využívány jak rigorozní tak agilní metodiky, je potřeba také studenty naučit na tyto fakta reagovat. Návrhové vzory se využívají např. u vývoje bezpečnostních systémů, naopak u spotřebitelských webových služeb se během vývoje softwaru volí agilnější přístup s možností pružně reagovat na změny. Studenti pracují na zadaných projektech, během kterých jsou vychováváni k stálé komunikaci a diskuzi na začátku projektu i během celého procesu vývoje softwaru. Studenti jsou vedeni k osvojování si návrhových vzorů a architektonických stylů, ale s každým problémem, který v daném projektu vznikne, dochází také k jeho induviduálnímu řešení. Např. vznikne-li problém v oblasti duplicit, jsou společně se studenty navrhovány možné kompromisy pro jeho řešení. Pro grafickou část projektu je využíváno moderní vývojové prostředí a studenti si tím lépe dokáží pracovat na grafických návrzích, protože se učí kinesteticky a vytvářejí praktická řešení. (Chatley, 2017)

Ve druhém ročníku jsou vyučovány následující kurzy související s moderním vývojem softwaru:

- Softwarové inženýrství (návrhy) - předmět je zaměřený na vývoj softwaru s využitím TDD, refaktoringu, mock objektů a návrhových vzorů
- Kompilátory - vyučuje se, jak kompilátory fungují, jak je návrh programovacích jazyků jimi ovlivněn, jak je ovlivněna tvorba architektury
- Úvod do C++ - předmět zaměřený na základy vývoje software v jazyce C++
- Laboratorní cvičení II - praktická cvičení zaměřená na vývoj softwaru v jazycích C, Java, Prolog, Maple, LTSA
- Softwarové inženýrství (algoritmy) - kurz zaměřený na algoritmické mylšení a řešení problémů vznikajících při vývoji softwaru s využitím Komplexní analýzy, “Conquer and divide”, Dynamického programování a Náhodných algoritmů
- Úvod do Prologu - předmět zaměřený na základy vývoje software v jazyce Prolog (Computing, 2017)

2.3 Třetí ročník

Třetí ročník začíná velkým významným projektem trvajícím po dobu tří měsíců, kde studenti pracují v týmech pěti až šesti členů a zároveň s projektem probíhá také blok přednášek, které jsou studenti povinni absolvovat. Každá skupina řeší jiný typ projektu se zaměřením na vývoj softwaru, který se týká konkrétního problému nebo poskytnutí služby zákazníkovi. Každá skupina má přiděleného klienta, buď je jím člen fakulty nebo partner z praxe, který řídí výrobu softwaru. Cílem této práce je naučit studenty pracovat v týmu a osvojit si proces vývoje softwaru, který takovým způsobem probíhá také v praxi. Dále studenti navštěvují kurz Softwarové inženýrské praxe, kde jsou jim představovány a mohou si zde osvojovat vývojové metody, nástroje, zajištění kvality, techniky

projektového a produktového řízení. Na Imperial College začali více klást důraz na učení se využití nástrojů a postupů práce, které by mohli dát společně využít jak z hlediska technického kódu, tak z hlediska správy softwaru a zároveň byly využitelné také v obecném projektovém řízení. Tato hlediska jsou poté aplikována ve zmiňovaném skupinovém projektu. (Chatley, 2017)

Ve třetím ročníku jsou vyučovány následující kurzy související s moderním vývojem softwaru:

- Softwarové inženýrství (praxe) - kurz je zaměřený na výuku agilních metodik - extrémní programování, scrum, kanban a další techniky např. vývojové procesy a zajištění kvality, průběžná integrace a kvalitativní a kvantitativní evaluace
- Informace a teorie kódování - předmět zaměřený na detekci chyb v kódu, lineární a cyklické kódy a reprezentaci informací
- Verifikace systémů- předmět zaměřený na verifikaci modelů návrhů systému v nástrojích NuSMV model checking toolkit a NCMAS model chcecker
- Databáze pro pokročilé - kurz rozvíjející znalosti z oblasti databází, konkrétně data minigu, OLAP operátorů, Big data a znalosti v oblasti kódování v jazyce SQL (Computing, 217)

