СЪВРЕМЕННИ АЛТЕРНАТИВНИ ПОДХОДИ ЗА ПРЕПОДАВАНЕ НА УВОД В ПРОГРАМИРАНЕТО

Ивайло Дончев

1. Увод

Изучаването на дисциплината „Програмиране“ включва както усвояването на технологиите, така и тяхната научна основа. Към технологиите спадат инструменти, техники и стандарти, които ни позволяват да програмираме. Научната основа съдържа обширната теория, с чиято помощ можем да разберем програмирането. Тази основа трябва да бъде и практическа, за да обясни технологиите и да ги направи приложими за работещия програмист.

Да се преподава програмиране означава да се преподава и наука, и технология. Изненадващо, анализът на учебната литература, направен в [31], открива, че това почти никога не се прави така. В проучените учебници са открити три различни начина за преподаване на програмиране:

– **като занаят**: Това е най-често срещаната ситуация. Преподава се една единствена програмна парадигма и нейната реализация в конкретен език за програмиране;

– **като раздел от математиката**: При този подход научните основи или са ограничени до лимитиран език, или са прекалено фундаментални, за да бъдат от практическа полза;

– **от гледна точка на лежащите в основата понятия:** Концепциите се въвеждат и развиват, като се започва с по-простите и постепенно се преминава към по-сложните. Това дава на студентите добро разбиране за практическото програмиране. Недостатъците произтичат от слабата формална семантика и пропускането на някои важни понятия.

Известни са множество реализации на различни подходи за изграждане на уводни курсове по програмиране. Постоянно се докладват и нови резултати от успешни педагогически експерименти. Малко са обаче тези реализации, които са възприети и се прилагат не само от техните откриватели. Повечето решават специфични проблеми в конкретна обучаваща институция или програма. Като класически са се утвърдили три подхода [3, с. 18]:

– *imperative-first*: реализира увода в програмирането, следвайки историческото развитие на парадигмите и езиците за програмиране – започва с основните понятия от процедурното програмиране, след което преминава към обектите. Изучаваните теми в уводния курс включват типове данни, управляващи конструкции, функции, масиви, файлове, както и техники за тестване и дебъгване на програмите;

– *objects-first*: най-популярният в последните 10-12 години подход. От самото начало се започва с нотацията на обектите и наследяването, за да свикнат студентите с тази идея. Чак след това се въвеждат традиционните процедурни конструкции, но винаги в контекста на приложението им за реализиране на обектния модел. За по-горни курсове са оставени алгоритмите, структурите от данни и детайлното изучаване на основите на софтуерното инженерство;

– *functional-first*: въвежда алгоритмичните понятия с помощта на прост, функционален език (SML, Scheme, Haskell), след което се прави преход към обектно-ориентиран език (най-често Java, C++ или C#).

В [3] има описани още три класически подхода за уводен курс, но в този случай става въпрос на „Увод в информатиката“, а не в програмирането. Всеки от тези подходи има своите предимства и недостатъци. Развитието на технологиите и езиците за програмиране от една страна, както и традиционните проблеми, неизменно съпътстващи обучението по програмиране, от друга, са предпоставка за постоянното търсене на нови, алтернативни подходи, които да избегнат недостатъците на старите. За това в тази статия ще анализираме някои от набиращите популярност нови подходи за увод в програмирането.

2. Подходът *concepts-first*

Принципите, залегнали в основата на този подход са [11]:

1) При въвеждането на новите програмни концепции се изхожда от ежедневния опит на студентите.

2) Позволява се на студентите да работят в една и съща предметна област по-дълго време, преди да се добави нова.

3) Разделяне на концепциите от синтаксиса на езика.

Тези принципи са повлияни от конструктивистката теория за ученето. Очакваните предимства от прилагането на подхода, според [11] са:

– повишаване на ефективността при усвояване на синтаксиса на езика – с добра концептуална основа езикът се изучава по-бързо;

– повишено разбиране на програмните конструкции, базирано на по-доброто осъзнаване на лежащите в основата им принципи;

– повишено ниво на комфорт на студентите.

Нивото на комфорт често се пренебрегва, но според [34] това е най-точният индикатор за успех във всеки курс по компютърни науки.

