Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Analýza současných trendů ve výuce programování

BAKALÁŘSKÁ PRÁCE

Student : Lukáš Hrách

Vedoucí : Ing. Rudolf Pecinovský, CSc.

Oponent : Ing. Jarmila Pavlíčková, Ph.D.

2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 2. května 2017 . . . . . .

Lukáš Hrách

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Rudolfovi Pecinovskému za jeho cenné rady, připomínky a užitečné konzultace. Dále bych rád poděkoval Ing. Olze Kaiferové, která mi poskytla několik doplňujících informací důležitých pro tuto bakalářskou práci.

Abstrakt

Předmětem této bakalářské práce je zmapování trendů v programování a jejich využití ve výuce. Tohoto cíle je dosaženo analýzou trendů vývoje programování jak v přítomnosti, tak s výhledem do blízké budoucnosti, následnými rozhovory s odborníky zodpovědnými za výuku programování na českých školách. Další částí práce je rozbor metodik výuky programování s bližším zaměřením na metodiku Architecture First, jejíž tvůrcem je vedoucí této práce pan inženýr Pecinovský.

Na základě provedených průzkumů byla zjištěná data okomentována, popsány různé problémy výuky programování a navrhnuta řešení, jak vést výuku studentů, jak implementovat trendy do výuky a tím zjednodušit pochopení látky ze strany studenta. Hlavním přínosem práce je tedy soubor doporučení pro výuku programování se zaměřením na úvodní kurzy.

Klíčová slova

Architecture First, metodiky výuky, OOP, trendy, výuka programování, vzdělávání.

Abstract

The aim of this Bachelor's thesis is to map trends in programming and their use in education. This objective is achieved by analysing trends in programming in the present as well as predictions for the near future and by subsequent interviews with experts that teach programming in Czech schools. The next part of the thesis analyses methodologies for teaching programming. In more detail, it focuses on the methodology called Architecture First created by the supervisor of this thesis Ing. Pecinovský.

Based on the conducted survey, the obtained data were discussed and various problems related to teaching programming were described. Furthermore, the thesis offers solutions to teaching students and implementing trends in education that could make learning easier for students. Therefore the main contribution of this thesis is a set of recommendations for teaching programming with emphasis on introductory courses.

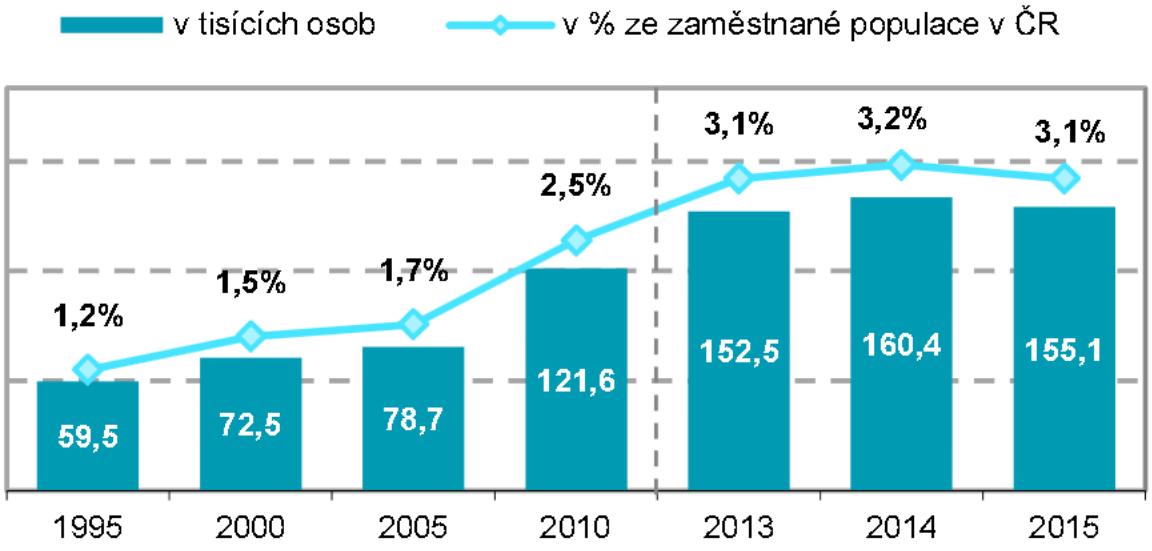
Keywords

Architecture First, education, methodologies of education, OOP, teaching of programming, trends.

Obsah

1 Úvod

Informační technologie jsou nedílnou součástí našich dnešních životů. Aby došlo k automatizaci veškerých výpočetních procesů, je potřeba veškerou techniku kolem nás správně naprogramovat. Bohužel při programování se někdy stane, že ne vždy se řeší daný problém tím správným přístupem. Programovací jazyk je nástroj jako každý jiný a je potřeba se s ním naučit správně zacházet. Na středních školách je tomuto tématu věnováno dotačně málo hodin a výuka je velmi rychlá a nepružná vůči dnešním trendům. Napříč tomu je potřeba programátorů na trhu práce stále vyšší. Americká společnost WalletHub prozkoumala, která povolání jsou pro případné uchazeče nejvíce atraktivní. Už na čtvrtém místě se objevuje vývojář webových aplikací. [1] Dle Českého statistického úřadu v roce 2015 bylo v ICT (Informační a komunikační technologie) sektoru zaměstnáno 3,1 % populace z celkového počtu zaměstnanců v České republice. [2] Během deseti let se počet obyvatel zaměstnaných v tomto sektoru zdvojnásobil, stejně jako se zdvojnásobil i podíl daného sektoru na trhu práce. V posledním roce 2015 můžeme vidět mírný pokles, obdobnou tendenci ale za rok 2016 osobně neočekávám (data za rok 2016 prozatím nejsou k dispozici).



Obrázek 1: Graf vývoje zaměstnanosti v ICT sektoru [2]

Z celkového počtu 155,1 tisíc osob dělá analytickou a vývojářskou práci softwaru a počítačových aplikací 44,9 tisíc zaměstnanců a jejich průměrná mzda činí u vývojářů softwaru 55 216,- Kč a u programátorů 49 620,- Kč, což je částka vysoce nad průměrem mzdy v tuzemsku. V České republice bylo v roce 2016 evidováno na úřadu práce 1184 nabídek pro analytiky a vývojáře softwaru počítačových aplikací. Počet těchto nabídek není konečný, firmy používají i jiné cesty k získání pracovníků. Proč je na trhu práce takový nedostatek kvalitních odborníků na dané téma, když přitom každý rok z univerzit odchází s diplomem přibližně čtyři tisíce absolventů, je nejasné.

Informační technologie se stále vyvíjejí, dostávají se i do dalších odvětví, příkladem toho může být mobilní zdravotnictví, tzv. mHealth. Jak je vidět, důležitost informačních technologií stále roste a jinak tomu nebude ani v příštích letech.

Výběr tématu této bakalářské práce odůvodňuji tím, že mně samotnému při studiu Informačních technologií právě programování dělalo největší problémy, a tento problém v podstatě stále trvá. Předmět jako takový je zajímavý a znalost tohoto odvětví je velmi ceněná, ale bohužel to není ideální volba pro každého. Doufám, že studentům, kteří s tímto předmětem mají problém, dále zmíněné trendy pomohou.

1.1 Cíle práce

Hlavním cílem této bakalářské práce je analyzovat trendy ve výuce programování na školách, a to se zvláštním zaměřením na vstupní kurzy. Před popisem těchto trendů popisuji v první části bakalářské práce aktuální stav výuky na českých školách, aby si čtenář mohl udělat svůj komplexní pohled na věc.

Dílčím cílem práce je provedení výzkumu u začínajících programátorů a následné zjištění, v čem měli problém při výuce programování, zda pociťují nějaké trendy ve výuce programování a co je během jejich praxe překvapilo. Dalšími dotazovanými budou pedagogičtí pracovníci, kteří programování vyučují.

V bodech se dají cíle práce vyjádřit takto:

- analýza výukových procesů – popis metod výuky;
- popis současných trendů ve výuce programování;
- provedení průzkumu a jeho vyhodnocení.

1.2 Cílová skupina

Práce je určena každému, koho daná problematika programování a trendů zajímá. Hlavní cílovou skupinou této práce jsou studenti oboru Informatika. Další skupinou jsou pedagogičtí pracovníci, kteří chtějí pomoc svým studentům v průběhu studia programování a přinést nové rady do výuky tak, aby odpovídala požadavkům praxe.

1.3 Použité metody

Pro teoretickou část jsou potřebné informace získány z odborné literatury, jež je zaměřená na problematiku vývoje softwaru, výuku programování a další související témata.

Praktická část je zpracována na základě strukturovaných rozhovorů, na které jsem pozval odborníky z praxe z vybraných českých škol, vyučující předmětu programování. V další části je použito dotazníkového řešení ke zhodnocení současné metodiky výuky. Na dotazníky odpovídali studenti informatických oborů. V ojedinělých případech je využito e-mailové komunikace.

1.4 Struktura práce

Bakalářská práce je strukturována do několika samostatných kapitol. Druhá kapitola se nazývá rešerše, kde se zabývám tuzemskou a zahraniční literaturou na dané téma, kterou využívám z větší části v této práci. Ve třetí kapitole se zabývám obecně tvorbou učebního plánu k předmětu programování na českých školách. V další kapitole čtenáře seznámím s hlavními důvody, proč některým studentům dělá předmět programování tolik potíží. V následující části práce se zaměřuji na metodiky výuky programování, kde rozeberu programovací paradigmata a obecné formy výuky. Ve své šesté kapitole pojednávám o trendech současných a budoucích v programování. A v sedmé kapitole popisuji své dotazníkové řešení. Na závěr navrhuji několik řešení (trendů) pro zlepšení výuky a komentuji zjištěná data z rozhovorů.

2 Komentovaná rešerše informačních zdrojů

V této kapitole se zabývám literaturou na téma výuky programování a témata tomu přidružená. Na trhu můžeme nalézt mnoho publikací, většina je v anglickém jazyce. Nejvýznamnější doporučenou literaturu níže vypíši a přidám svůj popis, kterým se pokusím vystihnout podstatu a obsah každého z děl. Ostatní použité zdroje v seznamu použité literatury byly pro tvorbu práce též důležité, avšak ne stěžejní. Naopak neméně důležitou součástí této práce, zejména její praktické části jsou lidské zdroje; respektive získané názory, informace, myšlenky a postoje všech, kteří byli ochotni na téma bakalářské práce diskutovat a bez nichž by praktická část nemohla vzniknout.

2.1 Tuzemské publikace

Důležitým zdrojem při zabývání se metodikami výuky je vědecký článek *Metodika Architecture First*, autora Ing. Rudolfa Pecinovského, Csc. [3] Článek má za cíl seznámit čtenáře s metodikou Architecture First a jejími jednotlivými etapami. Vychází ze zkušeností autora a stížností firem, které mají odlišné požadavky po absolventech škol. Podle nich nejsou studenti schopni myslet abstraktně a dívat se na problémy komplexně. V rámci závěrečného dotazníkového šetření budou respondenti dotazování, zda by jim byla tato metodika vhodná a proč.

Odborný článek *Současné trendy v metodice výuky programování* je druhým informačním zdrojem od autora Ing. Rudolfa Pecinovského, Csc. [4] Jedná se o odborný článek, který popisuje jednotlivé přístupy k výuce programování (hardware-first, algorithms-first, imperative-first, functional-first, objects-first, breadth-first), které jsou zmíněny i v rámci této bakalářské práce. Autor zde popisuje i rozdíly mezi strukturovaným a objektově orientovaným programováním a jeho osobní zkušenosti s výukou programování.

Výuka programování v jazyce Python pro střední školy je diplomová práce Hany Horáčkové, která mě zaujala z důvodu zaměření na programovací jazyk Python pro střední školy.[5] Jedná se o souhrnnou práci o tomto jazyce, popisuje výhody a nevýhody výuky v něm. Dále je v ní doporučeno, jaké učebnice používat a čemu se vyvarovat. Autorka dále práci doplnila o základy výuky tohoto jazyka i s příklady.

2.2 Zahraniční publikace

Publikace *Reflections on the Teaching of Programming* od kolektivu skandinávských vědců, je významným informačním pramenem této práce. [6] Tato publikace je psána v anglickém

jazyce a doporučuji ji především pro odborníky v oboru. Autoři zde popisují problémy výuky programování – špatná orientace výuky, málo materiálů na proces programování a řešení problémů, ale také to, že programování je opravdu těžká nauka („mix umění a vědy“) a ne každý ji může vyučovat. Kniha obsahuje sbírku článků o tom, jak předávat vzdělávání v oboru programování, informace o pedagogických experimentech a další. V případě řešení problematiky výuky programování, mohu tuto publikaci jenom doporučit.

Computing Curricula 2001 je dokument v anglickém jazyce, popisující obor Computer Science. Autoři dokumentu zde diskutují o výuce úvodních kurzů programování a vystihují výhody a nevýhody různých variant přístupů k programování. Jako nejideálnější přístup volí autoři začít výukou objektů. [7]

Pedagogical Patterns: Advice For Educators od Josepha Bergina, je sbírka úspěšných technik, které pomáhají při výuce a učení zejména technických předmětů. [8] Pro profesionální pedagogy se tyto vzorce mohou zdát zřejmé, dokonce až triviální. Já jsem díky této sbírce získal pohled na věc od zkušených učitelů.

3 Tvorba učebního plánu v České republice

Na úvod představím, jak probíhá tvorba učebního plánu v České republice a dále se budu zaobírat tím, jaké jsou největší problémy při výuce programování. Čtenáři této bakalářské práce tak umožním udělat si svůj pohled na věc, proč je třeba složité dále zmíněné trendy ve výuce programování promítnout. Ve spolupráci s Ing. Kaiferovou ze Smíchovské střední průmyslové školy jsem prošel veškeré kroky tvorby učebního plánu a krok po kroku je rozepíši. Již z této kapitoly může čtenář vydedukovat, že není lehké reagovat na trendy v programování, protože administrativní úkony prodlužují dobu nasazení těchto trendů do výuky.

3.1 Rámcový vzdělávací program

Celé školství se řídí velkým množstvím zákonů a předpisů. Nejdůležitějším dokumentem, spolu s Národním programem rozvoje vzdělávání v České republice (tzv. Bílou knihou), je Rámcový vzdělávací program. [9] Ten školám určuje:

- *konkrétní cíle, formy, délku a povinný obsah vzdělávání, a to všeobecného a odborného podle zaměření daného oboru vzdělání, jeho organizační uspořádání, profesní profil, podmínky průběhu a ukončování vzdělávání a zásady pro tvorbu školních vzdělávacích programů;*

- *podmínky pro vzdělávání žáků se speciálními vzdělávacími potřebami a nezbytné materiální, personální a organizační podmínky a podmínky bezpečnosti a ochrany zdraví.*

Rámcové programy tedy závazně určují „rámce“ pro jednotlivé etapy vzdělávání mládeže (předškolní, základní, střední vzdělání). Jsou zavedeny školským zákonem č. 561/2004 Sb., § 3 a § 4, o předškolním, základním, středním, vyšším odborném a jiném vzdělávání, kde se dočteme, že rámcové vzdělávací programy vydané ministerstvem školství mládeže a tělovýchovy jsou povinné: „*Vymezují povinný obsah, rozsah a podmínky vzdělávání; jsou závazné pro tvorbu školních vzdělávacích programů, hodnocení výsledků vzdělávání dětí a žáků…*“ [10] Než může kantor začít vyučovat daný předmět, musí se jednotlivé tematické plány přizpůsobit rámcovému vzdělávacímu programu. Pro každý vzdělávací obor je zpracovaný samostatný rámcový vzdělávací program. Co se týče výuky programování a vývoje aplikací, cílem tohoto okruhu je naučit žáka vytvářet algoritmy a pomocí programovacího jazyka zapsat zdrojový kód. [5]

Tabulka 1: Rámcový vzdělávací program pro obor Informační technologie [11]

Výsledky vzdělávání	**Učivo**
Žák: - zná vlastnosti algoritmu; - zanalyzuje úlohu a algoritmizuje ji; - zapíše algoritmus vhodným způsobem;	**1 Algoritmizace** - význam, prvky algoritmu
- použije základní datové typy; - použije řídící struktury programu; - vytvoří jednoduché strukturované programy;	**2 Strukturované programování** - datové typy - řídicí struktury
- rozumí pojmům třída, objekt a zná jejich základní vlastnosti; - použije jednoduché objekty;	**3 Úvod do objektového programování** - třída, objekt, vlastnosti tříd
- zná výhody použití jazyka SQL; - použije základní příkazy jazyka SQL;	**4 Základy jazyka SQL** - základní příkazy (SELECT, UPDATE, INSERT, DELETE)
- aplikuje zásady tvorby WWW stránek; - orientuje se ve struktuře HTML stránky; - vytvoří webové stránky včetně optimalizace a validace; - použije formuláře a skriptovací jazyk.	**5 Tvorba statických a dynamických webových stránek**

Pro pokrytí veškeré látky získávají školy hodinové dotace na předměty. Rozvržení učebních hodin v oboru 18 – 20 – M/01 Informační technologie naleznete na následujícím obrázku č. 2.

Vzdělávací oblasti a obsahové okruhy	Minimální počet vyučovacích hodin za celou dobu vzdělávání	
	týdenních	celkový
Jazykové vzdělávání - český jazyk - cizí jazyk	 5 10	 160 320
Společenskovědní vzdělávání	5	160
Přírodovědné vzdělávání	6	192
Matematické vzdělávání	12	384
Estetické vzdělávání	5	160
Vzdělávání pro zdraví	8	256
Vzdělávání v informačních a komunikačních technologiích	4	128
Ekonomické vzdělávání	3	96
Hardware	5	160
Operační systémy	6	192
Aplikační software	8	256
Počítačové sítě	4	128
Programování a vývoj aplikací	8	256
Disponibilní hodiny	39	1 248
Celkem	**128**	**4 096**

Obrázek 2: Hodinová dotace pro obor Informační technologie [11]

Na výuku programování mají na středních školách k dispozici celkově 256 hodin po dobu čtyř let studia. Toto číslo se může navýšit o disponibilní hodiny, které škola rozděluje do jednotlivých vzdělávacích oblastí pro podporu zájmové orientace žáků.

Na školní úrovni si učební plán upravuje samotná škola pomocí školního vzdělávacího programu.

3.2 Školní vzdělávací program

Učivo předepsané rámcovým vzdělávacím programem si samotná škola rozvrhne do vyučovaných předmětů a vytvoří tzv. Školní vzdělávací program (dále jen ŠVP), podle kterého je uskutečňováno vzdělávání studentů. To znamená, že ŠVP musí být v souladu s daným Rámcovým vzdělávacím programem. Legislativně je ŠVP uveden ve školském zákoně číslo 561/2004 Sb. Díky ŠVP může škola odlišit a vyprofilovat svou výuku, a získat tím další žáky; formulovat vlastní připomínky; vynechat zbytečné duplicity a spolupracovat v rámci mezioborového vzdělávání uvnitř pedagogického sboru. Jak vyplývá z výše řečeného, ŠVP by měly být obsahově propracovanější a přesnější než RVP. ŠVP je vydáván ředitelem dané

školy a je veřejně přístupný komukoliv – rodičům, studentům i široké veřejnosti. Níže uvedený školní vzdělávací program pochází ze Smíchovské střední průmyslové školy Preslova, která se před deseti lety podílela na tvorbě RVP pro obor Informační technologie.

Níže ukázka Školního vzdělávacího programu k předmětu Programování na Smíchovské střední průmyslové škole.