2.4 Čtvrtý ročník

Ve čtvrtém ročníku probíhá další kurz nazvaný Softwarové inženýrství pro průmysl. Cílem je ukázat studentům problémy, které vznikají v oblasti průmyslového softwarového inženýringu a řeší je specialisté v praxi. Dále je v kurzu vyučováno, jak efektivně pracovat se starším kódem a jak ho dále využít. Většina softwarových inženýrů totiž v praxi častěji přetváří starý kód a neprogramují nové řešení. Je to běžný postup úspěšně navržené systémy totiž potřebují průběžně aktualizovat, takže je žádoucí, aby programátor byl zvyklý na úpravu kódu. Nejvíce je ve čtvrtém ročníku kladen důraz na rozvoj kritického myšlení studentů rozvoj argumentace. Náplní kurzu je také zadané téma zkoumat na odborných blozích, článcích, příspěvcích a např. veřejně přístupných videokonferencích. Jde o to aby studenti dokázali vyhledávat názory odborníků z praxe. Studenti dále píší krátká týdenní stanoviska na zadané diskuzní téma, které zkoumali v dostupných zdrojích. Do tzv. diskuzní třídy jsou dále vyzýváni odborníci s prosbou o vyjádření k diskuznímu tématu na daný týden. Odborníci do diskuze vnáší zkušenosti, příklady a případové studie, které studenty mají podporovat vnímat více reálné případy než jen strohou teorii. (Chatley, 2017)

Ve čtvrtém ročníku jsou vyučovány následující kurzy související s moderním vývojem softwaru:
Prezentace a reporting v průmyslu - předmět venující se komunikačním dovednostem v oblasti průmyslového softwaru
Software reliability - kurz zaměřený na spolehlivost softwaru, na její verifikaci pomocí automatických nástrojů. Umožní náhled na algoritmy SAT a SMT, vyučován v jazyce C
Individuální projekt (MEng) - individuální projekt studenta, s možností seberealizace a s využitím technik naučených během čtyř let na univerzitě (Computing, 2017)

3 Moderní vývoj softwaru na VŠE v Praze

Na Vysoké škole ekonomické v Praze je studentům nabízeno hned několik kurzů zabývajícími se moderním vývojem softwaru. Studenti bakalářského oboru Aplikovaná informatika,si mohou v rámci skupiny povinných předmětů vybrat tyto semestrální kurzy:

- Programování v Javě - zaměřený na programovací jazyk Java
- Programování ve Visual Basic - zaměřený na programovací jazyk Visual Basic
- Softwarové inženýrství - zaměřený na programovací jazyk Java

V rámci oborově volitelných předmětů si mohou vybrat kurzy:

- Webové aplikace - zaměřený na jazyk PHP.
- Klient/ server aplikace v Javě - zaměřený na rozšiřování znalostí v jazyce Java
- Moderní programovací techniky - zaměřený na rozšíření znalostí v jazyce Java, např. serializace, parametrické datové typy, regulární výrazy a návrhové vzory
- Základy testování SW aplikací

V magisterském navazujícím programu na Fakultě informatiky a statistiky jsou studentům na některých oborech nabízeny tyto předměty zabývající se moderním vývojem softwaru:

- Agilní vývoj webových aplikací - zaměřený na návrh a vývoj moderních webových aplikacích v jazycích PHP a Javascript

Pro porovnání výuky moderního vývoje softwaru použijeme kurzy, jež autoři úspěšně absolvovali v rámci bakalářského studia jako povinné, a které tedy osobně znají, konkrétně se jedná o kurz Programování v Javě (dříve Základy programování) a Softwarové inženýrství. (java.vse.cz, 2017; VŠE, 2017)

3.1 Představení kurzů

Oba semestrální kurzy jsou rozděleny na dva devadesátiminutové úseky a na část přednášek, kde se studenti seznamují s teoretickou částí základů moderního vývoje softwaru a na část praktických cvičení, kde studenti zkouší programovat jednoduché úlohy v jazyce Java. Kurzy probíhají po dobu třinácti týdnů a jsou zaměřeny na osvojení si objektově orientovaného programování.

3.2 Kurz Programování v Javě

Cílem předmětu je předat studentům základy programování v jazyce Java konkrétně jde o osvojení si základních konstrukcí metod, datových struktur (List, Map, Set, Pole,

Map), jednoduchých datových typů, dědičnosti, výjimek, tříd, rozhraní, polymorfismu, jednotkových testů, lambda výrazů.

V předmětu jsou studenti povinni odevzdat výstupy v podobě naprogramovaného semestrálního úkolu s předem daným zadáním nebo s tím, které si sami vymyslí a je schváleno cvičícím. V průběhu cvičení ještě musí splnit několik domácích úkolů týkající se programování jednoduchých softwarů.

3.3 Kurz Softwarové inženýrství

Cílem předmětu je představit studentům základy z oblasti softwarového inženýrství a zaměřit se na návrh a implementaci nového softwarového řešení.