Като *concepts-first* можем да определим и подхода, предложен в [27] и [31]. Той е приложим за всички програмни парадигми, включително паралелно програмиране и използва специално проектиран за целта изчистен език (*Kernel language*) – подмножество на мултипарадигмения език *Oz3*. *Kernel* позволява да се направи въведение в основните програмни парадигми плюс набиращите сила сега такива още в уводния курс. Парадигмите се появяват естествено, в зависимост от програмните концепции, които се използват за конкретния решаван проблем. Това позволява на студентите да възприемат парадигмите в по-обща рамка, чрез връзките между тях и как взаимно се допълват.

Макар, че този подход изглежда много достъпно и привлекателно, не е широко разпространен и се счита за екзотичен.

3. Подходът *fundamentals-first*

Друг интересен подход е *fundamentals-first* (първо основите). За негов автор е сочен Daniel Liang с неговия популярен учебник по програмиране на Java [20], който вече има 8 издания. Идеята на този подход е да се започне първо с основните концепции и принципи, след което да се премине към решаването на проблеми (*problem solving*) и обектно-ориентиран дизайн. Под „основни“ тук се имат предвид процедурните концепции, така че този подход можем да определим като модификация на класическия *imperative-first*. Разликата с *imperative-first* е, че тези концепции, поне според идеята на Liang, трябва да се въведат преди каквито и да са специфични за езика особености. По това подходът прилича на описания по-горе *concepts-first*. Разликата с него е, че *fundamentals-first* препоръчва да се започне от най-лесните теми и постепенно да се повишава нивото на сложност. Според [30] това ще помогне на студентите да бъдат по-уверени във възможностите си.

Още щом разлистим учебника [20], обаче, откриваме, че „основните концепции и принципи“ са всъщност не концепции и принципи на програмирането (или, както ги определя [36], на ООП), а по-скоро на компютърната наука – организацията и работата на компютъра, операционната система и транслацията на програмите. Тези концепции са

определени като лесни за възприемане от студентите, което също е спорен въпрос. В крайна сметка въведението е кратко, в стил *hardware-first*, след което се преминава към класически *imperative-first* подход с малко по-ранно въвеждане на обектите и графичния интерфейс.

Учебен курс, следващ подхода на Liang трябва отрано да запознае студентите с важни техники за решаване на проблеми, използвайки език за програмиране, след което да измести фокуса към обектно-ориентираните концепции и визуалното програмиране.

Важен елемент на подхода *fundamentals-first* са примерите. Това е и най-силната му страна. Концепциите се усвояват чрез тези примери и, може би, на това се дължи успехът на подхода. Известен факт е, че е много трудно да се намерят подходящи примери за демонстриране на обектно-ориентираните концепции в ограниченото време на един учебен курс. Примерите в [20] определено са такива. Те са два вида:

– малки, прости и стимулиращи: демонстрират концепциите и техниките;

– обширни примери от реални приложения: излизат извън рамките на традиционните математически-ориентирани задачи, които откриваме в повечето учебници и помагала, например [38] и [37].

По-големите примери са оформени като казуси и са съпътствани от дискусии и обяснения на програмния код ред по ред, така че да не се затруднят начинаещите студенти. Могат да се ползват както за лабораторни занятия, така и за самостоятелна работа.

Подходът *fundamentals-first* е замислен и реализиран за езика Java, но лесно може да се приложи и за други обектно-ориентирани езици, например *C++*. През януари 2012 г. излезе и учебникът [21] на Liang, който имплементира същия подход за езика Python. Отчитайки неговите особености, графичните възможности са изтеглени малко по-рано в учебния курс. Интересно решение е предложено и в [16], където процедурните аспекти се преподават на Python, след което се преминава към ООП на Java.