Název předmětu: **Programování**
Obor vzdělání: **Informační technologie**
Ročník: **1.** **Hodinová dotace: 2**

UČIVO	VÝSLEDKY VZDĚLÁVÁNÍ	POČET HODIN
1. ALGORITMIZACE – PRŮŘEZOVĚ BĚHEM UČIVA		***10***
- význam, prvky algoritmu - principy algoritmizace - analýza úlohy - zápis algoritmizace	Žák - zná vlastnosti algoritmu; - zanalyzuje úlohu a algoritmizuje ji; - zapíše algoritmus vhodným způsobem; - zná principy algoritmizace; - ovládá principy algoritmizace úloh a sestavuje algoritmy řešení konkrétních úloh;	
2. HISTORIE PROGRAMOVÁNÍ – PRŮŘEZOVĚ BĚHEM UČIVA		***2***
- historie programování	Žák - vysvětlí a popíše historii programování;	
3. TŘÍDĚNÍ PROGRAMOVÁNÍ – PRŮŘEZOVĚ BĚHEM UČIVA		***2***
- programovací jazyky	Žák - zná různé typy programovacích jazyků;	
4. KONVENCE – PRŮŘEZOVĚ BĚHEM UČIVA		***2***
- programovací konvence	Žák - zná programovací konvence; - používá programovací konvence při vlastním programování;	
5. STRUKTUROVANÉ PROGRAMOVÁNÍ		***16***
- datové typy - řídicí struktury - proměnné - datové typy proměnných - operátory - jednoduché rozhodovací struktury	Žák - použije základní datové typy; - použije řídicí struktury programu; - vytvoří jednoduché strukturované programy;	
6. ÚVOD DO OBJEKTOVÉHO PROGRAMOVÁNÍ – APLIKACE WINDOWS		***20***
- vývojové prostředí pro programování - ovládací prvky - vlastnosti ovládacích prvků - proměnné - datové typy proměnných - operátory - barvy - grafické prvky - jednoduché rozhodovací struktury	Žák - zná vývojové prostředí pro programování; - ovládá práci ve vývojovém prostředí; - zná ovládací prvky a jejich vlastnosti; - používá ovládací prvky při programování; - umí měnit vlastnosti ovládacích prvků za běhu programu; - chápe význam proměnných; - zná datové typy proměnných; - používá proměnné při programování; - zná operátory; - používá operátory při programování; - zná práci s barvami; - používá barvy při programování;	

Obrázek 3: Školní vzdělávací program Smíchovské střední průmyslové školy

3.3 Tematické plány

Tematický plán je dokument, který provází učitele a jeho žáky průběhem roku. V tematickém plánu pro daný předmět musí být zahrnuto veškeré učivo z ŠVP daného ročníku. Jedná se o pomůcku pro vyučujícího, který si může předem rozvrhnout (a třeba i přeskládat) učivo tak, aby mělo logickou návaznost. Tento plán obsahuje zejména název vyučovaného předmětu, hodinovou dotaci, časový rozvrh výuky pro daný školní rok, pravidla hodnocení, požadované učebnice, další učební pomůcky apod.

Tematický plán

Název: **Programování a vývoj aplikací**
Obor: **Informační technologie**
Ročník: 1. **Hodinová týdenní dotace: 2** **Školní rok: 2015/2016**

Časový rozvrh	Téma	Počet hodin
1-10	Algoritmizace	10
	Algoritmu, analýza, značky vývojového diagramu	2
	Příklady – možné rozložit do celého školního roku	8
11-12	Programovací jazyky, vývojové prostředí (C# Console Application)	2
13-14	Programovací konvence, syntaxe, sémantika	2
15-18	Primitivní datové typy, datum a čas – DateTime	4
19-20	Proměnné, konstanty	2
21-22	Formátovací řetězce – základní úroveň	2
23-26	Operátory, Třída Math (základní fce)	4
27-30	Konverze typů (Convert(), Parse), Metoda TryParse	4
31-34	Náhodné číslo, if, if –else, if- else if -else	4
35-36	Switch	2
37-38	Metody – úvod do problematiky metod	2
39-40	For	2
41-46	Pole jednorozměrné, pole dvourozměrné, foreach	6
47-50	While, do-while	4
51-54	Práce s řetězci	4
55-58	Práce s textovými soubory	4
59-62	Výjimky a jejich ošetřování, přetečení checked, unchecked	4
63-68	Opakování	6
Celkem		68

Obrázek 4: Tematický plán Smíchovské střední průmyslové školy

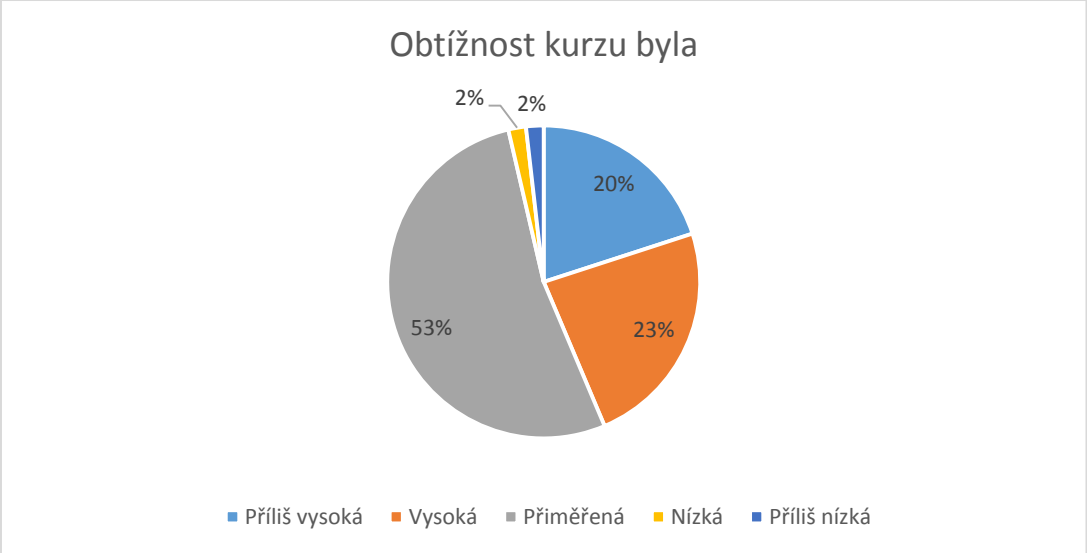
Soulad a dodržování veškerých plánů má na starost Česká školní inspekce.

4 Problémy při výuce

Pokud máme materiál, podle kterého chceme učit, je potřeba se zaměřit na odbourání všech brzd, které studenty nebo kantory trápí při výuce programování. Některé problémy mohou odbourat právě trendy zmíněné v dalších kapitolách.

4.1 Hodinová dotace

Na středních školách je výuce programování průměrně věnováno 280 hodin (dle rozhodnutí ředitele) studia po dobu 4 let. Na Vysoké škole ekonomické v Praze probíhá výuka povinně volitelného předmětu Programování v jazyce Java pro studenty oboru Aplikovaná informatika. Tento kurz je určen pro začátečníky a je ohodnocen 7 ECTS kredity, což odpovídá 182 hodinám studijní zátěže. Je ale počet hodin věnovaných výuce programování dostačující? Ve vstupních kurzech nejsou od studentů předpokládány žádné předchozí znalosti, nicméně výuka probíhá od prvních hodin velmi rychle, což vyplývá i z mého dotazníkového šetření. Proti této teorii mluví data z předmětové ankety VŠE pro zimní semestr 2015/2016, kde více než 53 % dotázaných studentů obtížnost kurzu hodnotí jako přiměřenou. Rozdílné výsledky mohou být zapříčiněny jiným vzorkem respondentů a jinak položenou otázkou.



Obrázek 5: Předmětová anketa VŠE, ZS 2015/2016 [12]

Na základních školách je situace ještě horší. V rámci RVP je algoritmizace začleněna v oblasti „Informační a komunikační technologie“. Zde jako jeden z cílů výuky najdeme „*schopnosti formulovat svůj požadavek a využívat při interakci s počítačem algoritmické myšlení*“. Výuka programování je tedy z velké části na rozhodnutí ředitelů základních škol. Z článku Modern approaches to teaching programming vyšlo najevo, že ze vzorku zkoumané skupiny základních škol a víceletých gymnázií jen 32 % respondentů zařadilo programování do svých tematických plánů v rámci předmětu výpočetní techniky, 22 % dotázaných vyučovalo programování v dřívějších dobách a celých 46 % programování nikdy nevyučovalo. [13].

Asi každý souhlasí s tím, že velkým motivátorem a tím, kdo z velké části stojí za úspěchem pochopení látky, je samotný učitel. Důvodem, proč výuka nezačíná již na prvním stupni a nechává se až na střední školu, je především nedostatečná aprobace učitelů. Jako svou původní aprobaci uvádí informatiku pouze 15 % respondentů. [13] Zajímavý je také průzkum provedený na samotné pedagogické fakultě, kde 67,4 % studentů uvedlo, že vzdělávání v oblasti nových technologií u začínajících učitelů je v České republice nedostatečné. [14]

V rámci této práce jsem udělal malý průzkum, kdy jsem se na dvaceti náhodně vybraných základních školách zeptal, zda zařadili programování do výuky na 2. stupni výuky. Pouze osm z dotázaných škol zařadilo programování v rámci výuky informatiky.

Ministerstvo školství v tomto ohledu plánuje do roku 2020 významnou změnu. [15] Už za několik let by se mohli žáci seznámit povinně s výukou programování na prvním stupni. Výuka by probíhala zábavnou formou – zadávání příkazů robotům, práce s některým výukovým program (např. Baltík a další). Na druhém stupni by se ve výuce pokračovalo v některém programovacím jazyce komplexněji. Spolu se zavedením nových osnov se očekává navýšení hodin informatiky. Dle mého názoru kromě počtu vyučujících hodin je důležitější samotná kvalita výuky.

Zajímavým příkladem může být Finsko, kde se učí děti pronikat do fungování technologií již od útlého věku. Programování je zde zakomponováno do finských rámcových vzdělávacích programů. Oproti zbytku světa zde jdou trochu jiným směrem. Nejde o izolovaný předmět, ale o multidisciplinární výuku. Například ve výuce výtvarné výchovy se studenti učí plést a sestavovat vzory, během tělesné výchovy zase sestavují sekvence pohybů nebo se učí základní funkce pomocí příběhů. Linda Liukas (finská programátorka, autorka série knih o rozmarné Ruby, která provází děti taji kódování i programování) jako příklad uvádí [16]: Táta řekne Ruby, ať se obleče. Holčička uposlechne příkaz a obleče další vrstvu oblečení na své pyžamo. Student musí opravit svůj požadavek a specifikovat, že chce, aby si Ruby nejdřív svlékla pyžamo a pak teprve oblékla. Jako nevýhodu programátorka vidí vysoké nároky na učitele napříč všemi obory. Každý z učitelů od přírodovědných předmětů po učitele výtvarné výchovy musí pochopit základy programování. Na druhou stranu učitelé ve Finsku mají větší autonomii, pokud se bavíme o tom, jak učit, a učitelství je zde prestižní a dobře ohodnocený obor zaměstnání. Na zdejší pedagogické fakulty se tak dostane jen 10 % nejlepších uchazečů a ti se učí, jak být svým studentům průvodci, ne přednášejícími.

4.2 Motivace

Jako další bod, který musím zmínit, je motivovanost studentů. Vždy záleží na studentově inciativě zabývat se danou problematikou. Žáci na středních školách jsou ve věku, kdy vrcholí různorodost zájmů těchto věkových skupin a velmi snadno získávají nový zájem. Motivace se postupem času prohlubuje v činnosti, ve které student dosahuje určitých cílů a posunů ve své práci. Bohužel vývoj aplikací je spojen s častým nezdarem projektů. A i toto

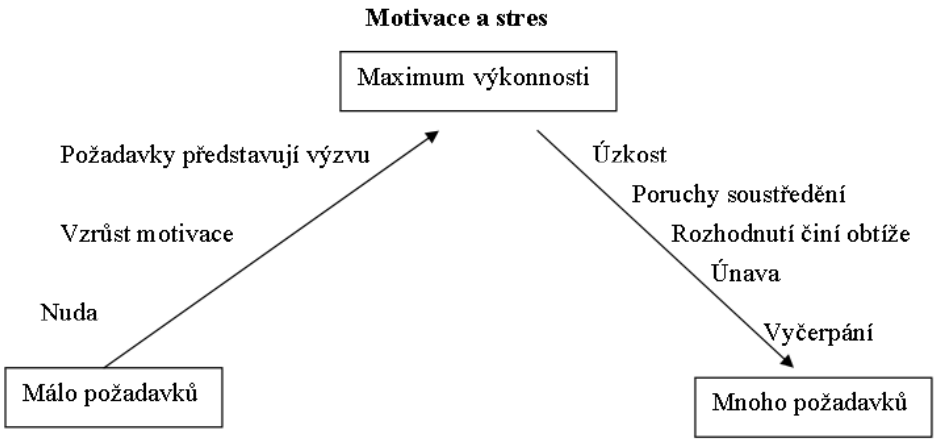
působí velmi rušivě a několik studentů odradí od prohlubování znalostí. Žáci by neměli být motivováni strachem ze svého neúspěchu, ale touhou uspět.

4.2.1 Postoj učitele

Důležitým prvkem během výuky je samotný učitel. Může to pro nás být životní vzor, který ovlivní naše další směřování, volbu povolání, ale můžeme se setkat i s učitelem, který své žáky soustavně nemotivuje. Díky tomu může žák ztratit o dříve oblíbený předmět zájem. Učitel musí odstranit všechny studentovy demotivátory a znát základní motivační činitele. Učitel může motivaci pojmout ze dvou hledisek [17]:

- *„Učitel prostřednictvím motivace zvyšuje efektivitu učení, pracuje s motivační sférou žáka tak, aby vzbudil jeho zájem o školu o vyučování.“*
- *„Učitel musí současně tuto motivační sféru žáka dále rozvíjet, kultivovat ji, zakládat nové motivy apod.“*

Motivovanost žáků velmi negativně ovlivňuje stres a nezájem o dané téma. Dalším z činitelů je míra požadavků ze strany učitele. Závislost těchto činitelů je zobrazena na následujícím obrázku.



Obrázek 6: Výše výkonu v závislosti na míře požadavků [18]

Velmi důležitý je i postoj, který učitel vůči žákovi zaujme. Žák, který je u učitele zaškatulkován jako špatný, je přehlížen a málo chválen, málo je tedy motivován. Na základě této zkušenosti se postupem času žák přestane snažit ve svém výkonu. Oproti tomu, kdo je u učitele brán jako dobrý student, dostává více pozornosti, více je chválen, a celkově je k němu zaujímán lepší postoj. Díky tomu se pak může zlepšit jeho postoj k výuce. [19]

4.2.2 Odměny a tresty

Odměna je pro nás vyjádřením pozitivního hodnocení a přináší studentovi radost a uspokojení některé z jeho potřeb. Oproti tomu trest si spojujeme s chováním nebo jednáním studenta, které vyjadřuje negativní hodnocení a přináší frustraci, demotivaci nebo omezení některých jeho potřeb. Dle mnoha učitelských, ale také manažerských knih platí, že odměna zvyšuje frekvenci chování, které k ní vede, a pozitivní feedback je mnohem účinnější než negativní nebo korektivní. Spisovatel Marvin Pasch ve své knize Od vzdělávacího programu k vyučovací hodině popisuje podoby odměn a trestů pro žáky. [20]

Odměny:

- prodloužení přestávky, pochvala do žákovské knížky, představením žáka jako vzoru ostatním, pochválení u rodičů, materiální odměna.

Tresty:

- zkrácení přestávky, zapsání jména na tabuli, telefonát rodičům, sezení s ředitelem.

4.3 Výukový materiál

Dalším tématem, na které vedení škol poukazuje, je nedostatečný výukový software, chybějící učební materiál a inovativní metodiky výuky. V této kapitole jsou uvedeny některé podpůrné programy pro vstupní výuku programování. Jejich předností od profesionálních výukových prostředí je snadná orientace v programu a možnost okamžitého uvedení studenta do programování, kdy není nutno zatěžovat pojmy, jakými je balíček nebo třída.

4.3.1 Scratch

Scratch je navržen tak, aby ho pochopil každý. Jedná se o úvodní nástroj k výuce programování, je tedy určen pro uživatele, kteří nemají žádné zkušenosti s programováním. Hlavní cílovou skupinou jsou děti ve věku 8 až 16 let. Provoz je možný jako aplikace na Windows, macOS i Linux, jde také spustit i v samotném prohlížeči.

Pořízení programu

Projekt je zcela bezplatně poskytován laboratoří MIT Media Lab. Jako hlavní výhodu vidím, že poskytuje cloudové řešení, které je k dispozici přes internetový prohlížeč. Stačí jen spustit příslušnou webovou stránku, přihlásit se (není podmínkou použití) a začít programovat.

Prostředí

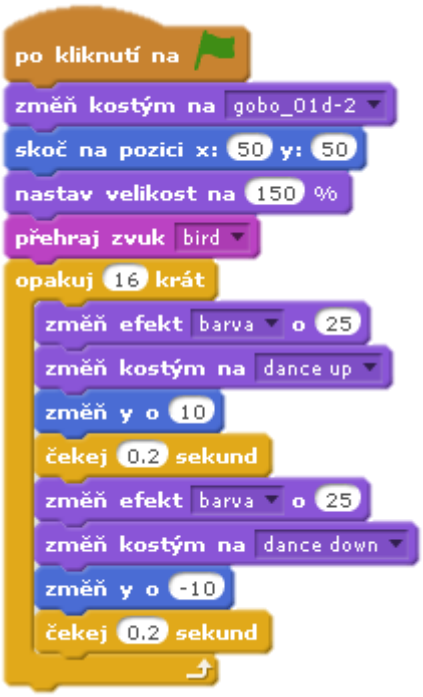
Desktopové a webové prostředí jsou víceméně totožná.Student zde zadává příkazy průvodci programem, kočce, která plní vaše požadavky. Od pouhého pobíhání postavy se dítě přesune

k podmínkám, dialogům nebo proměnným. Žák nemusí napsat jakýkoliv kód, vše probíhá spojováním jednotlivých bloků příkazů, které se do sebe vkládají podobně jako puzzle. Díky tomu se odbouraly syntaktické chyby. V programu Scratch ihned vidíme, že pokud nám příkazy do sebe nezapadnou, nemůžeme je použít v tomto pořadí.

Program seznámí žáky se všemi základními prvky programování. Zápis a úprava algoritmů je zde jednodušší než v dále popsaném programu Baltík.

Uživateli je k dispozici celá řada ukázkových příkladů a hotové dílo může tvůrce jednoduše umístit na webovou stránku. Prostředí Scratch nabízí mnoho jazykových verzí včetně té české.

Ukázka zdrojového kódu



Obrázek 7: Ukázka zdrojového kódu programu Scratch [vlastní zpracování]

4.3.2 Baltík

Jedná se o programovací a kreslící nástroj pro děti od 4 let. Produkt pochází z dílny společnost SGP a na internetových stránkách společnosti můžete program stáhnout ve zkušební verzi 3.0.71, která oproti zakoupené verzi neumožňuje ukládat soubory.

Pořízení programu

Pořízení licence programu vyjde od 3.500,- Kč za roční používání pro všechny školní počítače až po 25.000,- Kč za licenci časově neomezenou. Společnost nabízí i učebnici pro výuku programování od autorů Rudolfa Pecinovského a Jiřího Váchy za cenu 199,- Kč.

Prostředí programu

Program Baltík používá syntaxi jazyka C. Pro snadnější pochopení zde neprogramujeme pomocí textových příkazů. Ty jsou nahrazeny grafickými ikonkami. Prostředí není pro vytváření složitých projektů, ale pro pochopení programovacích základů. Program nabízí několik režimů, jak vést výuku – skládat scénu, čarovat scénu, programovat začátečník, programovat pokročilí. Každý režim je určen pro určitou věkovou kategorii studentů. Program obsahuje základní příkazy jazyka C (for, if, while, do-while a další). Dále můžeme používat proměnné, konstanty a pole různých datových typů. Jsou zde k dispozici také procedury a funkce.

Jako nevýhodu vnímám špatnou rozpoznatelnost příkazů. Při zkoušení programu jsem musel použít nápovědu a až poté jsem pochopil, co daný symbol představuje. Na druhou stranu nápověda je velmi dobře zpracovaná a uživatel z ní rozhodně pochopí, jak příkaz funguje.

Další nevýhodou je, že tento program není určen pro objektové programování. S objektovým programováním se mohou žáci seznámit v programu Baltie, který je volným pokračovatelem Baltíka 3. Oproti Baltíkovi je založen na C# a obsahuje kompilátor, který nám umožňuje vytvářet .exe soubory.