Během semestru studenti odevzdávají výstupy v podobě naprogramovaného grafického uživatelského rozhraní, následně v týmech zpracovávají designový model pro tuto semestrální úlohu a poté v týmu vytváří samotný kód, kde využívají úložiště Subversion, které umožňuje spravovat jednotlivé verze naprogramovaného kódu. V průběhu semestru dále odevzdává student menší projekty v podobě domácích úkolů a jednoduchého naprogramovaného softwaru.

4 Aplikace pro předměty na VŠE v Praze

Na základě seznámení se s přístupem výuky vývoje softwarových aplikací na Imperial College v Londýně jsme vybrali následující doporučení, která by mohla současnou podobu kurzů programování a přístup k výuce těchto kurzů ještě více ozvláštnit. Do sady těchto doporučení jsme zvážili i současnou situaci poptávky po IT specialistech na českém trhu a další možné formy vzdělávání vysokoškolských studentů. I z toho důvodu mohou působit následujícící doporučení více komplexněji, ale je důležité mít na paměti, že tato doporučení nemají za cíl měnit stávající přístup k výuce, ale zejména doplnit výuku o další prvky, které mohou samotnou výuku daných kurzů a také studenty, obohatit.

4.1 Zvané přednášky externistů z praxe

Mimo nutných teoretických znalostí, které studenti často postrádají z předchozího středoškolského vzdělání, je dle našeho názoru od začátku nutné provázat tyto teoretické poznatky s poznatky z praxe. Stejně tak jako v jiných předmětech i v programování je velkou výhodou, když si student může danou problematiku aplikovat na reálný příklad. Ideální by byla možnost vyzkoušet si dané modelové situace přímo na cvičeních na zadaném úkolu.

4.2 Rozšířená výuka programovacích jazyků využitelných při implementaci řešení ERP a BI

Velikou příležitost vidíme také v zavedení rozšířené výuky programovacích jazyků využitelných při implementaci řešení ERP a BI. Zejména při současném nedostatku odborníků v tomto odvětví. Studenti oboru informatiky mají dle našeho názoru výhodu ve znalosti předmětů z ekonomické oblasti - jako je například účetnictví, řízení podniku, management, atd. Mají tudíž velmi dobrý základ pro pochopení business logiky a provázání této logiky s technologickou částí zmíněných technologických řešení, které podporují cíle, řízení a fungování podniku.

4.3 Využití metodik v praxi a organizované řízení týmů

Z našeho zpětného pohledu po absolvování mnoha kurzů nejen z oblasti programování v rámci bakalářského i magisterského studia na VŠE vidíme jako klíčové seznámení studentů s prací v týmu už od prvních týmových projektů na bakalářském studiu. Zejména v kurzu Softwarového inženýrství se nám zdá vhodné rozdělit role před začátkem projektu. Je podle nás důležité, aby byl cíleně stanoven vedoucí týmu, který zodpovídá za odevzdané výstupy,prezentaci výsledného produktu a ohodnocení ostatních členů týmu. Další členové týmu by obsadili například pozici vývojáře, testera, atp. Týmy by mohli mimo výsledné aplikace rovněž odevzdat průběžné zprávy o stavu jejich projektů a naučit se základy řízení projektu a týmu dle některých výše zmíněných metodik. Hlavním nedostatkem tohoto návrhu vidíme možný nedostatek času na zapojení těchto aktivit do standardního učebního plánu.

4.4 Založení klubu programátorů

Po vzoru založení různých klubů a spolků, které působí na VŠE, jako například Klub investorů nebo Klub mladých manažerů, vidíme příležitost pro založení Klubu programátorů. Tento klub by vystupoval jako komunita lidí, kteří se o programování zajímají více i ve volném čase, chtějí sdílet svoje nápady, řešit společně s ostatními problémy z oblasti vývoje aplikací a účastnit se zvaných přednášek se zajímavými hosty. Tyto přednášky by mohly být zaměřeny nejen na programování jako takové, ale také na práci v týmu programátorů, na oblast moderních trendů v oblasti programování a to jak formou přednášek, tak například možnosti navštívení firem, které působí na trhu IT. V klubu programátorů by bylo možné pořádat tzv. dev meetupy zaměřené přímo na praktické ukázky programování a rozvíjení nových technik.

4.5 Založení dev fóra

Stejně jako na Imperial College by bylo dle našeho názoru zajímavé založení dev fóra, kde by bylo možné přímé komunikace s externisty a odborníky z praxe, kteří by mohli studentům zodpovědět jejich dotazy a případně pomoci s jejich problémy při řešení úloh v rámci kurzů vývoje SW aplikací.