4. Подходът *GUI-first*

Подходът *GUI*[1]*-first* поддържа идеята, че обучението по програмиране трябва да започне директно със създаването на приложения с графичен потребителски интерфейс. В наши дни огромната част от

[1] GUI (Graphical User Interface) – Графичен потребителски интерфейс

приложенията са точно такива и за това уменията за разработването им се считат за също толкова важни, колкото и правилното разбиране на процедурните и обектно-ориентираните концепции [8]. В практиката най-често за този подход се използват Java и Visual Basic – за тях има най-много методически материали. Поддръжниците на *GUI-first* подхода твърдят, че студентите усвояват по-лесно концепциите на ООП, ако първо свикнат да използват обектите (да им изпращат съобщения и да проследяват поведението им), преди да започнат да създават нови класове [28]. Чак след това паралелно се въвеждат теоретичните основи на процедурното и обектно-ориентираното програмиране.

В подкрепа на този подход можем да изтъкнем следните предимства:

– приложенията с графичен интерфейс са по-интересни за студентите (и учениците). Те са свикнали да ги ползват и очакват точно такива приложения да се научат да разработват. Това е добър начин да се задържи тяхното внимание;

– повечето езици са проектирани именно за създаването на графични приложения;

– кодът на GUI програма е труден за разбиране от студенти, които нямат предварителна подготовка по ООП, но въпреки това могат да се направят множество интересни GUI програми, без да е необходимо да се обсъжда генерирания от средата код в детайли;

– има достатъчно литература и дидактически материали, подпомагащи такъв подход.

Най-сериозният недостатък е, че много дълго се отлага във времето изграждането на алгоритмично мислене. Късното усвояване на процедурните концепции води до трудности при имплементирането на алгоритмите в програмата. В учебния курс е трудно да се отдели нужното внимание и на принципите на обектно-ориентирания дизайн, което пък е фактор за проблеми при проектирането на структурата на програмата – изборът на правилни класове, обекти и връзки между тях. В курсовете от средно и високо ниво обикновено се включват теми от дизайна и проектирането, но това може да се окаже недостатъчно за преодоляването на пропуските от уводния курс. За това едно добро решение е да се включи на междинно ниво специализиран курс по ООАП.

Друг проблем е, че студентите могат да останат с впечатлението, че програмирането е лесна работа и да подценят дисциплината. В този

случай преподавателят трябва да успее да ги ангажира в достатъчна степен чрез по-обемни домашни и курсови работи, изискващи повече внимание.

5. Подходът *games-first*

Games-first подходът цели да провокира въображението на първокурсниците и да задържи интереса им по начин, който ги вълнува, като в същото време им даде солидна основа в областта на програмирането и информатиката. Неговите идеи се явяват естествено продължение на идеите на *GUI-first* подхода. Тезата е, че е възможно въведение в компютърните науки чрез програмиране на игри. Появата на този подход е свързана, от една страна, с чувствителното намаляване на интереса към компютърните специалности в първото десетилетие на XXI век [32] [2, с. 9] и необходимостта да се намери начин да се мотивират младите хора да изберат професия в това направление. Подобни са и съображенията в CS2008 игрите и софтуерът за забавление да се разглеждат като част от областта знания „Интелигентни системи [2, с. 17]. От друга страна, разрастването на индустрията на компютърните игри, в частност за мобилни устройства, изисква подготвени кадри. В резултат на това доста институции са създали специалности, чиито учебни планове са организирани около разработването на игри [4] [26] [13]. Обучението в тази насока започва още от уводния курс, като след него знанията се задълбочават в по-горни курсове по „Компютърна графика“, „Програмиране на игри“ и „Операционни системи“. Това има смисъл, защото в игрите са заложени много алгоритми (например за определяне на следващ ход или победител). Особено подходящи като предмет за изучаване в курсовете по „Изкуствен интелект“ са логическите игри.

Има голямо разнообразие от езици и среди, които се използват в уводни курсове, следващи стратегията *games-first*. Имплементацията на подхода, описана в [19] използва ActionScript. Тъй като мнозинството първокурсници имат слаба или нямат никаква подготовка по програмиране, най-лесно е да се започне с Macromedia Flash – запознаване със средата за разработка на Flash и синтаксиса на ActionScript. Студентите бързо се научават да създават и местят по екрана графични обекти. Важно е да се започне с прости обекти, за да могат студентите веднага да видят резултата и да проверят коректността на работата си. Чрез

простите графични примери на ActionScript могат да се въведат и усвоят основни процедурни понятия (променлива, присвояване, условен преход, цикъл, масив, функция) и да се добие първоначална представа за обектно-ориентирани (клас, обект, метод, съобщение). Използват се и елементи на събитийното програмиране (обработват се събития, свързани с мишката, клавиатурата и прозорците).