Ukázka zdrojového kódu

Obrázek 8: Ukázka zdrojového kódu v programu Baltík [vlastní zpracování]

4.3.3 BlueJ

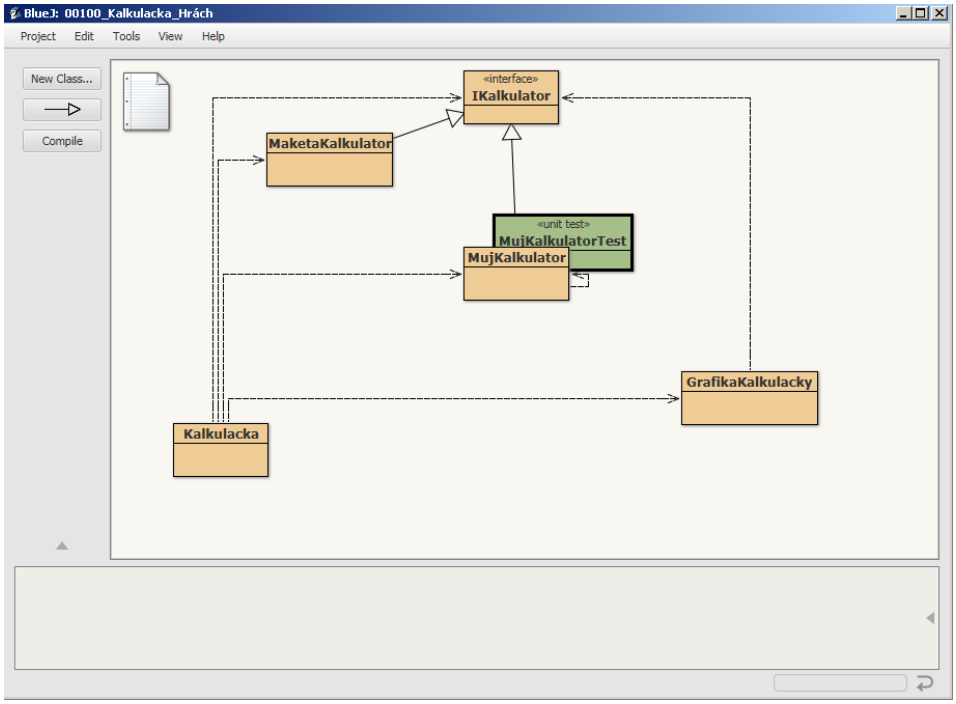
Jedná se o velice známé multiplatformní vývojové prostředí pro podporu výuky objektově orientovaného programování.

Pořízení programu

Toto prostředí je volně šiřitelné a je dostupné ve verzi 4.0.1 pro Windows, macOS a pro ostatní operační systémy s podporu Javy. Proces instalace je relativně přímý a rychlý. Pro výuku objektově orientovaného programování v BlueJ je na trhu i několik knižních titulů *Myslíme objektově v jazyku Java, 2.vyd.* a *OOP: Naučte se myslet a programovat objektově.* Autorem těchto knižních titulů je Ing. Rudolf Pecinovský, CSc.

Prostředí programu

Student pracuje v grafickém prostředí, díky němuž snadno získá znalosti o objektech, dědičnosti nebo rozhraní. Svým určením není zacílen na žáky základních škol, ale spíše na studenty středních, respektive vysokých škol. Prostředí dále obsahuje textový editor kódu, včetně zvýraznění syntaxe a chyb. Jako nespornou výhodu je třeba uvést možnost vypsání chybného řádku při zpracování programu, nápovědy k chybám a možnost oprav chyb při běhu. Jako nevýhodu BlueJ vidím absenci našeptávače nebo nabízení tříd a importu.

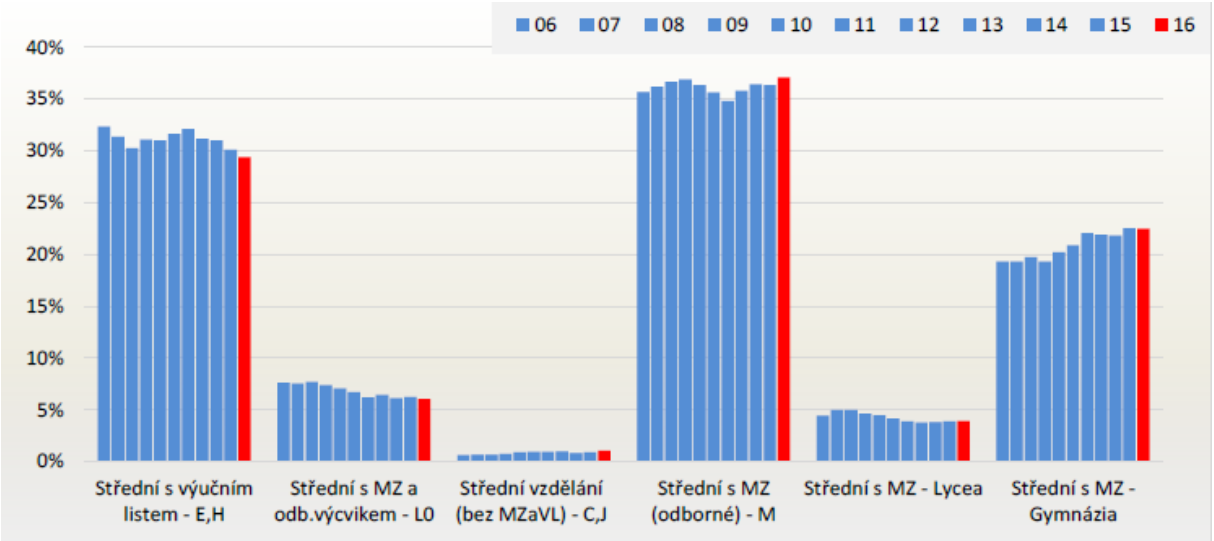


Obrázek 9: Prostředí aplikace BlueJ [vlastní zpracování]

4.4 Schopnosti (úroveň) studentů

Programování vyžaduje určitou míru logického a analytického myšlení, základem pro správné programování je totiž řešení problémů. Je dnešní generace ale schopna samostatně řešit problémy? Výuka programování na středních školách probíhá zejména na gymnáziích a průmyslových školách. V posledních letech je na trhu práce všeobecně známo, že chybí

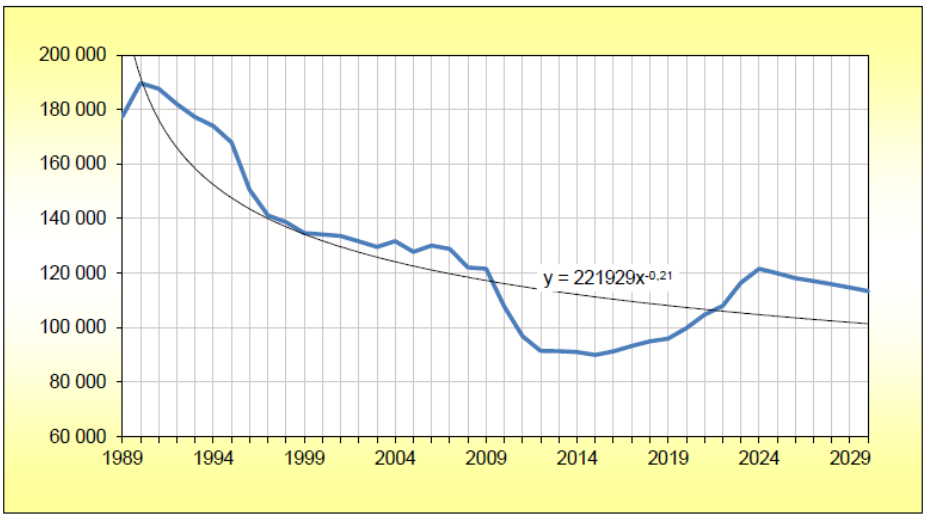
mnoho učňů a řemeslníků. Většina rodičů chce po svých dětech ukončení studia minimálně maturitní zkouškou, kterou ale ne všechny učební obory nabízejí. Uznávaný český pedagog a spisovatel Petr Koukal uvedl, že velkým problémem českého školství je, snadná možnost získání svého místa na střední nebo vysoké škole. *„Pokud na vysoké školy odchází 60 a více procent populačního ročníku, pak na nich podle Gaussovy křivky musí nevyhnutelně studovat a dostudovat je také lidé s IQ 90.“* [21]



Obrázek 10: Graf podílů žáků na středních školách [22]

Z přiloženého grafu získaného z Českého statistického úřadu můžeme zjistit, že podíl žáků vstupujících do 1. ročníků středního vzdělávání klesá u oborů s vyúčním listem a naopak se zvyšuje u škol, kde je studium ukončeno státní maturitní zkouškou. Mezi obory zakončené státní maturitní zkouškou patří i obor Informační technologie. Problém nedostatku dostatečného počtu žáků v učebních oborech aktuálně analyzuje MŠMT zavedením jednotných přijímacích zkoušek od roku 2016 a změnou financování škol. [23]

Z dalšího grafu ČSÚ můžeme vidět dlouhodobý vývoj počtu 15letých osob žijících v České republice.



Obrázek 11: Graf vývoje počtu 15letých osob v České republice (vč. trendu), [22]

V roce 2015 bylo dosaženo historického minima a můžeme očekávat nárůst počtu žáků přicházejících na střední školy. Slabší populační ročníky v minulých letech měly několik důsledků:

- přebytek kapacit;
- snížení počtu žáků na učitele;
    - negativní finanční dopad pro školu;
- **snazší získání místa na žákem zvolené škole.**

Z výročních zpráv vybraných středních škol zaměřených na výuku programování byly zjištěny následující počty přijatých žáků v roce 2015.

Tabulka 2: Výsledky přijímacích řízení

Škola	Počet přijatých žáků	Počet nepřijatých žáků
SPŠE Ječná, Praha	67	114
SSPPŠ, Praha	259	86
SPŠE V Úžlabině, Praha	109	14
SPŠE Plzeň	462	90

Jak je vidět, získat místo na střední škole nebylo v roce 2015 nijak obtížné. S tímto tvrzením souhlasilo i několik pedagogů vyučujících na středních školách, že z důvodu negativního

finančního dopadu na školu jsou přijímáni i žáci, kteří by dříve přijati nebyli. Takovéto financování školských zařízení může být již brzy změněno. Dne 8. března 2017 schválil senát novelu školského zákona, která bude garantovat financování skutečného rozsahu vzdělávání. Stát bude podle předlohy, kterou nyní dostane k podpisu prezident, nově platit školy podle počtu odučených hodin na základě vzdělávacích programů, nikoli pouze podle počtu žáků. Dosavadní systém nepostihoval veškerá specifika jednotlivých regionů. Změny ve financování škol by měly platit od 1. 1. 2019; pouze u soukromých škol bude zachován systém dosavadní. [24]

5 Metodiky výuky programování

Počítačová věda, na rozdíl od mnoha jiných technických oborů, nemá dobře popsaný seznam témat, která by se měla objevit prakticky ve všech úvodních kurzech.

Jedním z problémů je, že existuje mnoho typů programátorů a každý se odlišuje v požadavcích od typů ostatních. Společným kritériem je znalost informačních technologií jako celku. Základní otázkou je, jestli budoucí práce programátora bude vytváření složitých algoritmů a části komplexních programů, zda je nutnost, aby měl velkou znalost prostředí a technologií na úrovni operačního systému a jeho architektury. Takoví programátoři se nazývají vývojáři komplexních systémů.

Do další skupiny programátorů můžeme zařadit specialisty síťových technologií. Programování zde není hlavní prioritou, ale je stále součástí pracovní náplně. Jejich úkolem je sledování statistik a vylepšování sítě. Zabývají se i konfigurací nově zaváděných síťových prvků a stávajících síťových prvků. Vystupují spíše z pozice administrátora nebo uživatele. Základní znalost programování je zde důležitá, je nutné, aby znali principy a limity technologií, které ostatní programátoři používají.

Jako další skupinu uvedu programátory – analytiky. Jejich pracovní náplní je hlavně zapojení kreativního myšlení při určování směru, jakým se má řešený projekt zaobírat a jak bude vypadat finální řešení.

Opomíjenou skupinou programátorů jsou testeři. Jejich úkolem je ověřit, zda byly požadavky na projekt splněny, a nalézt i tu nejmenší chybu v práci programátora nebo analytika. Mohou se specializovat i na bezpečnost softwaru, kde testují všechna možná slabá místa. Mají za úkol přípravu scénářů testování, následné provedení testů, zpracování výsledků a neustálou komunikaci s vývojovým týmem.

Jedním způsobem, jak studentům ulehčit průchod předmětem programování, je zvolení vhodné metodiky na výuku programování v počátečních etapách výuky.

5.1 Computing Curriculum 2001

Computing Curriculum 2001 klasifikovalo úvodní kurzy do šesti obecných metod přístupu k programování: Object-first, Functional-first, Breadth-first, Algorithms-first a Hardware first. [7] Pomocí různých metodik dochází při výuce k získání rozdílných vědomostí a praktických schopností absolventů škol zaměřených na výuku informačních technologií. Přestože tato klasifikace pochází již z roku 2001, jsou tyto metody přístupů stále aktuální, pouze se změnilo procentuální zastoupení. Novější verze osnov Computing Curricula 2013 tyto metody nemění, pouze doplňuje, že se vzhledem k rozvoji oblasti IT pravděpodobně obje-

vily i další přístupy, avšak bez bližší specifikace. [25] V následujících odstavcích jsou shrnuty metodiky, kterými lze k výuce programování přistupovat. V praktické části práce jsou uvedeny některé názory učitelů a studentů programování na tyto metodiky.

5.1.1 Nejdříve hardware (Hardware-first)

Zastánci této metodiky tvrdí, že student programování musí nejdříve umět popsat, jak je počítač konstruován. Díky této vědomosti dokáže student pochopit, jak bude jím vytvořený program vykonávat jednotlivé úkony.

Výuka začíná výkladem spínacích obvodů, konstrukcí registrů a aritmetických jednotek. Poté se přesuneme k výkladu konstrukcí programu. Samotné programování začíná výukou na strojovém kódu a až poté se dostaneme k vyšším programovacím jazykům. [7]

Metodika je vhodná jen pro absolventy některých odvětví. Při tvorbě rozsáhlých aplikací je optimální, je-li programátor od hardwaru co nejvíce odstíněn.

5.1.2 Nejdříve příkazy (Imperative-first)

Dle vedoucího této práce, pana Pecinovského, se jedná o stále nejpoužívanější metodiku výuky v ČR. Student se seznámí s klasickými programovými konstrukcemi například v jazycích Pascal a C a až následně s případnou objektově orientovanou nadstavbou. Imperativní zápis přímo popisuje algoritmus neboli postup řešení problému. Nevýhodou je nutnost znalosti syntaxe daného jazyka a v případě tvorby obsáhlejších programů nepřehlednost kódu. Tento přístup je na českých školách běžný, což dokazují i některé odpovědi v dotazníku této práce.

Dosavadní zkušenosti ukazují, že takto vyučovaný student se nedokáže sžít s objektově orientovaným přístupem tak dobře, jako studenti, kteří začali výukou hned prací s objekty. Tento handicap můžeme brát jako vodítko, proč tento přístup nepoužívat. [7]

5.1.3 Nejdříve algoritmy (Algorithms-first)

Přístup nevyužívá k výkladu některého z existujících programovacích jazyků, ale pseudokód. Student řeší například matematické i nematematické úlohy jen za použití tužky a papíru. Cílem je, aby se student naučil řešit problémy a dokázal vysvětlit algoritmy, aniž by se musel zabývat syntaxí konkrétního jazyka. Takto získané znalosti může student aplikovat na téměř jakýkoliv programovací jazyk. Další výhodou je snazší pochopení výpočtu složitosti a náročnosti algoritmů.

Nevýhodou tohoto přístupu je motivace studentů. Vytváření pseudokódů není pro studenty tak atraktivní, jako vytvoření spustitelných kódů, a to z toho důvodu, že student nevidí výsledek své práce. Další nevýhodou je náročnější nalezení potencionálních chyb v pseudokódu. [7]

5.1.4 Nejdříve funkce (Functional-first)

Jedná se o přístup zavedených v osmdesátých letech na MIT. Výhodou je, že sjednocuje počáteční úroveň studentů, kteří se zde setkají s jazykem, jehož filozofie je odlišná od filozofie jazyků hlavního proudu. Tato odlišnost ale může mnoho studentů demotivovat, ti se nechtějí učit něco, co ve své budoucí praxi přímo nepoužijí. Je doporučeno začít s jednoduchým jazykem, například Scheme, na kterém se vysvětlí základy funkcionálního programování. [7] Hansen a Kristensen popsali v jedné z kapitol knihy Reflections on the Teaching of Programming, jak učili funkcionální přístup v začátečnických programátorských kurzech. Dle jejich zkušeností se studenti školeni tímto přístupem následně lehce naučili objektově orientovanému programování. [6] Program je chápán jako matematická funkce obsahující vstupní parametry a mající jediný výstup. Pro pochopení tohoto paradigmatu je nezbytná znalost funkcí z matematiky. To je velkou překážkou pro výuku na středních školách, kdy je tato látka zařazena až do vyšších ročníků.

5.1.5 Nejdříve objekty (Objects-first)

Tato metodika vychází ze skutečnosti, že objektově orientované programování je nejrozšířenější metodikou programování. Pokud si ji mají studenti opravdu osvojit, musí se s ní setkávat od samého počátku výuky. V objektově orientovaném programování je při řešení úloh v počítači snaha vycházet co nejvíce z principů reálného světa. Základní jednotkou je objekt, který odpovídá objektu z reálného světa.

Nevýhodou přístupu může být, že objektově orientované jazyky bývají koncipovány komplexně. To může zapříčinit při výuce zahlcení studentů, pokud vyučující začne studenty seznamovat s daným jazykem v jeho plné šíři. I přes tuto nevýhodu patří přístup Object-first mezi nejrozšířenější. [7]

5.1.6 Nejdříve celkový přehled (Breadth-first)

Podporovatelé této metodiky uvádí, že student by se měl nejdříve seznámit komplexně s problematikou počítačové vědy, a až poté se zaměřit na detaily, jakým je např. programování. Student poté dokáže k problému přistoupit k řešení problému z větší perspektivy a je

schopen v celé jeho šíři. Úvod by měl být sestaven nejen z programování, algoritmů a datových typů, ale také z logiky, operačních systémů, komunikace, grafiky, pravděpodobnosti a matematiky.

Nevýhodou může být časová náročnost úvodního školení, která odkládá výuku programování a navazující předměty až o dva semestry výuky. [7]

5.1.7 Nejdříve architektura (Architecture First)

Tvůrcem metodiky je vedoucí práce pan Ing. Rudolf Pecinovský CSc. a tuto metodiku ve zmíněném zdroji Computing Curriculum 2001 nenajdeme.

Metodika vychází z pedagogické zásady ranního ptáčete (early bird pattern), která udává*: "Organize the course so that the most important topics are taught first. Teach the most important material, the "big ideas", first (and often). When this seems impossible, teach the most important material as early as possible."*. [26] Dle této zásady je doporučováno kurz organizovat tak, aby nejdůležitější témata byla vysvětlena jako první, aby měl student co největší prostor pro osvojení tohoto tématu. V praxi se často setkáváme s absolventy škol, kteří byli učeni jako kodéři a pracovní pozici architekta je obtížné obsadit. Oproti tomu při využití metodiky Architecture First kurzy programování nezačínají samotným kódováním, ale výkladem architektury. V začátku se nemusejí rozptylovat pravidly syntaxe daného programovacího jazyka a mohou se soustředit na návrh správné architektury vytvářeného programu. Navržené programy vytváří automatický generátor kódu, který je součástí vývojového prostředí. Na kódování se přechází až v okamžiku, kdy řešené problémy jsou již za hranicemi použitého generátoru kódu. V tomto okamžiku by již měl student mít osvojenou řadu znalostí a dovedností. [3]

Jak můžeme vidět, výuka probíhá ve čtyřech etapách.