4.6 Grow shopping

Nejen studenti, kteří se chtějí vzdělávat, ale i studenti, kteří chtějí vzdělávat ostatní, by měli dostat prostor pro seberealizaci. K tomu by sloužil tzv. grow shopping. Jeho podstatou je sdílení svých zkušeností z teorie a praxe s ostatními. Studenti by tedy měli vedle možností vzdělávání se na odborných kurzech se senior specialisty, možnost navyšování znalostí ve svém volném čase se svými kamarády či spolužáky. Myslíme si, že Vysoká škola ekonomická má dostatek prostor na to, aby mohla v tomto studentům vyjít vstříc. Grow shopping vidíme jako vhodný nejen pro studenty, kteří se chtějí něco naučit, ale zároveň pro studenty, kteří učí ostatní. Díky tomu získají více sebejistoty při komunikaci s ostatními, rozvinou schopnost přednášení, což vidíme jako klíčové například pro prezentaci řešení zákazníkovi. Zároveň se také zdokonalují v oblasti zjišťování požadavků, protože každý student přichází s jinou znalostí a zkušenostmi z programování a přednášející student by byl nucen zjistit co vše je potřeba ve své ukázce odprezentovat jak teoreticky, tak prakticky.

4.7 Přednášky pro rozvoj komunikace, vystupování, vyjednávání

V rámci navazujícího magisterského programu vidíme jako vhodné seznámit studenty mimo jiné s problematikou řízení projektu také s oblastmi komunikace, správného vystupování a jednání se zákazníkem a případné vyjednávání. Pro budoucí konzultanty, ať už v oblasti vývoje individuálních či typových řešení SW aplikací, informačních systémů či řešení business intelligence, vidíme jako důležité vzdělání v oblasti identifikace a zaznamenání klientských požadavků. Pro budoucí vedoucí projektů je zase důležité mít základy vyjednávání a mít tak možnost vyjednat si výhodné podmínky pro svůj tým se zachováním maximální možné míry spokojenosti zákazníka. Vzdělání této oblasti by mohlo být omezené na zvané přednášky odborníků z praxe, kterých působí na naší Vysoké škole celá řada a vedou úspěšné kurzy na vedlejších specializacích.

Závěr

Cílem práce bylo představit techniky moderního vývoje softwaru vyučované na zahraniční univerzitě a vybrané techniky z modelu aplikovat na Vysoké škole ekonomické v Praze.
Nejdříve se práce věnovala metodikám využívaných při výuce na představované univerzitě v zahraničí, dále se zaměřila na samotný model výuky a kurzy spadající do kategorie moderního vývoje softwaru a v závěrečných kapitolách představila model výuky na Vysoké škole ekonomické v Praze a navrhla doporučení, které by výuku na základě informací ze zahraničního modelu mohly zefektivnit. Jelikož autoři neznají všechny procesy, které se váží, ke změnám v učebním plánu na českých státních univerzitách, mohou být stanovená doporučení limitována z hlediska zmiňovaných faktorů.

Literatura

Soupis použité literatury. Použijte Harvardský systém (forma jméno,datum) jako metodu citování a odkazování při dodržení aktuální normy ISO 690.

Aplikovaná informatika. Vysoká škola ekonomická v Praze [online]. .: ., . [cit. 2017-05-07]. Dostupné z: http://fis.vse.cz/wp-content/uploads/2014/10/aplikovana_informatika_bc_2016_www.pdf

CHATLEY, Robert. Teaching Modern Software Development Techniques at University. Infoq [online]. , 1 [cit. 2017-03-12]. Dostupné z: https://www.infoq.com/articles/teaching-software-development-university?utm_source=infoqWeeklyNewsletter&utm_medium=WeeklyNL_EditorialContent_culture-methods&utm_campaign=10182016news

Computing (MEng) [online]. London: Imperial College, 2017 [cit. 2017-05-14]. Dostupné z: http://www.imperial.ac.uk/computing/prospective-students/courses/ug/beng-meng-computing/meng-comp/

POLLICE, Gary. Teaching software development vs. software engineering [online]. [cit. 2017-03-12]. Dostupné z: https://www.ibm.com/developerworks/rational/library/dec05/pollice/

Software development methodologies. ITINFO [online]. -: -, 2016 [cit. 2017-03-12]. Dostupné z: http://www.itinfo.am/eng/software-development-methodologies/#chapter2

Výuka programování a softwarového inženýrství na KIT VŠE. *Java.vse.cz* [online]. -: -, 2017 [cit. 2017-03-12]. Dostupné z: https://java.vse.cz/Main/HomePage

BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů*. Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3.

BRUCKNER, Tomáš, BUCHALCEVOVÁ, Alena. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.