Подходът *games-first* може да се реализира и със смяна на езика и средата по време на курса, например от Flash на C++ или Java, за да могат вече по-задълбочено да се разгледат понятия като указатели, псевдоними, динамична памет, рекурсия, структури от данни. Този преход е възможен, тъй като елементарният синтаксис на тези езици е близък.

Друга възможност за език е Python, който добива все по-голяма популярност в уводните курсове, следващи не само *games-first* подход. Успешна реализация под формата на последователност от лабораторни занятия с Python е докладвана в [33].

6. Подход с програмиране за мобилни устройства

Мобилните устройства се превърнаха в част от всекидневието на съвременните студенти. Разработката на приложение за мобилен телефон под формата на курсова работа може да помогне на първокурсниците да видят веднага връзката между науката и реалните технологии. В сравнение с програмирането на игри за компютър, разработката им за мобилни устройства е по-малко сложно и е напълно по силата на начинаещи. Опит с прилагането на такъв подход откриваме в [18]. В [1] дори е предложен цялостен уводен курс на базата на програмиране за мобилни устройства. Според автора мотивацията е основният фактор за успеха на курса. Самият курс е разделен на две части: В първата се използва инструмент (Google App Inventor), позволяващ много високо ниво при разработката на приложения (визуално програмиране). По този начин синтаксисът не е доминиращ в учебния процес и фокусът се измества към уменията за решаване на проблеми. У студентите обаче се изгражда представа за основни понятия като променливи, декларации, условни оператори, цикли. Тези понятия се въвеждат по-детайлно във втората част на курса, тоест подходът е спираловиден. Практическата работа включва разработване на приложения с Android SDK.

Уводен курс по програмиране за мобилни устройства (конкретно за BlackBerry смартфони) предлага и [23]. В него, освен софтуерните

инструменти, се използва и специално разработен комплект в помощ на преподавателите, решили да интегрират мобилното програмиране в учебните си програми – CMER Academic Kit [10]. Този комплект се разпространява безплатно и съдържа учебни материали (слайдове с лекции, упражнения, ръководства, тестове и задачи). Акцентира на Java ME, разработка на приложения за BlackBerry и web-базирани мобилни приложения.

7. Подходът *model-first*

Този подход представлява вариант на *objects-first*, при който се отделя повишено внимание на концептуалното моделиране. Може да се счита, че названието му е дадено от Bennedsen и Caspersen в статията им [6]. Те предлагат да се добави концептуално моделиране към уводния курс. Целта е да се усвоят обектно-ориентираните концепции чрез моделиране и превеждане на моделите към генерични кодови шаблони.

Причините да се търсят промени се коренят в незадоволителните резултати от прилагането на *objects-first* подхода. Според [7] проблемът се дължи на факта, че дори да се преподава по *objects-first* подход, акцентът отново пада върху детайлите по имплементацията (програмните елементи), вместо върху усвояването на обектно-ориентираното мислене като инструмент за разработка на софтуер. За това решението е да се започне с моделиране и дизайн, което ще даде на студентите по-широк поглед върху обектно-ориентираните концепции, извън конкретиката и техническите детайли на езиците за програмиране.

Идеята да се добавят елементи на моделиране в курс по програмиране е свързана с особеностите на обектно-ориентираните програми. Те не се разглеждат като „алгоритми + структури от данни“ [35], а като „физически модели, симулиращи поведението на реална или имагинерна част от света“ [22, с. 16]. Ударението пада върху модела – обектно-ориентираната програма е модел. Този модел може да се разглежда от различни перспективи на различни нива на абстракция и детайлност. Концептуалният модел е от най-високото ниво на абстракция (съответно, най-ниско ниво на детайлност) – концептуалното ниво, и е неформален. Той описва ключовите абстракции (обекти) от проблемната област и връзките между тях. Второто ниво на абстракция (ниво на спецификациите) е по-детайлно и дава подробен план на проблема и неговото

решение, а третото ново е имплементацията на решението на конкретен език за програмиране.