První etapa

Na začátku je student seznámen se základními prvky objektového programování. V této etapě je důležité, aby student pochopil, že v programech je za objekt považováno vše. Studentovi je vysvětlena povaha objektů a základní konstrukce, jako např. skládání objektů, dědění rozhraní, dědění implementace, implementace rozhraní, potřeby zavedení abstraktních tříd a některé další. Výuka probíhá v interaktivním režimu, kdy student zasílá zprávy objektům a sděluje tak, co mají dělat. Tím ukáže vývojovému prostředí, jak se má navržený program chovat. Úloh, které již není třeba programovat, protože je umí naprogramovat generátor kódu, je stále více. Automatizaci stále vzdoruje – a ještě nějakou dobu jí vzdorovat bude – návrh architektury programu. Díky tomu může v počátcích ignorovat nutnost zakódování daných konstrukcí, ale pouze řešit tyto konstrukce na úrovni architektury. Kód vytváří generátor, který je součástí zvoleného vývojového prostředí. [3]

Druhá etapa

Studentům jsou vysvětlována syntaktická pravidla programovacího jazyka a základní programovací konstrukce jazyka. Student opakuje látku probíranou v první etapě a učí se zapsat program, který v první etapě tvořil generátor kódu vývojového prostředí. Díky tomu, že program mají již z první etapy navržený, mohou se více soustředit právě na syntaxi a konstrukci jazyka. Cílem druhé etapy je tedy naučení se kódování při zachování architektonického přístupu. [3]

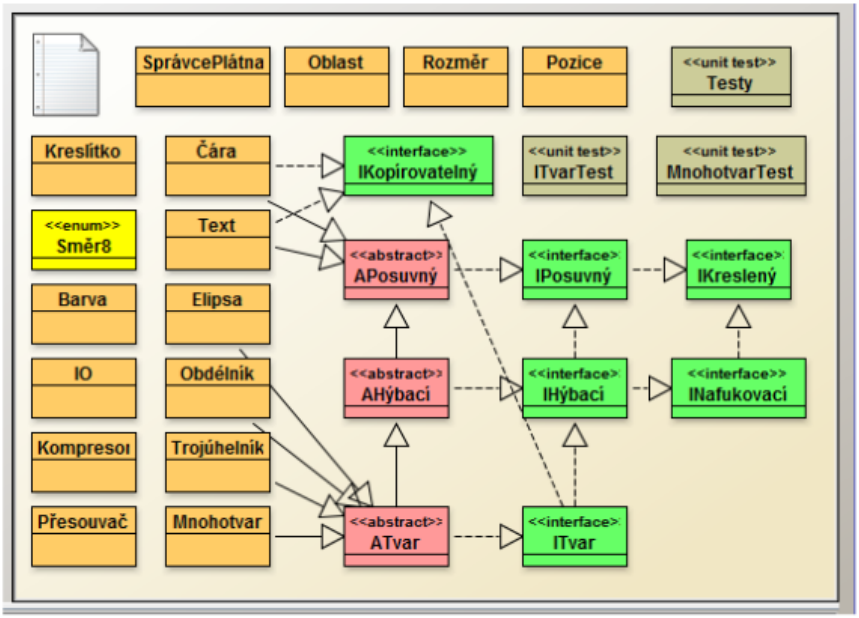
Třetí etapa

Student přechází na profesionální vývojové prostředí. V této etapě se dále prohlubují znalosti principů architektury a kódování. Je doporučeno studentům uvést návrhové vzory, včetně vhodného doplnění alternativních vzorů k danému problému. Na řadu se také dostává vysvětlení a předvedení dědičnosti. Stále je dbán zřetel na to, aby i tato problematika co nejvíce zůstávala v rovině architektury. [3]

Čtvrtá etapa

V poslední etapě se student seznámí se základními algoritmickými konstrukcemi (podmíněné příkazy, cykly a rekurze) a naučí se jejich používání ve svých programech. Postup zasvěcení studenta do problematiky programování je tedy téměř opačný od jiných, běžně vyučovaných metodikách. [3]

Příkladem projektu, se kterým se pracuje na konci první etapy, odpovídá program na obrázku níže.



Obrázek 12: Příklad projektu [3]

Autor udává, že studenti zpočátku nedůvěřují zvolené metodice, ale zpětně oceňují její přínosy a hodnotí ji kladně.

Problémem tohoto přístupu (Architecture-first) může být nutnost vyučovat ve výukovém prostředí, které je schopno generovat kód v optimální úrovni. Dále je zapotřebí, aby i pedagog této metodice naplno důvěřoval a dokázal aplikovat její postup.

5.2 Obecné metodiky výuky

V předchozí podkapitole 5.1 byla popsána jednotlivá programovací paradigmata. Nyní je vhodné probrat obecné formy výuky bez ohledu na zvolené paradigma.

Běžnou formou výuky programování je výklad učitele, kterým je zprostředkována probíraná teorie, včetně demonstrace dílčích úseků programovaného kódu. Tento kód ukazuje vzorovou syntaxi a aplikace. Výuka může pokračovat samostatnou prací studentů, zadáním skupinového projektu nebo formou domácího úkolu.

Moderní metody výuky dělají učební proces efektivnější. A jakému stylu výuky dávají přednost sami studenti? Žáci jsou rádi aktivní a odmítají pasivní metody. Osmdesát procent dotázaných žáků dalo přednost skupinové diskuzi. Je vidět, že žáci rádi hovoří mezi sebou. Údaje zpracoval v roce 1990 M. Hebditch na základě vlastního dotazníku pro žáky Gillinghamské školy (Dorset, jižní Anglie) ve věku 11–18 let. [27] Je potřeba využívat ale více než jedné metody. Díky rozmanitosti žáků je potřeba použít i více metod, jinak bude učitel nudit sebe i své žáky.

Firmy využívají skupinu různých inovativních metodik, které zefektivňují tvorbu kódu a celkově ulehčují práci zaměstnancům. Seznámení žáky s těmito metodikami by bylo přínosem pro zefektivnění výuky programování a smazání rozdílu mezi tím, co je učeno na školách a jaká je realita v praxi.

V následujících odstavcích jsou stručně zmíněny některé postupy použitelné při výuce.

5.2.1 PBL metoda – Problem based learning

Jedná se o formu vysokoškolské výuky, při které se student seznámí s novou látkou při řešení komplexního problému, který odpovídá realitě, což zvyšuje porozumění a motivaci. [6] Fáze výuky jsou rozděleny do tří samostatných částí:

- úvodní setkání (představení úkolu, identifikace problémů, brainstorming účastníků, návrh modelu popisující úkol, vytyčení studijních cílů);
- samostatné studium;

- uzavírající setkání (dvouhodinové setkání s prezentací výsledků a seznámení s navrženým řešením).

Pro úspěšnou aplikaci této metody je potřeba zkušeného pedagoga, který ovládá předmět v celé jeho šíři. Vystupuje zde jako moderátor a zapojuje studenty do aktivní diskuze.

Studie dokazují úspěšné nasazení PBL metody v úvodních kurzech programování na vysokých školách. Otázkou je, zda by tento přístup uspěl i na střední škole. Metoda je totiž vhodná v menších kolektivech vysoce motivovaných studentů. Klasický výklad je zde částečně nebo v plné míře nahrazen samostudiem.

5.2.2 Agilní metodiky

Agilní metodiky umožňují rychlý vývoj softwaru a zároveň dokáží reagovat na změnu požadavků v průběhu vývojového cyklu. Klade se důraz na fungující software místo perfektní dokumentace, spolu-práci se zákazníkem a reagování na změny místo dogmatického dodržování původního návrhu. Podle těchto metodik se správnost systému ověří jedině pomocí rychlého vývoje, předložení zákazníkovi a následných úprav dle zpětné vazby. Na začátku projektu se určí nejdelší možný čas a náklady na vývoj. Agilní přístup našel své uplatnění také v Business Intelligence a v marketingovém plánování. Proces vývoje je postavený na týmové spolupráci, otevřené komunikaci týmu, inovacím, zapojení zákazníka a otevřenosti změnám.

Tento přístup je zároveň vhodný i při výuce programování. Student při řešení programátorského úkolu pravidelně konzultuje zadání a řešení s vyučujícím. Ten zadání upravuje v průběhu projektu tak, aby odpovídalo jeho didaktickým záměrům, schopnostem studenta a potřebám směřování jeho dalšího rozvoje. Student pravidelně odevzdává část řešení, nikoliv hotové dílo. Následně opravuje nedostatky a s kódem opakovaně pracuje a vylepšuje ho, místo pouhého ohodnocení špatnou známkou, které nemá žádný didaktický efekt vyjma demotivace. Negativem tohoto přístupu je časová náročnost pro vyučující i studenty. Na vyučující jsou kladeny vysoké nároky a studenti musí být ochotni věnovat se programování intenzivně. [28]

5.2.3 Miniprojekty

Autoři v knize Reflections on the Teaching of Programming označují miniprojekty jako nejefektivnější alternativu klasického hodnocení. Skupině dvou až čtyř studentů je zadán úkol, vytvořit kompletní program včetně dokumentace. Zadaný úkol musí být náročný tak, aby jeho řešení netrvalo příliš dlouho, ale na druhou stranu komplexní tak, aby studenti prokázali odborné znalosti. [6] Studenti následně své řešení obhajují na ústní zkoušce. Hodnocení probíhá na základě těchto kritérií:

- konstruktivista (student má schopnost konstruktivně využívat možnosti jazyka);
- úplnost (řešení pokrývá kompletní rozsah zadání);
- individualita (prověřují se individuální znalosti konkrétního studenta);
- prezentace (student získá zpětnou vazbu na výsledky své práce).

Problém této metody je, že učitel musí mít připraveno dostatek témat zadání, aby předešel kopírování řešení z předchozích let. Zamezení podvodu se může předejít i cílenými otázkami při studentově prezentaci. [6]

5.2.4 Rozšiřování stávajícího kódu

Je metodika, která si získala oblibu při výuce objektově orientovaného programování. Student nemůže pochopit objektově orientované programování na příkladu, který obsahuje malé množství tříd. Na druhou stranu je nemožné, aby začínající student tvořil hned obsáhlou aplikaci. Vyučující studentům připraví množství ukázkových programů, do kterých studenti doplňují ty části kódu, kterým může porozumět na základě probrané látky. Toho je nám umožněno díky OOP metodě zapouzdření. Výhodou je příprava studentů na budoucí praxi, kdy v prostředí firmy je pravděpodobnější, že budoucí programátor bude pracovat na úpravě (a rozšíření) již existujícího kódu, než na tvorbě zcela nové aplikace. [6]

5.2.5 Skupinová výuka

Skupinová práce je jednou z organizačních forem výuky, která reaguje na požadavky dnešní doby na programátora a může se použít i při výuce programování. Hlavním přínosem této metody je rozvíjení komunikace a kooperace, členové se navzájem mohou obohacovat o získané zkušenosti, podněcuje vzájemnou pomoc, nabízí prostor pro uplatnění individuálních schopností studentů.

Rizika skupinové práce jsou převážně: časová náročnost, neschopnost komunikace mezi členy týmu, obtížnost hodnocení pro učitele, nevhodné složení skupiny.

Žáci vytvoří 3-4členné skupiny, které budou na řešení zadaného problému spolupracovat a učitel zde funguje pouze jako rádce, který pomáhá na vyžádání. Rozdělení do skupin může určit: náhoda, přátelství mezi žáky, zkušenosti žáků nebo zasedací pořádek. Po analýze problému a jeho vyřešení je nutné provést diskusi, k jakému výsledku skupina došla, jak se jim spolupracovalo atd. Pokud je ve hře prvek soutěživosti, motivovanost studentů je ještě větší.

Brainstorming

V rámci skupinové výuky je nezbytné zmínit také Brainstorming. Jedná se o skupinovou kreativní techniku, kdy cílem je generování co nejvíce nápadů na řešení daného problému. Žáci říkají všechny své nápady a žádná z myšlenek nesmí být kritizována. Díky uvolněné atmosféře přicházejí náměty nové, netradiční. Brainstorming vede žáky k rozvoji kreativity, podporuje tvořivé myšlení a může zaktivizovat a motivovat třídu k dalšímu učení.

Brainstorming je velmi zábavný, avšak výzkumy ukazují, že lidé mívají až dvakrát tolik nápadů, když pracují o samotě. [27]

5.2.6 Škola hrou

Již roku 1651 Jan Amos Komenský vytvořil učebnici Škola hrou. V minulém století bylo toto heslo opomíjeno a až v posledních letech se školství snaží jej opět do vzdělání zakomponovat. Hry mohou zapojovat žáky velmi intenzivně do výuky a přimět je získat k předmětu a k učiteli kladný vztah. Příklad takové hry si můžeme ukázat opět na předmětu programování.

Vytvořte počítačový program, který bude obsahovat X věcí a zároveň nebude delší než Y řádků.

Žáci mají takové hry velmi rádi, protože jim vytyčují jasně daný cíl. Další zábavnou formou může být hlavně pro žáky na základních školách programování v některém ze speciálních programů popsaných v kapitole 4.3.

Přínos využití her ve výuce vidím především v motivačním a aktivizujícím prvku.

5.2.7 Grafická znázornění tématu

Mezi grafická znázornění můžeme zařadit: myšlenkové mapy, vývojové (postupové) diagramy, Vennovy diagramy, maticové diagramy a jiné vizuální způsoby uspořádání předávaných in-formací. [27]

Myšlenkové mapy

Myšlenkové mapy jsou používány k učení, plánování nebo řešení složitých problémů. Můžeme je použít, když si chceme zanalyzovat náročný projekt. Jedná se o vysoce účinnou analytickou techniku, která podporuje paměť a kreativitu.

Zachycováním a znázorněním našich nápadů a myšlenek je vytvářena myšlenková mapa. Náš hlavní objekt pozornosti zachytíme uprostřed mapy. Ze středu mapy vedeme různé větve. Začátky větví jsou naše hlavní témata, která souvisejí s objektem ve středu mapy,

následně se rozvíjí na podrobnější motivy. Na každé větvi zaznamenáme klíčová slova, obrázky a ilustrace. Pro tvorbu stačí papír a tužka, nebo se může použít jeden z mnoha sofistikovaných softwarů.

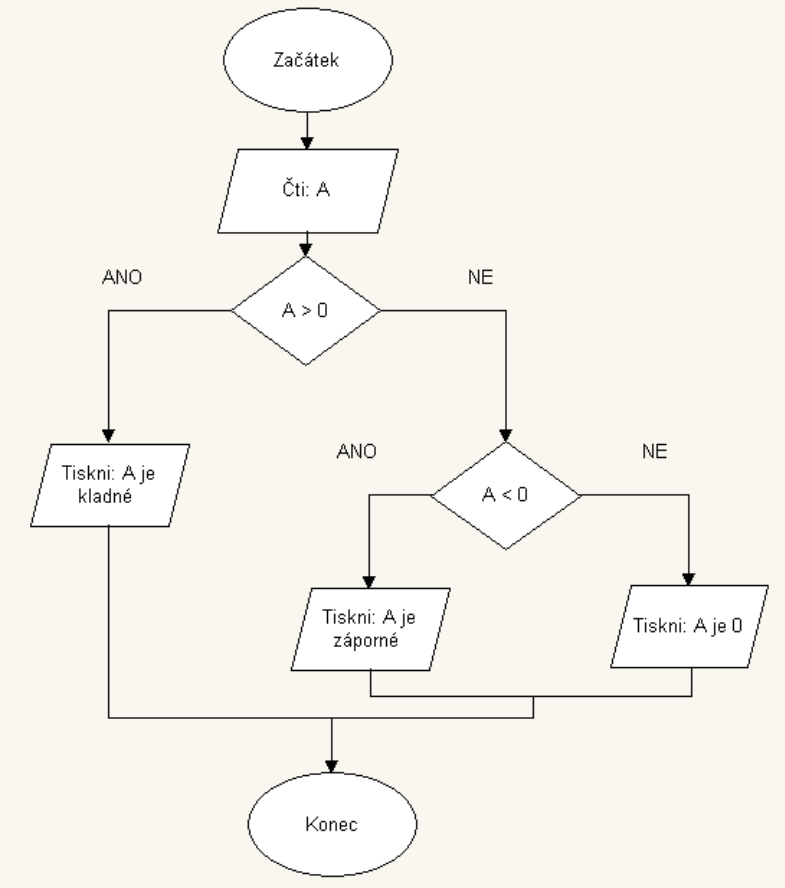
Obrázek 13: Myšlenková mapa [vlastní zpracování]

Jak využít myšlenkové mapy v programování? Pro odlišné oblasti programování se uplatnění myšlenkových map odlišuje.

Myšlenkové mapy můžeme použít místo ULM (grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů). Myšlenkové mapy mohou odhalit i slabá místa projektu, při fázi jeho testování. V algoritmizaci dosáhneme díky myšlenkovým mapám přehledného zobrazení jednotlivých příkladů, kdy snáze pochopíme jednotlivé vnořování cyklů, podmínek.

Vývojové diagramy

Vývojové diagramy se používají pro znázornění jednotlivých kroků algoritmů nebo obecného procesu. Jedná se o grafické ukázání logické struktury řešeného problému. Použití tohoto diagramu se všeobecně doporučuje u velkých projektů a při práci ve skupinách. Skládá se z grafických značek, které se kombinují, a tím simulují různé situace a příkazy. U začínajících programátorů může být problémem to, že si dostatečně nerozmyslí, co chtějí řešit a jak bude program pracovat. Tomu může zamezit právě vývojový diagram.



Obrázek 14: Ukázka vývojového diagramu [vlastní zpracování]

5.2.8 Multimediální výuka

Výuka probíhá pomocí e-learningového prostředí a multimediálních materiálu. Multimédia je doporučeno používat jako doplněk k učebnicím, nikoliv jako jediný prostředek výuky. Příkladem takovéto výuky můžou být vyučujícím připraveny nahrávky výkladu v délce 5 až 15 minut, na které navazují online testy a kvízy.

Tyto pomůcky mají za úkol upoutat žákovou pozornost zejména prostřednictvím zraku a sluchu. Používání některých vyjmenovaných pomůcek před pár lety bylo pro nás nemyslitelné. Dnes se využívání moderních technologií při výuce stává skutečností. Díky dotačnímu programu EU peníze školám, u kterého bylo jedním z cílů zlepšení využívání ICT, se můžeme na školách setkat například s interaktivní tabulí, která žáky aktivně zapojuje do výuky.

Mezi další moderní výukové pomůcky patří:

- interaktivní tabule;
- tablet, notebook;
- výukové pořady;

- flipchart;
- rozmnožované materiály;
- zpětný projektor.

Výhodou takových pomůcek je upoutání pozornosti, přinášejí změnu a zájem, jsou snáze zapamatovatelné. Použití těchto pomůcek je ale zároveň časově a finančně náročné. Časová náročnost přípravy těchto materiálů je mimo rozsah možností středoškolského učitele.

6 Trendy v programování

V této kapitole představím trendy v programování. Popisuji zde současnost a blízkou budoucnost programování. Jak jsem již zmínil v úvodu práce, odvětví IT je jedno z nejrychleji rostoucích průmyslových odvětví vůbec. Z tohoto důvodu je velmi obtížné zmapovat současné, natož budoucí trendy v této oblasti. Co platí dnes, zítra již nemusí. S tímto problémem se potýkáme i při výuce programování na školách, kdy školství na tyto rychlé změny není schopné dostatečně rychle reagovat. Chce to čas, investice a úsilí naučit se nová témata, nové jazyky nebo seznámit se s novými prostředky pro vývoj. A pokud budeme investovat čas do učení něčeho, po čem bude vysoká poptávka v budoucnu, můžeme na tom jako společnost velmi profitovat.

6.1 Programovací jazyk

Při výuce programování je důležitým bodem zvolit správný programovací jazyk. To, jak probíhá výuka na českých školách, jsme zjistili v předešlých kapitolách. Na Českém vysokém učení technickém začínají studenti výukou jazyka C, který je procedurální. Oproti tomu na Vysoké škole ekonomické v Praze začíná student v rámci oboru Aplikovaná informatika povinným předmětem 4IT101 Programování v Javě. Správné zvolení jazyka je důležité pro plánovanou náročnost kurzu, ale i kvůli tomu, že po různých programovacích jazycích je různá poptávka na trhu práce.

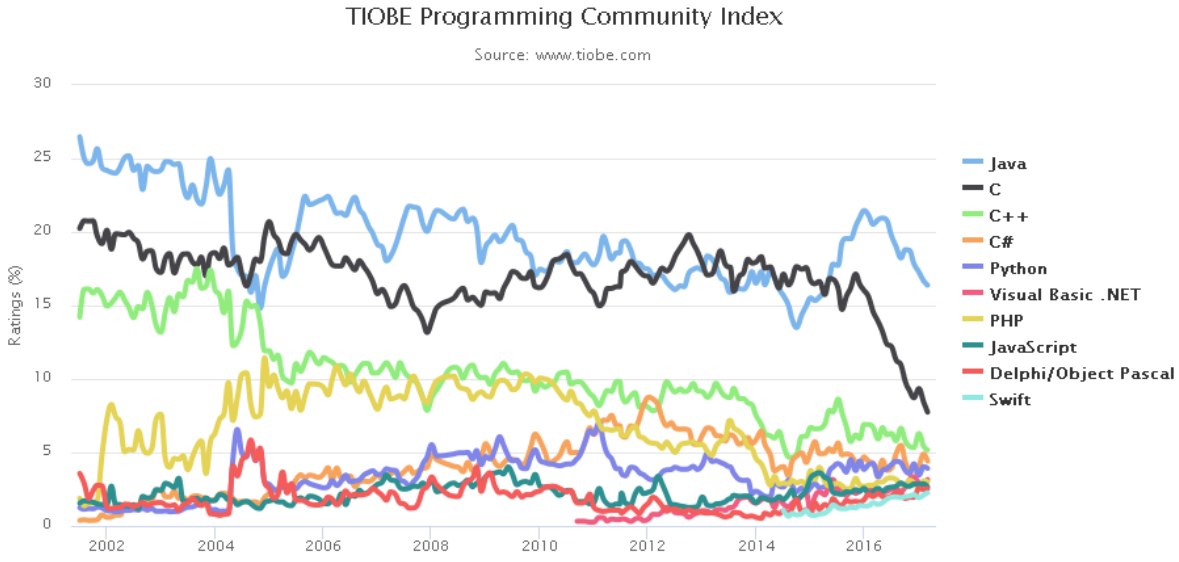
TIOBE index je ukazatelem popularity programovacích jazyků. Ratingy jsou založeny na počtu kvalifikovaných programátorů na celém světě, nabídky kurzů na výuku a dodavatelů třetích stran. Ke stanovení ratingu jsou použity vyhledávací enginy, jako je Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube a Baidu. Index je aktualizován každý měsíc a neříká nám nic o kvalitě daného programovacího jazyka, ale o jeho celkové popularitě. [29]

Dle ukazatele TIOBE indexu je pořadí prvních 10 programovacích jazyků následující:

Tabulka 3: Tiobe index, meziroční porovnání [29]

Únor 2017	Únor 2016	Jazyk	Hodnocení	Změna
1	1	Java	16.676 %	-4.47 %
2	2	C	8.445 %	-7.15 %
3	3	C ++	5.429 %	-1.48 %
4	4	C#	4.902 %	+0.50 %
5	5	Python	4.043 %	-0.14 %
6	6	PHP	3.072 %	+0.30 %
7	9	JavaScript	2.872 %	+0.67 %
8	7	Visual Basic .NET	2.824 %	+0.37 %
9	10	Delphi/Ojbect Pascal	2.479 %	+0.32 %
10	8	Perl	2.171 %	-0.08 %

Pro podrobnější analýzu použijeme graf, který ukazuje oblíbenost prvních deseti jazyků v období červen 2002 až březen 2017.



Obrázek 15: Tiobe index, dlouhodobý vývoj [29]

Na grafu během těchto let můžeme vidět klesající trend u C. I oblíbenost jazyku Java v minulém desetiletí klesala, až v posledních letech můžeme zpozorovat dramatický vzestup, který je následován dalším postupným klesáním. Vzhledem k rychlosti obměny trendů mu-

síme při předvídání budoucnosti vycházet z delšího období, než je meziroční srovnání udávané v tabulce č. 3. U nejoblíbenějších pěti programovacích jazyků pozorujeme dlouhodobé zastavení vývoje, či spíše sestupný trend s občasnými pozitivními výkyvy směrem vzhůru. V grafu i tabulce vidíme jen prvních 10 jazyků, avšak celkově sledovaných je 246 a jejich počet nadále roste. Je vidět, že trendem je rostoucí počet programovacích jazyků, což přirozeně vede k větší roztříštěnosti trhu. Na toto musí zareagovat i samotní programátoři a školství při tvorbě učebních plánů. Úspěšný programátor by měl aktuálně zvládat více než jeden programovací jazyk. Díky tomu získá nový pohled na řešení problému, a zvýší se tím i jeho šance na uplatnění na trhu práce. V následujících odstavcích prvních pět jazyků krátce představím.

Java

Jedničkou na seznamu je Java, objektově orientovaný jazyk, který oslavil již přes 22 let své existence. Java je velmi oblíbená díky její čitelnosti a jednoduchosti. Díky tomu můžeme pro tento jazyk najít široké použití od čipových karet, přes mobilní telefony, až po aplikace běžící na serveru. Tento jazyk je stále vyvíjen a na rok 2017 je naplánováno vydání Javy 9, která zahrnuje například projekt Jigsaw (systém modulů) či podporu pro reaktivní programování. [31] V grafu je vidět, že tento jazyk je dlouhodobě první (až na malé odchylky) a za poslední dva roky si znovu upevnil svoji pozici na trhu.

C

Programovací jazyk C byl vyvinut pro potřeby operačního systému Unix již počátkem 70. let 20. století. Výsledek TIOBE indexu C je překvapivý z hlediska stáří programovacího jazyka, ne však z hlediska jeho funkčnosti. Je používán zejména pro systémový software, jako jsou ovladače, operační systémy a podobně, ale je velmi oblíbený i pro aplikace. Díky tomu, že Unix je open source, je tedy jako platforma pro uživatele zdarma a díky tomu je stále více využívána. Na základě tohoto faktu můžeme předpovídat silnou pozici jazyku C i nadále.

C++

Jazyk C++ podporuje více programovacích stylů (paradigmat), tedy nejen objektově orientované programování, ale také procedurální programování a generické programování. C++ vznikl rozšířením dřívějšího C v roce 1983 v AT&T Bell Laboratories. [31] V začátcích jazyku C++ se jednalo o nadstavbu nad jazykem C a kód se před vlastním překladem převáděl do zdrojového kódu jazyka C. Nutno podotknout, že některé programy v jazyce C nešlo překládat překladači pro C++. Později vznikl samostatný překladač a jazyk se osamostatnil.

C#

Vysokoúrovňový objektově orientovaný programovací jazyk C# byl vyvinut v roce 2000, autorem je softwarový gigant Microsoft. [31] Jazyk C# je založen na jazycích C++ a Java. Hlavní využití můžeme vidět u databázových programů, webových aplikací, stránek nebo softwaru pro mobilní zařízení.

Python

Python vešel ve známost coby univerzální programovací jazyk, který vznikl v roce 1991. Jedná se o hybridní jazyk, podporující různá scriptovací paradigmata včetně objektově orientovaného, imperativního, procedurálního nebo funkcionálního. Volba závisí na zkušenostech a preferencích programátora. Python je open source projekt pro většinu běžných platforem (Unix, Windows, Mac OS). Jako významná vlastnost tohoto jazyka je uváděna jednoduchost z hlediska učení, a to díky tomu, že kód programu v Pythonu je krátký a vyznačuje se dobrou čitelností. Na základě této vlastnosti je hojně vyučován v úvodních kurzech programování na mnoha prestižních univerzitách. [5] Možnosti využití Pythonu jsou rozsáhlé. Ze známých produktů jsou to nástroje pro 3D modelování – Cinema4D, Maya; grafické a vektorové editory – GIMP, Inkscape, PaintShop Pro nebo společně s Javou je využíván Python v kancelářském balíku GoogleDocs.

6.2 Low-code development platforms

Vývojová platforma low-code development představuje typ technologie, která umožňuje vytváření obchodních aplikací zejména prostřednictvím nastavování parametrů, než samotným kódováním těchto funkcí díky automatickému generování kódu. Použitím low-code je možné například rychle změnit konfiguraci aplikace nebo přidávat nové prvky uživatelského rozhraní dle našich potřeb. Generace těchto nástrojů pro programování bez znalosti samotného programování je na IT scéně již nějakou dobu, k většímu rozšíření dochází až v poslední době. V roce 2016 vydal Google sadu App Maker, Microsoft představil nástroj PowerApps a Oracle ukázal nástroj Project Visual Code. Dle společnosti Forrester hodnota low-code trhu vzroste na 15,5 miliardy $ již v roce 2020. [32] Růst těchto aplikací lze připsat své pružnosti a snadnosti. Uživatel může v případě potřeby nebo zvláštního požadavku použít i vlastní kód. Vzhledem k minimálnímu množství požadavků znalosti kódování snižuje tato technologie nároky na programátorské znalosti, tím pádem i rozpočty finanční a časové na práci týmu. Kritici hodnotí tuto technologii prozatím jako nepoužitelnou pro vytváření rozsáhlých a důležitých podnikových aplikací.

Hlavní přednosti shrnuji takto:

- minimální ruční kódování;

- experimentování a testování nových možností;
- rychlost nasazení aplikace, rychlá tvorba prototypů;
- udržitelný kód aplikace;
- lepší čitelnost kódu, nemusíme dbát na správnou syntax = méně chyb;
- možnost zapojení více „nezkušených“ zaměstnanců do vývoje;
- snížení nákladů na vývoj.

Základní přehled Low-code produktů [33]:

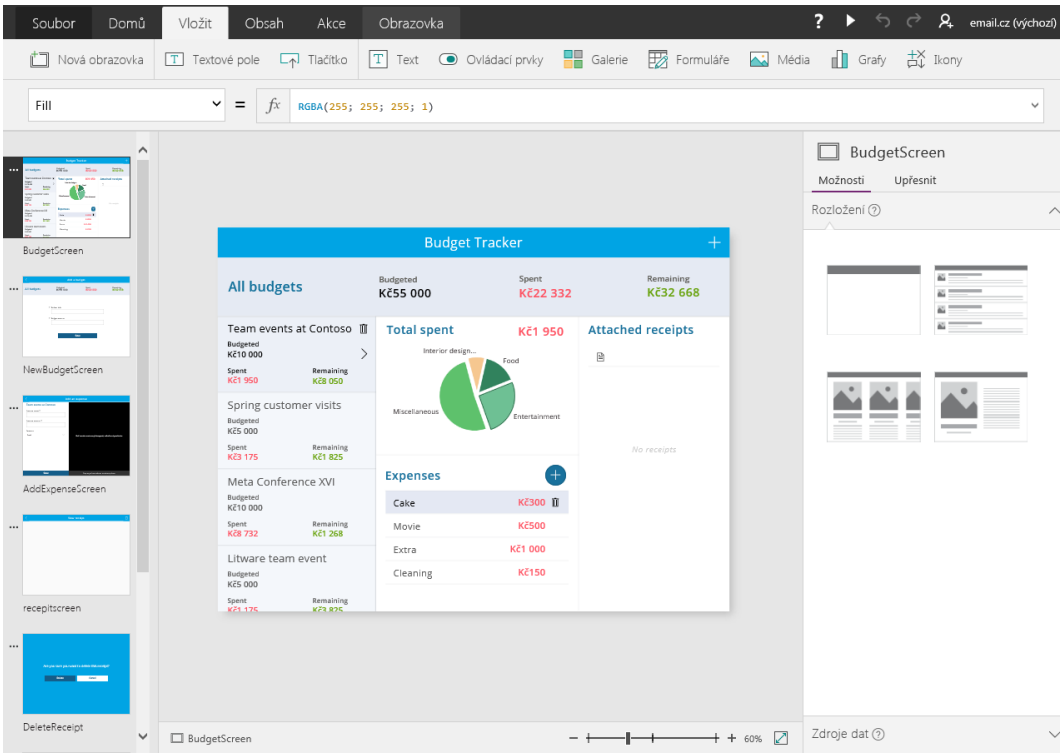
- Uniface (http://www.uniface.com/);
- Agile Point (http://agilepoint.com/);
- Appian (http://appian.com/);
- Bizagi;
- Caspio (https://caspio.com/);
- Microsoft PowerApps (https://powerapps.microsoft.com/);
- K2 (https://k2.com/);
- MatsSoft (https://matssoft.com/);
- Mendix (https://mendix.com/);
- MicroPact (https://micropact.com/);
- Nintex (https://nintex.com/);
- QuickBase (https://quickbase.com/);
- Salesforce (https://salesforce.com/eu/platform/);
- ServiceNow (https://servicenow.com/);

V následujících podkapitolách krátce představuji dva z produktů.

6.2.1 Microsoft PowerApps

Microsoft PowerApps nabízí vytváření podnikových aplikací bez znalosti programování. Nástroj podporuje tvorbu programů pro operační systémy iOS, Android a webové prohlížeče. Velmi oceňuji možnost použití webového rozhraní pro tvorbu aplikací.

Během prvního přihlášení do služby PowerApps uživatel může projít tutoriálem, jak na tvorbu jednoduchých aplikací. Pokud s vývojem aplikací teprve začíná, může začít sestavením jednoduché aplikace automaticky vygenerované ze zdroje dat a aplikaci si pouze přizpůsobit, aby vyhovovala řešenému problému. Program si můžete vytvořit i z již hotové šablony, která je založena na fiktivních datech. Po získání základních zkušeností může uživatel vyvíjet zcela vlastní aplikace. Může je připojit k různým zdrojům dat, přidat ovládací prvky uživatelského rozhraní a na závěr určit chování aplikace sestavením vzorců. Zdroje dat jsou podporovány místně i v cloudu, například: SharePoint, Dynamics 365, SQL server, Excel, OneDrive, Office 365 Users, Office 365 Outlook, Google Drive, Dropbox. List všech podporovaných připojení je mnohem obsáhlejší a jak je vidět, obsahuje i služby konkurenčních firem. Vytvořené aplikace můžete snadno a bezpečně sdílet v rámci své organizace. Další možností je získat mobilní aplikace PowerApps z Microsoft AppSource, ve kterém naleznete stovky již vyvinutých a vyzkoušených aplikací jiných zákazníků a partnerů Microsoftu. Na AppSource můžete vyhledávat aplikace podle různých kritérií, filtrovat podle používaných zdrojů dat, podle využívaných produktů nebo dle typu použití. Řešení PowerApps je vhodné pro podniky, které pro svůj provoz využívají databázový systém Access. PowerApps totiž značnou část funkcionalit přebírá, rozšiřuje je a umožňuje jejich přenesení na různá mobilní zařízení.



Obrázek 16: PowerApps prostředí [vlastní zpracování]

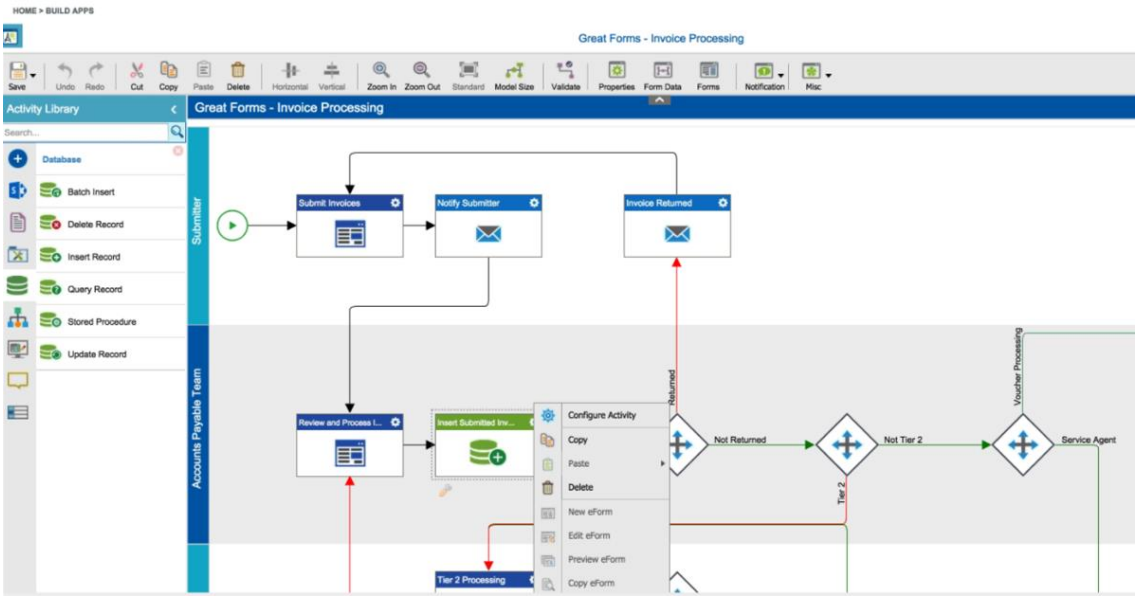
Cenová politika prostředí PowerApps je příjemným překvapením. Cena za používání je $7 (176,- Kč) za měsíc pro podnikové uživatele, a až $40 (1 004,- Kč) pro tvůrce aplikací a

administrátory za měsíc. Dále je možnost využití bezplatného programu s omezenými funkcemi. Rozdílné plány obsahují i rozdílné možnosti prostředí. Příkladem může být velikost datového uložiště, počet datových připojení a podpora.

6.2.2 AgilePoint NX

V začátcích AgilePoint začalo svou existenci jako nástroj pro implementaci systémů BPM (Business Process Management), v nejnovější verzi AgilePoint NX najdeme program pro tvorbu jednoduchých pracovních prostředí až po sofistikované business systémy, které se přizpůsobují řadě různých podmínek bez nutnosti znát kód pro vytváření aplikací. Vývojové prostředí uživatelům umožňuje snadnou konfiguraci aplikace z předem vytvořených komponent. Můžete vytvářet i aplikace, které zahrnují data z jiných aplikací a systémů, jako jsou SharePoint nebo Salesforce.

AgilePointNX můžete používat v soukromém cloudu nebo jako SaaS (Software as a Service - model pořízení softwaru, kdy dochází k hostování aplikace provozovatelem služby zákazníkům přes internet) nebo může být spuštěna v soukromém cloudu. Náklady na provoz se pohybují v rozmezí od $5 (128,- Kč) do $125 (3 168,- Kč) za licenci, v závislosti na počtu a typu žádostí o licence. [34]

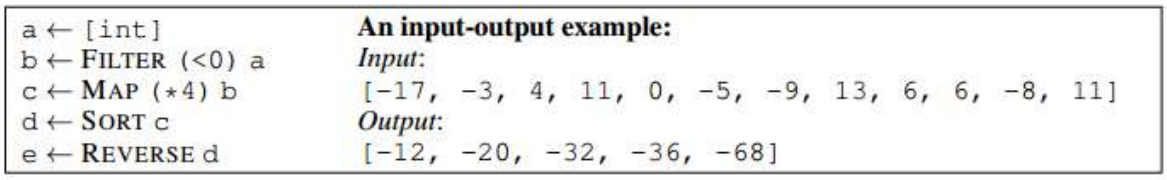


Obrázek 17: AgilePoint [34]

Dle mého názoru se bude low-code vývoji dostávat stále většího prostoru a samotný uživatel zde bude už jen pro snadnější nadefinování toho, co, jak a proč má daný program dělat. Tento krok beru já osobně jako největší trend a očekávám masivní rozšíření této technologie v následujících letech.

6.3 Umělá inteligence

Z Redmondské dílny společnosti Microsoft vychází i další zajímavý projekt zaměřený na programování. Přesněji ve spolupráci s Cambridgeskou univerzitou vyvinuli umělou inteligenci, která dokáže řešit problémy v programování opakováním řádků kódu z jiných programů. Projekt DeepCoder je v programování mnohem účinnější než člověk, jeho limitem učení je výpočetní výkon, jaký poskytuje počítač. [35] V porovnání s člověkem je programování umělé inteligence neporovnatelně rychlejší. Systém se také postupně učí a zlepšuje podle svých předešlých výběrů. To znamená, že brzy bychom mohli vytvářet programy, které najdou bugy v softwaru nebo objekty na fotografiích mnohem rychleji a programátoři nebudou muset ani hnout prstem. Prozatím zvládá neuronová síť lehké výpočetní úlohy, které reálně zabírají pět řádků kódu. Jedná se ale o velký pokrok, který stojí za zmínku.



Obrázek 18: Zadání výpočetní úlohy [35]

Neuronová síť dostala zadání:

- Vstup, sada čísel: -17, -3, 4, 11, 0, -5, -9, 13, 6, 6, -8, 11
- Výstup chceme: -12, -20, -32, -36, -68,

Neuronová síť pochopila, že nejprve odfiltruje kladná čísla, ostatní vynásobí čtyřmi a seřadí.

6.4 Preprocesor

Při zrodu nového programovacího jazyka museli programátoři zformulovat celý ekosystém pro transformaci kódu na bity zasílané čipům. Dnešní praxí je napsat preprocesor, který jej přeloží do vybraného staršího programovacího jazyka, který má již bohatou sadu knihoven rozhraní API. Díky tomuto trendu se zefektivňuje vývoj nového programovacího jazyka, za využití již stávajících. Příkladem může být CoffeeScript, který byl vytvořen pro JavaScript a Groovy, jednodušší verzi Javy. CoffeeScript pomáhá při kódování bez chybné interpunkce. Existuje několik jazyků, jako je např. Clojure nebo Scala, které běží na existující Java virtual machine (JVM) – ale existuje jen jeden stroj JVM. [36]

6.5 Framework

Jedná se o softwarovou strukturu, která slouží k podpoře při programování a vývoji. Může obsahovat knihovny API, podpůrné programy nebo doporučené postupy při vývoji programu. Příchod těchto struktur ulehčil programátorům práci. Ti totiž nemuseli již každou funkčnost dopisovat ručně, ale podívali se do knihovny příslušeného frameworku, tuto funkci našli a poté pouze implementovali. Využili tedy práce, kterou udělal někdo jiný. Aplikace může využívat buďto celý framework a nebo pouze konkrétní knihovnu pro práci jen s určitou skupinou funkcí. Ukázkou problémů, které můžeme pomocí frameworku řešit je: autorizace/autentizace, přístup do zabezpečené administrační sekce webu, validace formulářových vstupů, práce s daty, sessions/cookies a další. Příkladem dnešních frameworků je: jQuery, Kendo, Sencha, Netty a další. [36]

6.6 Mobilní webové aplikace

Mnoho programátorů při vývoji obsahu pro mobilní zařízení se rozhoduje, zda navrhovat responsivní webové stránky, které jsou funkční napříč vícero zařízeními a platformami, nebo se zaměřit výhradně na budování nativní mobilní aplikace. V poslední době se těší oblibě právě responsivní webové stránky. Při tvorbě aplikace je potřeba napsat samostatnou verzi pro iOS, Android, Windows 8 a BlackBerry OS pokud chceme cílit na co nejvíce uživatelů. Každá verze vyžaduje samostatný tým, který používá jiný programovací jazyk. Výhodou nativní mobilní aplikace je, že můžete využít všechny funkce smartphonů.