По принцип учебният курс по програмиране трябва да отделя повече внимание и усилия на имплементацията (писането на код). Това не означава, че останалите нива трябва да се пренебрегват. Когато става въпрос за уводен курс, ролята на концептуалното моделиране е още по-важна, тъй като това е подходът към намирането на програмни решения на проблемите. Тук обаче има възможност да се изпадне в крайност – ако студентите използват инструмент за моделиране (*case tool*), който генерира готов код е много вероятно да имат проблеми с писането на собствен такъв. Другата крайност е да се отдели внимание само на детайлите по имплементацията. Тогава студентите ще овладеят синтаксиса на езика, но няма да усвоят концептуалните основи на програмната парадигма. За това подходът *model-first*, разработен в рамките на проекта COOL [12], предлага учебен курс, при който чрез включване на концептуалното моделиране са балансирани трите гледни точки към езика за програмиране, определени в [17] и съответстващи на трите нива на абстракция:

– инструктиране на компютъра (кодиране): езикът за програмиране се разглежда като машинен език от високо ниво. Фокусът е върху аспекти, свързани с изпълнението на програмата – състояние на паметта, последователност на изпълнение;

– управление на описанието на програмата: езикът се използва за преглеждане и разбиране на програмата. Фокусът е върху аспекти като видимост, капсулиране, модулност, разделно компилиране;

– концептуално моделиране: езикът за програмиране служи за изразяване на концепциите.

Като основни предимства на този подход се изтъкват [5, с. 119]:

– систематичен подход към програмирането;

– по-дълбоко разбиране на процеса на програмиране;

– фокусиране върху основните програмни концепции, вместо върху езиковите конструкции на конкретен език за програмиране.

Така описан, подходът *model-first* има всички предпоставки да бъде успешен – набляга на концептуализацията на обектно-ориентираните модели, както и на процеса на разработка на софтуер, свързан с тях, преди да се навлезе в детайлите на семантиката и синтаксиса на

езика за програмиране. Въпреки това, този подход почти не е прилаган извън рамките на проекта COOL на университета в Осло. Няма и данни как се отразява това на процеса на учене в дългосрочен план, но тази критика може да се отправи към повечето съвременни подходи.

Подробно разписани последователности от учебни дейности, които трябва да се провеждат на всеки етап от процеса на обучение по подхода *model-first* и могат да ориентират преподавателите как да ги използват в своите курсове, могат да бъдат открити в [14]. Те включват:

– последователността и разпределението на времето на всяка фаза;

– учебни дейности, които студентите и преподавателите трябва да извършват на всеки етап;

– методи на взаимодействие учител-студент и студент-студент (например, „лице в лице“, асинхронно, текстово или звуково ориентирано);

– методи на взаимодействие между студентите и учебните ресурси и инструменти.

8. Подходът *design-first*

Друга вариация на *objects-first* подхода е така наречения *design-first* подход. Той е много близък до *model-first* и залага на идеята, че студентите научават най-добре това, което учат първо. За това те трябва да започнат с придобиване на умения за решаване на проблеми в обектно-ориентиран стил, тоест, с основите на обектно-ориентирания анализ и проектиране. „Студенти, които научат как да откриват изискванията на проблема, как да го разлагат на „управляеми“ части и как да проектират решението са по-добре подготвени да прилагат концепциите, които ще изучават и могат по-добре да разберат тези концепции чрез тяхното прилагане“ [25].

Учебното съдържание на *design-first* курс трябва да включва елементи на UML за онагледяване на техниките за решаване на проблеми. Задължително е използването и на помощни инструменти, както за UML моделиране и генериране на код от диаграмите, така и за визуализиране на изпълнението на програмата. Полезно е използването и на външни библиотеки, например за работа с графични обекти.

Имплементацията на подхода, предложена в [25] разпределя учебния материал в 6 основни раздела и покрива: UML диаграми на случаи на употреба и на класове; понятията обект и клас; откриването на атри-

бути и методи и проектиране на клас; процедурни концепции (присвояване, условни оператори, цикли, текстов вход и изход); графичен потребителски интерфейс и събитийно програмиране. По време на курса студентите разработват стъпка по стъпка средни по обем приложения, моделиращи ситуации от реалния живот. Първите два раздела са предвидени за въвеждане на идеите на софтуерното инженерство. Класовете и обектите се разглеждат само като абстрактни понятия, но студентите могат да ги „пипнат“, използвайки библиотека с готови обекти (в случая, геометрични фигури, които те създават, местят и променят). Елементите от процедурното програмиране са необходими, за да могат да се имплементират проектираните методи на класовете. Тяхното въвеждане се прави постепенно и започва в третия раздел. Усвоените чрез по-прости примери процедурни концепции веднага се прилагат в разработвания проект. Интересното в тази имплементация е, че въпреки наличните средства и инструменти, първите създавани проекти са с текстов интерфейс. Операторите за цикъл попадат в пети раздел. При неговото завършване трябва да е готова и изтествана текстовата версия на проекта. Графичните приложения попадат в последния, шести раздел, наред с теми за напреднали като многонишково програмиране.

Въпреки докладваните добри резултати, слабост на тази имплементация на подхода, е че е използваните примери всъщност не представят добър обектно-ориентиран дизайн. На практика става въпрос за един голям клас, който предоставя цялата функционалност. Няма го характерния за ООП разпределен контрол. Много е трудно да се намерят подходящи примери, които хем да представят набор сътрудничещи си класове, хем имплементирането им да не е прекалено сложно за начинаещи.

Характерен проблем за *design-first* и *model-first* подходите е, че за да могат студентите да мислят за дизайна, те трябва да имат известна представа и от неговото реализиране на ниво език за програмиране. Дори в началото да не се интересуват от имплементацията на методите, самото определяне на функционалността на класовете предполага известни познания за това как се реализира тя.

Всички подходи, които залагат първо на дизайна изглеждат като естествено възникнали от принципите на заложения в тях обектен модел.

Този модел е близък до моделите в реалния свят, но изграждането на разбиране за абстрактни понятия е възможно само след натрупването на голям опит и добро познаване на по-конкретни понятия [15]. С други думи, няма лесен начин за придобиване умения за така важното за всеки програмист абстрактно мислене.

9. Заключение

Обучението по програмиране, традиционно е съпътствано от множество трудности. Независимо от повече от 50-годишния опит, то все още е голямо предизвикателство [9, p. 111], особено що се отнася до уводните курсове [24]. Много изследователи определят изучаването на програмиране като изключително трудна дейност [29]. Причините за тази ситуация имат дълбоки корени. Безспорно е само, че програмирането наистина е сложна сплав от изкуство и наука – трудно да се прави и още по-трудно да се преподава [5]. Част от проблема се дължи на това, че въпреки многото научни изследвания, не е открито достатъчно за програмирането и неговото преподаване. Нужни са повече изследователска работа, експерименти и дискусии. Разгледаните в настоящата статия алтернативни подходи за увод в програмирането безспорно са крачка напред в тази посока, но тепърва им предстои да се доказват. Необходимо е да продължи тестването им в практиката и приспособяването им към конкретните условия и изисквания на образователната система и индустрията.

ЛИТЕРАТУРА

1. *Abukmail, A.* A Dual Abstraction Mobile Computing Framework for Developing Programming Skills, In Proceedings of the 2011 ASEE Southeastern Section Conference 10-12 April 2011, 2 – 82.

2. ACM/IEEE CS2008 Review Taskforce, Computer Science Curriculum 2008: An Interim Revision of CS 2001, Technical report, ACM/IEEE CS, 2008.

3. ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 2001*, Journal on Educational Resources in Computing, Volume 1 Issue 3es, Fall 2001, ACM New York, NY, 2001.

4. *Argent, L., Depper, B., Fajardo, R., Ghertson, S., Leutenegger, S., Lopez, M., Rutenbeck, J.* Builidng a Game Development Program, IEEE Computer, Vol 39, no 2, 2006, pp. 52 – 61.

5. *Bennedsen, J., Caspersen M., Kölling, M.* Reflections on the Teaching of Programming. Methods and Implementations, Springer, 2008, 261 pages.