Druhým řešením je vytvoření jedné HTML aplikace a umístit ji na webové stránky. Multiplatformní mobilní web spustíte na jakémkoliv mobilním zařízení a zasáhnete podstatně větší cílovou skupinu než s jednou mobilní aplikací pro konkrétní platformu. Jedná se o cenově dostupnější řešení a tvorba takového webu trvá mnohem kratší dobu než udělat mobilní aplikaci. [37]

V dnešní době jsou responsivní webové stránky právě jedním z trendů a tento přístup může konkurovat nativním aplikacím. Konečné rozhodnutí, zda zvolit responsivní web či mobilní aplikaci, není ale vždy snadné; v konečném důsledku nejvíce záleží na konkrétním vytyčení cílů projektu. Další variantou je udělat obojí – jak mobilní aplikaci, tak také responsivní web.

6.7 Cloudové řešení

Jak roste odvětví IT, dlouhodobě se zvyšují nároky i na výpočetní techniku. Programátoři vyvíjejí komplexnější software, který zpracovává obrovské množství dat a kvůli tomu je potřeba stále většího výkonu. Avšak vývoj čipů narazil na svůj technologický strop a nesetkáváme se každý rok s dvojnásobným výkonem, jak tomu bylo v 80. a 90. letech.

V dnešní době se většina prací provádí v cloudu. Veškerá data a výpočetní výkon budeme mít předplacena na serveru a budeme k nim přistupovat za pomoci levného klienta, který zobrazí výsledek na našem monitoru.

Výzvou pro programátory je najít chytré způsoby, jak optimalizovat aplikaci tak, aby pronajaté prostředky (výpočetní výkon, úložné systémy, přenosové pásmo a další) využívala z hlediska nákladů co nejvíce efektivně. Cloudové firmy nám umožní zvládnout uspokojení mnoha uživatelských požadavků a jedná se o jedno ze směřování na softwarovém poli.

7 Rozhovory a dotazník

Na základě rozhovoru s vedoucím práce panem Pecinovským a dalšími odborníky jsem získal náhled na současný stav výuky programování a přehled trendů v programování.

Tvorba dotazníků probíhala ve spolupráci s vedoucím práce tak, aby pokryla cíle této bakalářské práce. Tento dotazník byl vytvořen za účelem zjištění, zda mají studenti a učitelé přehled o trendech v programování a co si o těchto trendech případně myslí; a zjištění analýzy současného stavu výuky programování v ČR. Dotazníková anketa probíhala dvěma formami, a to:

- elektronická forma – emailová komunikace s využitím nástroje Google Forms, byla využita při dotazování studentů;
- osobní rozhovory s učiteli předmětu Programování a vývoj aplikací.

Cíle rozhovorů a dotazníkové ankety můžeme shrnout takto:

- Zjistit, jaké jsou současné trendy ve vývoji aplikací a výuce programování.
- Zjistit, jaké jsou nedostatky ve výuce programování.
- Zjistit názor odborníků a studentů z praxe na výuku programování pomocí metodiky Architecture-first.
- Zjistit názor odborníků na low-code development platforms.
- Zjistit, jaká programovací paradigmata jsou vyučována na českých školách.

Získané odpovědi pocházejí zejména od učitelů a studentů těchto státních a soukromých škol:

- Vysoká škola ekonomická v Praze (VŠE);
- Česká zemědělská univerzita v Praze (ČZU);
- České vysoké učení technické v Praze (ČVUT);
- Smíchovská střední průmyslová škola (SSPŠ);
- Střední průmyslová škola elektrotechnická Ječná (SPŠE Ječná);
- Střední průmyslová škola elektrotechnická V Úžlabině (SPŠE V Úžlabině);
- Unicorn College (UC).

Většina dotazníku je vystavěna z otázek s otevřenou odpovědí. To znamená, že respondent nebyl omezen předem danou škálou odpovědí a mohl se volně vyjádřit k dané problematice.

Část otázek je formou otázky s výběrem z více možností, kde lze zvolit právě jednu odpověď. Veškeré odpovědi v rámci rozhovorů a dotazníků jsou čistě anonymní, s výsledky tedy nejsou spojována jména ani jiné identifikační údaje.

7.1 Rozhovory s odborníky – vyučujícími

U rozhovorů s učiteli bylo cílem zjistit, jak probíhá jejich výuka programování, jak motivují studenty, co si myslí o trendech v programování a jestli se tyto trendy v jejich výuce promítají.

7.1.1 Výběr výzkumného vzorku

Dotázaní učitelé byli zvoleni náhodně a podle dostupnosti, se snahou o co největší rozmanitost vzorku respondentů – vyučující předmětu programování na středních a vysokých školách. Rozhovorů proběhlo pět. Většina z dotázaných projevila zájem o toto téma z důvodu jejich domnění nutné změny vyučování programování na českých školách a aktivnější práce s trendy ve výuce.

7.1.2 Otázky výzkumného šetření – učitelé

Následující otázky byly použity během rozhovorů s učiteli:

Paradigmata

Jaká programovací paradigmata se vyučují u Vás na škole?

Co děláte pro to, aby si studenti daná programovací paradigmata opravdu osvojili?

Jaké trendy v programování v poslední době pociťujete, a jak se tyto trendy odráží ve vaší výuce?

Low-code platforms

Některé z agentur zabývajících se trendy ve vývoji programování očekávají v nejbližší době masivní nástup vývojových prostředí typu low-code development platforms. Jedná se o prostředí, která poskytují možnost automatického generování kódu, díky němuž umožňují vytváření aplikací spíše prostřednictvím nastavování konfigurace a specifikací požadovaných funkcí než samotným kódováním těchto funkcí. Co si o tomto trendu myslíte?

Za jak dlouho se podle vašeho názoru takovéto vývojové nástroje prosadí jako main stream?

Architecture First

Setkal(a) jste se s metodikou výuky Architecture First?

Většina dotázaných z řad studentů i odborníků, neznala metodiku Architecture First v souvislosti s výukou. Proto byla metodika krátce nastíněna, viz níže a dále byla účastníkům položena otázka, která z výukových metodik jim připadá vhodnější.

Metodika Architecture first staví na předpokladu, že tvorbu kódu vyvíjených programů převezmou zanedlouho počítače, a že je proto důležité, aby se studenti naučili především navrhovat architekturu programů. Podle pedagogické zásady ranního ptáčete, která tvrdí, že nejdůležitější témata se mají probírat jako první, anebo alespoň co nejdříve [8], *prosazuje začít výklad a procvičování návrhu architektury hned od počátku výuky s tím, že v prvních etapách přebírá zakódování navržených programů generátor kódu, který je součástí použitého vývojového prostředí. K výuce syntaxe a pravidel kódování se má podle této metodiky přistoupit až ve chvíli, kdy složitost navrhovaných programů přesáhne možnosti aktuálně používaného generátoru kódu.* [13]

Po zjištění, co je to Architecture first, je podle vás vhodnější tato metodika nebo je vhodnější tradiční přístup (začít u výuky kódování)? Pokuste se vysvětlit proč.

Odpovědi na otázky jsou u každého respondenta shrnuty do těchto částí:

- Programovací paradigmata
- Trendy v programování
- Architecture first

7.1.3 Vybrané rozhovory

Na základě získaných odpovědí jsem vybral tři rozhovory, které se zdály být nejzajímavější a k tématu. Zbylé dva rozhovory kopírovaly u většiny otázek názory svých kolegů.

Odpovědi respondentů ponechávám v původním znění bez jakýchkoliv úprav.

UČITEL 1

Programovací paradigmata

„U nás na škole se vyučuje programovací jazyk C# celé 4 roky. V prvním ročníku začínáme logikou programu a vytvářením vývojových diagramů. Dále se programy píší převážně strukturovaně (i když si pro začátek pomáháme příkazem goto). Teprve na konci třetího čtvrtletí se studenti učí vytváření metod a postupně se přechází na programování objektové. Od druhého ročníku se již píší programy objektově a studenti vytváří vlastní třídy. Využívají dědičnost a polymorfismus.

Studenti po probrání teorie pokračují na zadaném úkolu a po uběhnutí časového limitu jsou ohodnoceni na základě toho, kam daleko projekt posunuli.“

Trendy v programování

„Jedna věc je sledovat trendy a jiná je jejich složité zavádění do výuky. V současné době je největší problém přinutit studenty, aby přemýšleli. Jsou zvyklí informace přijímat, v lepším případě si část z nich zapamatovat. Ale pochopit princip a ten pak aplikovat je pro mnohé problém.

Někteří studenti si přejí jiné programovací jazyky, dle „aktuálních trendů“. Tedy podle toho, že si někde přečetli jeden článek a mají pocit, že výuka C# je zastaralá. Mimochodem tento programovací jazyk nám byl doporučen při spolupráci s Unicorn College, když vedení školy (na popud studentů) uznalo, že VisualBasic je zastaralý.

V programování na střední škole je dle mého důležité pochopit principy a ty aplikovat. A je skoro jedno, jaký je používán jazyk. Malý kluk se také nejdříve musí naučit jezdit na kole a pak teprve začít závodit. My studentům ukážeme základ a už je na ni nich se současným trendům přizpůsobovat.

***Low-code development** mi připadá jako hezká pomůcka, ale pokud člověk kódu nerozumí, může se dostat do problému.*

Příklad: Ve čtvrtém ročníku učím PHP. Když studenti přijdou, nejdříve musíme zopakovat HTML. Pokud nerozumí struktuře, jak je kód zapsaný, nelze do ní zapisovat další scripty.

Jako další příklad bych uvedla osobní zkušenost: Školní web má redakčním systém. I když je dobře propracovaný, občas dojde k situaci, že stránka nevypadá, jak bych si představovala. V takovém případě se přepnu do režimu kódu a upravím si jej podle sebe. Pokud bych mu nerozuměla, nebylo by to možné. Byla bych odkázána pouze na to, co „pro mě vymyslel někdo jiný“.

Nástup tohoto trendu očekávám bohužel brzy. Je pohodlnější klikat v menu, než přemýšlet. A přesně to vede k degradaci myšlení. Což je bohužel někdy vidět na našich studentech.“

Architecture First

„Tuto metodiku neznám, ale dle mého názoru není pro výuku na středních školách popřípadě v úvodních kurzech programování vhodná. V začátcích je dle mého názoru výklad architektury pro studenty příliš složité téma.“

UČITEL 2

Programovací paradigmata

„V úvodním kurzu programování u nás na škole využíváme Algorithms-first a Imperative-first.Student dané paradigma skutečně pochopí až tehdy, když v duchu daného paradigmatu napíše několik programů. Po teoretickém úvodu studenti pracují samostatně na individuálních projektech za mé pomoci. Zkoušeli jsme na škole zavést tvorbu týmových projektů, ale z důvodu příliš nízké hodinové dotace výuky jsme od tohoto projektu časem upustili. Tyto projekty byly časově náročnější na přípravu.“

Trendy v programování

„Z oblasti programování se zajímám o budoucnost samotného programování, kdy příkladem může být umělá inteligence, která sama skládá jednoduché programy tím, že „vykrádá“ kód z již vyvinutých programů. Na tyto trendy a zajímavosti není při výuce čas, na to máme bohužel málo vyučovacích hodin. Na škole se zaměřujeme na to, abychom podchytili potřeby budoucích zaměstnavatelů. V tomto směru máme navázanou spolupráci i s univerzitou ČVUT. Od studentů mám pozitivní i negativní zpětnou vazbu na to, jak jsme je připravili na svá budoucí povolání. Dle několika názorů příliš podceňujeme rozvoj měkkých dovedností. Bohužel nevím, jak toto zrovna na hodinách programování změnit.
Vámi popisované ***Low-Code development*** *prostředí jsou dle mého názoru další budoucností z části. Dnes existují stále dokonalejší generátory kódu, ale nikdy se nebude dle mého názoru generovat kód všechen. Z tohoto důvodu by měl být student schopen v případě jeho potřeby kód napsat sám, bez použití generátoru. Může nastat i situace, kdy bude student potřebovat implementovat něco nestandardního, a není vhodné kvůli této mimořádné události vyvíjet, rozvíjet generátor.“*

Architecture First

„Existují různé typy studentů. Jednomu studentovi metodika Architecture First vyhovovat může, protože pochopí problematiku komplexně, ale druhému vůbec ne. Já osobně tento přístup neznám.

Na této metodice se mi líbí, že pokud naučíme studenta navrhnout architekturu, může tuto znalost využít na všechny již známé programovací jazyky a bude to umět i s těmi, které se na trhu teprve objeví. Pokud studenta pouze naučíme bezhlavě kódovat v jednom jazyku, tak je to dle mého ztráta času. V dnešní době jak říkáte, je samotného kódování méně než dříve.

Pokud si nejsem jistý syntaxí, stačí napsat dotaz do Googlu a máte vše vyřešeno. Naopak pokud nevíme jak na architekturu, tak s tím nám nikdo tak rychle nepomůže.“

UČITEL 3

Programovací paradigmata

„Výuku studentů začínáme na Algorithms-first a poté se pokračuje Object-first. Problémem je, že někteří studenti se nedokáží „přeorientovat“ z algoritmického na objektové myšlení. Na hodinách se nestíhá probrat vše, co bych chtěla. Studenti dostávají malé i komplexní domácí úkoly, tak aby si probíranou látku osvojili. Na základě výsledku v hodinách a domácí přípravě jsou dále hodnoceni.“

Trendy v programování

*„Nemyslím si, že školství je tady od toho studenty seznamovat se všemožnými trendy a novinkami v tomto rychle rozvíjejícím se odvětví. Studentům dáváme nutný základ a je na nich, jak budou dále pokračovat. Spíše než samotné nové technologie, studentům ukazujeme cestu k programování. Vámi zmiňovaná platforma **Low-code development** je zajímavou budoucností, ale až za několik příštích let. Stále si myslím, že student by měl znát, jak kódování funguje, aby porozuměl řešenému problému. Word vám taky poradí, kde máte chybu, ale základ psaní musíte umět sám.“*

Architecture First

„Já osobně jsem principy programování pochopila na metodice Imperative First a o této metodice slyším od vás prvně. Tento přístup se mi líbí, ale za předpokladu, že studenti mají již získané dostatečné vědomosti. Dle mého názoru je nepředstavitelné s tímto přístupem začínat již na střední škole. Ve výuce je potřeba se přizpůsobit většině a volit takovou metodiku, která bude vyhovovat větší části studentů a nebude hrozit nepochopení probírané látky. Pro určitou skupinu studentů, kteří chtějí být v budoucnu vývojáři a jsou do informatiky jako celku velmi zapálení, bych tento přístup doporučila.“

7.2 Dotazníkové šetření – studenti

Zhodnocení, nakolik současné metodiky výuky vyhovují studentům, a zda je využíváno některých trendů ve výuce programování, jsou otázky, jejichž zodpovězení náleží studentům oboru informatiky, kteří prošli předměty programování a mohou mít za sebou získanou již nějakou praxi.

7.2.1 Výběr výzkumného vzorku

Pro dotazníkové šetření mezi studenty bylo využito osobních rozhovorů s dotázanými a pro větší rozmanitost odpovědí a získání většího vzorku respondentů jsem dále využil online nástroje Google Forms. Na dotazník odpovědělo 100 respondentů, kteří mají středoškolské nebo vysokoškolské vzdělání v oblasti IT.

7.2.2 Otázky výzkumného šetření – studenti

Výsledky dotazníku jsou rozděleny do logických skupin dotazování v rámci souvisejících témat. Následné zjištěné výsledky jsou vyhodnoceny, graficky zpracovány nebo slovně okomentovány. Získané hodnoty jsou uváděny v celých číslech (počet respondentů, kteří zvolili tuto odpověď). Následující otázky byly použity během dotazníkového šetření se studenty.

Jaké je vaše nejvyšší dosažené vzdělání?

Programovací jazyky

S výukou jakých programovacích jazyků jste se během vašeho studia setkali?

Bylo něco, co vás při výuce programování zaujalo?

Do jaké míry vás v začátcích studia programování odváděl „zápas se syntaxí“ od koncentrace na návrh výsledného řešení?

Zaměření ve výuce

Na co by se podle vašeho názoru měla výuka programování více zaměřit? Jsou nějaké trendy, které by se měly do výuky promítnout?

Zabýváte se programováním v komerční sféře i po skončení studií?

Jak vás překvapila praxe jako programátora začátečníka, oproti tomu, co jste se naučil ve škole a co vás nejvíce zaujalo?

Návrh architektury

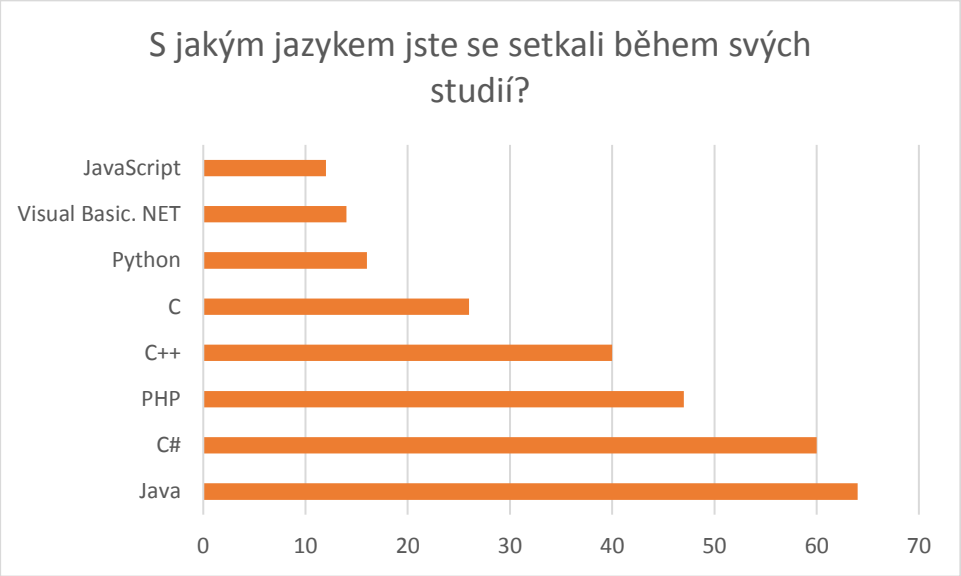
Setkali jste se při výuce s výukovou metodikou Architecture first?

Po zjištění, co je to Architecture first (definici naleznete v kapitole 7.1.2)*, je podle vás vhodnější tato metodika nebo je vhodnější tradiční přístup (začít u výuky kódování)? Pokuste se vysvětlit proč.*

7.2.3 Programovací jazyky

Tato kategorie zahrnuje výsledky otázek zaměřených na výuku programovacích jazyků (otázky č. 2, 3 a 4). Na otázku č. 1 odpovědělo 29 respondentů, že získali vysokoškolský diplom, 41 respondentů ukončilo své studium maturitní zkouškou a 30 respondentů je stále studentem. Ve vzorku dotázaných nikdo nezvolil odpověď pouze základní školy.

Na obrázku č. 19 můžete vidět respondenty nejčastěji (osm nejčastěji objevených odpovědí) uvedené programovací jazyky, se kterými se setkali během svých studií. Zajímavostí je, že několik ze studentů uvedlo, že se během studií naučili až 9 programovacích jazyků (všechny tyto odpovědi jsou zaznamenány v grafu).



Obrázek 19: Graf nejčastěji vyučovaných programovacích jazyků [vlastní zpracování]

Nejčastěji se v anketě objevoval jazyk Java (64 dotázaných). Tento výsledek pro mě není překvapením. Programovací jazyk Java je v současné době jedním z nejpoužívanějších jazyků i dle TIOBE indexu (viz kapitola 6.1). Také se tento jazyk často ve školách učí jako první programovací jazyk. Jinak tomu není ani na Vysoké škole ekonomické. Na dalších místech můžeme vidět rozdíly mezi výsledkem dotazníkového šetření a seznamem nejčastěji používaných programovacích jazyků uvedeným v kapitole 6.1. Tento rozdíl může být způsobený odchylkou způsobenou menším počtem respondentů. Zajímavým faktem je, že všech osm nejčastěji zodpovězených programovacích jazyků je v seznamu top deseti nejpoužívanějších programovacích jazyků ve světě.