6. *Bennedsen, J., Caspersen, M.* Programming in context – a model-first approach to CS1, In Proceedings of the 35th SIGCSE technical symposium on Computer science education, ACM, New York, 2004, pp. 477 – 481.

7. *Berge, O., Borge, R., Fjuk, A., Kaasbøll, J., Samuelsen, T.* Learning Object-Oriented Programming, available at http://halshs.archives-ouvertes.fr/hal-00190184_v1

8. *Bishop, J., Horspool, N.* Developing principles of GUI programming using views, ACM SIGCSE Bulletin, Volume 36, Issue 1, March 2004, pp. 373 – 377.

9. *Caspersen, M., Bennedsen, J.* Instructional design of a programming course: A learning theoretic approach, Proceedings of the Third International Workshop on Computing Education Research, September 15-16, Atlanta, Georgia, USA, 2007, pp. 111 – 122.

10. CMER Academic Kit, http://cmer.uoguelph.ca/kit.html

11. *DuHadway, L., Clyde, S., Recker, M., Cooley, D.* A concept-first approach for an introductory computer science course, Journal of Computing Sciences in Colleges, Volume 18 Issue 2, December 2002, pp. 6 – 16.

12. *Fjuk, A., Kaasbøll, J, Karahasanovic, A.* Comprehensive Object-Oriented Learning: The Learner's Perspective, Informing Science, 2006, 242 pages

13. *Fullerton, T.* Play-centric Games Education, IEEE Computer, Vol 39, no 2, 2006, pp. 36 – 42.

14. *Georgantaki, S., Psaromiligkos, Y., Retalis, S., Dendrinos, V., Adamopoulos, D.* Developing a blended learning strategy for teaching Object-Oriented Programming using the "Model First" approach, In proceedings of International Conference on the state of Informatics Education in Europe, Thessaloniki, November 2007, pp. 87 – 96.

15. *Hadar, I., Hadar, E.* An Iterative Methodology for Teaching Object Oriented Concepts, Informatics in Education, 2007, Vol. 6, No. 1, pp. 67–80.

16. *Jayal, A., Lauria, S., Tucker, A., Swift, S.* Python for teaching introductory programming: A quantitative evaluation, ITALICS 10 (1), 2011, pp. 86 – 90.

17. *Knudsen, J., Madsen, O.* Teaching Object-Oriented Programming Is More than Teaching Object-Oriented Programming Languages, In Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag, London, 1988, pp. 21 – 40.

18. *Kurkovsky, S.* Engaging students through mobile game development, ACM SIGCSE Bulletin – SIGCSE '09, Volume 41, Issue 1, March 2009, pp. 44 – 48.

19. *Leutenegger, S., Edgington, J.* A games first approach to teaching introductory programming, ACM SIGCSE Bulletin, Volume 39, Issue 1, March 2007, pp. 115 – 118.

20. *Liang, D.* Introduction To Java Programming. Comprehensive (8th Edition), Prentice Hall, 2010, 1368 pages

21. *Liang, D.* Introduction to Programming Using Python, Prentice Hall, 1-st edition, 2012, 576 pages

22. *Madsen. O., Moller-Pedersen, B., Nygaard, K.* Object-Oriented Programming in the Beta Programming Language, ACM, 1993, 400 pages

23. *Mahmoud, Q.* A mobile web-based approach to introductory programming, In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, ACM, New York, 2011, pp. 334 – 334.

24. *Meyer, B.* The Outside-In method of teaching introductory programming, in Perspective of System Informatics, Proceedings of fifth Andrei Ershov Memorial Conference, Akademgorodok, Novosibirsk, 9-12 July 2003, ed. Manfred Broy and Alexandr Zamulin, Lecture Notes in Computer Science 2890, Springer-Verlag, 2003, pages 66 – 78.

25. *Moritz, S., Blank, G.* A Design-First Curriculum for Teaching Java in a CS1 Course, ACM SIGCSE Bulletin, Volume 37 Issue 2, June 2005, ACM, New York, pp. 89 – 93

26. *Murray, J., Bogost, I., Mataes, M., Nitsche, M.* Game Design Education: Integrating Computation and Culture , IEEE Computer, Vol 39, no 2, 2006, pp. 3 – 51.