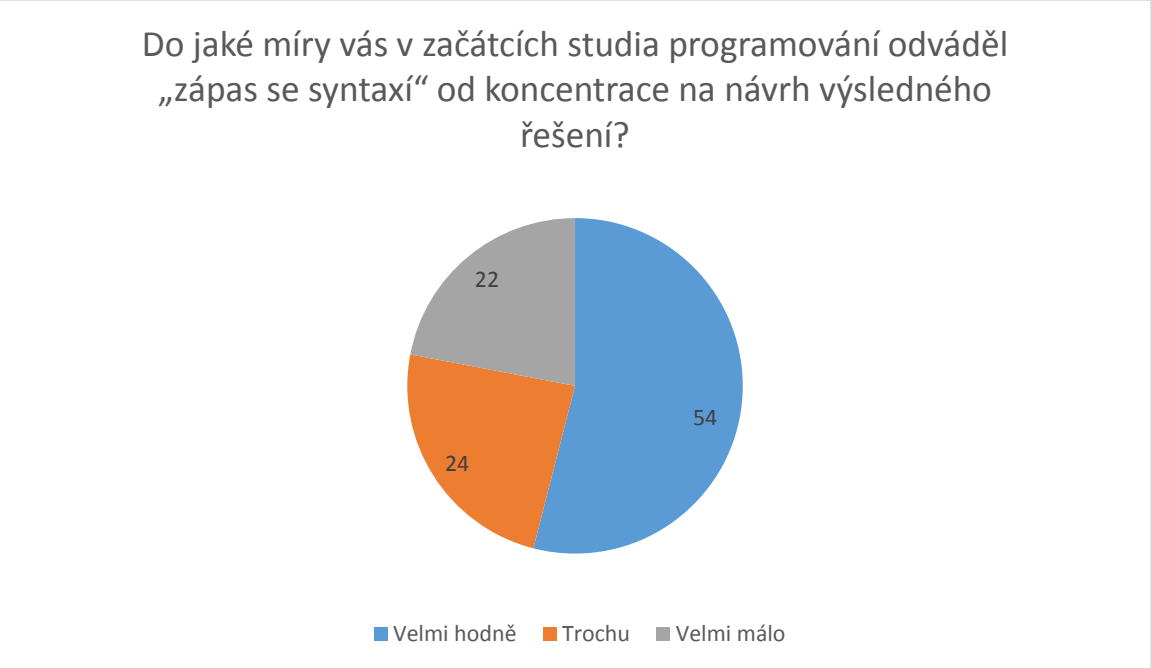
Pozitivním trendem je, že na prvních dvou místech můžeme vidět jazyky podporující OOP. Školy by měly dbát na zahrnutí objektově orientovaného přístupu ve výuce, aby studenti naplňovali očekávání pracovního trhu.

V odpovědích se objevilo i několik méně obvyklých jazyků jako je: Swift, Smalltalk, F# a další. Je vidět, že studenti mají během svého studia na některých školách rozsáhlý výběr, jakým směrem se budou vyvíjet.

Další otázky této kategorie byly určeny k obsáhlejší odpovědi. Na otázku, co studenty zaujalo při výuce programování, většina z dotázaných odpověděla, že hodiny byly koncipovány nezajímavým způsobem a většina času byla strávena nad syntaxí. V několika odpovědích bylo zmíněno objektově orientované programování jako nová, nikoliv nejrozšířenější metodika výuky.

Na studenty vysokých škol působilo negativně, že výuka je zaměřena na přílišnou specializaci v jednom programovacím jazyce. Veškerá úskalí jim jsou zde vysvětlena na příkladech platných právě v tomto jazyce a v dalších už ne. Nezískávají tedy komplexnější znalosti z této problematiky. Respondenti byli seznámeni se skutečností, že komplexnost vzdělání by byla na úkor specializovaných znalostí v jednotlivých jazycích. I přesto studenti a HR specialisté preferují nižší znalost více programovacích jazyků, než výbornou znalost pouze jednoho z nich.

Vzhledem k častým odpovědím ohledně syntaxe byla další otázka nasnadě, jak moc studenty v začátcích studia programování odváděl „zápas se syntaxí“ od koncentrace na návrh výsledného řešení?



Obrázek 20: Graf názoru účastníků na vyučování programování [vlastní zpracování]

Nadpoloviční většina studentů zodpověděla, že zaměření na syntaxi odvádělo jejich pozornost od návrhu výsledného řešení.

Jedním příkladem studentovy odpovědi je tato: „*Velmi, při výuce se bral převážně ohled na syntaxi. Vhodné řešení bylo až na druhém místě. Byla teda tvořena řešení, která byla krkolomná a neoptimální.*“. Další odpověď potvrzuje slova předešlé: „*Pamatuji si na svoji první hodinu programování, kdy hned na začátku jsme byli zahlceni různými pravidly programovacího jazyka. Ještě teď si nejsem 100 %.“.*

Dle mého názoru není správným směrem studenty učit hlavně kódování. Pokud naučíme studenta navrhovat architekturu, může tuto znalost aplikovat i na programovací jazyky, které teprve přijdou. Oproti tomu při učení kódování bude umět kódovat právě v naučeném jazyce a možná i špatně, protože mu nevysvětlíme architekturu. V dnešní době máme zároveň spoustu možností různých frameworků a samotného kódování je mnohem méně než dříve.

7.2.4 Zaměření ve výuce

Otázky č. 5, 6 a 7 se zabývaly trendy v programování a také problematikou, jestli je realita odlišná od vyučování na školách po příchodu do zaměstnání, tedy komerční programátorské sféry.

První otázka řešila, na co by se mělo při výuce programování v budoucnu zaměřit, popřípadě jestli studenti pociťují trendy v programování, které by se mohly ve výuce promítnout.

Na první část otázky bylo nejčastěji zodpovězeno, že výuka by se měla přiblížit co nejvíce praxi, na což navazuje otázka č. 6.

„*Školy by se měly více zaměřit na objektově orientované programování. Jazyk Java nám zařadili do výuky až po nesčetných prosbách studentů. Ze strany zaměstnavatelů je to velice oblíbený a hodnocený jazyk*“

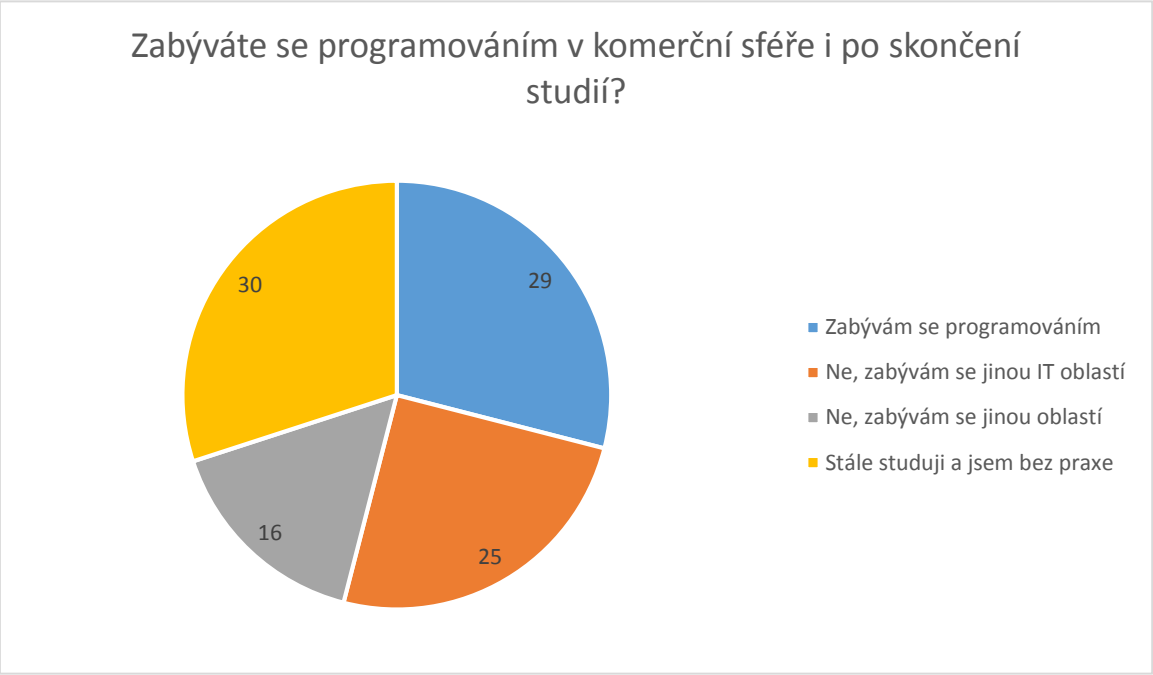
Další častou odpovědí bylo, že v kurzech pro začátečníky jsou opomenuty základy, například jak funguje paměť, procesor, komunikace mezi CPU a pamětí, bezpečnost sítí a jiné.

Několik z odpovědí předpovídá větší pozornost generátorům kódu, tak jak uvádím i já v této bakalářské práci.
„Celkově by se výuka programování měla více zaměřit na přípravu řešení a jeho optimalizaci, to především z důvodu rozvoje nástrojů, pro psaní programů, které budou čím dál více pomáhat se samotným psaním a syntaxí.“

Další z respondentů by ocenil interaktivnější metodu výuky.
„Výuka by se měla více zaměřit na praktické příklady a podrobné návody sloužící k samostudiu. Když je skupina studentů vědomostně nevyrovnaná a je někdo, kdo už všechno ví, tak nám „nováčkům“ nezbývá nic jiného, než samostudium. Pro mě osobně je výuka ve škole velmi rychlá.“

Dotázaní měli základní přehled o současných a budoucích trendech v programování. 39 dotázaných respondentů vidí budoucnost v umělé inteligenci a 11 respondentů uvedlo jako trend této doby rozšíření nástrojů na způsobu low-code development platforms, o těchto trendech píši v kapitole č. 5. Alarmujících 50 dotázaných nemělo o trendech představu.

Druhou otázkou této kategorie byl dotaz na skutečnou praxi v komerční sféře. Abych získal přesný pohled na realitu, nejdříve jsem vyselektoval respondenty, zda se ve sféře programování stále pohybují i po svých studiích.



Obrázek 21: Graf respondentů o setrvání v IT sféře [vlastní zpracování]

Jak je vidět z výše uvedeného grafu, v otázce praxe pokračovalo pouze 29 respondentů. Mezi těmito dotazovanými většina odpověděla, že výuka na školách je nepřipravila na realitu běžné programátorské práce.

Obrázek 22: Graf respondentů na téma praxe [vlastní zpracování]

Velké množství absolventů praxi v oboru během svých studií vůbec nezíská a většina respondentů se shodla, že realita programátorské práce je odlišná. Jedna ze zkušeností HR pracovníka, který má na starost nábor zaměstnanců do IT oddělení v nadnárodní společnosti:

„Pracuji jako HR zaměstnanec ve středně velké IT firmě. Při náboru na jednu z programátorských pozic se zaměřujeme spíše na soubor soft skills kandidátů a jejich relevantní praxi. Oboje dnešním absolventům chybí.“.

Další názor podporuje již řečené:

„Realita v běžném životě je bohužel jiná, než mě naučila škola. Studia jsem zanechal po studiu bakalářském a myslím si, že studijní plány by měly být upraveny tak, aby student mohl získat plnohodnotné vzdělání bez nutnosti pokračovat na magisterský obor studia. Ne totiž každý má takovou možnost. Dále bych zavedl povinnou praxi, se kterou jsem se nesetkal na střední ani vysoké škole. Sama naše společnost tyto praxe pro studenty nabízí. Nově příchozí zaměstnance musíme učit práci v týmu a základním komunikačním dovednostem. Jsou to věci, které jsou při studiu velmi podceňovány a školy se někdy až moc zaměřují na technickou stránku.“.

Někteří respondenti byli opačného názoru a praxe je nijak významně nepřekvapila.

„Ze školy jsem měl jen základní znalosti s jazykem Java a měl jsem pracovat s PHP. Firma si mě sama vyškolila k obrazu svému. Teoretické znalosti ze školy ale fungovaly a brzo jsem se mohl plnohodnotně zapojit do týmového projektu.“

Obdobně se vyjádřil jiný respondent:

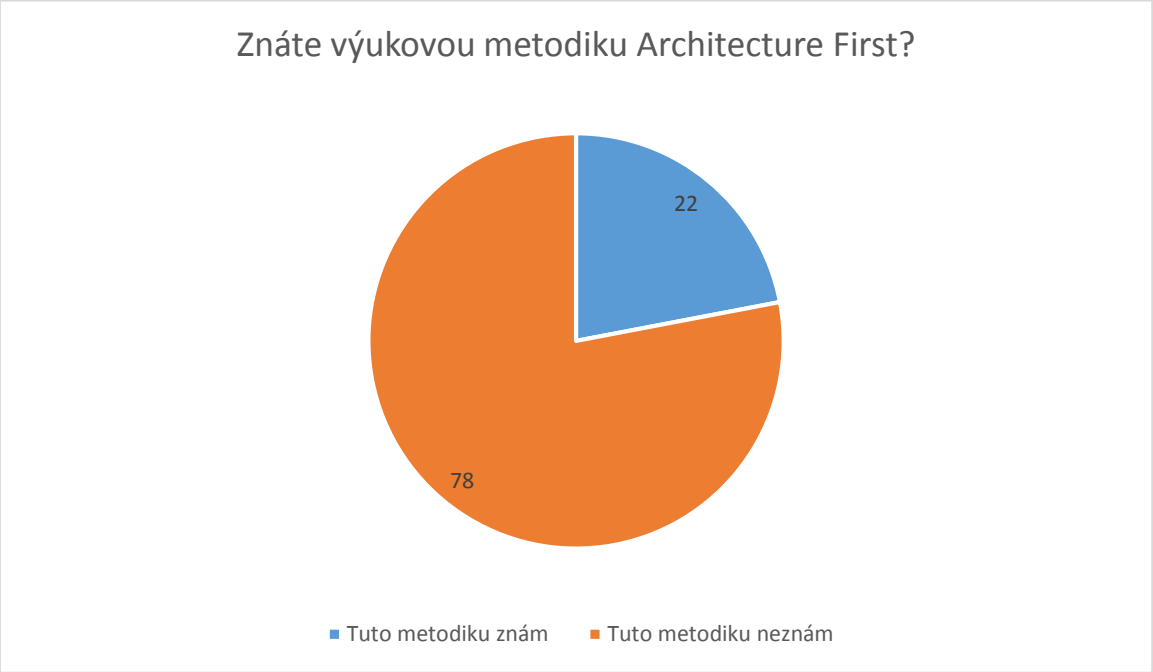
„Myslím si, že mě v základu škola naučila, co měla. Jak zúročíme nabrané zkušenosti je již na nás samotných. Ve firmě mě zaujala rychlost vývoje. Ve škole jsme mohli vše přepracovat. Tady se pro změnu velmi dbá na produktivitu práce.“.

Jako doporučení absolventi zejména uváděli, že školy by měly studenty více nabádat ke stážím ve firmách nebo pracím na firemních projektech, např. v rámci maturitního projektu.

Z mého pohledu je alibistické vinit samotné školy, že nepřipravují studenty na budoucí povolání tak, jak by studenti (popř. zaměstnavatelé) chtěli. Každý z nás má možnost zvolit si tu školu, která mu nejvíce vyhovuje. Studenti by si měli hledat z vlastní vůle i různé placené nebo neplacené stáže sami a zde strávený čas vnímat jako investici do budoucna.

7.2.5 Návrh architektury

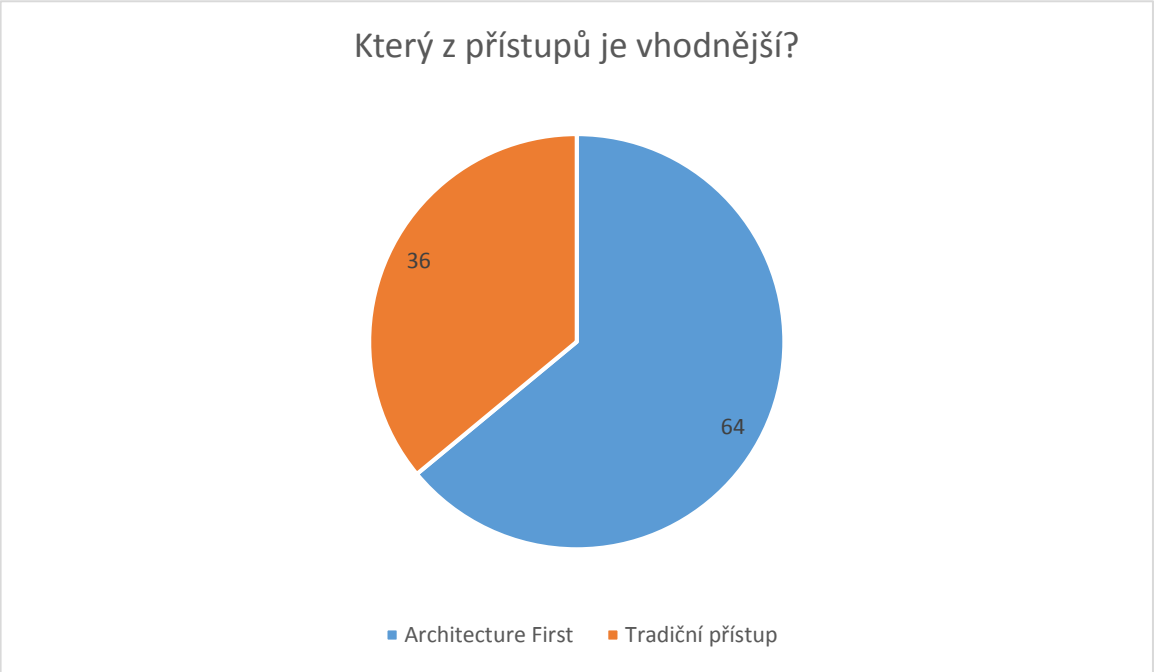
Poslední dvě dotazníkové otázky (otázky č. 8 a 9) směřovaly k samotné metodice výuky. V první otázce jsou studenti dotazováni, zda znají koncept Architecture First v souvislosti s výukou programování.



Obrázek 23: Graf povědomí o metodice Architecture First [vlastní zpracování]

Ti, kteří metodiku znali, studovali předmět 4IT101 Programování v Javě na Vysoké škole ekonomické v Praze. Většina respondentů metodiku Architecture First v souvislosti výukou programování ovšem nezná. Dotyčným dále bylo vysvětleno, že jde o to učit studenty nejdříve navrhovat architekturu a až poté se zaměřit na kódování.

Na základě tohoto výkladu jim byla položena otázka další, zda jim přijde vhodnější tato metodika, nebo je vhodnější tradiční přístup (začít u výuky kódování). Ze strany studentů u této otázky nedocházelo ke shodě. Respondent většinou obhajoval metodiku, kterou si během svého studia zažil. Přístup se setkával s příznivci i odpůrci.



Obrázek 24: Graf programovacích přístupů [vlastní zpracování]

Příklady detailních studentských odpovědí:

„Myslím si, že je lepší tradiční přístup. Z počátku, kdy člověk nemá sebemenší představu o programování, je složité myslet na úrovni architektury. Naopak pokud má již za sebou něco nakódováno, může se již odrážet od základní syntaxe, se kterou se seznámil a spojovat si vše dohromady.“

„Metodika Architecture first, je z mého pohledu vhodnější. Je to velmi podobné studiu jazyků. Zde je také důležitější umět gramatiku a slovíčka jsou až na druhém místě. Celkově by se měla výuka snažit studentům vštěpit představu, že vše by mělo vycházet z dobře navrhnutého konceptu. Pokud by programátor neměl dobře sestavenou architekturu programu tak i s výbornou znalostí syntaxe nevytvoří tak kvalitní program jako programátor, který bude mít výborně navrhnutý program.“

„Po zjištění detailů o metodice Architecture first bych upřednostnil tuto metodiku oproti tradičnímu přístupu z důvodu, že bude následné navrhování programů snazší pro pochopení i pro zaměstnance, kteří se programováním nezabývají popřípadě pro ty, kteří nemají hluboké znalosti programování.“.

Následuje souhrn argumentů pro a proti metodice Architecture-first, získaných z detailních odpovědí studentů:

Argumenty pro Architecture-first:

- student se naučí komplexně navrhovat architekturu, a je tak připraven na možnou budoucnost generátorů složitých kódů;

- při tvorbě aplikací je především požadována schopnost architektonicky myslet; tato schopnost je ovšem u některých studentů nedostatečná;

- metodika učí analyzovat a řešit problémy;

- po pochopení architektury je jednodušší pochopit konkrétnosti a detaily vývoje.