27. *Reinfelds, J.* Programming as an engineering discipline, Frontiers in Education, 2002. FIE 2002. 32nd Annual, vol.2, pp. F2G-5 – F2G-9

28. *Roumani, H.* Practice What You Preach: Full Separation of Concerns in CS1/CS2, ACM SIGCSE Bulletin Volume 38, Issue 1, March 2006, pp. 491 – 494.

29. *Simon, S., Fincher, S., Robins, A., Baker, B, Box, I., Cutts, Q., de Raadt., M., Haden, P., Hamer, J., Hamilton, M., Lister, R., Petre, M., Sutton, K., Tolhurst, D., Tutty, J.* Predictors of success in a first programming course. ACM International Conference Proceeding Series; Proceedings of the 8th Australian conference on Computing education, January 16–19, 2006, Hobart, Tasmania, Australia, pp. 189 –196.

30. *Smolarski, D.* A first course in computer science: Languages and goals, In Teaching Mathematics and Computer Science 1(1), 2003, pp. 137 – 152.

31. *Van Roy, P., Haridi, S.* Teaching Programming with the Kernel Language Approach, Workshop on Functional and Declarative Programming in Education (FDPE02), Available at http://www.sics.se/~seif/

32. *Vegso, J.* Drop in CS Bachelor's Degree Production, Computing Research News, Vol 18, No 2, March 2006, p. 5, available at: http://archive.cra.org/CRN/issues/0602.pdf

33. *Wang, H.* Teaching CS1 with Python GUI Game Programming, IAENG TRANSACTIONS ON ENGINEERING TECHNOLOGIES: Volume 4: Special Edition of the World Congress on Engineering and Computer Science-2009. AIP Conference Proceedings, Volume 1247, 2010, pp. 253 – 260.

34. *Wilson, B., Shrock, S.* Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors, SIGCSE 2001, pp. 184 – 188.

35. *Wirth, N.* Algorithms + Data Structures = Programs, Prentice-Hall, 1976, 366 pages.

36 *Yau, J., Joy, M.* Introducing Java: A Case for Fundamentals-first, In proceedings of EISTA '04: International Conference on Education and Information Systems: Technologies and Applications, vol, 2, 2004, pp. 229 – 234.

37. *Тодорова, М., Армянов, П., Петкова, Д., Георгиев, К.* Сборник от задачи по програмиране на C++. Първа част – Увод в програмирането. София, ТехноЛогика, 2008. 357 с.

38. *Тодорова, М.* Обектно-ориентирано програмиране на базата на езика C++. София, Ciela, 2011. 395 с.

СЪВРЕМЕННИ АЛТЕРНАТИВНИ ПОДХОДИ ЗА ПРЕПОДАВАНЕ НА УВОД В ПРОГРАМИРАНЕТО

ИВАЙЛО ДОНЧЕВ

Резюме

Изграждането на уводен курс по програмиране винаги е било трудна задача, но в наши дни към традиционните предизвикателства се прибавят и много нови, свързани предимно с развитието на технологиите, изискванията на индустрията и мотивацията на студентите. За това преподавателите и изследователите постоянно търсят нови педагогически подходи. Повечето от тях, обаче, решават само специфични проблеми в конкретна обучаваща институция или програма и се прилагат само от откривателите им. За това в тази статия правим критичен анализ на седем подбрани съвременни алтернативни подхода за увод в програмирането, за които считаме, че имат шансовете да се наложат като класически.

Ключови думи: увод в програмирането, педагогически подходи, педагогика.

CONTEMPORARY ALTERNATIVE APPROACHES TO TEACH INTRODUCTORY PROGRAMMING

IVAYLO DONCHEV

Summary

Construction of an introductory course in programming has always been difficult task, but new challenges have been added to the traditional ones, mostly related to the technology development, industry requirements and student motivation. So teachers and researchers are constantly looking for new pedagogical approaches to deal with this. Most of these approaches, however, solve only specific problems in a specific educational institution or program and are used only by their inventors. This article makes a critical analysis of seven selected contemporary alternative approaches to teaching introductory programming which have good chances to be imposed as a classical.

Keywords: introductory programming, teaching approaches, pedagogy.