Argumenty proti Architecture-first:

- student nemusí problematice rozumět, pro začátek velmi abstraktní;

- nevhodná a náročná metoda pro studenty, kteří svoji specializaci směřují jiným směrem IT odvětví;

- naučit studenty navrhování komplexních systémů nelze z časových důvodů na školách.

Z výběru toho nejzajímavějšího z dotazníkového šetření je tedy patrné, že více než dvě třetiny dotázaných by preferovalo výuku, kde je kladen důraz na pochopení návrhu architektury, než učení strojového kódování.

8 Doporučení pro výuku studentů

Na základě provedených rozhovorů lze tvrdit, že výuka na českých školách je na každé škole velmi rozdílná.

Z dotazníkového šetření a rozhovorů vyplynulo, že studenti jsou ve výuce příliš zatíženi samotnou syntaxí a na další složky programování se často zapomíná. Studenti mají slabý přehled o trendech v programování a praxe v zaměstnání se většinou velmi liší od výuky. Studentům scházejí soft skills a odborné i praktické znalosti. Na druhou stranu často znají věci, které v budoucnu nebudou potřebovat. Na tyto problémy se zaměřují doporučení v následujících odstavcích.

V kapitole 8.1 jsou uvedena doporučení pro školy, jak přistupovat k výuce programování na českých školách. Veškerá níže uvedená doporučení jsem konzultoval se vzorkem studentů vysokých škol, kteří změny připomínkovali a schvalovali. V další kapitole 8.2 uvádím doporučení pro studenty středních a vysokých škol, kteří se v budoucnu chtějí věnovat programování.

8.1 Doporučení pro školy a budoucí výuku studentů

Jako největší problém vnímám zaměření škol na jednu technologii nebo jazyk, které považují za důležité a díky nimž se profilují od škol ostatních. Již zde ale není brán ohled na různorodost studentů, kdy každého zajímá něco jiného. Jedna z možností je, že každá škola se zaměří na něco jiného. Osobně za východisko považuji zavedení povinného absolvování základů informatiky, které by bylo nutné absolvovat před další profilací studenta. Mezi takové základy, které zmiňovali studenti programování, patří: fungování počítače, znalosti matematiky – převody mezi číselnými soustavami, internetová bezpečnost, fungování sítí, znalosti algoritmizace a další. Poté by si mohl student vybrat své další směřování pomocí volitelných předmětů. Pokud se bude student zabývat tou technologickou oblastí, ke které má blíže, bude do větší míry motivovaný, aktivní a přístupný k přemýšlení.

Mezi metodikami programování nelze jednoznačně určit tu nejvhodnější. Každý přístup má své výhody i nevýhody a může každému studentovi jinak vyhovovat. Metodika Architecture First získala od dotázaných rozporuplné reakce. Celých 64 % dotázaných studentů by tuto metodiku volilo před tradičním přístupem a studenti si výuky architektury cenní. Základním faktorem pro výběr konečného přístupu by mělo být budoucí zaměření studenta dané školy.

Pro přiblížení k budoucí praxí navrhuji několik doporučení. Prvním z nich je spolupráce škol s firmami. Tato spolupráce může mít několik podob. Student se může seznámit s budoucí praxí u jemu zadaných praktických úkolů – v rámci maturitního projektu nebo semestrální práce. Studenti mohou získat představu fungování firem i na pracovních stážích a pro sa-

motné firmy je to příležitost, jak získat budoucí zaměstnance. V dnešní době má mnoho absolventů potíže vstoupit do běžného pracovního procesu. Možnost stáží by tento stav mohla vylepšit a zjednodušit tak nástup do práce.

Možnost spolupráce s firmami může probíhat i zcela odlišně. V teoretické části této práce byly zmíněny některé problémy při výuce programování, např. nedostatečná aprobace nebo zastaralé metody výuky učitelů na některých školách. A i studenti v dotazníkovém šetření připomínkovali samotný přístup k výuce, kdy je student veden tradičním stylem se zaměřením především na teorii a kódování. Kromě aktualizace osnov mohou změnu učinit odborníci z praxe v učitelských řadách. Tito lidé vědí, na co se zaměřit, co je opravdu důležité a jaké budoucí trendy v programování se dají očekávat. Přicházejí s novými nápady a způsoby, jak oživit výuku. Zapojení těchto odborníků při změnách osnov by přineslo přiblížení praxe, a školy by tak poznaly (a vyučovaly) aktuální trendy a technologie používané při vývoji softwaru.

Důležitým bodem pro přiblížení se praxi a seznámení studentů s aktuálními trendy v programování je neustálé doplňování a obnovování znalostí učitelů. Nezřídka jsou zkušenosti některých žáků velmi hluboké a přesahující znalosti učitele. V tomto směru mě zaujal projekt „*Škola učitelů informatiky*„. Jedná se o intenzivní dvoutýdenní školení pro učitele informatiky. Projekt je organizován pod záštitou Univerzity Karlovy a tým lektorů tvoří pracovníci Matematicko-fyzikální fakulty, středoškolští učitelé a odborníci z praxe. Hlavním cílem tohoto projektu je doplnit učitelům vysokoškolské studium informatiky a seznámit je s novinkami tohoto oboru. Tato specifická škola získala pro léta 2016–2018 akreditaci od MŠMT v rámci programu Dalšího vzdělávání pedagogických pracovníků. Tato akreditace usnadňuje ředitelům škol financování účasti jejich zaměstnanců. Více informací o programu http://ksvi.mff.cuni.cz/skola.

Změnit by se měl přístup škol i v trénování soft skills studentů. Některé z těchto dovedností se dají vylepšit stylem výuky. Vyučující může po studentech vyžadovat více týmových prací nebo prezentací na zadané téma. Tyto a další prvky se mohou implementovat nejen do výuky programování, ale i dalších běžně vyučovaných předmětů. Student se tak seznámí s prací v týmu dříve, než během svého pracovního života.

8.2 Doporučení pro studenty

Pro studenta je důležité, aby byl silný v informatických základech, znalostech matematiky a anglického jazyka. Mimo to by si měl student během svých studií zvolit, takový programovací jazyk, který ho bude zajímat. Pro studenta je důležité i samostudium – sledováním technologických novinek, čtení článků z oblasti vývoje nebo účast na konferencích a technologických přednáškách. Existuje i mnoho soutěží, kde mohou studenti porovnat své zkušenosti a posunout své znalosti o krok dál. Příkladem může být Google Code Jam. Jedná se o prestižní soutěž pořádanou společností Google. Cílem soutěže je zapojit studenty do

open source projektů. Díky moderním technologiím je v dnešní době samostudium významně jednodušší. Existuje nespočet způsobů, jak efektivně získat relevantní informace. Při řešení problému stačí zadat dotaz do vyhledávače nebo požádat o radu v programátorské komunitě.

Důležitým bodem studia je ovšem samotná praxe. Taková zkušenost se ve studentově životopisu velmi cení. Většina středních a větších firem tyto praxe pro studenty nabízí nebo jsou minimálně otevřeni společné práci na závěrečných školních projektech. Studenti by měli volit takové týmové projekty, kde se naučí kooperaci, koordinaci a komunikaci v rámci týmu. Pro získání zkušenosti z korporátní kultury doporučuji sledovat webové stránky a nástěnky škol, webové stránky samotných firem nebo specializující se kariérní stránky.

9 Závěr

Někteří absolventi a studenti nemají ani ponětí o trendech v programování, natož co je čeká v reálném světě programování. Studijní plány jsou neflexibilní, staré 10 a více let, což je v rychle se rozvíjejícím IT odvětví skutečný problém. Není to však vina jen samotných škol - firmy a studenti se na této situaci podílejí samozřejmě také. Významným problémem je propojení firem s českým školstvím, neboť takové propojení téměř neexistuje.

Přínos této bakalářské práce je zejména v porovnání problematiky ze dvou stran – učitelů programování z několika českých škol a studentů informatických oborů, taktéž z různých středních a vysokých škol. Je zajímavé uvědomit si rozdíly, které strany vnímají, ač v některých otázkách jsou stejného názoru.

Hlavním přínosem této bakalářské práce je zjištění těchto nedostatků, analýza trendů a předání výsledků školám zaměřených na informatiku, potažmo výuku programování. Pro hodnotnější závěry by však výzkum vyžadoval více respondentů - učitelů, které nebylo lehké získat pro účely této bakalářské práce. Zajímavé by bylo přizvat ve větším měřítku i zaměstnance komerční sféry. Výsledky práce mohou sloužit dále studentům, kteří si uvědomí, na co se mají při svých studií zaměřit, co po nich praxe vyžaduje a že získaná teorie ze školy v dnešním světě zdaleka není všechno.

Výzkumy tohoto druhu jsou velice důležité, ale jejich provedení v malém rozsahu, jednou za pár let není příliš přínosné. V ČR je situace u absolventů programátorů z hlediska jejich schopností velmi špatná. Odborné výzkumy současné situace se vyskytují jen ve velmi malé míře. Tato práce má proto iniciovat další diskuze, jak zlepšit výuku pro snazší pochopení ze strany studentů a jak se více přiblížit praxi.

Terminologický slovník

Termín	Význam [zdroj]
Cloud	Cloud computing charakterizuje poskytování služeb a aplikací servery dostupnými pomocí internetu s tím, že uživatel k těmto službám přistupuje vzdáleně, pomocí webového prohlížeče nebo klienta. [38]
ECTS	Jednotka náročnosti, kterou musí student vynaložit na získání tohoto kreditního bodu. [vlastní definice autora]
Metodika	Pojem metodika představuje obecně pracovní postup. Příkladem metodiky může být v oblasti pedagogiky popis přístupu k výuce.[vlastní definice autora]
mHealth	mHealth je využívání mobilních a bezdrátových zařízení s cílem zlepšit výsledky v oblasti zdraví, zdravotních služeb a zdravotního výzkumu. [39]
Programátor	Programátor je člověk, jehož pracovní činností je vývoj aplikací ve zvoleném programovacím jazyce. Kromě programování má na starost dále testování nebo ladění produktu. [vlastní definice autora]
Programovací jazyk	Programovací jazyk je nástrojem pro komunikaci mezi programátorem, který pomocí něho formuluje postup řešení zvoleného problému a počítačem, který tento postup interpretuje a problém řeší. [40]
Pseudokód	Pomocí pseudokódu popisujeme algoritmy bez závislosti na programovacím jazyku. [vlastní definice autora]
Rating	Rating je jiné označení pro hodnocení. Díky ratingu můžeme určit pořadí subjektů dle různých kritérií. [vlastní definice autora]
RVP	Rámcový vzdělávací program. Definuje úroveň vzdělávání v České republice [vlastní definice autora]
ŠVP	Školní vzdělávací program. Jedná se o učební dokument, splňující požadavky RVP. [vlastní definice autora]

Použité informační zdroje

[1] KIERNAN, J. S. *2016's Best & Worst Entry-Level Jobs*. 2016 [cit. 2017-Leden-19]. Dostupné z: https://wallethub.com/edu/best-entry-level-jobs/3716/.

[2] Český statistický úřad. *Informační ekonomika v číslech* [online]. 2016 [cit. 2017-02-10]. ISBN 978-80-250-2749-3. Dostupné z: https://www.czso.cz/documents/10180/34217534/06300516.pdf/d798efc5-199c-4af3-9e0d-a1802be5e301?version=1.2.

[3] PECINOVSKÝ, R. *Metodika Architecture First*. *Journal of Technology and Information Education: Časopis pro technickou a informační výchovu.* 2013. ISSN 1803-537X.

[4] PECINOVSKÝ, R. *Současné trendy v metodice výuky programování* [online]. 2006 [cit. 2016-12-20]. Dostupné z: http://vyuka.pecinovsky.cz/prispevky/2006-PS_Soucasne_trendy_v_metodice_vyuky_programovani.pdf.

[5] HORÁČKOVÁ, H. *Výuka programování v jazyce Python pro střední školy* [online]. 2015 [cit. 2017-Únor-08]. Dostupné z: https://is.muni.cz/th/39773/fi_m/dp.pdf.

[6] JENS, B. M. E. CASPERSEN a K. MICHAEL. *Reflections on the Teaching of Programming*. Springer, 2008. ISBN 978-3540779339.

[7] CHANG, C. et al. *Computing Curricula 2001* [online]. 2001 [cit. 2016-Prosinec-15]. Dostupné z: https://www.acm.org/education/curric_vols/cc2001.pdf.

[8] BERGIN, J. *Pedagogical Patterns : Advice for Educators*. CreateSpace Independent Publishing Platform, 2012. ISBN 978-1479171828.

[9] Národní ústav pro vzdělávání. *Rámcové vzdělávací programy* [online]. [cit. 2017-Leden-29]. Dostupné z: http://www.nuv.cz/t/rvp.

[10] EUROUNION. *Školské zákony 2010*. Praha: Eurounion, 2010. ISBN 978-80-7317-085-1.

[11] MINISTERSTVO ŠKOLSTVÍ, M. A. T. *Rámcový vzdělávací program pro obor vzdělání 18 – 20 – M/01 Informační technologie* [online]. 2008 [cit. 2016-Prosinec-15]. Dostupné z: http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf.

[12] Integrovaný studijní informační systém. *Evaluace předmětů* [online]. 2016 [cit. 2017-02-27] Dostupné z: https://insis.vse.cz/.

[13] FOJTÍK, R. *Modern approaches to teaching programming. Journal of Technology and Information Education* [online]. 2013 [cit. 2016-Prosinec-02]. ISSN: 1803-537X. Dostupné z: https://jtie.upol.cz/pdfs/jti/2013/01/08.pdf.

[14] Peďák.cz. *Technologie na peďáku*. [online]. 2017 [cit. 2017-Leden-25]. Dostupné z: http://pedak.cz/2017/01/technologie-na-pedaku/.

[15] HROMOVÁ, M. a A. SKOUPÁ. *Děti se budou učit programovat. Novinka ve výuce má být povinná už od první třídy* [online]. Economia, a.s. 2015 [cit. 2016-Listopad-10]. Dostupné z: http://archiv.ihned.cz/c1-64734570-deti-se-budou-ucit-programovat-novinka-ve-vyuce-ma-byt-povinna-uz-od-prvni-tridy.

[16] DERUY, E. *In Finland, Kids Learn Computer Science Without Computers* [online]. The Atlantic Monthly Group. 2017 [cit. 2017-Březen-05]. Dostupné z: https://www.theatlantic.com/education/archive/2017/02/teaching-computer-science-without-computers/517548/.

[17] FRANCLOVÁ, M. a J. BÁRTOVÁ. *Učitelé učitelů* [online]. Praha: Portál, 2008 [cit. 2017-Březen-14]. ISBN 978-80-7367-513-4. Dostupné z: http://www.ucitelske-listy.cz/2009/09/marta-franclova-jana-bartova-motivace.html.

[18] LOKŠOVÁ, I. a J. LOKŠA. *Pozornost, motivace, relaxace a tvořivost dětí ve škole*. Praha: Portál, 1999. ISBN 80-717-8205-X.

[19] PRŮCHA, J. *Moderní pedagogiga*. Praha: Portál, 2009. ISBN 978-80-7367-503-5.

[20] PASCH, M. T. GARDNER a G. LANGER. *Od vzdělávacího programu k vyučovací hodině*. Portál, 2005. ISBN 80-7367-054-2.

[21] PETŘÍK, L. *Zkušený pedagog hovoří o tom, jak čelit spratkům, tělesných trestech i o strašné úrovni …*. OUR MEDIA a.s. 2014 [cit. 2017-Březen-10]. Dostupné z: http://www.parlamentnilisty.cz/arena/rozhovory/Zkuseny-pedagog-hovori-o-tom-jak-celit-spratkum-telesnych-trestech-i-o-strasne-urovni-vysokoskolaku-a-gymnazii-301633.

[22] VOJTĚCH, J. a D. CHAMOUTOVÁ. *Vývoj vzdělanostní a oborové struktury žáků a studentů ve středním a vyšším odborném vzdělávání v …*[online]. Národní ústav pro vzdělávání. 2017 [cit. 2017-Březen-29]. Dostupné z: http://www.nuv.cz/uploads/Vzdelavani_a_TP/VYVOJ2016_pro_www_fin.pdf.

[23] Ministerstvo školství mládeže a tělovýchovy. *Jednotné přijímací zkoušky na střední školy s maturitou* [online]. [cit. 2017-Březen-12]. Dostupné z: http://www.msmt.cz/ministerstvo/jednotne-prijimaci-zkousky-na-stredni-skoly-s-maturitou.

[24] ČTK. České noviny. *Senát schválil reformu financování škol beze změn* [online]. 2017 [cit. 2017-Březen-10]. Dostupné z: http://www.ceskenoviny.cz/zpravy/senat-schvalil-reformu-financovani-skol-beze-zmen/1458673.

[25] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Computer Science Curricula 2013: Guidelines for Undergraduate Degree Programs in Computer Science* [online]. ACM/IEEE-CS, 2013 [cit. 2017-10-12]. ISBN: 978-1-4503-2309-3. DOI: 10.1145/2534860. Dostupné z: https://www.acm.org/education/CS2013-final-report.pdf.

[26] BERGIN, J. *Pedagogical Patterns : Advice for Educators*. CreateSpace Independent Publishing Platform, 2012. ISBN 978-1479171828.

[27] PETTY, G. *Moderní vyučování*. Portál, 2006. ISBN 978-80-7367-427-4.

[28] RÝDLO, L. *Programovací jazyky pro výuku programování na SŠ* [online]. 2012 [cit. 2017-Leden-20]. Dostupné z: https://is.muni.cz/th/139514/fi_m/dp.pdf.

[29] TIOBE. TIOBE. *TIOBE Index for March 2017* [online]. 2017 [cit. 2017-Březen-07]. Dostupné z: https://www.tiobe.com/tiobe-index/.

[30] Wikipedia the free encycloped. *Java (programovací jazyk)* [online]. [cit. 2017-Únor-27]. Dostupné z: https://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk).

[31] BENEŠOVSKÁ, M. *Devět nejžádanějších programovacích jazyků roku 2016* [online]. 2016 [cit. 2017-Březen-02]. Dostupné z: http://www.helloworld.cz/devet-nejzadanejsich-programovacich-jazyku-pro-rok-2016/.

[32] RICHARDSON, C. *Vendor Landscape: The Fractured, Fertile Terrain Of Low-Code Application Platforms [online]*. 2016 [cit. 2017-Březen-07]. Dostupné z: http://informationsecurity.report/Resources/Whitepapers/0eb07c59-b01c-4399-9022-dfc297487060_Forrester%20Vendor%20Landscape%20The%20Fractured,%20Fertile%20Terrain.pdf.

[33] RICHARDSON, C. *The Forrester Wave™: Low-Code Development Platforms, Q2 2016 [online]*. 2016 [cit. 2017-Březen-07]. Dostupné z: https://www.forrester.com/report/The+Forrester+Wave+LowCode+Development+Platforms+Q2+2016/-/E-RES117623.

[34] AgilePoint NX | Low-Code aPaaS [online], 2017 [cit. 2017-Leden-20]. Dostupné z: http://agilepoint.com/.

[35] BALOG, M. et al. *DeepCoder: Learning to write programs*. 2017 [cit. 2017-Březen-10]. Dostupné z: https://openreview.net/pdf?id=ByldLrqlx.

[36] WAYNER P. *Trendy v programování: Co je in a co je out?* [online]. IDG Czech Republic,a. s. 2015 [cit. 2016-Listopad-15]. Dostupné z: http://computerworld.cz/software/trendy-v-programovani-co-je-in-a-co-je-out-51749.

[37] HONIGMAN, B. The Next Web. *How to decide between a responsive website or a native mobile app* [online]. 2014 [cit. 2017-Leden-05]. Dostupné z: https://thenextweb.com/dd/2014/02/08/decide-responsive-website-native-mobile-app/.

[38] BusinessIT. *Cloud Computing pro každého: Jasně a stručně* [online]. 2011 [cit. 2017-Únor-25]. Dostupné z: http://www.businessit.cz/cz/cloud-computing-vysvetleni-jasne-a-strucne.php.

[39] himss. *Definitions of mHealth* [online]. 2012 [cit. 2017-Březen-21]. Dostupné z: http://www.himss.org/definitions-mhealth?ItemNumber=20221.

[40] Wikipedia the free encyclopedia. *Programovací jazyk* [online]. [cit. 2017-Únor-24]. Dostupné z: https://cs.wikipedia.org/wiki/Programovac%C3%AD_jazyk.

Seznam obrázků a tabulek

Seznam obrázků

Seznam tabulek